

**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL
DE HUAMANGA**

ESCUELA DE POSGRADO

**UNIDAD DE POSGRADO DE LA FACULTAD DE
CIENCIAS DE LA EDUCACIÓN**



TESIS:

**Aprendizaje basado en problemas para mejorar las
competencias de programación en los estudiantes de
Ingeniería de Sistemas, UNSCH 2024**

Para optar el grado académico de:
MAESTRO EN DOCENCIA UNIVERSITARIA

PRESENTADO POR:
Bach. Pelayo QUISPE BAUTISTA

ASESORA:
Dra. Elinar CARRILLO RIVEROS

AYACUCHO - PERÚ

2026

Con profunda gratitud y amor, dedico este trabajo de investigación a mi familia, pilar fundamental en mi formación académica y personal.

A mi madre, por su amor incondicional y apoyo constante a lo largo de este camino.

A mi querida abuelita Paulina, por sus enseñanzas y sabiduría que han guiado mis pasos.

Y de manera especial, a la memoria de mi padre Rofino, quien en vida fue ejemplo de perseverancia y dedicación. Su legado continúa inspirando cada uno de mis logros.

Pelayo

Agradecimientos

A Dios, por la vida y por permitirme continuar alcanzando mis objetivos académicos y profesionales.

A mi familia, por su apoyo incondicional durante toda mi formación como profesional y académica.

A mi alma mater, la Universidad Nacional de San Cristóbal de Huamanga, por brindarme la oportunidad de crecer intelectualmente.

De manera especial, a la Dra. Elinar Carrillo Riveros, por su patrocinio, guía experta y dedicación constante durante el desarrollo de este proyecto de investigación.

Índice general

Agradecimientos	vii
Índice general	ix
Índice de tablas	xiii
Índice de figuras	xiv
Índice de anexos	xv
Resumen	xvi
Abstract.....	xvii
Introducción.....	xviii
I. Planteamiento del problema	21
1.1. Descripción de la situación problemática	21
1.2. Formulación de problemas.....	24
1.2.1. Problema general	24
1.2.2. Problemas específicos.....	24
1.3. Formulación de objetivos.....	25
1.3.1. Objetivo general	25
1.3.2. Objetivos específicos	25
1.4. Justificación	25
1.4.1. Teórica	25
1.4.2. Práctica	26
1.4.3. Metodológica	27
II. Marco teórico.....	28
2.1. Antecedentes	28
2.1.1. Internacional	28
2.1.2. Nacional.....	31

2.1.3.	Local	33
2.2.	Bases teóricas.....	34
2.2.1.	Aprendizaje Basado en Problemas	34
2.2.2.	Planificación de Actividades	35
2.2.3.	Ejecución de Estrategias	36
2.2.4.	Evaluación del Proceso.....	37
2.2.5.	Competencias de programación.....	38
2.2.6.	Pensamiento Lógico-Algorítmico	39
2.2.7.	Habilidades de Codificación.....	40
2.2.8.	Depuración y Prueba	40
2.2.9.	Fundamentos pedagógicos del Aprendizaje Basado en Problemas.....	41
2.2.10.	El enfoque universitario en el proceso de enseñanza.....	43
2.3.	Bases conceptuales	45
III.	Metodología.....	48
3.1.	Formulación de hipótesis	48
3.1.1.	Hipótesis general	48
3.1.2.	Hipótesis específicas.....	48
3.2.	Definición conceptual de las variables	48
3.3.	Operacionalización de variables	50
3.4.	Tipos de Investigación	52
3.5.	Nivel de Investigación	53
3.6.	Métodos	53
3.6.1.	Método experimental.....	53
3.6.2.	Método estadístico.....	54
3.7.	Diseño de Investigación.....	54
3.8.	Población y muestra.....	56

3.8.1.	Población	56
3.8.2.	Muestra	57
3.8.3.	Técnica muestral.....	58
3.9.	Técnicas e instrumentos.....	61
3.9.1.	Técnicas de recolección de datos.....	61
3.9.2.	Instrumento de recolección de datos	61
3.10.	Validez y confiabilidad de instrumentos.....	63
3.10.1.	Validez de los instrumentos	63
3.9.3.	Confiabilidad de los instrumentos	66
3.11.	Técnicas de procesamiento de datos	69
3.12.	Plan de experimentación	70
3.13.	Aspectos éticos.....	71
IV.	Resultados y discusión	74
4.1.	Resultados descriptivos.....	74
4.2.	Resultados a nivel inferencial	84
4.2.1.	Prueba de normalidad	85
4.2.2.	Contraste de hipótesis de pretest (grupo control y experimental)	86
4.2.3.	Contraste de hipótesis de postest (grupo control y experimental).....	89
4.2.4.	Contraste de hipótesis de pretest y postes grupo experimental	92
4.2.5.	Prueba de hipótesis general	92
4.2.6.	Prueba de hipótesis específica 1	94
4.2.7.	Prueba de hipótesis específica 2	95
4.2.8.	Prueba de hipótesis específica 3	97
4.3.	Discusión de resultados	98
V.	Propuesta innovadora	104
5.1.	Introducción	104

5.2.	Fundamento Filosófico	105
5.3.	Fundamento Psicopedagógico	106
5.4.	Fundamento Pedagógico	107
5.5.	Enfoque Tecnológico	108
5.6.	Objetivos de la Propuesta.....	111
5.6.1.	Objetivo General.....	111
5.6.2.	Objetivos Específicos	111
5.7.	Descripción de la propuesta	112
5.7.1.	Modelo de cinco fases ABP aplicado a la programación	113
5.7.2.	Secuencia de sesiones de aprendizaje.....	114
5.7.3.	Estrategias de evaluación.....	118
5.8.	Impacto esperado y sostenibilidad	118
VI.	Conclusiones.....	120
	Recomendaciones	123
	Referencias bibliográficas	124
	Anexos	131

Índice de tablas

Tabla 1 Estudiantes de la serie 100 matriculados en el semestre 2024-II	56
Tabla 2 Distribución de la muestra por grupos de estudio	57
Tabla 3 Criterios de inclusión y exclusión de la muestra de estudio	59
Tabla 4 Resumen de la conformación de la muestra de estudio	60
Tabla 5 Técnicas de recolección de datos.....	61
Tabla 6 Estructura del instrumento de evaluación.....	62
Tabla 7 Ficha técnica del instrumento de investigación	63
Tabla 8 Los resultados de la prueba binomial se obtuvieron mediante Python.	64
Tabla 9 Relación de expertos validadores del instrumento	66
Tabla 10 Interpretación del coeficiente de confiabilidad KR-20.....	68
Tabla 11 Resultado de confiabilidad del instrumento	68
Tabla 12 Estadísticos descriptivos del pretest por grupo y variable.....	74
Tabla 13 Estadísticos descriptivos del postest por grupo y variable	79
Tabla 14 Resultados de la prueba de normalidad Shapiro-Wilk postest	85
Tabla 15 Prueba de normalidad Shapiro-Wilk - pretest	87
Tabla 16 Prueba U de Mann-Whitney - pretest por variable.....	88
Tabla 17 Prueba de normalidad Shapiro-Wilk - postest.....	90
Tabla 18 Prueba U de Mann-Whitney - postest por variable	90
Tabla 19 Prueba de Wilcoxon - grupo experimental (pretest vs. postest)	93
Tabla 20 Prueba de Wilcoxon - Pensamiento Lógico-Algorítmico, grupo experimental (pretest vs. postest)	95
Tabla 21 Prueba de Wilcoxon - Habilidades de Codificación, grupo experimental (pretest vs. postest)	96
Tabla 22 Prueba de Wilcoxon - Depuración y Prueba, grupo experimental (pretest vs. postest).....	98
Tabla 23 Herramientas tecnológicos de la propuesta experimental	109
Tabla 24 Objetivos específicos, dimensiones e indicadores de logro	111
Tabla 25 Modelo de cinco fases ABP adaptado a la enseñanza de programación en Python.....	113
Tabla 26 Secuencia de actividades y sesiones de la propuesta experimental.....	114

Índice de figuras

Figura 1 Esquema del diseño cuasiexperimental	55
Figura 2 Comparación del pretest en PLA entre el grupo control y el grupo experimental.	75
Figura 3 Comparación del pretest en HC entre el grupo control y el grupo experimental.	76
Figura 4 Comparación del pretest en DP entre el grupo control y el grupo experimental.	77
Figura 5 Comparación del pretest en el CP entre el grupo control y el grupo experimental.	78
Figura 6 Comparación del posttest en PLA entre el grupo control y el grupo experimental.	80
Figura 7 Comparación del posttest en HC entre el grupo control y el grupo experimental	81
Figura 8 Comparación del posttest en DP entre el grupo control y el grupo experimental.	82
Figura 9 Comparación del posttest en el CP entre el grupo control y el grupo experimental.	83

Índice de anexos

Anexo 1: Matriz de coherencia.....	132
Anexo 2: Constancia de autorización.....	134
Anexo 3 Instrumentos (Prueba pedagógica)	135
Anexo 4 Confiabilidad de instrumento	139
Anexo 5 Validación de instrumento	141
Anexo 6 Procesamiento estadístico	150
Anexo 7 Artículo científico.....	152
Anexo 8 Guía de sesiones	170

Resumen

La presente investigación tuvo como objetivo implementar el Aprendizaje Basado en Problemas (ABP) para mejorar las competencias de programación en Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024. El estudio fue de enfoque cuantitativo, tipo aplicada y diseño cuasiexperimental con pretest y postest. La muestra estuvo conformada por 90 estudiantes del curso de Algoritmos y Solución de Problemas del semestre 2024-II, distribuidos en un grupo experimental ($n = 45$) y un grupo control ($n = 45$). Se utilizó una prueba de desarrollo de 30 ítems, validada mediante juicio de expertos y con un coeficiente KR-20 promedio de 0.89, estructurada en tres dimensiones: Pensamiento Lógico-Algorítmico, Habilidades de Codificación y Depuración y Prueba. La intervención consistió en 11 sesiones ABP con integración de herramientas tecnológicas como Python, Google Colab, GitHub y PyCharm. Dado que los datos no siguieron una distribución normal según la prueba de Shapiro-Wilk ($p < 0.05$), se empleó la prueba U de Mann-Whitney para la comparación entre grupos y la prueba de Rangos con Signo de Wilcoxon para la comparación intragrupal. La equivalencia inicial entre grupos fue confirmada en el pretest ($p > 0.05$ en todas las variables). Los resultados del postest evidenciaron diferencias estadísticamente significativas ($p < 0.001$) en las tres dimensiones, con una diferencia de 8.4 puntos en el Promedio General entre el grupo experimental ($\bar{x} = 15.7$) y el grupo control ($\bar{x} = 7.3$). Se concluye que el ABP mejoró significativamente las competencias de programación en los estudiantes de Ingeniería de Sistemas de la UNSCH.

Palabras clave: aprendizaje basado en problemas, competencias de programación, pensamiento lógico-algorítmico, habilidades de codificación, depuración y prueba, Python.

Abstract

This research aimed to implement Problem-Based Learning (PBL) to improve programming competencies in Python among Systems Engineering students at the Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024. The study followed a quantitative approach, applied type, and quasi-experimental design with pretest and posttest. The sample consisted of 90 students from the Algoritmos y Solución de Problemas course during the 2024-II semester, distributed into an experimental group ($n = 45$) and a control group ($n = 45$). A 30-item development test was used, validated through expert judgment with an average KR-20 coefficient of 0.89, structured in three dimensions: Logical-Algorithmic Thinking, Coding Skills, and Debugging and Testing. The intervention comprised 11 PBL sessions integrating technological tools such as Python, Google Colab, GitHub, and PyCharm. Since the data did not follow a normal distribution according to the Shapiro-Wilk test ($p < 0.05$), the Mann-Whitney U test was used for between-group comparisons and the Wilcoxon Signed-Rank test for within-group comparisons. Initial equivalence between groups was confirmed in the pretest ($p > 0.05$ for all variables). Posttest results showed statistically significant differences ($p < 0.001$) across all three dimensions, with an 8.4-point difference in the Overall Average between the experimental group ($\bar{x} = 15.7$) and the control group ($\bar{x} = 7.3$). It is concluded that PBL significantly improved programming competencies in Systems Engineering students at UNSCH.

Keywords: problem-based learning, programming competencies, logical-algorithmic thinking, coding skills, debugging and testing, Python.

Introducción

La enseñanza de la programación constituye uno de los mayores desafíos en la formación de profesionales de Ingeniería de Sistemas. A nivel internacional, diversos estudios evidencian elevadas tasas de desaprobación en los cursos introductorios de programación, situación que limita el desarrollo de competencias necesarias para afrontar las demandas de la industria tecnológica. Esta problemática también se manifiesta en el contexto peruano, donde persiste una brecha entre las competencias adquiridas durante la formación universitaria y las exigidas por el mercado laboral, especialmente en el dominio de lenguajes de programación de amplia utilización como Python.

En la Universidad Nacional de San Cristóbal de Huamanga, en la Escuela Profesional de Ingeniería de Sistemas, se han encontrado dificultades asociadas con el desarrollo del pensamiento lógico-algorítmico, las habilidades de codificación y los procesos de depuración y prueba de programas. Estas limitaciones afectan el rendimiento académico de los estudiantes y restringen la capacidad de resolver problemas computacionales de forma eficiente. Lo anterior demuestra la necesidad de implementar nuevas estrategias metodológicas que favorezcan un aprendizaje más activo y significativo.

En este contexto, se presenta el Aprendizaje Basado en Problemas (ABP) se presenta como una alternativa pedagógica centrada en el estudiante, que fomenta la construcción del conocimiento mediante la resolución de situaciones problemáticas reales. Este enfoque metodológico promueve el desarrollo de competencias cognitivas, procedimentales y actitudinales, al tiempo que potencia habilidades tales como el trabajo colaborativo, el pensamiento crítico y la capacidad de tomar decisiones. Asimismo, encuentra sustento en los principios del constructivismo, la teoría sociocultural y el

aprendizaje experiencial, los cuales reconocen al estudiante como protagonista de su propio proceso formativo.

En este contexto, la presente investigación tuvo como problema general determinar de qué manera el Aprendizaje Basado en Problemas mejora las competencias de programación en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional de San Cristóbal de Huamanga durante el año 2024. En correspondencia con ello, el objetivo general fue implementar el Aprendizaje Basado en Problemas para mejorar las competencias de programación utilizando el lenguaje Python en dichos estudiantes.

La investigación se justifica desde el punto de vista teórico porque contribuye al fortalecimiento del conocimiento sobre la aplicación del ABP en la enseñanza de la programación; desde la perspectiva práctica, porque ofrece una alternativa metodológica para mejorar el aprendizaje de los estudiantes; y desde el ámbito metodológico, porque proporciona evidencias empíricas sobre la efectividad de esta estrategia pedagógica mediante un diseño de investigación riguroso.

Metodológicamente, el estudio se desarrolló bajo un enfoque cuantitativo, de tipo aplicada y con diseño cuasiexperimental de pretest y posttest con grupo control y grupo experimental. La muestra estuvo conformada por estudiantes del curso de Algoritmos y Solución de Problemas de la Escuela Profesional de Ingeniería de Sistemas, a quienes se aplicó una intervención pedagógica basada en el ABP para evaluar su influencia en el desarrollo de las competencias de programación.

Finalmente, la tesis se encuentra estructurada en seis capítulos. El primero aborda el planteamiento del problema, los objetivos y la justificación de la investigación. El segundo desarrolla el marco teórico y conceptual que sustenta el estudio. El tercero presenta la metodología empleada. El cuarto expone y analiza los resultados obtenidos.

El quinto contiene la propuesta innovadora derivada de los hallazgos de la investigación. Finalmente, el sexto capítulo presenta las conclusiones y recomendaciones correspondientes.

I. Planteamiento del problema

1.1. Descripción de la situación problemática

A nivel mundial, la enseñanza de la programación enfrenta desafíos significativos en la educación superior. Según el estudio realizado por Bennedsen y Caspersen (2019), las tasas globales de no aprobación en cursos introductorios de programación se sitúan alrededor del 28%, basado en un estudio multinacional que abarcó 161 universidades.. Los métodos de enseñanza tradicionales han fallado, sobre todo, en el contexto actual donde Python ha pasado a ser el lenguaje más demandado. De acuerdo con el índice TIOBE (julio de 2025), Python se mantiene como el lenguaje de programación más utilizado a nivel mundial con un 26,98% de popularidad TIOBE Index - TIOBE, impulsado en particular por su aplicación en inteligencia artificial y aprendizaje automático. Sin embargo, Becker y Quille (2019) señalan que el 67% de los estudiantes no logran desarrollar las competencias necesarias para implementar soluciones efectivas en proyectos que involucran IA.

Este escenario mundial se refleja de forma especial en carreras como Ingeniería de Sistemas e Ingeniería de Software, donde la formación en programación se ha vuelto crítica. Las universidades están bajo presión para formar profesionales para un entorno cada vez más competitivo, lo que queda claro con las altas tasas de desaprobación ya mencionadas. Paradójicamente, mientras la inteligencia artificial se presenta como herramienta de apoyo al aprendizaje, también pone en evidencia las deficiencias en competencias básicas que los estudiantes deben desarrollar. Por lo tanto, es indispensable desarrollar metodologías pedagógicas innovadoras que superen los métodos tradicionales, a fin de garantizar las habilidades que exige el mercado laboral actual.

La enseñanza de la programación en el Perú también se enfrenta a particularidades. De acuerdo con el Instituto Nacional de Estadística e Informática INEI (2023), el sector de tecnología de la información ha tenido un crecimiento del 15.3% en lo que se refiere a la demanda de programación, haciendo hincapié sobre tecnologías emergentes como Python e inteligencia artificial. Ahora bien, no obstante este crecimiento, el mismo contrasta con las limitaciones del sistema educativo nacional. Las universidades peruanas reproducen a nivel local las mismas limitaciones del escenario internacional, lo que hace que la brecha entre formación académica y necesidades empresariales se haga evidente cuando muchas empresas no encuentran programadores competentes en Python y otros lenguajes de programación. Por otra parte, las metodologías de enseñanza en muchas instituciones educativas continúan siendo tradicionales, lo que acentúa las tasas de desaprobación y atenta contra el desarrollo de competencias prácticas, en particular en cuanto al pensamiento lógico-algorítmico, la aplicación de conceptos abstractos a situaciones reales y viceversa.

En la Escuela Profesional de Ingeniería de Sistemas de la UNSCH, a nivel local, esta problemática se concreta en tres áreas: pensamiento lógico-algorítmico, habilidades de codificación y depuración y prueba de código. Agravan la situación factores como la heterogeneidad en la preparación previa de los estudiantes, los recursos tecnológicos limitados y la necesidad de actualización constante. Lo que resulta preocupante es que los estudiantes que sobresalen como programadores competentes, por lo general, han desarrollado esas capacidades mediante la autoformación o en institutos externos, no en las aulas de las universidades. Esto refleja que las metodologías actuales no son efectivas en el desarrollo de estas competencias, lo que impacta tanto en el rendimiento académico como en la inserción laboral de los egresados.

En los tres niveles de formación se observan las limitaciones estructurales de la enseñanza de programación en la UNSCH. En el nivel básico los estudiantes presentan dificultades para aprender conceptos como estructuras de datos y algoritmos, ya que la metodología se centra en la transmisión pasiva de contenidos, privilegiando la memorización en vez de la comprensión, lo que produce baja retención, desmotivación y deserción. En el nivel intermedio, la falta de actividades prácticas que se contextualicen en problemas reales del desarrollo de software limita el pensamiento algorítmico y la resolución de problemas. En la enseñanza avanzada, la desconexión entre el plan de estudios y los requerimientos de la industria deja a los estudiantes sin las destrezas de trabajo en equipo y el uso de herramientas tecnológicas que serán necesarias en su futuro desempeño profesional.

Este problema no ocurre solamente en la programación, sino que se reproduce en muchas áreas de la educación superior donde la enseñanza tradicional está llegando a sus límites. De acuerdo con Bustamante Vega y Mejía Mejía (2023), las estrategias constructivistas, particularmente el ABP, influyen significativamente en el fortalecimiento de competencias específicas y genéricas en estudiantes universitarios, superando las deficiencias de los enfoques pedagógicos tradicionales que priorizan la memorización por encima de la comprensión conceptual. Asimismo, Araya Cabrera y Martínez Clares (2023) evidencia que la implementación del ABP no solo mejora el rendimiento académico, sino que permite evidenciar y retroalimentar el logro de competencias del perfil de egreso, facilitando la identificación de fortalezas y debilidades en habilidades transversales como el trabajo en equipo, la autorregulación y la comunicación. Estos hallazgos sugieren que la adopción de metodologías activas representa una alternativa viable para abordar las deficiencias estructurales identificadas

en la enseñanza de la programación, proporcionando un marco pedagógico que favorece el desarrollo integral de competencias profesionales y la articulación efectiva entre la teoría y la práctica en contextos auténticos de aprendizaje.

En conclusión, los problemas identificados en la enseñanza de programación en la Universidad Nacional San Cristóbal de Huamanga evidencian la necesidad de transformar las prácticas pedagógicas tradicionales. La implementación de metodologías activas como el ABP ofrece una alternativa viable para desarrollar competencias técnicas y transversales que realmente preparen a los estudiantes para el ejercicio profesional

1.2. Formulación de problemas

1.2.1. Problema general

¿De qué manera el Aprendizaje Basado en Problemas mejora las competencias de programación en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024?

1.2.2. Problemas específicos

- a. ¿De qué manera la aplicación del Aprendizaje Basado en Problemas mejora el Pensamiento Lógico-Algorítmico?
- b. ¿Cómo la aplicación del Aprendizaje Basado en Problemas mejora las Habilidades de Codificación?
- c. ¿De qué manera la aplicación del Aprendizaje Basado en Problemas mejora la Depuración y Prueba?

1.3. Formulación de objetivos

1.3.1. *Objetivo general*

Implementar el Aprendizaje Basado en Problemas para mejorar las competencias de programación utilizando el lenguaje Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

1.3.2. *Objetivos específicos*

- a. Utilizar el Aprendizaje Basado en Problemas para mejorar el Pensamiento Lógico-Algorítmico en los estudiantes.
- b. Utilizar el Aprendizaje Basado en Problemas para mejorar las Habilidades de Codificación en los estudiantes.
- c. Utilizar el Aprendizaje Basado en Problemas para mejorar la Depuración y Prueba de codificación en los estudiantes.

1.4. Justificación

La investigación surge en respuesta a la necesidad de mejorar las competencias de programación en los estudiantes de Ingeniería de Sistemas de la UNSCH. En el contexto actual, donde la programación se ha convertido en una habilidad fundamental para el desarrollo profesional, es crucial implementar metodologías que permitan a los estudiantes desarrollar un pensamiento lógico-algorítmico eficaz. El ABP se presenta como una estrategia pedagógica innovadora que fomenta el aprendizaje activo y la resolución de problemas reales, alineándose con las demandas del mercado laboral en tecnologías como Python y la inteligencia artificial

1.4.1. *Teórica*

El presente trabajo aporta a la pedagogía universitaria al relacionar los principios constructivistas del ABP con la enseñanza de la programación. Esta metodología se

remonta teóricamente al aprendizaje experiencial de John Dewey y la teoría sociocultural de Vygotsky, coincidiendo ambas en que el conocimiento se construye de forma activa mediante la interacción social y la resolución de problemas reales (Loyens et al. 2012). En el contexto de la educación en programación, el ABP facilita a los estudiantes la comprensión de conceptos complejos como algoritmos y depuración, mediante su aplicación práctica en escenarios reales.

1.4.2. Práctica

La presente investigación surge de las necesidades concretas que enfrentan diariamente los profesores y estudiantes en la enseñanza y el aprendizaje de la programación. Los estudiantes tienen dificultades en la codificación, la depuración y el desarrollo del pensamiento lógico-algorítmico, y las presentan de forma recurrente. Ante esta situación, el ABP sugiere utilizar problemas reales que reflejen escenarios del mundo laboral, lo cual aumenta la motivación y la retención de los conocimientos adquiridos (O'Grady et al., 2012). Investigaciones anteriores han demostrado que esta metodología mejora significativamente las calificaciones y la satisfacción académica de los estudiantes, más que los métodos tradicionales, a la vez que fortalece habilidades como el pensamiento crítico, la resolución de problemas y el trabajo en equipo, disminuyendo la brecha entre la formación universitaria y la práctica profesional en ingeniería de software (AlSideiri et al., 2023). En ese sentido, los resultados de este estudio ofrecerán a los docentes un marco metodológico validado que incluye orientaciones para el diseño de problemas, estrategias de facilitación y criterios de evaluación aplicables en el aula.

1.4.3. Metodológica

Desde el punto de vista metodológico, la investigación realizada constituye un referente útil para la evaluación sistemática de intervenciones pedagógicas en la enseñanza de la programación. El diseño cuasiexperimental utilizado, los instrumentos validados para la evaluación y el proceso de implementación debidamente documentado, conforman un modelo que futuras investigaciones en el área pueden tomar como referencia y adaptar. A través de pruebas aplicadas antes y después de la intervención, la investigación permite medir de manera objetiva el impacto del ABP en las competencias de programación y establecer relaciones de causa y efecto entre la metodología empleada y los resultados obtenidos (Osmanović y Maksimović, 2022). Asimismo, el diseño cuasiexperimental con grupos control y experimental ha demostrado su eficacia en la evaluación del ABP mediante instrumentos validados como pruebas de ensayo y rúbricas de evaluación, confirmando la efectividad del modelo en el desarrollo de habilidades de resolución de problemas y competencias técnicas especializadas (Sari et al., 2021).

II. Marco teórico

2.1. Antecedentes

2.1.1. Internacional

Alshaye et al. (2023) en el artículo *The Effectiveness of Online Problem-Based Learning Tasks on Riyadh's Secondary School Students' Problem-Solving Ability and Programming Skills*. El objetivo fue examinar la efectividad de tareas de ABP en línea en la capacidad de resolución de problemas y habilidades de programación de estudiantes de secundaria. La metodología fue un diseño de investigación cuantitativo cuasi-experimental, involucrando 120 estudiantes de undécimo grado en escuelas secundarias de Riyadh, divididos en grupos experimental (ABP en línea vía grupos de Facebook) y control (centrado en el docente, no en línea), con un período de tratamiento de 8 semanas. Se recolectaron datos usando un cuestionario de inventario de resolución de problemas para capacidad de resolución de problemas y pruebas de desempeño pre-post para habilidades de programación. Los resultados mostraron que el PBL en línea mejoró la capacidad de resolución de problemas y habilidades de programación de los estudiantes, con diferencias significativas en puntuaciones post-test favoreciendo al grupo experimental, y una correlación positiva significativa entre la capacidad de resolución de problemas y habilidades de programación en ese grupo. Se concluyó que las tareas de ABP en línea vía Facebook impactan positivamente en la capacidad de resolución de problemas y habilidades de programación, sugiriendo su efectividad para mejorar habilidades de resolución de problemas y programación en estudiantes de secundaria aprendiendo Visual Basic.

Rattanakha y Chatwattana (2023) en el artículo *The Problem-based Learning Model: PBL Model via Cloud Technology to Promote Programming Skills*. El estudio tuvo como objetivo sintetizar el marco conceptual, desarrollar y evaluar la efectividad del modelo de aprendizaje basado en problemas a través de tecnología en la nube para promover habilidades de programación en estudiantes vocacionales. La metodología adoptada fue un enfoque de desarrollo basado en el método sistémico, dividido en tres etapas: Síntesis del marco conceptual a partir de documentos y estudios relacionados, diseño y desarrollo del modelo PBL mediante tecnología en la nube, y estudio de resultados con evaluación por expertos. Participaron 7 expertos en tecnología educativa para validar la idoneidad del modelo mediante la aplicación de instrumentos de evaluación cuantitativa con análisis descriptivos de medias y desviaciones estándar. Al ser un estudio que busca diseñar y validar un modelo educativo innovador, se clasifica como una investigación aplicada de nivel descriptivo-evaluativo con enfoque cuantitativo. Los resultados mostraron un alto nivel de idoneidad en los elementos generales del modelo (Media = 4.77, DE = 0.44) y en su estructura total (Media = 4.74, DE = 0.39), destacando la adecuación de componentes como el factor de entrada (análisis de características de usuarios, cursos, estudiantes, docentes, medios y tecnología en la nube), el proceso de aprendizaje ABP en seis pasos (definir problema, entender problema, contenido autodirigido, síntesis, resumen, presentación y evaluación), la salida (logro de aprendizaje y habilidades de programación) y la retroalimentación. Las conclusiones indican que el modelo ABP via cloud technology emerge como una alternativa prometedora para enriquecer el proceso de aprendizaje en cursos de programación, fomentando habilidades críticas como el pensamiento analítico sistemático, la resolución de problemas y el uso efectivo de tecnologías, lo que resulta en una experiencia educativa

más flexible, accesible en cualquier momento y lugar, y con implicaciones positivas para el desarrollo de competencias vocacionales alineadas con estrategias nacionales de educación en Tailandia.

AlSideiri et al. (2023) en el artículo *Examining the Impact of Software Engineering Problem-Based Learning on Student Performance*. El estudio tuvo como objetivo de la investigación fue implementar y evaluar la efectividad del enfoque Software Engineering Problem-Based Learning (SEPBL) para mejorar el aprendizaje en cursos de ingeniería de software. La metodología consistió en un estudio comparativo donde se realizaron encuestas a estudiantes para recopilar retroalimentación y se compararon sus calificaciones con métodos tradicionales de enseñanza, además de analizar el impacto en los Resultados de Aprendizaje del Estudiante (SLOs). Al ser un estudio que busca validar una nueva metodología de enseñanza, se clasifica como una investigación aplicada de nivel descriptivo-correlacional con enfoque cuantitativo. Los resultados mostraron un alto nivel de satisfacción de los estudiantes con el método SEPBL en comparación con los métodos tradicionales, así como una mejora significativa en las calificaciones y en el desempeño de los SLOs alineados con el marco SEPBL. Las conclusiones indican que el enfoque SEPBL emerge como una alternativa prometedora para enriquecer el proceso de aprendizaje en cursos de ingeniería de software, fomentando la aplicación práctica y mejorando habilidades críticas como el pensamiento crítico, la resolución de problemas y el trabajo en equipo, lo que resulta en una experiencia educativa más efectiva y participativa con importantes implicaciones para el logro y los resultados de aprendizaje de los estudiantes.

Palomino y Osorio (2023) en el artículo *El aprendizaje basado en problemas para el logro de competencias en educación superior*. El estudio tuvo como objetivo realizar

una revisión bibliográfica sobre el Aprendizaje Basado en Problemas (ABP) en educación superior, especialmente durante la pandemia de COVID-19. La metodología empleada fue de tipo descriptiva, realizando una revisión de literatura nacional e internacional entre 2020-2021, consultando bases de datos académicas como Scopus, EBSCO, Google Académico, Latindex y Dialnet, además de tesis de maestría y doctorado, organizando los hallazgos en tablas comparativas. Los resultados y conclusiones evidenciaron que el ABP favorece el desarrollo integral de los estudiantes mediante el uso adecuado de estrategias, demostrando una mejora significativa no solo en la adquisición de conocimientos sino también en el fortalecimiento de habilidades blandas y competencias procedimentales. Se encontró que los estudiantes alcanzaron una tasa de asimilación del 95.23% después de implementar ABP en comparación con un 58.62% inicial, particularmente en estudios contables. La investigación también destacó la importancia de la integración tecnológica durante la pandemia, con implementaciones exitosas en diversas disciplinas como matemáticas, ingeniería y ciencias de la salud, enfatizando la necesidad de capacitación docente continua y adaptación a metodologías de enseñanza virtual.

2.1.2. Nacional

Santiago (2022) en la tesis *Python y la mejora del pensamiento algorítmico en los alumnos de la escuela profesional de Ingeniería de Sistemas, 2022*, objetivo general determinar en qué medida la enseñanza del programa Python mejora el pensamiento algorítmico en los alumnos del primer año de la Escuela Profesional de Ingeniería de Sistema de la UNHEVAL. Es un enfoque cuantitativo y es de tipo aplicada, y se seleccionó como diseño el tipo cuasi – experimental - longitudinal. En este estudio, se empleó la técnica de encuestas y se diseñó un test con 10 preguntas para evaluar el nivel

de pensamiento algorítmico, considerado como la variable independiente. La muestra consistió en 30 alumnos seleccionados de manera no probabilística y no intencional. La confiabilidad del instrumento se aseguró mediante el juicio de expertos, validando así su eficacia. Las hipótesis planteadas fueron sometidas a prueba mediante la prueba de t para muestras relacionadas. El resultado del estudio al aplicar la prueba de t nos da el valor de significancia menor a 0.05 ($0.000 < 0.05$) con lo cual se concluye que existe influencia entre la variable independiente “Lenguaje de Programación Python” y la variable dependiente “Pensamiento algorítmico”.

García et al. (2021) en el artículo *Influencia del aprendizaje basado en problemas en las competencias adquiridas en el curso de base de datos: caso universidad peruana*. El objetivo fue determinar el efecto de la metodología del Aprendizaje Basado en Problemas (ABP) en el logro de competencias en el curso de Base de Datos de la Escuela Profesional de Ingeniería de Sistemas de una universidad estatal peruana en 2020, con objetivos específicos centrados en evaluar su influencia en el diseño del modelo Entidad Relación, el diseño de bases de datos y el diseño de bases de datos a partir de la normalización. La metodología fue de tipo básica, cuantitativa, con un diseño cuasi-experimental, involucrando 45 alumnos divididos en un grupo de control (11 alumnos) y un grupo experimental (34 alumnos), seleccionados mediante muestreo no probabilístico por conveniencia. Se utilizaron pre-test y post-test evaluados con rúbricas, midiendo tres dimensiones de competencias: diseño del modelo Entidad Relación, diseño de bases de datos y diseño de bases de datos a partir de la normalización. Los instrumentos fueron validados mediante prueba piloto, Alfa de Cronbach (0.792) y pruebas KMO y Bartlett. Los datos se analizaron con estadística descriptiva e inferencial (prueba T de Student) usando Excel y SPSS 25. Los resultados mostraron que en el pre-test, ambos grupos

obtuvieron promedios deficientes (13.0 para control, 13.5 para experimental). En el post-test, el grupo de control alcanzó un promedio moderado de 27.0, mientras que el grupo experimental obtuvo un promedio sobresaliente de 43.5. Las diferencias fueron significativas en las tres dimensiones evaluadas, con aumentos notables en el grupo experimental. La prueba T de Student mostró un valor de $\text{Sig}=0.000<0.05$, rechazando la hipótesis nula y confirmando que el ABP mejora significativamente las competencias. Se concluyó que la metodología del ABP tiene un efecto significativo positivo en el logro de competencias en el curso de Base de Datos, mejorando el desempeño en las dimensiones evaluadas y preparando mejor a los alumnos para enfrentar el mundo laboral. Se recomienda promover el uso de metodologías colaborativas como el ABP en cursos especializados, reforzar el enfoque por competencias mediante capacitación y fortalecer el diseño de bases de datos con esta metodología, incluyendo el uso de rúbricas para evaluar competencias.

2.1.3. Local

Aguilar, (2014) en la tesis *El aprendizaje basado en problemas y el rendimiento académico de los estudiantes del C.E.P. Ciberneta, Huamanga - Ayacucho. 2012*, La investigación se propuso demostrar el efecto positivo y significativo de la técnica didáctica del Aprendizaje Basado en Problemas sobre el rendimiento académico en estudiantes del curso de Aritmética del 1er al 4to año de educación secundaria del C.E.P. CIBERNETA en Huamanga-Ayacucho durante el 2012. A través de un estudio cuasi-experimental, longitudinal y explicativo, se trabajó con una muestra intencional de 196 estudiantes divididos en un grupo control (100 estudiantes) y un grupo experimental (96 estudiantes), distribuidos en ocho aulas: 1A-4A para el control y 1B-4B para el experimental. La recolección de datos se realizó mediante pruebas pedagógicas, y el

análisis estadístico de igualdad de medias para datos pareados demostró un incremento significativo de 3.7 puntos en el rendimiento académico tras aplicar la técnica, pasando de 10.53 puntos en la evaluación control a 14.23 puntos en la evaluación final, diferencia confirmada mediante la prueba T de Student. Con un 95% de nivel de confianza, se concluyó que la aplicación adecuada del Aprendizaje Basado en Problemas incrementa significativamente el rendimiento académico ($T_e = 18.61$; $\alpha = 0.05$; P-valor < 0.001). Se identificaron, también, a través de un análisis de conglomerados cuatro perfiles de estudiantes: grupo control con rendimiento deficiente, grupo control con rendimiento bajo, grupo experimental con rendimiento medio y grupo experimental con rendimiento alto.

2.2. Bases teóricas

2.2.1. *Aprendizaje Basado en Problemas*

Luy-Montejo (2019), el Aprendizaje Basado en Problemas (ABP o PBL) como una metodología activa centrada en el estudiante, cuya característica distintiva es promover el aprendizaje a través de la resolución de problemas reales. Tienen como referentes teóricos a John Dewey, quien defendió la importancia del aprendizaje experiencial, y a Vigotski, cuya teoría sociocultural se centra en la participación del estudiante en comunidades cognitivas para aprender. En la universidad, el ABP es especialmente interesante ya que los estudiantes tienen que enfrentarse a demandas complejas, como la autonomía en su propio proceso formativo o el desarrollo de habilidades sociales. Esta metodología les ofrece un marco a la vez estructurado y flexible, dentro del cual pueden ir adquiriendo contenidos académicos mientras interactúan activamente con otros y resuelven problemas auténticos.

Desde otro punto de vista,, Sari et al. (2021) afirman que el ABP fomenta un proceso de aprendizaje significativo que trasciende la memorización o el cúmulo de información. Los estudiantes actúan con gran intensidad en la resolución de situaciones concretas siguiendo etapas de carácter científico lo cual les permite construir conocimiento propio y desarrollar la capacidad de sustentar soluciones. El proceso se estructura en cinco etapas: orientación al problema, organización del aprendizaje, investigación individual y grupal, desarrollo y presentación del trabajo, análisis y evaluación del proceso de resolución. De este modo, el ABP contribuye a mejorar la calidad del aprendizaje en la educación superior, ya que los estudiantes no solo comprenden los conceptos teóricos sino que también adquieren herramientas prácticas para abordar problemas del mundo real de forma sistemática.

Hidajat (2023), define el ABP como una metodología de aprendizaje autónomo orientada a la resolución de problemas o preguntas poco estructuradas y no rutinarias. Con este enfoque, el problema no es sólo el punto de partida de la actividad, sino la mediadora que activa y sostiene las capacidades de autorregulación de los estudiantes, quienes deben recurrir a conocimientos previos y generar conocimiento nuevo de forma independiente. El autor destaca que el ABP se caracteriza por mantener un alto grado de orientación hacia el aprendizaje creativo y el pensamiento de orden superior, condiciones que precisamente favorecen la aparición de esas capacidades de autorregulación que se requieren en la formación universitaria.

2.2.2. Planificación de Actividades

Para Luy-Montejo (2019), las actividades en el ABP equivalen a la preparación de la situación problemática sobre la que se va a desarrollar toda la experiencia de aprendizaje. En esta fase se requiere en primer lugar, identificar los hechos e ideas

relevantes que constituyan el sustento y contexto del problema planteado. A continuación, se explicitan los objetivos del ABP de manera clara para que sean coherentes con las competencias que se espera que los estudiantes desarrollen. La fase concluye con la elaboración de los sílabos y los instrumentos de evaluación que orientarán y medirán el proceso. En conjunto, estas acciones preparatorias son las que establecen el andamiaje sobre el cual se construirá toda la experiencia formativa.

Hasbi et al. (2024) abordan esta misma fase desde un ángulo complementario. Para estos autores, la planificación comienza con la observación del entorno educativo y la identificación del grupo al que va dirigida la intervención. Luego se realiza un análisis curricular que permite reconocer las competencias y estándares básicos del área, a partir del cual se diseñan los materiales didácticos: planes de lección, hojas de trabajo y recursos de enseñanza adaptados al contenido. Finalmente, se elaboran los instrumentos de recojo de información con indicadores específicos para evaluar aspectos como el interés de los estudiantes y el desarrollo de su pensamiento crítico a lo largo del proceso.

2.2.3. Ejecución de Estrategias

Luy-Montejo (2019), plantea que la ejecución de estrategias en el ABP no opera de forma aislada, sino que se articula directamente con el establecimiento de la situación problemática, conformando así un proceso integrado. En esta fase, el punto de partida es presentar y explicar el problema a los estudiantes junto con los instrumentos de evaluación que acompañarán el proceso. A partir de ahí, se forman los equipos de trabajo y se recogen los primeros intentos de los estudiantes por aproximarse a una solución. Esta articulación entre la ejecución de estrategias y la situación planteada es lo que sostiene el funcionamiento del ABP como metodología activa.

Chueh y Kao (2024) describen este proceso desde una secuencia más detallada. En su propuesta, la ejecución comienza con una fase de enseñanza directa seguida de una evaluación inicial que permite establecer un punto de referencia. Luego, los alumnos realizan trabajo semanal en grupos colaborativos donde comparten y discuten problemas actuales, asumiendo cada uno responsabilidades diferenciadas dentro del equipo. De este modo, la metodología imita la lógica de división del trabajo propia del entorno profesional, impulsando a los estudiantes a desarrollar las competencias asociadas a distintos roles laborales.

2.2.4. Evaluación del Proceso

Luy-Montejo (2019) describe la evaluación en el ABP como un proceso que se lleva a cabo en cinco momentos que se encuentran estrechamente relacionados con la resolución del problema. La primera es descubrir una pregunta que les cause dificultad e interés a los alumnos. En el segundo momento se plantean hipótesis, es decir, respuestas tentativas que los estudiantes construyen activando sus conocimientos previos y organizando esquemas conceptuales. La tercera etapa es cuando se recoge la información dirigida por dichas hipótesis, momento en que los estudiantes planifican sus estrategias de búsqueda y desarrollan habilidades para organizar y comunicar lo que van encontrando, lo que favorece la metacognición. En el cuarto momento se evalúan las hipótesis, los estudiantes contrastan sus propuestas con los datos recogidos y se enfrentan a la ambigüedad propia de las situaciones reales. El proceso termina con la generalización, en donde se sintetizan las conclusiones a las que se ha llegado y se hace una reflexión sobre el recorrido realizado, lo que permite que ese aprendizaje se transfiera a otras situaciones académicas y a la vida cotidiana.

Desde otro punto de vista,, Amanda et al. (2024) firman que la evaluación en el marco del proceso ABP es establecida como la última etapa del modelo, en donde se aplica principalmente la evaluación entre pares como una forma de valoración. Esta metodología es relevante ya que permite a los estudiantes recibir retroalimentación de sus compañeros durante el proceso de aprendizaje. Asimismo, este tipo de evaluación contribuye al desarrollo de la autoconfianza en el trabajo grupal, promueve el proceso de pensamiento crítico, ofrece un proceso de evaluación transparente y ayuda a reducir la pasividad de los estudiantes durante las actividades de aprendizaje.

2.2.5. Competencias de programación

Según Vinueza-Morales et al. (2025), la competencia de programación son habilidades y conocimientos que los estudiantes adquieren para resolver problemas mediante la escritura y comprensión de código, incluyendo conceptos básicos, estructuras de control, objetos, clases, procedimientos, funciones, métodos, parámetros, condicionales y bucles, entre otros. Estas competencias son fundamentales para el desarrollo de habilidades algorítmicas y de resolución de problemas en programación.

Para Ahirrao et al. (2024), es un conjunto integral de habilidades y conocimientos técnicos que son fundamentales en el panorama tecnológico actual. Fundamentalmente, estas competencias abarcan el dominio de diversos lenguajes de programación, marcos de trabajo y herramientas que son esenciales para el desarrollo de software. Por ende, incluyen la capacidad de adaptarse a las demandas cambiantes de la industria y la habilidad para resolver problemas de manera efectiva.

Holenko Dlab Hoic-Bozic (2021) entienden las competencias de programación como un conjunto integral de habilidades que va más allá del dominio de los procedimientos estándar de desarrollo de programas. Para estos autores, dichas

competencias abarcan también dimensiones como la innovación, el emprendimiento y la iniciativa en el diseño y desarrollo de nuevos modelos y productos mediante tecnología computacional. A esto se suma que su cultivo fortalece en los estudiantes la autoconfianza, perseverancia y precisión en el momento de corregir errores, así como la capacidad de comunicarse y trabajar de manera colaborativa para alcanzar metas concretas.

2.2.6. *Pensamiento Lógico-Algorítmico*

Lu et al. (2022) consideran que el pensamiento lógico-algorítmico es una habilidad fundamental del pensamiento computacional, a través de la cual los estudiantes construyen y comprenden una serie de acciones preestablecidas que incluyen elementos tales como ciclos, arreglos y secuencias. Es una manera estructurada de pensar, que permite dividir un problema en etapas lógicas y ordenadas. En este proceso entran en juego operadores que funcionan como constructos para la manipulación de datos, aunque su expresión obedece a una perspectiva semántica.

Milková y Hulková (2013), por su parte, el pensamiento secuencial como la base sobre la que se construye todo razonamiento lógico-algorítmico. Estos autores afirman que pensar de manera algorítmica es tomar las ideas, datos y conclusiones relevantes de un problema y organizarlos en una progresión encadenada que adquiera por sí misma sentido. En otras palabras, pensar lógicamente es pensar en pasos secuenciales. Esta capacidad se puede reforzar con un trabajo sistemático sobre las capacidades de pensamiento secuencial y profundizar mediante asignaturas relacionadas con la optimización combinatoria. De ahí que sea imprescindible el pensamiento lógico-algorítmico para que los estudiantes puedan desarrollar una verdadera capacidad de resolución estructurada de problemas.

2.2.7. Habilidades de Codificación

La capacidad de escribir código constituye una parte esencial de las habilidades de programación. Hacen referencia a la habilidad del estudiante para plasmar soluciones algorítmicas en código funcional utilizando un lenguaje de programación. Estas habilidades, como señalan Robins et al. (2003), implican no sólo el conocimiento sintáctico del lenguaje, sino también la comprensión semántica de las instrucciones y la capacidad de estructurar de forma eficiente y legible el código. En Python, las habilidades de codificación implican el uso apropiado de estructuras de control (condicionales y bucles), la definición y utilización de funciones, el manejo de estructuras de datos como listas y diccionarios, y la aplicación de buenas prácticas en la escritura del código. Vinueza-Morales, et al. (2025) señalan que estas habilidades son fundamentales para que el estudiante pueda implementar soluciones concretas a problemas reales, constituyendo el núcleo operativo del desempeño en programación en educación superior.

2.2.8. Depuración y Prueba

La depuración y prueba son una faceta avanzada de las habilidades de programación que combina dos capacidades estrechamente vinculadas: localizar, examinar y subsanar los errores en el código, y asegurarse de que el programa se comporte de manera correcta en distintas situaciones. Según McCauley et al. (2008) la depuración es un proceso cognitivo de gran complejidad, ya que requiere una comprensión profunda del programa, razonamiento lógico y aplicación sistemática de estrategias de diagnóstico. La prueba, por otra parte, se dedica a diseñar y realizar casos de evaluación que permitan constatar que el software realiza lo que se espera de él.

En la formación en ingeniería de sistemas el dominio de esta dimensión pesa de manera particular ya que los errores que no se detectan a tiempo pueden derivar en

consecuencias técnicas de consideración. Vinueza-Morales et al., (2025) afirman que la depuración y la prueba, vistas en conjunto, expresan la madurez técnica del programador y su capacidad para entregar soluciones sólidas y confiables en escenarios reales de desarrollo.

2.2.9. Fundamentos pedagógicos del Aprendizaje Basado en Problemas

La pedagogía activa constituye el principal sustrato teórico del ABP como estrategia de enseñanza en educación superior. Sus raíces intelectuales se remiten a su principal inspirador, Dewey (1969), quien en su libro *Experiencia y educación* apuntó que el aprendizaje auténtico no puede estar separado de la experiencia directa del sujeto con su realidad y que la educación ha de estar configurada en torno a situaciones problemáticas con la exigencia de que el pensamiento sea reflexivo y de que la acción sea deliberada. Para Dewey, el aula no es un lugar contenedor de contenidos, es un laboratorio de resolución de problemas donde el estudiante aprende haciendo, principio que constituyen la base filosófica en la que se asienta el ABP y que en el ámbito de la programación cobra especial relevancia puesto que el desarrollo funcional del código es en esencia una actividad práctica, iterativa y con un enfoque a la solución de problemas concretos.

Freire (1970) aportó una nueva crítica fundamental al modelo tradicional al cuestionar lo que se conoce con el nombre de concepción bancaria de la educación, a través de la cual el docente se limita a realizar un depósito de información a un estudiante que resulta pasivo. Más bien intenta promover una pedagogía dialógica donde el aprendizaje surge del encuentro entre el sujeto y el mundo, realizándose a través de la problematización de la realidad. Esta idea enlaza directamente con la lógica del ABP: el problema auténtico es el que actúa como vínculo entre el estudiante y el conocimiento

disciplinar, derivando en la necesidad de aprender para poder resolver, y no en la acumulación de contenidos alejada de la acción. Existen evidencias empíricas que apoyan este carácter superador del ABP. Freeman et al. (2014), en un metaanálisis de 225 estudios en carreras de ciencias, tecnología, ingeniería y matemáticas, encontraron que los estudiantes que participaron en clases basadas en aprendizaje activo lograron unas notas más altas y tasas de desaprobación más bajas que sus compañeros que participaron en clases tradicionales, obteniendo un tamaño del efecto de 0.47 desviaciones estándar, de modo que existe un respaldo empírico suficientemente sólido para la adopción del ABP en la formación.

El enfoque por competencias complementa esta base pedagógica al plantear que la formación universitaria debe garantizar el desarrollo de capacidades transferibles a contextos profesionales reales. Perrenoud (2004) define la competencia como la capacidad de movilizar diversos recursos cognitivos para hacer frente a un tipo de situaciones, aclarando que las competencias no son conocimientos, habilidades o actitudes por separado, sino la integración y movilización de todos esos recursos ante situaciones específicas. Tobón (2013), desde el enfoque socioformativo, las entiende como actuaciones integrales orientadas a identificar, interpretar, argumentar y resolver problemas del contexto con idoneidad y ética, articulando el saber ser, el saber conocer y el saber hacer en una unidad inseparable. La relación entre el enfoque por competencias y el ABP es, en ese sentido, estructural. Hmelo-Silver (2004), en una revisión sistemática de esta metodología, finaliza que el ABP favorece el desarrollo de conocimiento flexible, habilidades de resolución de problemas, aprendizaje autodirigido y colaboración efectiva, competencias todas ellas alineadas con el perfil de egreso que los programas de ingeniería de sistemas contemporáneos demandan.

2.2.10. El enfoque universitario en el proceso de enseñanza

La educación superior actual se caracteriza por el ejercicio de cierta tensión latente entre unos modelos pedagógicos tradicionales más bien caracterizados por la transmisión sistemática de los conocimientos disciplinares y las exigencias de la sociedad del conocimiento, la cual pide un egresado que sea capaz de aprender autónomamente, de adquirir prácticas de trabajo en entornos diversos y colaborativos, así como de enfrentarse a los problemas complejos en condiciones de incertidumbre. Esta tensión ha propiciado, desde finales del siglo XX, el desarrollo de un modelo universitario en torno a un aprendizaje activo, a la formación por competencias y a la adopción de metodologías pedagógicas conformes con las realidades del ejercicio profesional actual.

La universidad como espacio de formación para la complejidad

En su obra “Los siete saberes necesarios para la educación del futuro”, Morin (1999) afirma que la educación universitaria debe preparar a los estudiantes para hacer frente a la incertidumbre y complejidad que presentan los problemas del mundo real, superando la fragmentación disciplinar del modelo tradicional. Este punto es especialmente importante en la educación para ingenieros en sistemas, ya que los problemas de desarrollo de software rara vez pertenecen a un solo campo del conocimiento. Para abordarlas se necesita una combinación de habilidades técnicas, capacidad de análisis lógico, competencias comunicativas y capacidad para el trabajo en equipo.

El rol del docente universitario en el proceso de enseñanza activa

La aplicación del ABP en el ámbito universitario implica una profunda redefinición del rol del profesorado. Deja de estar asociada a la transmisión central del conocimiento, para pasar a ser la de un diseñador de experiencias de aprendizaje y un

facilitador del proceso mediante el cual los propios estudiantes construyen el conocimiento. Branda, (2009) describe este nuevo rol como el del docente tutor en el ABP, cuya función principal no consiste en proporcionar respuestas correctas sino en formular preguntas que orienten el pensamiento del estudiante, identificar los momentos en que el grupo necesita andamiaje pedagógico y crear las condiciones para que la colaboración entre pares sea productiva. Esta transformación del rol docente tiene implicaciones directas para la enseñanza de la programación, donde el docente facilitador diseña los problemas ABP, selecciona las herramientas tecnológicas de soporte, estructura los equipos de trabajo y proporciona retroalimentación formativa sobre el proceso de solución, sin sustituir la búsqueda autónoma del estudiante ni revelar la solución antes de que el grupo la haya construido.

Evaluación universitaria orientada al desarrollo de competencias

La evaluación del aprendizaje en el nivel universitario constituye uno de los factores de mayor impacto sobre la calidad de la formación, dado que los estudiantes orientan su esfuerzo académico en función de lo que saben que será evaluado. Boud y Falchikov (2006)) argumentan que la evaluación universitaria tradicional, centrada en exámenes de reproducción de contenidos al final del periodo académico, no solo mide de manera inadecuada las competencias desarrolladas durante el proceso de aprendizaje, sino que activamente desincentiva el desarrollo de las capacidades de orden superior que los perfiles de egreso declaran como objetivos formativos. Estos autores proponen un modelo de evaluación sostenible orientado al aprendizaje a largo plazo, que incorpore instancias de evaluación formativa continua, autoevaluación y evaluación entre pares como componentes estructurales del proceso pedagógico y no como añadidos opcionales a la evaluación sumativa terminal.

La integración de tecnología en el proceso de enseñanza universitaria

La integración de herramientas tecnológicas en el proceso de enseñanza universitaria ha dejado de ser una opción pedagógica para convertirse en una condición de pertinencia curricular en los programas de formación en ingeniería de sistemas. Garrison y Vaughan (2008) desarrollaron el modelo de comunidad de indagación como marco para comprender el aprendizaje universitario mediado por tecnología, identificando tres presencias interrelacionadas que determinan la calidad de la experiencia educativa: la presencia cognitiva, referida al proceso de construcción del conocimiento mediante la reflexión y el diálogo; la presencia social, referida a la capacidad de proyectarse como miembros de una comunidad de aprendizaje; y la presencia docente, referida al diseño, facilitación y dirección de los procesos cognitivos y sociales hacia resultados de aprendizaje significativos. Este modelo describe con precisión la dinámica de las sesiones ABP de programación apoyadas en herramientas colaborativas, donde los estudiantes construyen conocimiento a través de la resolución de problemas auténticos, interactúan como miembros de un equipo técnico con roles definidos y cuentan con la facilitación continua del docente para orientar el proceso hacia los resultados de aprendizaje establecidos.

2.3. Bases conceptuales

Algoritmo: Secuencia organizada de acciones o directrices que se aplican para solucionar un problema determinado. Es como una receta: dice qué hacer primero, qué hacer después y cómo acabar.

Aprendizaje activo: Metodología de enseñanza donde el estudiante no solo escucha, sino que participa, investiga y construye su propio conocimiento a partir de la práctica y la colaboración.

Autorregulación: Capacidad del estudiante para ordenar su propio aprendizaje: saber qué debe aprender, buscar información por su cuenta y evaluar si aprendió lo suficiente.

Diagrama de flujo: Representación gráfica de los pasos seguidos por un proceso o algoritmo mediante figuras geométricas y flechas. Sirve para visualizar cómo va a funcionar un programa antes de escribirlo.

Estrategias heurísticas: Métodos prácticos que ayudan a encontrar una solución posible a un problema cuando no existe una fórmula exacta. Son atajos o pistas que guían el razonamiento.

Framework: Conjunto de herramientas y reglas prediseñadas que sirven de base para construir programas de manera más rápida y ordenada, sin tener que empezar desde cero.

GitHub: Plataforma en internet donde los programadores guardan, organizan y comparten su código. También permite ver los cambios que se hicieron al programa a lo largo del tiempo.

Google Colab: Entorno de programación en línea, gratuito, que permite escribir y ejecutar código Python directamente desde el navegador, sin instalar nada en la computadora.

Inteligencia artificial (IA): Área de la informática que busca crear programas capaces de aprender, razonar y tomar decisiones de forma similar a como lo haría una persona.

Machine Learning: Parte de la inteligencia artificial donde los programas aprenden solos a partir de datos, mejorando con la experiencia sin que alguien les programe cada regla manualmente.

Metacognición: Capacidad de una persona para reflexionar sobre cómo está aprendiendo, qué entiende y qué no, y qué puede hacer para mejorar su comprensión.

Modularización: Técnica de programación que consiste en dividir un programa grande en partes más pequeñas e independientes, llamadas módulos, donde cada una cumple una función específica.

POO (Programación Orientada a Objetos): Forma de programar que organiza el código en "objetos", que son como moldes que combinan datos y acciones. Por ejemplo,

un objeto "estudiante" puede tener datos como nombre y edad, y acciones como estudiar o rendir examen.

Pretest y posttest: Evaluaciones que se aplican antes y después de una intervención educativa para medir cuánto aprendieron los estudiantes con el método aplicado.

Pseudocódigo: Descripción en lenguaje sencillo de los pasos que seguirá un programa, escrita de forma parecida a la programación pero sin seguir las reglas exactas de ningún lenguaje. Sirve para planificar antes de escribir el código real.

PyCharm: Programa especializado para escribir y ejecutar código en Python. Tiene herramientas que ayudan a detectar errores y organizar mejor el trabajo del programador.

pytest: Herramienta de Python que permite comprobar automáticamente si un programa funciona como se espera, ejecutando una serie de pruebas diseñadas por el programador.

Python: Lenguaje de programación sencillo y muy popular, usado tanto para aprender a programar como para desarrollar aplicaciones avanzadas de inteligencia artificial, análisis de datos y desarrollo web.

TDD (Test-Driven Development): Método de programación en el que primero se escriben las pruebas que el programa debe pasar, y luego se escribe el código que las resuelve. Ayuda a desarrollar software más ordenado y con menos errores.

Zona de Desarrollo Próximo (ZDP): Concepto propuesto por Vygotsky que se refiere a la diferencia entre lo que un estudiante puede hacer solo y lo que puede lograr con la ayuda de alguien más experimentado. Es el espacio ideal donde ocurre el aprendizaje.

III. Metodología

3.1. Formulación de hipótesis

3.1.1. Hipótesis general

H1: La implementación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación utilizando el lenguaje Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

3.1.2. Hipótesis específicas

H1 La aplicación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación en relación al Pensamiento Lógico-Algorítmico en los estudiantes de Ingeniería de Sistemas de la UNSCH, Ayacucho 2024.

H2 La aplicación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación en relación a las habilidades de codificación en los estudiantes de Ingeniería de Sistemas de la UNSCH, Ayacucho 2024.

H3 La aplicación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación en relación a la Depuración y Pruebas en códigos en los estudiantes de Ingeniería de Sistemas de la UNSCH, Ayacucho 2024.

3.2. Definición conceptual de las variables

Variable independiente (VI): Aprendizaje basado en problemas

Según lo planteado por Xue et al. (2025), el aprendizaje basado en problema es un enfoque pedagógico que favorece el aprendizaje activo, la participación colaborativa

y la construcción autónoma del conocimiento. En lugar de recibir información de forma pasiva, los estudiantes trabajan en equipo para investigar, analizar y resolver desafíos, desarrollando así capacidades de pensamiento crítico, resolución de problemas y toma de decisiones.

Variable dependiente (VD): Competencia de programación

Según Guerrero-Rodríguez et al. (2025) la competencia de programación es la capacidad de codificar información en diferentes sistemas de informáticos para garantizar su correcta organización y acceso, identificar las etapas de una aplicación informática para gestionar proyectos de manera eficiente, analizar cambios en un sistema para evaluar recursos, costos, ventajas y desventajas, y utilizar los lenguajes de programación orientado a objetos y sistemas gestores de base de datos para solucionar errores o fallos detectados.

3.3. Operacionalización de variables

Variables	Definición conceptual	Definición operacional	Dimensiones	Indicadores	Escala de medición
Aprendizaje Basado en Problemas	Según Xue et al. (2025), es un enfoque pedagógico que favorece el aprendizaje activo, la participación colaborativa y la construcción autónoma del conocimiento, mediante el cual los estudiantes trabajan en equipo para investigar, analizar y resolver desafíos.	Se operacionalizó mediante la aplicación de la metodología ABP en tres fases (planificación, ejecución y evaluación), verificadas a través de una lista de cotejo / guía de observación, registrando su aplicación como de variable intervención pedagógica.	Planificación de actividades	Organización del problema, definición de objetivos, distribución de roles, diseño del plan de trabajo colaborativo	Nominal (Aplicado / No aplicado)
			Ejecución de estrategias	Trabajo en equipo, resolución colaborativa de problemas, construcción activa del conocimiento en Python	Nominal (Aplicado / No aplicado)
			Evaluación del proceso	Autoevaluación, coevaluación, retroalimentación, reflexión sobre los aprendizajes logrados	Nominal (Aplicado / No aplicado)
Competencias de programación	Según Guerrero-Rodríguez et al. (2025), es la capacidad de codificar información en diferentes sistemas informáticos, identificar etapas de una aplicación informática, analizar cambios en un sistema y	Se operacionalizó mediante una prueba de desarrollo de 30 ítems (1-30), distribuida en tres dimensiones, con puntuación dicotómica por ítem (correcto = 1,	Pensamiento lógico-algorítmico	Identifica tipos de datos y expresiones; diseña y evalúa expresiones lógicas; comprende estructuras condicionales e iterativas; reconoce conceptos algorítmicos (búsqueda binaria, divide y vencerás, programación dinámica)	Dicotómica (Correcto=1 / Incorrecto=0); Vigesimal (0-20 pts)

Variables	Definición conceptual	Definición operacional	Dimensiones	Indicadores	Escala de medición
	utilizar lenguajes de programación orientados a objetos para solucionar errores detectados.	incorrecto = 0) convertida a escala vigesimal (0-20 puntos).	Habilidades de codificación	Implementa funciones en Python; maneja listas, tuplas, diccionarios y conjuntos; aplica operaciones sobre estructuras de datos; comprende paso de parámetros y retorno de valores; aplica POO (herencia, polimorfismo)	Dicotómica (Correcto=1 / Incorrecto=0); Vigesimal (0-20 pts)
			Depuración y prueba	Identifica tipos de errores (TypeError, IndexError, ZeroDivisionError); detecta errores sintácticos y lógicos; corrige código defectuoso; aplica manejo de excepciones (try-except-finally); diseña y ejecuta pruebas unitarias; usa excepciones personalizadas	Dicotómica (Correcto=1 / Incorrecto=0); Vigesimal (0-20 pts)

3.4. Tipos de Investigación

La presente investigación se enmarca dentro del tipo aplicado, en la medida en que se orienta a la generación de conocimiento con fines prácticos y a la solución de problemas concretos dentro de un contexto determinado. Al respecto, Carrasco (2015) señala que este tipo de investigación se caracteriza por perseguir propósitos prácticos inmediatos y claramente definidos, pues su finalidad no es acumular conocimiento por sí mismo, sino actuar, transformar o modificar una determinada realidad. De acuerdo con este planteamiento, en el presente estudio se implementó el ABP como estrategia didáctica orientada a mejorar las competencias de programación en estudiantes de Ingeniería de Sistemas, constituyendo una intervención práctica para generar cambios concretos en el proceso de enseñanza y aprendizaje.

En cuanto a su enfoque, la investigación es de tipo cuantitativa ya que el fenómeno de estudio se aborda a partir de la medición numérica de variables y del análisis estadístico de los datos obtenidos. Hernández-Sampieri y Mendoza (2018) afirman que la aproximación cuantitativa emplea la recolección de datos para comprobar hipótesis, apoyándose en la medición numérica y el análisis estadístico, con el fin de identificar patrones de comportamiento y validar teorías. En este mismo sentido, Bernal, (2022) precisa que dicho enfoque se fundamenta en la medición de las características observables de los fenómenos sociales. De acuerdo con ello, en la presente investigación se midieron las competencias de programación, comprendidas en las dimensiones de pensamiento lógico-algorítmico, habilidades de codificación, y depuración y prueba, de manera cuantitativa, antes y después de la aplicación del ABP, empleando pruebas estadísticas para el contraste de las hipótesis formuladas.

3.5. Nivel de Investigación

El nivel de investigación se refiere al grado de profundidad con que se aborda un fenómeno u objeto de estudio Arias (2012). Según esta consideración, el presente estudio corresponde a un nivel experimental. Al respecto, Carrasco (2015) sostiene que la investigación experimental es aquella que se realiza luego de conocer las características del fenómeno o hecho que se investiga, así como las causas que han determinado dichas características; es decir, una vez identificados los factores que originan el problema, se procede a otorgarle un tratamiento metodológico específico.

En coherencia con este planteamiento, en la presente investigación se manipuló deliberadamente la variable independiente (Aprendizaje Basado en Problemas) para observar su efecto sobre la variable dependiente (Competencias de programación), estableciendo así una relación causa efecto entre ambas variables

3.6. Métodos

El método de investigación constituye el conjunto de procedimientos lógicos a través de los cuales se plantean los problemas científicos y se ponen a prueba las hipótesis y los instrumentos de trabajo de investigación (Bernal, 2022). En la presente investigación se emplearon los siguientes métodos:

3.6.1. Método experimental

Según Carrasco (2015), este método es un proceso lógico sistemático que responde a la pregunta ¿qué sucede con la variable dependiente cuando se manipula la variable independiente? Este método permite establecer relaciones de causalidad entre variables mediante el control de factores externos.

En la presente investigación, este método permitió comparar el efecto del Aprendizaje Basado en Problemas sobre las competencias de programación entre el grupo

experimental y el grupo control, cuyos procedimientos específicos se detallan en el apartado correspondiente al diseño de investigación.

3.6.2. Método estadístico

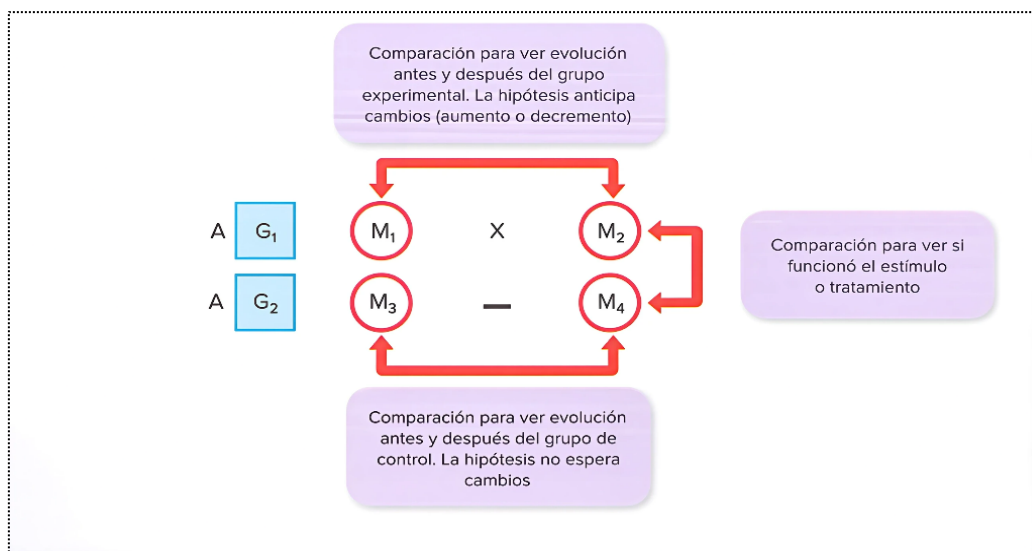
De acuerdo con Córdova (2003), el método estadístico consiste en una secuencia de procedimientos para el manejo de datos cualitativos y cuantitativos de la investigación. Dicho manejo tiene como propósito comprobar, en una parte de la realidad, una o varias consecuencias verificables deducidas de la hipótesis general de la investigación.

Este método se empleó para el procesamiento, análisis e interpretación de los datos recolectados, utilizando técnicas de estadística descriptiva e inferencial para determinar la significancia de los resultados.

3.7. Diseño de Investigación

El diseño de la investigación es de tipo experimental, específicamente cuasiexperimental, con la participación de dos grupos (experimental y control), y la aplicación de mediciones de pretest y posttest antes y después de la intervención. Según Hernández-Sampieri y Mendoza (2018), los diseños cuasiexperimentales manipulan deliberadamente al menos una variable independiente para observar su efecto sobre una o más variables dependientes, diferenciándose de los diseños experimentales puros en que los sujetos no se asignan aleatoriamente a los grupos.

Se optó por este diseño debido a que los grupos ya se encontraban constituidos previo al inicio del estudio, correspondiendo a los grupos regulares del curso de Algoritmos y Solución de Problemas. Esta condición impidió la asignación aleatoria de los participantes, situación que es frecuente en contextos educativos donde la institución responde a normas y acuerdos internos

Figura 1*Esquema del diseño cuasiexperimental*

Nota. Fuente: (Hernández-Sampieri y Mendoza, 2018)

Se optó por este diseño debido a que los grupos ya se encontraban constituidos previo al inicio del estudio, correspondiendo a los temas regulares del curso de Algoritmos y Solución de Problemas. Esta condición no se optó por la asignación aleatoria de los participantes, situación que es frecuente en los contextos educativos donde la institución responde a las normas o acuerdos internos.

El esquema del diseño es el siguiente:

$$\begin{array}{rcccl}
 \text{GE:} & O_1 & X & O_2 \\
 \text{GC:} & O_3 & - & O_4
 \end{array}$$

Donde:

- GE: Grupo experimental
- GC: Grupo control
- O_1, O_3 = Observaciones del pretest (Medición inicial de competencia de programación)
- O_2, O_4 = Observaciones del posttest (Medición final de competencias de programación)

- X = Tratamiento experimental (Aprendizaje Basados en Problemas)
- - = Sin tratamiento (enseñanza tradicional)

3.8. Población y muestra

3.8.1. Población

La población constituye un elemento metodológico fundamental en todo proceso de investigación científica. Al respecto, Bernal (2022) define la población como el conjunto de todos los elementos (personas, objetos, instituciones, documentos, etc.) que poseen características comunes, las cuales serán objeto de estudio y para las cuales serán válidas las conclusiones de la investigación.

La población de la presente investigación estuvo conformada por 140 estudiantes de la serie 100 de la Escuela Profesional de Ingeniería de Sistemas de la UNSCH, matriculados en el semestre académico 2024-II. Los estudiantes cursaban asignaturas del plan de estudios 2018.

Tabla 1

Estudiantes de la serie 100 matriculados en el semestre 2024-II

Grupos	Estudiantes matriculados (N)
Grupo A	50
Grupo B	50
Grupo C	40
Total	140

Nota: Reporte de estudiantes matriculados en el semestre 2024-II

La población presentó las siguientes características: estudiantes de ambos sexos, con edades comprendidas principalmente entre los 17 y 25 años, que cursaban la serie 100 correspondiente al primer año de formación universitaria en el programa de estudios de Ingeniería de Sistemas.

3.8.2. Muestra

Según Carrasco (2015), la muestra es una parte o fragmento representativo de la población, cuyas características esenciales son ser objetiva y reflejo fiel de ella, de tal manera que los resultados obtenidos en la muestra puedan generalizarse a todos los elementos que conforman dicha población. En coherencia con el diseño cuasiexperimental descrito anteriormente, la muestra estuvo conformada por grupos intactos, ya constituidos previamente al estudio.

La muestra estuvo conformada por 100 estudiantes, distribuidos en dos grupos (Grupo A y B) intactos correspondientes a la asignatura de Algoritmos y Solución de Problemas de la Escuela Profesional de Ingeniería de Sistemas, semestre 2024-II. El Grupo A fue asignada como grupo experimental ($n = 50$) y el Grupo B como grupo control ($n = 50$). Ambos grupos compartían el mismo plan de estudios, el mismo docente titular y fueron evaluadas en el mismo período académico, lo que garantizó condiciones iniciales comparables.

Tabla 2

Distribución de la muestra por grupos de estudio

Grupos	Grupo asignado	Estudiantes (n)
Grupo A	Grupo Experimental (GE)	50
Grupo B	Grupo Control (GC)	50
Total		100

Nota. Reporte de estudiantes matriculados del grupo A y B en la asignatura de Algoritmos y Solución de Problemas, semestre 2024-II, Escuela Profesional de Ingeniería de Sistemas, UNSCH.

En la Tabla 2 se puede apreciar la distribución de la muestra de estudio. De los 100 estudiantes que conformaron la muestra, 50 estudiantes del Grupo A fueron

asignados al grupo experimental, quienes recibieron la intervención pedagógica basada en el ABP. Los 50 estudiantes restantes del Grupo B conformaron el grupo control, quienes continuaron con la metodología de enseñanza tradicional. La igualdad numérica entre ambos grupos constituye una ventaja metodológica, ya que facilita la comparación estadística y reduce posibles sesgos asociados a diferencias en el tamaño de los grupos (Field, 2018).

3.8.3. Técnica muestral

Se empleó el muestreo no probabilístico de tipo intencional o por conveniencia. De acuerdo con Hernández-Sampieri y Mendoza (2018), en este tipo de muestreo la elección de los elementos no depende de la probabilidad, sino de las características de la investigación y de los criterios del investigador. Esta técnica es la más apropiada para los diseños cuasiexperimentales, en los que se trabaja con grupos ya conformados de manera natural (grupos intactos), dado que la aleatorización individual de los participantes no es factible en contextos educativos reales (Campbell y Stanley, 1966).

La selección de los grupos A y B de la asignatura de Algoritmos y Solución de Problemas obedeció a los siguientes criterios: ambos grupos pertenecían al mismo semestre académico, compartían el mismo plan curricular, estaban a cargo del mismo docente y tenían el mismo tamaño (50 estudiantes cada una). Estas condiciones permitieron asegurar la homogeneidad inicial entre los grupos, aspecto que fue posteriormente corroborado mediante la prueba de equivalencia en el pretest.

Criterios de selección: por inclusión y exclusión

Para la conformación definitiva de la muestra se establecieron criterios de inclusión y exclusión, los cuales se detallan a continuación:

Tabla 3*Criterios de inclusión y exclusión de la muestra de estudio*

Criterios de inclusión	Criterios de exclusión
1. Estudiantes matriculados en la asignatura de Algoritmos y Solución de Problemas de la Escuela Profesional de Ingeniería de Sistemas de la UNSCH, semestre académico 2024-II.	1. Estudiantes que se retiraron, desmatricularon o abandonaron la asignatura durante el período de intervención.
2. Estudiantes que asistieron de manera regular a las sesiones de aprendizaje durante el período de intervención.	2. Estudiantes con más del 30% de inasistencias a las sesiones de aprendizaje.
3. Estudiantes que completaron tanto la evaluación pretest como la evaluación posttest.	3. Estudiantes que no completaron alguna de las dos evaluaciones (pretest o posttest).
4. Estudiantes que aceptaron participar voluntariamente en la investigación mediante la firma del consentimiento informado.	4. Estudiantes que no aceptaron participar voluntariamente en la investigación.
	5. Estudiantes con matrícula condicional o que cursaban la asignatura por segunda o más veces (repitentes).

En la Tabla 3 se presentan los criterios de inclusión y exclusión aplicados para la conformación de la muestra de estudio. Los criterios de inclusión garantizaron que todos los participantes tuvieran las mismas condiciones de partida: estar formalmente

matriculados en la asignatura, asistir regularmente, completar ambas evaluaciones y participar de forma voluntaria. Por su parte, los criterios de exclusión permitieron eliminar posibles fuentes de sesgo, tales como la irregularidad en la asistencia, la incompletitud de los datos o la condición de repitencia, que podría generar diferencias previas en el dominio de la asignatura. La aplicación rigurosa de estos criterios aseguró que la muestra final de 90 estudiantes fuera homogénea y apta para la comparación entre los grupos experimental y control.

Tabla 4

Resumen de la conformación de la muestra de estudio

Grupos	Matriculados	Excluidos	Muestra final (n)	Grupo asignado
Grupo A	50	5	45	Experimental
Grupo B	50	5	45	Control
Total	100	10	90	

Nota. Reporte de estudiantes matriculados en el semestre 2024-II.

En la Tabla 4 se muestra el resumen de la conformación de la muestra. Los 90 estudiantes matriculados en ambas secciones cumplieron con todos los criterios de inclusión y ninguno fue excluido, por lo que la muestra final coincidió con el total de matriculados en las dos secciones seleccionadas. La Sección A ($n = 45$) fue asignada como grupo experimental, al cual se le aplicó la intervención basada en el Aprendizaje Basado en Problemas, mientras que la Grupos B ($n = 45$) fue asignada como grupo control, el cual continuó con la metodología de enseñanza tradicional expositiva. La equivalencia inicial entre ambos grupos fue posteriormente verificada mediante la prueba U de Mann-Whitney en el pretest ($p = 0.599$), lo que confirmó que los grupos partieron de condiciones estadísticamente similares antes de la intervención.

3.9. Técnicas e instrumentos

3.9.1. Técnicas de recolección de datos

Para la recolección de datos se utilizó la técnica de la prueba pedagógica. Díaz Barriga (2002) sostiene que las pruebas pedagógicas constituyen un pilar fundamental para la evaluación del aprendizaje, ya que permiten recoger información y medir el nivel de conocimiento, habilidades y competencia de los estudiantes en un área específica del conocimiento.

Esta técnica fue aplicada en dos momentos:

- ✓ **Pretest:** Aplicado antes de la intervención experimental para medir el nivel inicial de competencia de programación en ambos grupos.
- ✓ **Posttest:** Aplicado después de la intervención experimental para medir el nivel final de competencia de programación y determinar el efecto del tratamiento.

Tabla 5

Técnicas de recolección de datos

Técnica		Propósito	Momento de aplicación
Prueba (Pretest)	pedagógica	Medir el nivel inicial de competencias de programación	Antes de la intervención
Prueba (Posttest)	pedagógica	Medir el nivel final de competencias de programación	Después de la intervención

3.9.2. Instrumento de recolección de datos

Según Hernández Sampieri y Mendoza (2018), un instrumento de medición es un recurso que utiliza el investigador para registrar información o datos sobre las variables

que tiene en mente. En esta línea, Bernal (2022) complementa que dicho instrumento debe ser válido y confiable para garantizar la calidad de los datos recolectados. Por su parte García Ramos (1989) argumenta que las pruebas de desarrollo constituyen instrumentos que posibilitan la evaluación de procesos cognitivos superiores, destacando que estas permiten valorar la capacidad de análisis, síntesis, evaluación y pensamiento crítico de los estudiantes.

El instrumento utilizado fue una prueba de desarrollo diseñado para evaluar las competencias de programación. La prueba estuvo estructurada en función de las tres dimensiones de la variable independiente:

Tabla 6

Estructura del instrumento de evaluación

Dimensión	Indicadores	N° de ítems
Pensamiento Lógico-Algorítmico	Analiza problemas, diseña algoritmos, representa soluciones mediante pseudocódigo y diagramas de flujo	10
Habilidades de Codificación	Implementa soluciones en Python, utiliza estructuras de control, maneja tipos de datos, aplica buenas prácticas	10
Depuración y Prueba	Identifica errores, corrige bugs, realiza pruebas de código, valida resultados	10
TOTAL		30

Tabla 7*Ficha técnica del instrumento de investigación*

Aspecto	Descripción
Nombre del instrumento	Prueba de desarrollo de competencias de programación
Autor	Pelayo Quispe Bautista
Año	2025
Objetivo	Evaluar las competencias de programación en Python
Población objetivo	Estudiantes de Ingeniería de Sistemas - UNSCH
Forma de administración	Individual
Tiempo de aplicación	45 minutos
Número de ítems	30 ítems
Escala de medición	Selección múltiple
Dimensiones	Pensamiento Lógico-Algorítmico, Habilidades de Codificación, Depuración y Prueba
Niveles de logro	Aprobado y Desaprobado nota vigesimal (0-20)

3.10. Validez y confiabilidad de instrumentos

3.10.1. Validez de los instrumentos

Según Hernández Sampieri y Mendoza (2018), la validez se refiere al grado en que un instrumento mide realmente la variable que pretende medir. Por su parte, Bernal (2022) señala que un instrumento de medición es válido cuando mide aquello para lo cual fue diseñado, es decir, cuando permite obtener los datos que se requiere conocer.

Validez de contenido

Según Galicia et al. (2017), la validez de contenido, un tipo de validez ampliamente utilizado en la investigación educativa, se establece mediante el juicio de

expertos, que consiste en obtener la opinión informada de profesionales cualificados sobre el instrumento.

Juicio de expertos

Según Skjong y Wentworth, (2001), el juicio de experto es una opinión informada basada en la formación y experiencia del experto en un área específica. Los expertos pueden proporcionar información, evidencia, juicios y evaluaciones, y estos términos a menudo se usan de manera intercambiable, aunque siguieren diferentes grados de valoraciones.

El instrumento fue validado por cuatro (4) expertos en el área de Ingeniería de Sistemas y metodología de la investigación, quienes evaluaron los 30 ítems del instrumento que fueron aplicados al experimento. En dicho instrumento se utilizó la categoría dicotómica (SI= 1, NO= 0); esta categoría corresponde a una escala nominal que, según Siegel y Castellan (1998), es el nivel más elemental de medición en el que los números funcionan únicamente como etiquetas para clasificar objetos en categorías mutuamente excluyentes, sin posibilidad de establecer orden ni realizar operaciones aritméticas entre ellas.

Para el análisis estadístico se aplicó la prueba binomial exacta. Siegel y Castellan, (1998) señalan que esta prueba es apropiada cuando los datos se encuentran en escala nominal dicotómica y se desea contrastar si la proporción observada difiere significativamente de una proporción esperada. La fórmula empleada es:

$$P[Y = k] = \binom{N}{k} p^k (1 - p)^{N-k} \quad k = 0, 1, \dots, N$$

Donde:

N: El número total de ensayos

k: El número de éxitos observados (ítems calificados como 'SI' o 1).

$P0$: La probabilidad de éxito bajo la hipótesis nula (0.50 en este caso).

$\binom{N}{k}$: El coeficiente binomial, que representa el número de formas de elegir k éxitos de N ensayos, calculado como $\frac{N!}{k!(N-k)!}$

Tabla 8

Los resultados de la prueba binomial se obtuvieron mediante Python.

	Categoría	N	Prop. Observada	Prop. de prueba	Significación exacta (bilateral)	Decisión
Experto 1	SI	28	0.933	0.50	0.000	Válido
	No	2	0.067			
	Total	30	1.000			
Experto 2	SI	29	0.967	0.50	0.000	Válido
	No	1	0.033			
	Total	30	1.000			
Experto 3	SI	29	0.967	0.50	0.000	Válido
	No	1	0.033			
	Total	30	1.000			
Experto 4	SI	29	0.967	0.50	0.000	Válido
	No	1	0.033			
	Total	30	1.000			

Nota: Decisión basada en $p < 0.05$ (bilateral). Prop. de prueba = 0.50 (criterio de aleatoriedad). Si = ítems bien planteados (valor 1); No = ítems no bien planteados (valor 0).

Como se observa en la Tabla 8 todos los expertos obtuvieron una significancia exacta bilateral de $p = 0.000 < 0.05$, lo que indica que la proporción de ítems bien valorados es estadísticamente significativa. Por lo tanto, el instrumento es válido según el juicio de los 4 expertos.

Tabla 9*Relación de expertos validadores del instrumento*

N°	Nombres y Apellidos	Grado Académico	Especialidad	Dictamen
1	Edith Felicita Guevara Morote	Magíster	Maestro en Ingeniería de Sistemas Mención Informática	Aplicable
2	Karel Peralta Sotomayor	Magíster	Maestro en Ingeniería Informática con Mención Ingeniería de Software	Aplicable
3	Juan Carlos Cerreño Gamarra	Magíster	Maestro en Ingeniería de Sistemas Mención en Informática	Aplicable
4	Christian Lezama Cuellar	Magíster	Maestro en Ciencias de Ingeniería; Mención en Gestión de Tecnologías de la Información y Comunicación	Aplicable

3.9.3. Confiabilidad de los instrumentos

De acuerdo con Hernández-Sampieri y Mendoza (2018), la confiabilidad de un instrumento de medición se define como el nivel de consistencia que exhibe al generar resultados equivalentes cuando es aplicado de manera reiterada sobre el mismo sujeto u objeto de estudio. Bernal (2022) sostiene que la confiabilidad constituye la uniformidad que evidencian las calificaciones obtenidas por los mismos individuos al ser evaluados en diferentes momentos temporales empleando instrumentos idénticos.

Para estimar la confiabilidad del instrumento se utilizó el coeficiente Kuder-Richardson 20, dado que la prueba de resolución de ejercicios está compuesta por ítem puntuados de forma dicotómica (correcto =1, Incorrecto=0). Según Kuder y Richardson

(1937), este coeficiente es el más apropiado para medir la consistencia interna de prueba con ítems de respuesta dicotómica.

Según Corral (2022), el KR-20 es equivalente al Alfa de Cronbach ya que ambos son métodos de análisis de homogeneidad que miden la consistencia interna de un instrumento con una sola aplicación a una prueba piloto, con valores que oscilan entre 0 y 1. La diferencia reside en las escalas de respuesta: Kuder-Richardson 20 se aplica a instrumentos con respuestas dicotómicas, como Si-No, Verdadero-Falso, o pruebas objetivas de selección simple donde las respuestas se califican como correctos o incorrectas, mientras que el Alfa de Cronbach se utiliza cuando las alternativas de respuesta son politómicas, siendo común en escalas ordinales como Liker o Diferencia Semántico.

Según Kuder y Richardson (1937), la fórmula KR-20 se expresa de la siguientes manera:

$$r_{KR20} = \frac{k}{k-1} \left(1 - \frac{\sum p_i q_i}{\sigma_t^2} \right)$$

Donde:

- k : Número total de preguntas.
- p_i : Proporción de individuos que acertaron el ítem i (media de respuestas correctas).
- q_i : Proporción de individuos de que fallaron el ítem i ($1-p_i$).
- $\sum p_i q_i$: Suma de los productos de p y q para todos los ítems.
- σ_t^2 : Varianza de las puntuaciones totales de los sujetos.

Para la interpretación del coeficiente de confiabilidad KR-20 se considera los criterios por (George y Mallery, 2003):

Tabla 10*Interpretación del coeficiente de confiabilidad KR-20*

Valor de KR-20	Interpretación
0.90 - 1.00	Excelente confiabilidad
0.80 - 0.89	Buena confiabilidad
0.70 - 0.79	Aceptable confiabilidad
0.60 - 0.69	Cuestionable confiabilidad
< 0.60	Inaceptable confiabilidad

Nota. Fuente: George y Mallery (2003)**Prueba piloto**

Según Hernández-Sampieri y Mendoza (2018), la prueba piloto es una aplicación preliminar del instrumento de recolección de datos a un grupo pequeño de personas con características similares a las de la muestra o población objetiva de la investigación.

Se realizó una prueba pilo con 30 estudiantes de características similares a la muestra de estudio.

Tabla 11*Resultado de confiabilidad del instrumento*

Instrumento	Nº de ítems	KR-20	Interpretación
Pensamiento Lógico Algorítmico	10	0.89	Buena
Habilidades de Codificación	10	0.90	Excelente
Depuración y Prueba	10	0.88	Buena
Promedio general		0.89	Buena

Nota. El coeficiente KR-20 promedio obtenido fue de 0.89, lo cual indica una confiabilidad buena del instrumento en sus tres dimensiones, permitiendo su aplicación para evaluar las competencias de programación.

3.11. Técnicas de procesamiento de datos

Según Hernández-Sampieri y Mendoza (2018), el análisis de datos se efectúa sobre la matriz de datos utilizando un programa computacional de análisis estadístico. Para el procesamiento y análisis de los datos recolectados se utilizaron las siguientes técnicas:

a) Estadística descriptiva

Según Córdova (2003), la estadística descriptiva es el conjunto de métodos para organizar, resumir y presentar datos de manera informativa. Se empleó para:

- Medidas de tendencia central: media aritmética y mediana.
- Medidas de dispersión: desviación estándar, rango (nota mínima y máxima).
- Gráficos estadísticos: diagramas de cajas (Comparación).

b) Estadística inferencial

Según Hernández-Sampieri y Mendoza (2018), la estadística inferencial se utiliza para probar hipótesis y estimar parámetros. Se realizaron las siguientes pruebas:

- Prueba de normalidad de Shapiro-Wilk para determinar la distribución de los datos, cuyos resultados indicaron distribuciones no normales en todas las variables del pretest y posttest ($p < 0.05$) en ambos grupos.
- Prueba U de Mann-Whitney para la comparación entre el grupo control y el grupo experimental (muestras independientes), al no cumplirse el supuesto de normalidad.
- Prueba de Rangos con Signo de Wilcoxon para la comparación del pretest y posttest dentro del grupo experimental (muestras relacionadas), por la misma razón.

c) Software utilizado

Se empleó el lenguaje de programación Python (versión 3.x), mediante las bibliotecas pandas y numpy para la tabulación y el análisis estadístico descriptivo e inferencial, y matplotlib para la elaboración de representaciones gráficas.

3.12. Plan de experimentación

La presente investigación adoptó un diseño cuasiexperimental con dos grupos de estudio: un grupo control y un grupo experimental. El objetivo principal fue implementar la metodología de Aprendizaje Basado en Problemas (ABP) con la finalidad de mejorar las competencias de programación en el lenguaje Python en estudiantes de la carrera profesional de Ingeniería de Sistemas. La población estuvo conformada por estudiantes matriculados en el curso de Algoritmos y Solución de Problemas de la Escuela Profesional de Ingeniería de Sistemas. El grupo experimental participó en la intervención pedagógica propuesta, mientras que el grupo control continuó desarrollando sus actividades académicas bajo la metodología de enseñanza tradicional.

El plan de experimentación se estructuró en cuatro etapas secuenciales, descritas a continuación:

Etapas 1: Diagnóstico inicial (Pretest)

En esta etapa se aplicó el instrumento de evaluación previamente validado por juicio de expertos, el cual midió tres dimensiones: Pensamiento Lógico-Algorítmico, Habilidades de Codificación, y Depuración y Prueba. El propósito de esta fase fue establecer una línea base que permitiera comparar el nivel de competencias de programación entre el grupo control y el grupo experimental antes de la intervención. Los resultados obtenidos posibilitaron identificar las condiciones iniciales de ambos grupos y verificar su equivalencia previa al tratamiento.

Etapas 2: Intervención experimental

El grupo experimental fue sometido a la aplicación de la metodología de Aprendizaje Basado en Problemas orientada al desarrollo de competencias en programación con Python. Esta intervención pedagógica permitió a los estudiantes enfrentarse a situaciones problemáticas contextualizadas, a partir de las cuales debían

construir soluciones algorítmicas. Las sesiones de aprendizaje estuvieron diseñadas en correspondencia con las tres dimensiones evaluadas: Pensamiento Lógico-Algorítmico, Habilidades de Codificación, y Depuración y Prueba. Cada sesión tuvo una duración de 90 minutos, programada en coordinación con los docentes responsables del curso. El grupo control, por su parte, continuó con el enfoque tradicional de enseñanza, centrado en la identificación de sintaxis y la transmisión de conceptos teóricos.

Etapa 3: Evaluación final (Postest)

Al término de la intervención, se aplicó el mismo instrumento de evaluación a ambos grupos. El objetivo de esta fase fue medir y comparar el nivel de competencias en programación con Python alcanzado por cada grupo tras el período de experimentación, con el fin de determinar el efecto de la metodología implementada.

Etapa 4: Análisis de datos

Los datos recopilados en el pretest y postest fueron procesados mediante técnicas de estadística descriptiva e inferencial. Se aplicaron pruebas paramétricas para determinar la significancia estadística de las diferencias observadas entre el grupo experimental y el grupo control, lo que permitió contrastar las hipótesis planteadas en la investigación.

3.13. Aspectos éticos

Según Lema et al. (2010), ética en la investigación es un conjunto de morales y directrices que rigen el diseño, la realización, el análisis y la presentación de informes de estudios que involucran sujetos humanos o animales, con el fin de garantizar que la investigación sea conducida de manera responsable y justos, protegiendo los derechos y el bienestar de los participantes. Esta perspectiva es complementada por (Guillemin y Gillam, 2004), quienes fundamenta que la ética se refiere al conjunto de principios y marcos conceptuales que guían la conducta en la investigación, asegurando que esta se realice de manera responsable y justa.

La presente investigación se desarrolló respetando los principios éticos establecidos en el Código de Ética de la Universidad Nacional San Cristóbal de Huamanga y las normas internacionales de investigación científica. Se consideraron los siguientes aspectos:

a) Consentimiento informado

Todos los participantes fueron informados sobre los objetivos, procedimientos, beneficios y posibles riesgos de la investigación. Los participantes tuvieron la libertad de retirarse en cualquier momento sin que esto afecte su situación académica.

b) Confidencialidad

Se garantizó la confidencialidad de la información proporcionada por los participantes. Los datos personales fueron codificados y almacenados de manera segura, siendo utilizados únicamente con fines de investigación. Los resultados se presentan de forma agregada, sin identificar a ningún participante individual.

c) Anonimato

Se protegió la identidad de los participantes mediante el uso de códigos en lugar de nombres. En ningún momento se reveló información que permita identificar a los estudiantes participantes.

d) No maleficencia

Se garantizó que la participación en la investigación no causó ningún daño físico, psicológico o académico a los estudiantes. El grupo control también recibió una enseñanza de calidad siguiendo el método tradicional.

e) Originalidad y honestidad científica

El investigador se compromete a presentar resultados verídicos y a respetar la propiedad intelectual de otros autores, citando adecuadamente todas las fuentes consultadas según las normas APA 7ma edición. La investigación fue sometida a verificación de originalidad mediante software antiplagio.

f) Responsabilidad

El investigador asume la responsabilidad de cumplir con todos los procedimientos establecidos y de informar cualquier eventualidad que pudiera surgir durante el desarrollo de la investigación.

IV. Resultados y discusión

4.1. Resultados descriptivos

El análisis estadístico descriptivo de los datos obtenidos en las fases de pretest y posttest fue procesado mediante el lenguaje de programación Python (versión 3.x), empleando las bibliotecas pandas, numpy y matplotlib para la manipulación de datos, el cálculo de estadísticos y la generación de representaciones gráficas, respectivamente. Para cada variable: Pensamiento Lógico Algorítmico (PLA), Habilidades de Codificación (HC), Depuración y Prueba (DP) y Promedio General (CP); se reportan la media aritmética ($\bar{x}$), la mediana (Md), la desviación estándar (DE) y el rango (R), tanto para el grupo control como para el grupo experimental, acompañados de diagramas de caja que permiten comparar visualmente la distribución, tendencia central y dispersión de los puntajes en ambos momentos de evaluación.

Para la evaluación de entrada (pretest), se reportan la media aritmética, la mediana, la desviación estándar y el rango de cada variable, tanto para el grupo control como para el grupo experimental, con el propósito de verificar la equivalencia de las condiciones de partida previo a la intervención. Los resultados se presentan en la Tabla 12 y se ilustran mediante los diagramas de caja de las Figuras 2 a 5.

Tabla 12

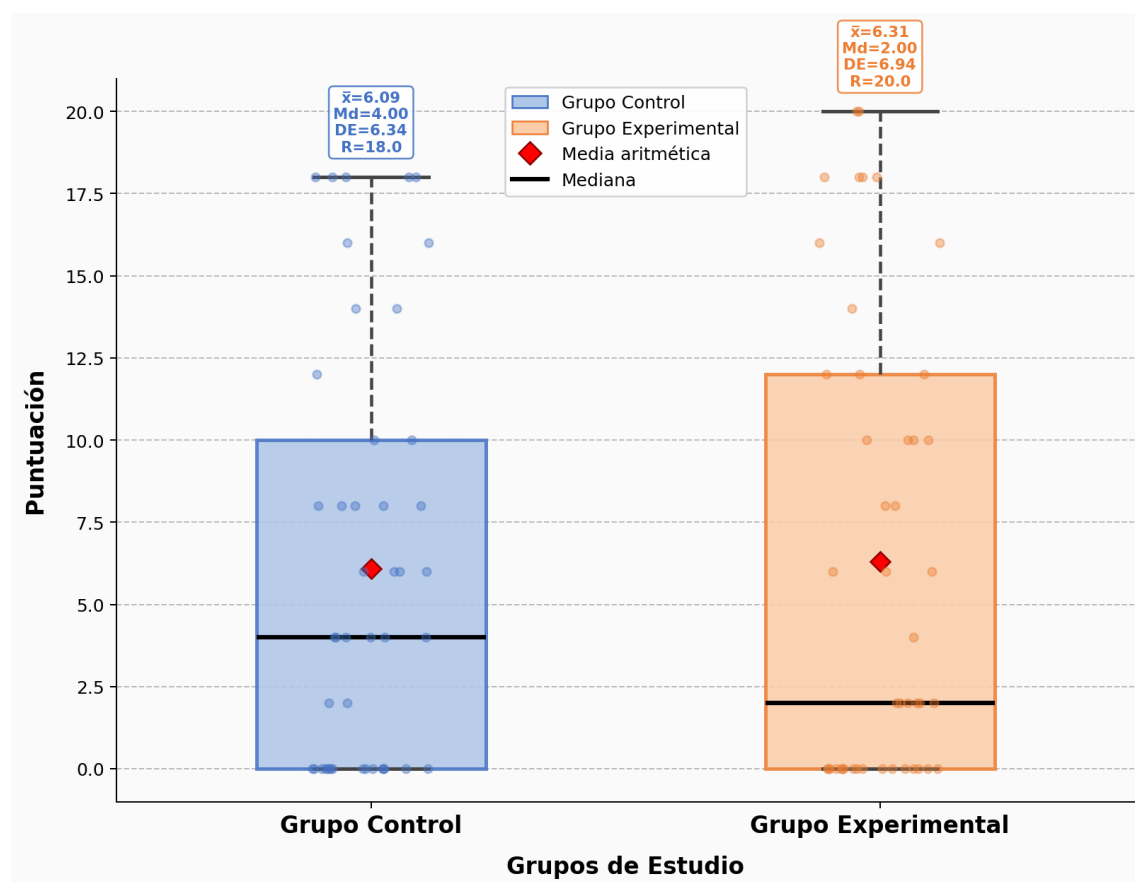
Estadísticos descriptivos del pretest por grupo y variable

	Pretest Control				Pretest Experimental			
	PLA	HC	DP	CP	PLA	HC	DP	CP
$\bar{x}$	6.09	6.08	6.58	6.25	6.31	6.22	6.40	6.31
Md	4	4	4	7.33	2	6	4	6
DE	6.33	6.38	6.05	3.50	6.94	5.15	6.45	3.68
R	18	18	18	14	20	18	20	13.3

Nota. $\bar{x}$ = media aritmética; Md = mediana; DE = desviación estándar; R = rango; PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General.

Figura 2

Comparación del pretest en PLA entre el grupo control y el grupo experimental.



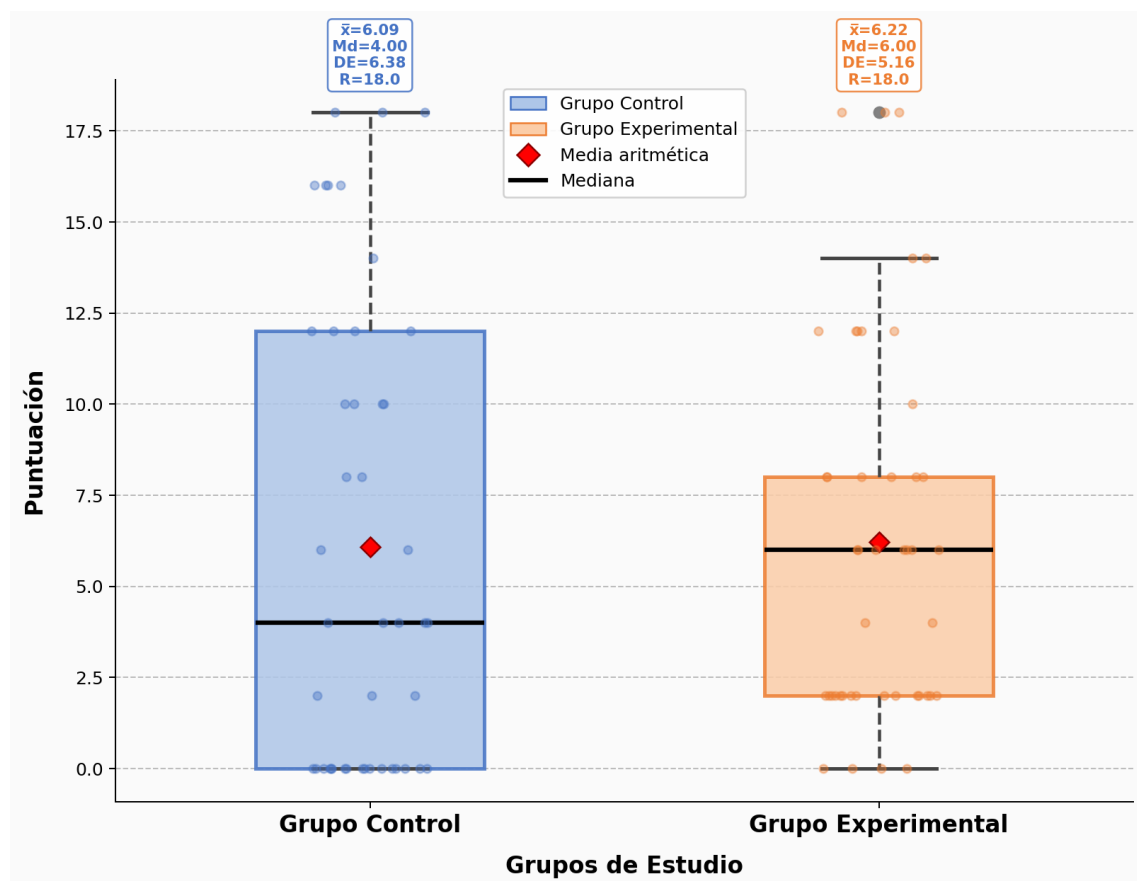
Nota. Generado con Python 3

Los resultados del pretest en la variable pensamiento lógico algorítmico muestran que ambos grupos partieron desde condiciones iniciales muy similares. El grupo control registró una media aritmética de $\bar{x} = 6.09$ (DE = 6.34), una mediana de Md = 4.00 y un rango de R = 18, con puntuaciones comprendidas entre 0 y 18 puntos. El grupo experimental presentó una media de $\bar{x} = 6.31$ (DE = 6.94), una mediana de Md = 2.00 y un rango de R = 20. En ambos grupos el primer cuartil se situó en 0 puntos, lo que indica

que al menos el 25% de los participantes no obtuvo ninguna puntuación en esta dimensión. La discrepancia entre la media y la mediana especialmente en el grupo experimental ($\bar{x} = 6.31$ vs. $Md = 2.00$) revela distribuciones con asimetría positiva. Las desviaciones estándar elevadas y similares ($DE = 6.34$ y $DE = 6.94$) confirman una alta heterogeneidad interna en ambos grupos. En conjunto, estos indicadores permiten concluir que ambos grupos al inicio del estudio, presentan condiciones de partida equivalentes en cuanto al pensamiento lógico algorítmico, sin diferencias sistemáticas relevantes entre ellos.

Figura 3

Comparación del pretest en HC entre el grupo control y el grupo experimental.

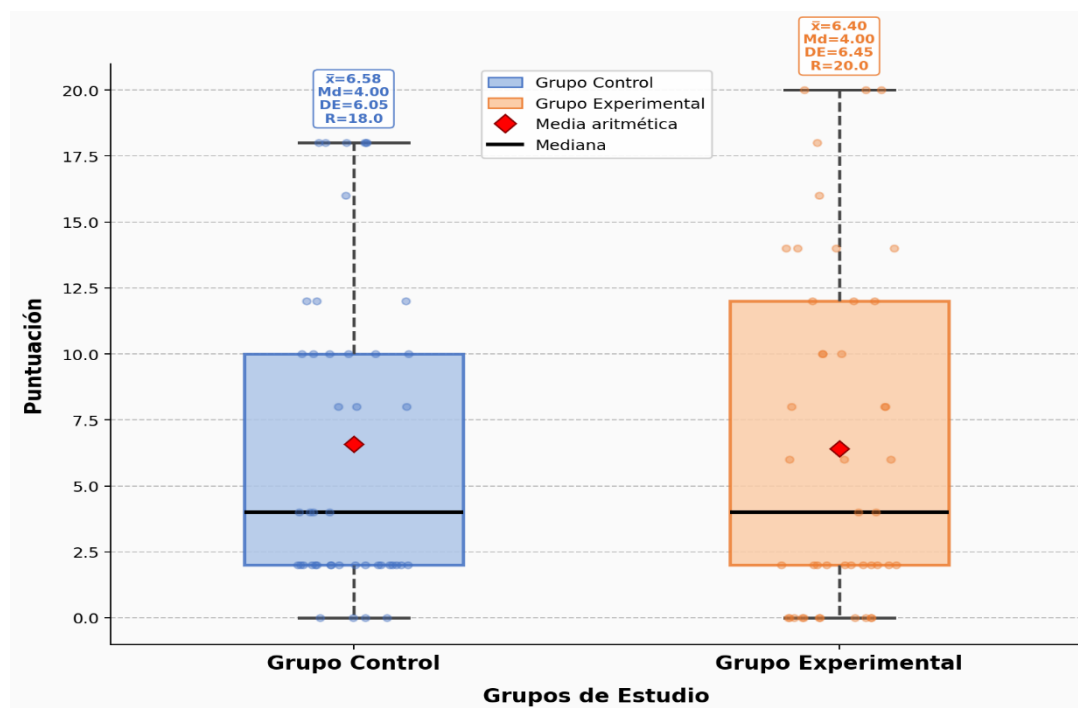


Nota. Generado con Python 3

En la variable Habilidades de Codificación, los datos del pretest reflejan una gran similitud entre los grupos. El grupo control obtuvo una media de $\bar{x} = 6.09$ (DE = 6.38), una mediana de Md = 4.00 y un rango de R = 18. El grupo experimental registró una media de $\bar{x} = 6.22$ (DE = 5.16), una mediana de Md = 6.00 y un rango de R = 18. Las cajas de ambos grupos se superponen considerablemente en la región baja-media de la escala. La distribución del grupo experimental resultó ligeramente más concentrada (DE = 5.16 frente a DE = 6.38 del control), aunque ambas desviaciones estándar siguen siendo elevadas en términos relativos. La similitud entre las medianas (Md = 4.00 y Md = 6.00) y las medias de ambos grupos corrobora la equivalencia inicial en esta variable, confirmando que las habilidades de codificación no diferían de manera sustancial entre los grupos antes de la intervención.

Figura 4

Comparación del pretest en DP entre el grupo control y el grupo experimental.

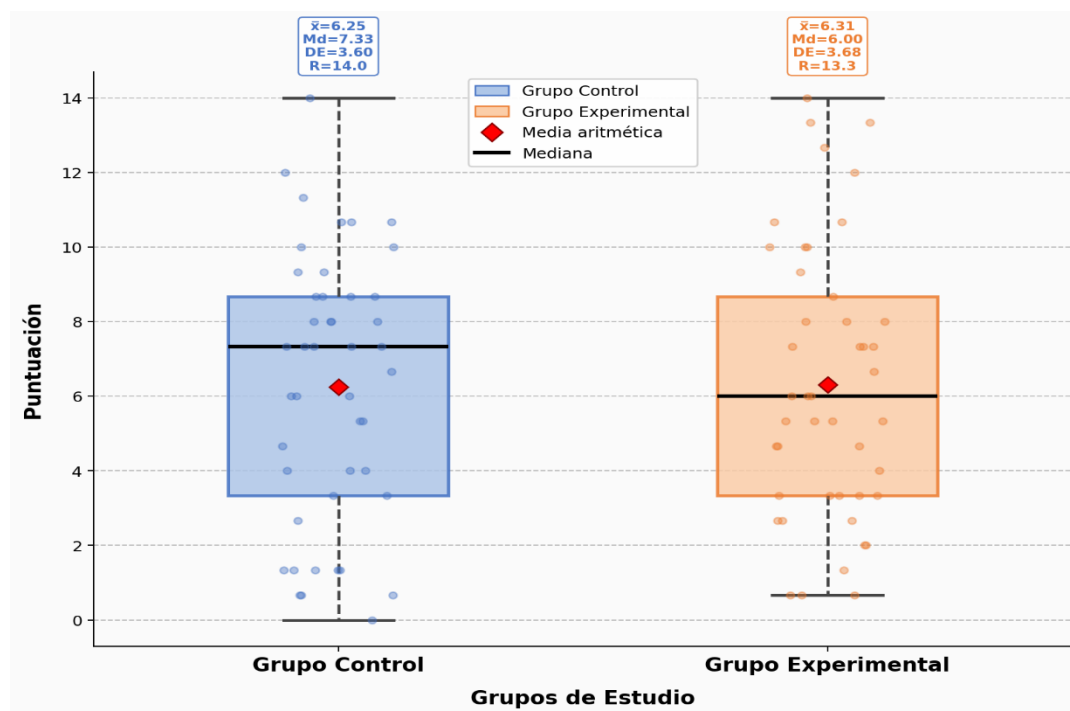


Nota. Generado con Python3

La variable Depuración y Prueba en el pretest es la que muestra mayor semejanza entre ambos grupos en todos sus estadísticos descriptivos. El grupo control presentó una media de $\bar{x} = 6.58$ (DE = 6.05), una mediana de Md = 4.00 y un rango de R = 18. El grupo experimental tuvo una media de $\bar{x} = 6.40$ (DE = 6.45), una mediana de Md = 4.00 y un rango de R = 20. El indicador más claro que permite asegurar la igualdad de condiciones iniciales es la coincidencia de las medianas de los dos grupos (Md = 4.00). Las cajas se superponen en gran medida y las desviaciones estándar comparables indican una dispersión equivalente. La ligera superioridad del rango del grupo experimental (R=20 frente a R=18) se explica por la presencia de valores máximos de 20 puntos en dicho grupo. Los datos corroboran que ambos grupos eran estadísticamente equivalentes en la dimensión de depuración y prueba al inicio del estudio.

Figura 5

Comparación del pretest en el CP entre el grupo control y el grupo experimental.



Nota. Generado con Python 3

El Promedio General del pretest incluye el desempeño inicial de los participantes en las tres dimensiones evaluadas. El grupo control obtuvo una media de $\bar{x} = 6.25$ (DE = 3.60), una mediana de Md = 7.33, y un rango de R = 14.00. El grupo experimental obtuvo una media de $\bar{x} = 6.31$ (DE = 3.68), una mediana de Md = 6.00 y un rango de R = 13.33. Los valores de ambos grupos se solapan casi por completo, con rangos intercuartílicos idénticos (Q1 = 3,33; Q3 = 8,67 en ambos casos) y valores máximos prácticamente idénticos (R = 14,00 en ambos grupos). Desviaciones estándar comparables (DE = 3.60 y DE = 3.68) verifican niveles similares de variabilidad interna. Estos resultados son una prueba fehaciente de que, antes de la intervención, ambos grupos mostraron un rendimiento inicial similar y comparable, lo cual fortalece la validez del diseño cuasiexperimental.

Los mismos estadísticos descriptivos $\bar{x}$, Md, DE y R se reportan para la evaluación de salida (postest) para cada variable y grupo, con la intención de examinar el rendimiento alcanzado por ambos grupos tras la intervención y evidenciar las diferencias en la distribución y tendencia central de los puntajes entre el grupo control y el grupo experimental. Los resultados se muestran en la Tabla 13 y se ilustran en las Figuras 6 a 9..

Tabla 13

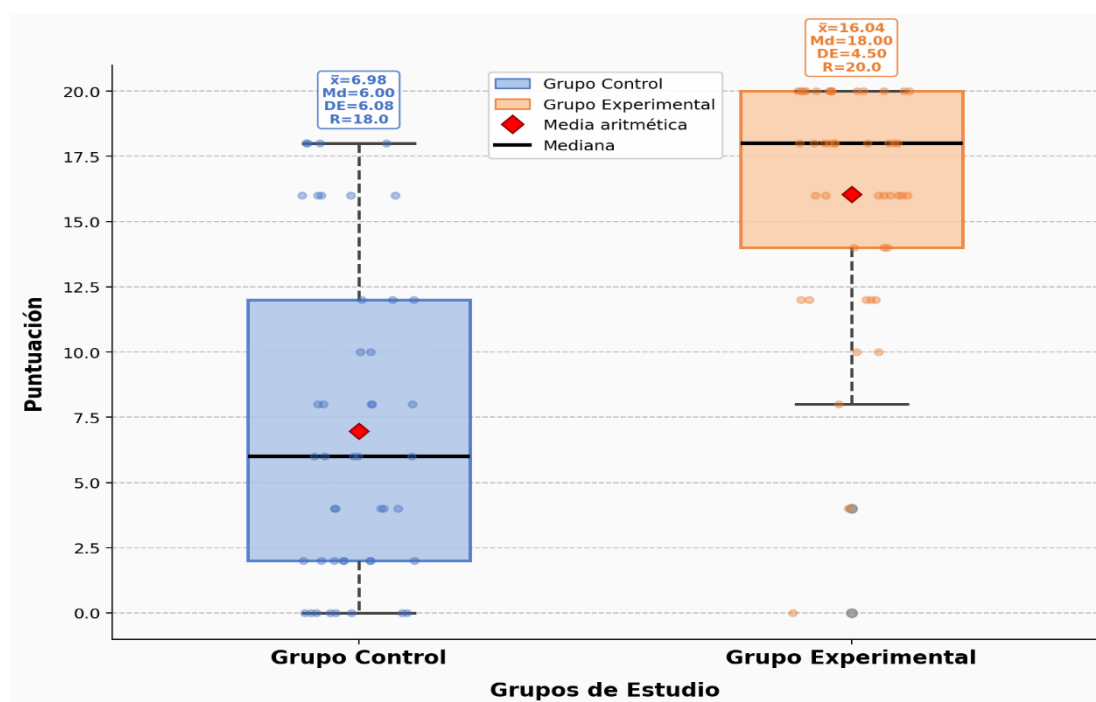
Estadísticos descriptivos del postest por grupo y variable

	Postest Control				Postest Experimental			
	PLA	HC	DP	CP	PLA	HC	DP	CP
$\bar{x}$	7	7	7.2	7.3	16	16	15.42	15.7
Md	6	6	4	6.7	18	18	18	16
DE	6.1	6.1	6.7	3	4.5	4.5	5.7	2.6
R	18	20	20	10.7	20	18	18	9.3

Nota. $\bar{x}$ = media aritmética; Md = mediana; DE = desviación estándar; R = rango; PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General.

Figura 6

Comparación del posttest en PLA entre el grupo control y el grupo experimental.



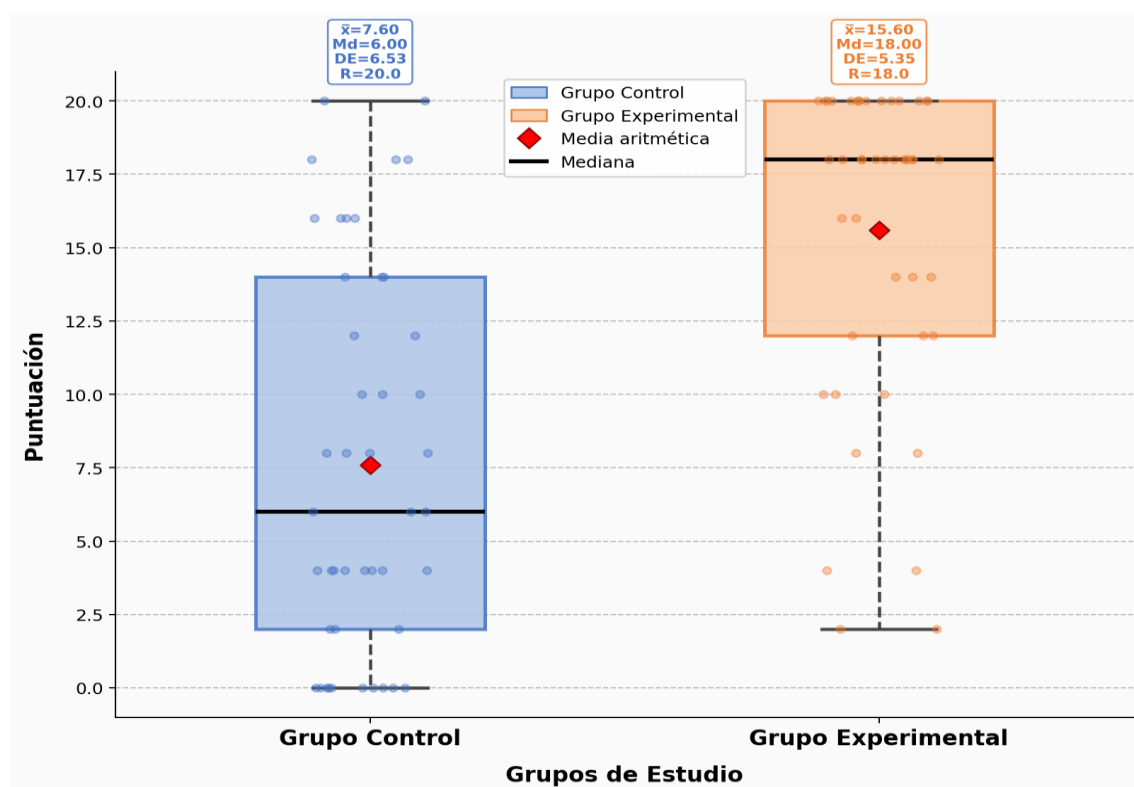
Nota. Generado con Python 3

El análisis del posttest en la variable Pensamiento Lógico Algorítmico muestra las diferencias de alta magnitud entre ambos grupos, contrastando de forma marcada con la equivalencia observada en el pretest. El grupo control obtuvo una media aritmética de $\bar{x} = 6.98$ (DE = 6.08), una mediana de Md = 6.00 y un rango de R = 18, con puntuaciones distribuidas entre 0 y 18 puntos. En contraste, el grupo experimental alcanzó $\bar{x} = 16.04$ (DE = 4.50), Md = 18.00 y R = 20, con puntajes concentrados en los valores superiores de la escala. La caja correspondiente al grupo experimental se desplazó de manera notoria hacia la parte alta del eje de puntuaciones, con el 50% central de los datos ubicado entre 14 y 20 puntos, lo que indica que la mayoría de los participantes de este grupo alcanzó

niveles de logro elevados en la variable. El grupo control, por su parte, concentró su 50% central entre 2 y 12 puntos, con una distribución más heterogénea y más extendida hacia los valores bajos de la escala. La mayor dispersión observada en el grupo control ($DE = 6,08$) frente al experimental ($DE = 4,50$) indica que los participantes de este grupo mostraron desempeños más variables y menos consolidados, mientras que el grupo experimental tendió a homogeneizarse en torno a niveles de logro superiores. Estos resultados permiten deducir que la intervención aplicada tuvo un efecto positivo, significativo y consistente en el desarrollo del pensamiento lógico algorítmico de los estudiantes del grupo experimental.

Figura 7

Comparación del posttest en HC entre el grupo control y el grupo experimental



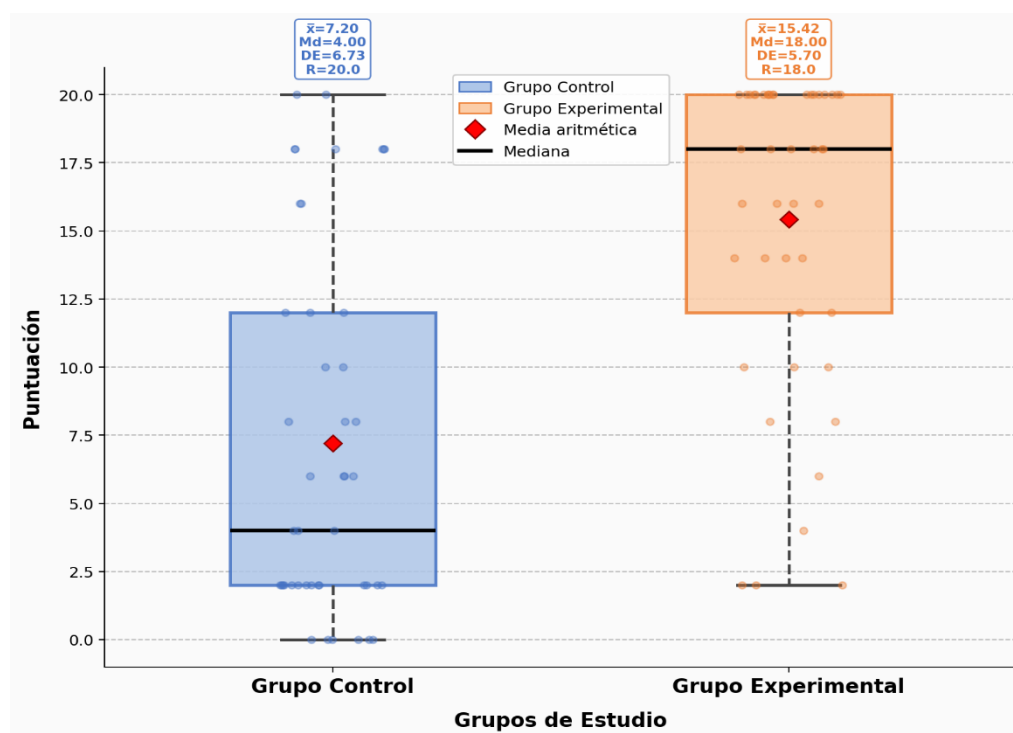
Fuente. Creado con Python 3

En la variable HC, el posttest muestra una diferencia notoria entre los grupos. El grupo control obtuvo $\bar{x} = 7.60$ ($SD = 6.53$), $Md = 6.00$ y $R = 20$, lo que indica gran

variabilidad y puntajes diseminados a lo largo de toda la escala. El grupo experimental obtuvo $\bar{x} = 15.60$ ($DE = 5.35$), $Md = 18.00$, $R = 18$, con distribución concentrada en los valores altos. La caja del grupo experimental está claramente por encima de la del grupo control, con su primer cuartil en aproximadamente 12 puntos y su tercer cuartil en 20 puntos. La mayor desviación estándar del grupo control ($DE = 6.53$) que del grupo experimental ($DE = 5.35$) indica una mayor dispersión, con participantes desde niveles muy bajos hasta niveles relativamente altos. Estos resultados indican que la intervención experimental promovió un desarrollo más uniforme de las habilidades de codificación y a un nivel de logro superior.

Figura 8

Comparación del postest en DP entre el grupo control y el grupo experimental.



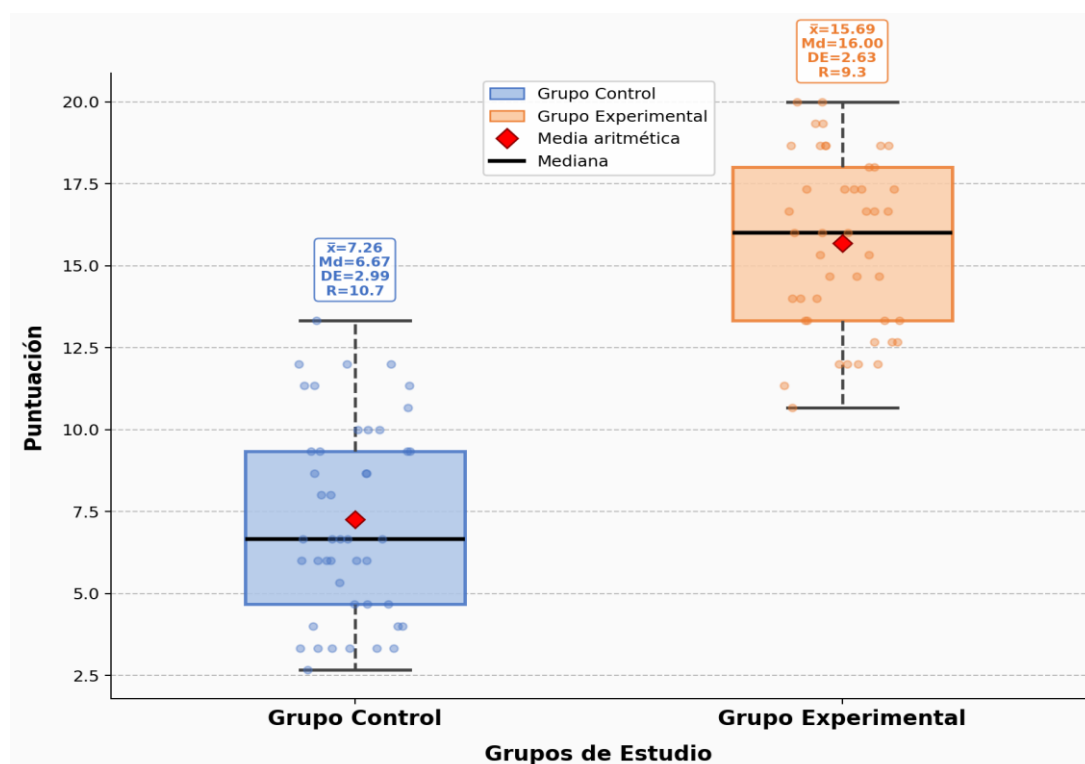
Nota. Creado con Python 3.

Los resultados del postest en la variable Depuración y Prueba muestran las diferencias más pronunciadas en tendencia central. El grupo control presentó $\bar{x} = 7.20$

(DE = 6.73), Md = 4.00 y R = 20, con distribución marcadamente sesgada hacia los valores bajos: la mitad de los participantes obtuvo puntajes iguales o inferiores a 4 sobre 20. El grupo experimental alcanzó $\bar{x}$ = 15.42 (DE = 5.70), Md = 18.00 y R = 18. La diferencia entre la media ($\bar{x}$ = 7.20) y la mediana (Md = 4.00) en el grupo control sugiere que la media fue influida por algunos participantes con puntuaciones altas; por lo tanto, la mediana podría ser un indicador más representativo. El hecho de que el grupo control tenga una desviación estándar más alta (DE = 6,73 frente a DE = 5,70) indica que hay más heterogeneidad interna en ese grupo. Los resultados señalan que las destrezas de depuración y prueba se desarrollaron de forma significativamente más sólida en el grupo experimental.

Figura 9

Comparación del posttest en el CP entre el grupo control y el grupo experimental.



Nota. Generado por Python 3.

Los resultados del Promedio General en el postest ofrecen la evidencia más clara y contundente del efecto de la intervención sobre el rendimiento final de los estudiantes en las tres dimensiones evaluadas. El grupo control obtuvo una media de $\bar{x}=7.26$ (DE=2.99), una mediana de Md=6.67 y un rango de R=10.67, con puntuaciones que oscilaron entre 2.67 y 13.33 puntos; mientras que el grupo experimental obtuvo $\bar{x}=15.69$ (DE=2.63), Md=16.00 y R=9.33, con valores comprendidos entre 10.67 y 20.00 puntos. El diagrama muestra una separación clara y prácticamente sin superposición: el mínimo del grupo experimental (≈ 10.67) está muy por encima de la mediana del grupo control (≈ 6.67). Las desviaciones estándar similares (2.99 vs 2.63) indican homogeneidad interna comparable entre ambos grupos, pero en rangos de rendimiento completamente diferentes. Estos resultados son una prueba consistente de que la intervención puesta en marcha tuvo un efecto positivo.

4.2. Resultados a nivel inferencial

El análisis inferencial tuvo como propósito contrastar las hipótesis planteadas en la investigación. Para ello, se siguió el protocolo estadístico establecido en la metodología, el cual se estructuró en dos etapas. En la primera etapa se verificó el supuesto de normalidad de los datos mediante la prueba de Shapiro-Wilk, recomendada para muestras de hasta 50 participantes por grupo (Shapiro y Wilk, 1965). Los resultados de esta prueba determinaron la selección del estadístico de contraste más adecuado. En la segunda etapa, dado que las distribuciones no cumplieron el supuesto de normalidad, se aplicó la Prueba U de Mann-Whitney para muestras independientes en las comparaciones entre grupos y la Prueba de Rangos con Signo de Wilcoxon para muestras relacionadas en la comparación pretest-posttest del grupo experimental, ambas pruebas no paramétricas

que no asumen normalidad en los datos. El nivel de significancia empleado en todas las pruebas fue $\alpha = 0.05$.

4.2.1. Prueba de normalidad

a. Planteamiento de hipótesis

H₀: Los datos del postest siguen una distribución normal.

H₁: Los datos del postest no siguen una distribución normal.

b. Nivel de significancia

Se estableció un nivel de significancia de $\alpha = 0.05$. Se concluye ausencia de normalidad cuando el valor p obtenido es menor a este umbral.

c. Prueba estadística a emplear

Se aplicó la prueba de Shapiro-Wilk a los puntajes del postest de ambos grupos en las tres variables. Los resultados se presentan en la Tabla 14.

Tabla 14

Resultados de la prueba de normalidad Shapiro-Wilk posttest

Variable	Grupo Control		Grupo Experimental		Distribución
	W	p	W	p	
PLA	0.883	0.000	0.815	0.000	No normal
HC	0.900	0.001	0.789	0.000	No normal
DP	0.841	0.000	0.791	0.000	No normal
CP	0.943	0.029	0.950	0.053	No normal / Normal*

Nota. GC = Grupo Control (n = 45); GE = Grupo Experimental (n = 45). Se concluye distribución no normal cuando $p < 0.05$. *En CP, únicamente el grupo control no cumple el supuesto (W = 0.943, p = 0.029). PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General).

Los resultados presentados en la Tabla 14 muestran que las variables PLA, HC y DP no siguen una distribución normal en ninguno de los dos grupos ($p < 0.05$ en todos los casos). En el Promedio General (CP), el grupo experimental evidenció una distribución aproximadamente normal ($W = 0.950$, $p = 0.053 > 0.05$); sin embargo, el grupo control no cumplió este supuesto ($W = 0.943$, $p = 0.029 < 0.05$). Este comportamiento es consistente con la naturaleza dicotómica del instrumento de evaluación, lo que genera distribuciones de puntajes totales con tendencia a la asimetría. Dado que $p < 0.05$ en al menos uno de los grupos para todas las variables, se rechaza H_0 y se acepta H_1 : los datos no siguen una distribución normal. En consecuencia, se descarta el uso de pruebas paramétricas y se opta por pruebas no paramétricas para el contraste de todas las hipótesis de investigación.

En resumen, las competencias de programación antes de la aplicación de aprendizaje basado en problemas eran deficiente tanto en el grupo control y experimental

4.2.2. Contraste de hipótesis de pretest (grupo control y experimental)

Antes de implementar la intervención pedagógica, se verificó estadísticamente la equivalencia inicial entre el grupo control y el grupo experimental mediante la Prueba U de Mann-Whitney. Esta verificación es fundamental en los diseños cuasiexperimentales, ya que permite confirmar que cualquier diferencia observada en el posttest es atribuible al tratamiento y no a condiciones previas distintas entre los grupos.

a. Sistema de hipótesis

H_0 : No existe diferencia significativa entre el grupo control y el grupo experimental en las competencias de programación antes de la aplicación del Aprendizaje Basado en Problemas.

H₁: Existe diferencia significativa entre el grupo control y el grupo experimental en las competencias de programación antes de la aplicación del Aprendizaje Basado en Problemas.

b. Nivel de significancia

Se estableció un nivel de significancia de $\alpha = 0.05$, con una prueba bilateral dado que el objetivo es detectar diferencias en cualquier dirección entre los grupos.

c. Estadígrafo

Se empleó la Prueba U de Mann-Whitney para muestras independientes, previa verificación del supuesto de normalidad mediante la prueba de Shapiro-Wilk. Esta elección se justifica dado que, como se muestra en la Tabla 15, las variables PLA, HC y DP no siguen una distribución normal en al menos uno de los grupos ($p < 0.05$), condición que invalida el uso de pruebas paramétricas como la t de Student.

Tabla 15

Prueba de normalidad Shapiro-Wilk - pretest

Variable	Grupo Control		Grupo Experimental		Distribución
	W	p	W	p	
PLA	0.840	< 0.001	0.825	< 0.001	No normal
HC	0.838	< 0.001	0.881	< 0.001	No normal
DP	0.818	< 0.001	0.857	< 0.001	No normal
CP	0.959	0.112	0.959	0.111	Normal

Nota. GC = Grupo Control (n = 45); GE = Grupo Experimental (n = 45). Se concluye distribución no normal cuando $p < 0.05$. PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General.

Los resultados muestran que PLA, HC y DP presentan distribuciones no normales en ambos grupos ($p < 0.001$). El Promedio General (CP) evidenció normalidad en ambos

grupos ($p = 0.112$ y $p = 0.111$, respectivamente); no obstante, en aras de la consistencia metodológica se aplicó la Prueba U de Mann-Whitney de manera uniforme para todas las variables.

d. Resultados

Tabla 16

Prueba U de Mann-Whitney - pretest por variable

Variable	U	Z	p	Decisión
PLA	1009.50	-0.024	0.984	No se rechaza H0
HC	907.00	-0.851	0.391	No se rechaza H0
DP	1078.50	0.533	0.590	No se rechaza H0
CP	1032.50	0.161	0.875	No se rechaza H0

Nota. U = estadístico de Mann-Whitney; Z = puntuación estandarizada; p = significancia bilateral. $\alpha = 0.05$. n = 45 por grupo. PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General.

Los resultados de la Prueba U de Mann-Whitney aplicada al pretest se presentan en la Tabla 16. En la variable Pensamiento Lógico-Algorítmico, el estadístico $U = 1009.50$, $Z = -0.024$ y $p = 0.984$ indican que no existe diferencia significativa entre los grupos. En Habilidades de Codificación, $U = 907.00$, $Z = -0.851$ y $p = 0.391$ confirman igualmente la equivalencia inicial. En Depuración y Prueba, $U = 1078.50$, $Z = 0.533$ y $p = 0.590$ muestran que ambos grupos partieron desde condiciones prácticamente idénticas en esta dimensión. Finalmente, en el Promedio General, $U = 1032.50$, $Z = 0.161$ y $p = 0.875$ corroboran la homogeneidad de los grupos antes de la intervención.

Dado que en todas las variables el valor p es ampliamente superior al nivel de significancia establecido ($p \gg \alpha = 0.05$), no se rechaza la hipótesis nula. Esto confirma

que, previo a la intervención, el grupo control y el grupo experimental presentaban condiciones iniciales estadísticamente equivalentes en las competencias de programación evaluadas. Este hallazgo es fundamental para la validez interna del diseño cuasiexperimental, pues garantiza que las diferencias observadas en el posttest sean atribuibles al efecto del Aprendizaje Basado en Problemas y no a diferencias preexistentes entre los grupos.

4.2.3. *Contraste de hipótesis de posttest (grupo control y experimental)*

Una vez aplicada la intervención pedagógica basada en el Aprendizaje Basado en Problemas, se procedió a contrastar las hipótesis de investigación mediante la comparación de los puntajes del posttest entre el grupo control y el grupo experimental en las cuatro variables evaluadas.

a. Sistema de hipótesis

H₀: No existe diferencia significativa entre el grupo control y el grupo experimental en las competencias de programación después de la aplicación del Aprendizaje Basado en Problemas.

H₁: Existe diferencia significativa entre el grupo control y el grupo experimental en las competencias de programación después de la aplicación del Aprendizaje Basado en Problemas.

b. Nivel de significancia

Se estableció un nivel de significancia de $\alpha = 0.05$ con una prueba unilateral, dado que la hipótesis de investigación establece que el grupo experimental obtendrá puntajes superiores al grupo control tras la intervención.

c. Estadígrafo

Se empleó la Prueba U de Mann-Whitney para muestras independientes, previa verificación del supuesto de normalidad mediante la prueba de Shapiro-Wilk. Como se aprecia en la Tabla 17, las variables PLA, HC y DP no siguen una distribución normal en ninguno de los dos grupos ($p < 0.05$). En el caso del Promedio General (CP), el grupo control tampoco cumple el supuesto ($p = 0.029 < 0.05$). En consecuencia, se aplicó la Prueba U de Mann-Whitney de manera uniforme para todas las variables.

Tabla 17

Prueba de normalidad Shapiro-Wilk - posttest

Variable	Grupo Control		Grupo Experimental		Distribución
	W	p	W	p	
PLA	0.883	< 0.001	0.815	< 0.001	No normal
HC	0.900	0.001	0.789	< 0.001	No normal
DP	0.841	< 0.001	0.791	< 0.001	No normal
CP	0.943	0.029	0.950	0.053	No normal / Normal*

Nota. GC = Grupo Control ($n = 45$); GE = Grupo Experimental ($n = 45$). Se concluye distribución no normal cuando $p < 0.05$. *En CP, solo el GE cumple el supuesto ($p = 0.053 > 0.05$). PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General.

d. Resultados

Tabla 18

Prueba U de Mann-Whitney - posttest por variable

Variable	U	Z	p	Decisión
PLA	261.50	-6.060	< 0.001	Se rechaza H_0
HC	352.50	-5.326	< 0.001	Se rechaza H_0

Variable	U	Z	p	Decisión
DP	362.50	-5.245	< 0.001	Se rechaza H_0
CP	28.00	-7.945	< 0.001	Se rechaza H_0

Nota. U = estadístico de Mann-Whitney; Z = puntuación estandarizada; p = significancia unilateral. $\alpha = 0.05$. n = 45 por grupo. PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General.

Los resultados de la Prueba U de Mann-Whitney aplicada al postest, presentados en la Tabla 18, evidencian diferencias estadísticamente significativas entre el grupo control y el grupo experimental en las cuatro variables evaluadas. En la variable Pensamiento Lógico-Algorítmico, el estadístico $U = 261.50$, $Z = -6.060$ y $p < 0.001$ indican que el grupo experimental obtuvo puntajes significativamente superiores al grupo control tras la intervención. En Habilidades de Codificación, $U = 352.50$, $Z = -5.326$ y $p < 0.001$ confirman la misma tendencia, evidenciando una mejora sustancial en el grupo experimental. En la variable Depuración y Prueba, $U = 362.50$, $Z = -5.245$ y $p < 0.001$ revelan que esta dimensión también registró diferencias significativas a favor del grupo experimental. Finalmente, en el Promedio General, $U = 28.00$, $Z = -7.945$ y $p < 0.001$ constituyen el resultado más contundente de todos, reflejando el impacto global de la intervención sobre el conjunto de competencias de programación evaluadas.

Dado que en todas las variables el valor $p < 0.001$ es inferior al nivel de significancia establecido ($\alpha = 0.05$), se rechaza la hipótesis nula y se acepta H_1 . Estos resultados demuestran que la aplicación del Aprendizaje Basado en Problemas generó diferencias estadísticamente significativas en las competencias de programación de los estudiantes de Ingeniería de Sistemas de la UNSCH, siendo el grupo experimental el que alcanzó niveles de rendimiento notoriamente superiores al grupo control en el postest.

4.2.4. *Contraste de hipótesis de pretest y postes grupo experimental*

Con la finalidad de evaluar el efecto de la intervención pedagógica dentro del propio grupo experimental, se compararon los puntajes del pretest y el posttest mediante la Prueba de Wilcoxon para muestras relacionadas. Esta prueba permite determinar si los cambios observados en los puntajes de un mismo grupo antes y después de la intervención son estadísticamente significativos.

4.2.5. *Prueba de hipótesis general*

a. Sistema de hipótesis

H₀: La implementación del Aprendizaje Basado en Problemas no mejora significativamente las competencias de programación en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

H₁: La implementación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación utilizando el lenguaje Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

b. Nivel de significancia

Se estableció un nivel de significancia de $\alpha = 0.05$ con una prueba unilateral, dado que la hipótesis de investigación establece una dirección específica: los puntajes del posttest serán superiores a los del pretest en el grupo experimental.

c. Estadígrafo

Se empleó la Prueba de Rangos con Signo de Wilcoxon para muestras relacionadas, prueba no paramétrica apropiada para comparar dos mediciones realizadas sobre el mismo grupo de participantes cuando los datos no siguen una distribución normal. La elección de esta prueba es consistente con la naturaleza dicotómica del

instrumento de evaluación, cuyos puntajes totales presentan distribuciones asimétricas, tal como fue verificado mediante la prueba de Shapiro-Wilk en el análisis previo.

d. Resultados

Tabla 19

Prueba de Wilcoxon - grupo experimental (pretest vs. posttest)

Variable	W	p	Decisión
PLA	53.00	< 0.001	Se rechaza H_0
HC	15.00	< 0.001	Se rechaza H_0
DP	60.00	< 0.001	Se rechaza H_0
CP	1.00	< 0.001	Se rechaza H_0

Nota. W = estadístico de Wilcoxon (suma de rangos negativos); p = significancia unilateral. $\alpha = 0.05$. n = 45. PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General.

En la Tabla 19 se muestran los resultados de la Prueba de Wilcoxon, los cuales revelan que existen diferencias estadísticamente significativas entre el pretest y el posttest del grupo experimental en las cuatro variables evaluadas. Los resultados del pensamiento lógico-algorítmico $W = 53.00$ y $p < 0.001$, demuestran que los puntajes obtenidos en el posttest fueron significativamente mayores que los puntajes obtenidos en el pretest. En Habilidades de Codificación, se obtuvo $W = 15.00$ y $p < 0.001$, confirmando una mejora altamente significativa, siendo este el valor W más pequeño entre las dimensiones, lo que refleja que prácticamente todos los participantes incrementaron su puntaje después de la intervención. En Depuración y Prueba, $W = 60.00$ y $p < 0.001$ revelan igualmente una mejora significativa. Finalmente, en el Promedio General, $W = 1.00$ y $p < 0.001$ constituyen el resultado más contundente: el valor W cercano a cero indica que la

totalidad de los estudiantes del grupo experimental obtuvieron puntajes más altos en el posttest que en el pretest.

Dado que en todas las variables $p < 0.001 < \alpha = 0.05$, se rechaza la hipótesis nula y se acepta H_1 . Se concluye que la implementación del ABP mejoró significativamente las competencias de programación en Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga durante el semestre académico 2024-II, en todas las dimensiones evaluadas: Pensamiento Lógico-Algorítmico, Habilidades de Codificación, Depuración y Prueba, y el Promedio General.

4.2.6. Prueba de hipótesis específica 1

a. Sistema de hipótesis

H₀: La aplicación del Aprendizaje Basado en Problemas no mejora significativamente las competencias de programación en relación al Pensamiento Lógico-Algorítmico en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

H₁: La aplicación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación en relación al Pensamiento Lógico-Algorítmico en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

b. Nivel de significancia

Se estableció un nivel de significancia de $\alpha = 0.05$ con una prueba unilateral, dado que la hipótesis establece una dirección específica: los puntajes del posttest serán superiores a los del pretest en el grupo experimental.

c. Estadígrafo

Se empleó la Prueba de Rangos con Signo de Wilcoxon para muestras relacionadas, prueba no paramétrica apropiada para comparar dos mediciones sobre el

mismo grupo cuando los datos no siguen una distribución normal. Las variables PLA, HC y DP presentaron distribuciones no normales tanto en el pretest como en el posttest del grupo experimental (Shapiro-Wilk, $p < 0.05$ en todos los casos), lo que justifica el uso de esta prueba.

d. Resultados

Tabla 20

Prueba de Wilcoxon - Pensamiento Lógico-Algorítmico, grupo experimental (pretest vs. posttest)

Variable	W	p	Decisión
PLA	53.00	< 0.001	Se rechaza H_0

Nota. W = estadístico de Wilcoxon (suma de rangos negativos); p = significancia unilateral. $\alpha = 0.05$. n = 45. PLA = Pensamiento Lógico Algorítmico.

Dado que $p < 0.001 < \alpha = 0.05$, se rechaza la hipótesis nula y se acepta H_1 . Se concluye que la aplicación del Aprendizaje Basado en Problemas mejoró significativamente las competencias de programación en relación al Pensamiento Lógico-Algorítmico en los estudiantes de Ingeniería de Sistemas de la UNSCH durante el semestre académico 2024-II.

4.2.7. Prueba de hipótesis específica 2

a. Sistema de hipótesis

H_0 : La aplicación del Aprendizaje Basado en Problemas no mejora significativamente las competencias de programación en relación a las Habilidades de Codificación en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

H₁: La aplicación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación en relación a las Habilidades de Codificación en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

b. Nivel de significancia

Se estableció un nivel de significancia de $\alpha = 0.05$ con una prueba unilateral, dado que la hipótesis establece una dirección específica: los puntajes del postest serán superiores a los del pretest en el grupo experimental.

c. Estadígrafo

Se empleó la Prueba de Rangos con Signo de Wilcoxon para muestras relacionadas, prueba no paramétrica apropiada para comparar dos mediciones sobre el mismo grupo cuando los datos no siguen una distribución normal. Las variables PLA, HC y DP presentaron distribuciones no normales tanto en el pretest como en el postest del grupo experimental (Shapiro-Wilk, $p < 0.05$ en todos los casos), lo que justifica el uso de esta prueba.

d. Resultados

Tabla 21

Prueba de Wilcoxon - Habilidades de Codificación, grupo experimental (pretest vs. posttest)

Variable	W	p	Decisión
HC	15.00	< 0.001	Se rechaza H ₀

Nota. W = estadístico de Wilcoxon (suma de rangos negativos); p = significancia unilateral. $\alpha = 0.05$. n = 45. HC = Habilidades de Codificación.

Dado que $p < 0.001 < \alpha = 0.05$, se rechaza la hipótesis nula y se acepta H_1 . Se concluye que la aplicación del Aprendizaje Basado en Problemas mejoró significativamente las competencias de programación en relación a las Habilidades de Codificación en los estudiantes de Ingeniería de Sistemas de la UNSCH durante el semestre académico 2024-II.

4.2.8. Prueba de hipótesis específica 3

a. Sistema de hipótesis

H_0 : La aplicación del Aprendizaje Basado en Problemas no mejora significativamente las competencias de programación en relación a la Depuración y Pruebas en códigos en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

H_1 : La aplicación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación en relación a la Depuración y Pruebas en códigos en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

b. Nivel de significancia

Se estableció un nivel de significancia de $\alpha = 0.05$ con una prueba unilateral, dado que la hipótesis establece una dirección específica: los puntajes del postest serán superiores a los del pretest en el grupo experimental.

c. Estadígrafo

Se empleó la Prueba de Rangos con Signo de Wilcoxon para muestras relacionadas, prueba no paramétrica apropiada para comparar dos mediciones sobre el mismo grupo cuando los datos no siguen una distribución normal. Las variables PLA, HC y DP presentaron distribuciones no normales tanto en el pretest como en el postest del

grupo experimental (Shapiro-Wilk, $p < 0.05$ en todos los casos), lo que justifica el uso de esta prueba.

d. Resultados

Tabla 22

Prueba de Wilcoxon - Depuración y Prueba, grupo experimental (pretest vs. posttest)

Variable	W	p	Decisión
DP	60.00	< 0.001	Se rechaza H_0

Nota. W = estadístico de Wilcoxon (suma de rangos negativos); p = significancia unilateral. $\alpha = 0.05$. n = 45. DP = Depuración y Prueba.

Dado que $p < 0.001 < \alpha = 0.05$, se rechaza la hipótesis nula y se acepta H_1 . Se concluye que la aplicación del Aprendizaje Basado en Problemas mejoró significativamente las competencias de programación en relación a la Depuración y Pruebas en códigos en los estudiantes de Ingeniería de Sistemas de la UNSCH durante el semestre académico 2024-II.

4.3. Discusión de resultados

Los resultados obtenidos en la presente investigación confirman que la implementación del Aprendizaje Basado en Problemas mejoró significativamente las competencias de programación en Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga durante el semestre académico 2024-II. En efecto, la diferencia de 8.4 puntos en el Promedio General entre el grupo experimental ($\bar{x} = 15.7$) y el grupo control ($\bar{x} = 7.3$), confirmada mediante la prueba U de Mann-Whitney ($p < 0.001$), constituye evidencia estadísticamente sólida del efecto positivo de la intervención pedagógica implementada. Estos resultados se corresponden con los expuestos por García et al. (2021), quienes en un estudio cuasiexperimental con

estudiantes de Ingeniería de Sistemas de una universidad pública peruana, dieron cuenta de que la intervención de ABP en el curso de Base de Datos supuso diferencias significativas en el posttest entre el grupo experimental y el grupo control ($\text{Sig} = 0,000 < 0,05$), evidenciando así que el ABP constituye una estrategia de tipo pedagógica para la formación de competencias técnicas especializadas, en un contexto análogo al de la investigación llevada a cabo.

En lo referente a la dimensión de Pensamiento Lógico-Algorítmico, el grupo en condiciones experimentales presentó una subida de $\bar{x} = 6.31$ del pretest a $\bar{x} = 16.04$ en el posttest, con una mediana de $Md = 18.00$ la que permite ver que la mayoría del alumnado logró puntuaciones cercanas a la puntuación máxima otorgada. Este resultado coincide con los encontrados por Alshaye et al. (2023), quienes observan que las tareas de ABP favorecen el aprendizaje de la capacidad de resolución de problemas pero también el aprendizaje de habilidades de codificación con un efecto positivo para el grupo experimental en el posttest. En consonancia con estos resultados, Lu et al. (2022) remarcaban que el pensamiento algorítmico tiene carácter jerárquico en el pensamiento computacional y que el ABP puede activarlo sistémicamente ya que plantea al estudiante una auténtica situación problema que requiere de análisis, planificación y representación algorítmica antes de pasar el algoritmo a un lenguaje de programación. También, Milková y Hulková (2013) sostienen que el pensamiento algorítmico se favorecerá cuando el alumnado reordene y encadene las ideas y los hechos y los presente en una serie secuencial con un sentido propio, proceso que las sesiones de ABP que hemos ido realizando en la presente propuesta pedagógica han ido potenciando.

Respecto a las Habilidades de Codificación, el grupo experimental obtuvo en el posttest una media de $\bar{x} = 15.60$ y una mediana de $Md = 18.00$, mientras que el grupo

control obtuvo $\bar{x} = 7.60$ y $Md = 6.00$. Por otra parte, la disminución de la desviación estándar en el grupo experimental revela una tendencia hacia la homogeneización del aprendizaje, con la mayoría de los estudiantes agrupándose en los niveles más altos de la escala. Este efecto es consistente con los hallazgos de AlSideiri et al. (2023), quienes reportaron mejoras significativas en el rendimiento académico y en el logro de competencias de los estudiantes que cursaron ingeniería de software bajo el enfoque ABP, en comparación con aquellos que recibieron instrucción tradicional. De igual manera, Rattanakha y Chatwattana (2023) evidenciaron que el modelo de ABP mediado por tecnología en la nube fomenta el pensamiento analítico sistemático y el uso eficaz de herramientas tecnológicas, competencias que la presente propuesta fomentó mediante una secuencia progresiva que incorporó Python, Google Colab, Visual Studio Code y GitHub. Al respecto, Robins et al. (2003) argumentan que las habilidades de codificación trascienden el conocimiento sintáctico del lenguaje para implicar comprensión semántica y capacidad de estructurar código de forma eficiente, precisamente el tipo de competencia que el ABP desarrolla al exigir al estudiante no solo escribir código, sino justificar sus decisiones de diseño, corregir errores y presentar soluciones funcionales ante sus pares.

La dimensión de Depuración y Prueba, por su parte, mostró las diferencias más marcadas en tendencia central entre ambos grupos. El grupo experimental obtuvo $\bar{x} = 15.42$ y $Md = 18.00$, mientras que el grupo control obtuvo $\bar{x} = 7.20$ y $Md = 4.00$, lo que indica que al menos la mitad de los miembros del grupo control no superó los 4 puntos de un total de 20 en esta dimensión. Esta discrepancia evidente señala las limitaciones de la enseñanza expositiva tradicional a la hora de desarrollar una competencia cognitiva avanzada que, según McCauley y cols. (2008), requiere una comprensión profunda del programa, razonamiento lógico y aplicación sistemática de estrategias de diagnóstico. En

contraste, las sesiones ABP orientadas al manejo de errores, la depuración sistemática con Python Debugger y el diseño de pruebas unitarias con pytest proporcionaron al grupo experimental un entorno de práctica deliberada y reflexiva que favoreció la adquisición de esta competencia, resultado confirmado mediante la prueba de Rangos con Signo de Wilcoxon ($W = 60.00$; $p < 0.001$).

Justamente desde la perspectiva teórica, los resultados de la presente investigación encuentran consuelo en los fundamentos psicopedagógicos del ABP. La teoría sociocultural de Vygotsky (1978) ilumina el mecanismo que apoya el efecto opuesto de los resultados de la investigación, en donde el trabajo en los equipos colaborativos con roles técnicos rotativos junto a problemas de complejidad creciente reproduce la dinámica de la Zona de Desarrollo Próximo de tal suerte que los estudiantes llegan a poder (cada vez más) a solucionar problemas que individualmente no hubieran podido superar. De manera complementaria, el ciclo de aprendizaje experiencial de Kolb (1988) muestra su huella en la homogeneidad de los puntajes del grupo experimental en el posttest, mostrando que los estudiantes van transitando a través de manera iterativa las fases de experiencia concreta, observación reflexiva, conceptualización abstracta y experimentación activa en cada sesión ABP, por lo que conducen aprendizajes sólidos y generales frente a las exposiciones. En esta línea, Luy-Montejo (2019) pone el foco en la relevancia contextual de los problemas en el ABP universitario como una de las condiciones necesarias de activación del aprendizaje genuino en situación, la que esta propuesta ha operacionalizado a partir de situaciones relacionadas con la gestión académica junto al modelamiento de sistemas universitarios en Python.

En cuanto a los aportes de la presente investigación al campo educativo, cabe destacar que los hallazgos contribuyen a fortalecer la evidencia empírica sobre la eficacia

del ABP en la enseñanza de la programación en el contexto de la educación superior tecnológica peruana, ámbito en el que la investigación experimental sigue siendo escasa. A diferencia de Santiago (2022), quien demostró mejoras en el pensamiento algorítmico atribuibles al uso del lenguaje Python sin una metodología activa articulada, la presente investigación demuestra que es la combinación de Python con el ABP como marco pedagógico la que genera incrementos sustantivamente superiores en el conjunto de las competencias de programación. Asimismo, la propuesta pedagógica desarrollada, que integra herramientas como GitHub, Google Colab, pytest y Python Debugger en una secuencia de 11 sesiones con el modelo de cinco fases ABP, constituye un aporte metodológico concreto y replicable para docentes de cursos de programación que buscan superar las limitaciones del modelo transmisivo tradicional. En este sentido, la investigación ofrece tanto evidencia cuantitativa de impacto como un diseño didáctico documentado que puede adaptarse a distintas asignaturas del plan curricular de Ingeniería de Sistemas.

No obstante, los hallazgos anteriores, la presente investigación presenta algunas limitaciones que deben considerarse al momento de interpretar y generalizar sus resultados. En primer lugar, el diseño cuasiexperimental, si bien apropiado para el contexto educativo en que fue implementado, no permite la asignación aleatoria de participantes a los grupos, lo que limita el control de variables extrañas y restringe la posibilidad de establecer relaciones de causalidad con la misma solidez que un experimento puro. En segundo lugar, la muestra estuvo circunscrita a dos secciones de una única asignatura en un solo semestre académico de la UNSCH, lo que limita la validez externa de los hallazgos y su generalización a otros cursos, instituciones o niveles educativos. En tercer lugar, el instrumento de evaluación, aunque validado mediante

juicio de expertos y con un coeficiente KR-20 de 0.89, mide competencias de programación en un momento puntual, sin dar cuenta del proceso de aprendizaje ni de la transferencia de las competencias adquiridas a contextos profesionales reales. Finalmente, la duración de la intervención, compuesta por 11 sesiones distribuidas en 13 semanas, podría no ser suficiente para consolidar competencias de mayor complejidad, aspecto que futuras investigaciones deberían abordar mediante diseños longitudinales con seguimiento posterior a la intervención.

V. Propuesta innovadora

5.1. Introducción

La formación en programación en educación superior demanda una transformación de las prácticas pedagógicas tradicionales. En la Escuela Profesional de Ingeniería de Sistemas de la UNSCH, los estudiantes del curso de Algoritmos y Solución de Problemas presentan dificultades persistentes para desarrollar competencias de programación que superen la memorización sintáctica y alcancen niveles de desempeño técnico alineados con las exigencias del mercado laboral en tecnología. Bennedsen y Caspersen, (2019b) documentaron tasas de desaprobación cercanas al 28% en cursos introductorios de programación a nivel global, situación que encuentra su correlato en la realidad local, donde la brecha entre la formación académica y las competencias requeridas por la industria del software se hace cada vez más evidente.

Ante este panorama, la presente propuesta pedagógica experimental propone la implementación del ABP como estrategia metodológica principal para el desarrollo de las tres dimensiones de las competencias de programación: el Pensamiento Lógico-Algorítmico, las Habilidades de Codificación y la Depuración y Prueba. El ABP plantea que el conocimiento no se transmite pasivamente, sino que se construye activamente cuando el estudiante se enfrenta a problemas auténticos, complejos y contextualmente relevantes que movilizan sus recursos cognitivos y colaborativos (Luy-Montejo, 2019b; Sari et al., 2021).

La propuesta se destaca por incorporar herramientas tecnológicas actuales como Python, GitHub, PyCharm y Google Colab como base del proceso de aprendizaje, configurando un entorno pedagógico que prepara a los estudiantes para trabajar en los entornos actuales de desarrollo de software. La intervención se implementó en el semestre

académico 2024-II con 90 estudiantes distribuidos en un grupo experimental ($n = 45$) y un grupo control ($n = 45$), a lo largo de 11 sesiones de aprendizaje.

5.2. Fundamento Filosófico

La propuesta aquí expuesta encuentra su base filosófica en el pragmatismo, corriente cuyo nombre proviene del vocablo griego *pragma*, que alude a la acción y al trabajo que genera la experiencia humana. Dewey (1969), máximo exponente del pragmatismo en educación, afirma que el conocimiento es un instrumento dinámico al servicio de la resolución de problemas, y que aprender es un proceso de investigación orientado hacia la acción. En la enseñanza de la programación, esta concepción implica que la competencia no se evidencia en la reproducción de definiciones formales, sino en la capacidad de construir soluciones tecnológicas efectivas ante situaciones problemáticas reales.

Charles Sanders Peirce, fundador del pragmatismo, agregó un ángulo metodológico complementario: el significado de un concepto radica en las consecuencias prácticas verificables. En programación, esto significa que la comprensión de un algoritmo o de una estructura de datos queda demostrada solamente cuando el estudiante lo aplica para resolver un problema concreto y obtiene resultados que pueden ser comprobados. William James amplió esa visión al admitir que la experiencia individual, incluso el error y su corrección a través de la colaboración, es fuente y testimonio del conocimiento construido.

En consecuencia, la propuesta adopta el pragmatismo como fundamento filosófico al concebir el aula de programación como un espacio de resolución de problemas reales, donde las herramientas tecnológicas son los instrumentos a través de los cuales los

estudiantes construyen conocimiento accionable y se preparan para intervenir de manera competente en el ecosistema tecnológico contemporáneo.

5.3. Fundamento Psicopedagógico

La propuesta integra tres corrientes psicopedagógicas que fundamentan el diseño de la intervención basada en ABP.

a. Teoría Constructivista

Según las vertientes del constructivismo cognitivo (Piaget) y significativo (Ausubel), el aprendizaje auténtico se produce cuando el estudiante vincula la información novedosa con estructuras cognitivas ya existentes. Según Jean Piaget, antes de poder llegar a la abstracción, el alumno debe resolver problemas concretos para desarrollar el pensamiento lógico. En programación, esto significa que los estudiantes tienen que manipular código real, ejecutarlo, y reflexionar acerca de los resultados antes de poder asimilar los conceptos algorítmicos. Para David Ausubel, el aprendizaje significativo es el más sólido y transferible; el ABP da cuerpo a este principio al plantear problemas contextualizados que los estudiantes asumen como relevantes para su formación profesional.

b. Teoría Sociocultural

Vygotsky (1978) demostró que el desarrollo cognitivo es esencialmente un proceso social: las funciones mentales superiores, tales como el razonamiento algorítmico y la habilidad para la depuración lógica, aparecen primero en el ámbito interpsicológico, es decir, entre personas, antes de ser internalizadas en el ámbito intrapsicológico, dentro del individuo. La idea de la Zona de Desarrollo Próximo es muy importante para el diseño de esta propuesta, ya que los estudiantes son capaces de resolver problemas mucho más complejos cuando cuentan con el apoyo pedagógico adecuado, que se logra a través del trabajo colaborativo entre compañeros, la ayuda del profesor y el acceso a herramientas inteligentes y documentación técnica actualizada.

Esta dinámica se reproduce fielmente en los equipos de trabajo del ABP, donde estudiantes con distintos niveles de dominio interactúan, discuten estrategias de solución y co-construyen código, reproduciendo así el mecanismo vygotskiano de aprendizaje en la ZDP. Esta teoría da cuenta de la programación colaborativa y la revisión de código entre pares, que son prácticas estándar en la industria del software.

c. Teoría del Aprendizaje Experiencial

Kolb (1984) planteó un ciclo del aprendizaje experiencial con cuatro etapas: experiencia concreta, observación reflexiva, conceptualización abstracta y experimentación activa. Este ciclo es el que ocurre de forma natural en el proceso ABP de programación: los estudiantes juegan con código real, analizan los errores que cometen, generalizan principios algorítmicos a partir de múltiples experiencias y los aplican a nuevos problemas. Jupyter Notebook, que permite documentar el proceso de solución junto con el código ejecutable, facilita esta espiral de aprendizaje y la metacognición.

5.4. Fundamento Pedagógico

El sustento pedagógico de la propuesta gira en torno a la pedagogía activa y al enfoque por competencias, que plantea que la formación universitaria debe asegurar el desarrollo de habilidades que el estudiante pueda transferir a contextos profesionales reales (Perrenoud, 2004; Tobón, 2013). La investigación en enseñanza de la programación ha documentado de forma consistente las limitaciones del modelo transmisivo tradicional para producir programadores competentes (McCauley et al., 2008; Robins et al., 2003). Los estudiantes formados bajo este paradigma suelen dominar la sintaxis del lenguaje, pero carecen de la capacidad de análisis y síntesis necesaria para abordar problemas de programación no rutinarios.

El ABP supera estas limitaciones estructurando el aprendizaje alrededor de problemas reales que no tienen una única solución evidente. Rattanakha y Chatwattana (2023) afirmaron que el modelo ABP mediado por tecnología ha sido eficaz en el desarrollo de habilidades de programación en estudiantes universitarios, fomentando el pensamiento analítico y la resolución de problemas de forma colaborativa. AlSideiri et al. (2023) confirmaron que la aplicación del Aprendizaje Basado en Problemas en ingeniería de software genera mejoras significativas en el rendimiento académico y en el desarrollo de habilidades como trabajo en equipo y pensamiento crítico. Desde el punto de vista pedagógico, la propuesta se apoya en el modelo de cinco etapas de Sari et al. (2021): orientación al problema, organización del aprendizaje, investigación individual y grupal, desarrollo y presentación del producto, y análisis y evaluación del proceso. Este modelo se ha adaptado al contexto de la programación en Python, incorporando en cada fase herramientas tecnológicas digitales que potencian la capacidad de los estudiantes para investigar, experimentar, colaborar y documentar sus procesos de solución.

5.5. Enfoque Tecnológico

La propuesta incorpora un conjunto de herramientas tecnológicas que reproducen las condiciones de trabajo de la industria del software actual. Se escogió como lenguaje de programación principal a Python, por su posicionamiento en el índice TIOBE (2025) como el lenguaje de mayor popularidad global, con un 26.98% de participación, impulsado por su protagonismo en inteligencia artificial y análisis de datos.

Google Colab proporciona acceso gratuito a entornos de ejecución en la nube, removiendo las barreras de infraestructura que históricamente han limitado la experimentación tecnológica en contextos universitarios con recursos limitados como la UNSCH. Como plataforma de control de versiones y colaboración distribuida, GitHub

permite introducir a los estudiantes en las prácticas estándar de la industria del software. La incorporación de estas herramientas obedece a una lógica de andamiaje tecnológico, cada herramienta cumple una función específica en el proceso de construcción de competencias y se va introduciendo progresivamente con el soporte didáctico necesario.

Tabla 23

Herramientas tecnológicos de la propuesta experimental

Herramienta Plataforma	/ Función pedagógica en el ABP	Dimensión
Python 3.x	Es el lenguaje principal para implementar soluciones algorítmicas y permite un rápido prototipado, así como acceder a bibliotecas de análisis de datos e inteligencia artificial.	PLA, HC, DP
Google Colab	Entorno de ejecución en la nube que permite la colaboración sin instalación local; soporta cuadernos interactivos y ejecución compartida.	HC
Visual Studio Code	Editor de código con extensiones para Python, depuración interactiva y soporte para control de versiones integrado.	HC, DP
PyCharm Education	IDE especializado para la enseñanza de programación; ofrece	HC, DP

Herramienta Plataforma	/ Función pedagógica en el ABP	Dimensión
	retroalimentación inteligente, detección de errores en tiempo real y ejercicios integrados con corrección automática.	
GitHub / Git	Gestión de repositorios colaborativos; seguimiento de cambios y revisión por pares del código.	HC, DP
Jupyter Notebook	Entorno interactivo que integra código, resultados y narrativa académica; favorece la documentación del proceso de solución.	PLA, HC

Nota. PLA = Pensamiento Lógico-Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba.

Incorporar estas herramientas no responde a una lógica de modernización superficial, sino a una pedagogía del andamiaje tecnológico, donde cada herramienta cumple una función específica en el proceso de construcción de competencias y se va introduciendo de forma progresiva, con el soporte didáctico necesario para que su uso potencie el aprendizaje y no lo obstaculice.

5.6. Objetivos de la Propuesta

5.6.1. *Objetivo General*

Implementar la metodología del ABP integrada con herramientas tecnológicas contemporáneas para mejorar las competencias de programación en Python, específicamente el PLA, las HC y la DP, en los estudiantes de la Escuela Profesional de Ingeniería de Sistemas de la UNSCH, semestre académico 2024-II.

5.6.2. *Objetivos Específicos*

Tabla 24

Objetivos específicos, dimensiones e indicadores de logro

Objetivo específico	Dimensión	Meta
Aplicar la metodología ABP para desarrollar el PLA mediante la resolución de problemas computacionales contextualizados en Python.	Pensamiento Lógico-Algorítmico	Diseña algoritmos eficientes y los representa mediante pseudocódigo y diagramas de flujo.
Implementar estrategias ABP que fortalezcan las HC en Python, integrando buenas prácticas de desarrollo de software profesional.	Habilidades de Codificación	Codifica soluciones funcionales en Python usando estructuras de control, funciones y POO.
Desarrollar la competencia de DP a través de la resolución colaborativa de problemas reales que demanden validación y corrección de código.	Depuración y Prueba	Identifica y corrige errores; diseña y ejecuta pruebas unitarias con pytest bajo el enfoque TDD.
Fomentar el uso de herramientas tecnológicas contemporáneas (GitHub, Google Colab, PyCharm, VS Code, pytest) como soporte del aprendizaje basado en problemas.	Competencias digitales transversales	Utiliza repositorios de control de versiones, IDEs y frameworks de pruebas con fluidez operativa.

Nota. PLA = Pensamiento Lógico-Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba. TDD = Test-Driven Development..

5.7. Descripción de la propuesta

La propuesta experimental denominada "ABP para el Desarrollo de Competencias de Programación en Python" está dirigida a los estudiantes de la serie 100 de la Escuela Profesional de Ingeniería de Sistemas de la UNSCH, cursantes de la asignatura de Algoritmos y Solución de Problemas en el semestre académico 2024-II. La intervención se estructuró en 11 sesiones de aprendizaje distribuidas a lo largo de 15 semanas académicas, desde el diagnóstico inicial hasta la evaluación final. Los materiales empleados incluyen guías de problemas ABP.

Cada sesión de aprendizaje sigue el modelo didáctico de cinco fases ABP adaptado al contexto de la programación en Python. El docente actúa como diseñador de problemas y facilitador del proceso. Los estudiantes trabajan en equipos colaborativos de 4 a 5 integrantes con roles rotativos, incluyendo líder de equipo, desarrollador, tester y documentador, lo que garantiza que todos desarrollen competencias en las tres dimensiones evaluadas. El grupo control recibió durante el mismo periodo la metodología expositiva tradicional, con el mismo docente titular.

5.7.1. Modelo de cinco fases ABP aplicado a la programación

Tabla 25

Modelo de cinco fases ABP adaptado a la enseñanza de programación en Python

Fase ABP	Descripción operativa	Competencia desarrollada
Fase 1: Presentación del problema	El docente presenta un escenario tecnológico real o simulado vinculado al contexto universitario. Los estudiantes identifican el problema y formulan preguntas guía.	PLA: análisis y descomposición del problema.
Fase 2: Organización del aprendizaje	Los equipos discuten conocimientos previos, identifican brechas de aprendizaje y elaboran un plan de investigación con roles técnicos asignados.	PLA: diseño de estrategias de solución.
Fase 3: Investigación y construcción	Los estudiantes investigan de forma autónoma y colaborativa, consultando documentación oficial de Python y recursos especializados disponibles.	HC: implementación de soluciones en Python.
Fase 4: Desarrollo y presentación	Los equipos codifican la solución en Python, identifican errores, realizan pruebas de validación y presentan el producto funcional ante sus pares.	DP: identificación de errores, validación y corrección.

Fase ABP	Descripción operativa	Competencia desarrollada
Fase 5: Evaluación y metacognición	El grupo evalúa la solución, reflexiona sobre el proceso, recibe retroalimentación del docente y documenta las lecciones aprendidas.	Integración PLA + HC + DP: evaluación integral del proceso de aprendizaje.

Nota. Fases propuestas de aprendizaje.

5.7.2. *Secuencia de sesiones de aprendizaje*

La secuencia de 11 sesiones fue diseñada para asegurar una progresión lógica de complejidad creciente, articulando el desarrollo del PLA en las primeras sesiones con las HC en las sesiones intermedias y la DP en las sesiones avanzadas, culminando en un proyecto integrador que moviliza simultáneamente las tres dimensiones de competencia. El período de implementación abarcó de agosto a noviembre del año 2024.

Tabla 26

Secuencia de actividades y sesiones de la propuesta experimental

Periodo	Contenido y problema ABP	Duración	Estrategia y herramientas
Semana 1	Aplicación de la prueba piloto y Pretest al grupo de control. Evaluación inicial de las tres dimensiones: PLA, HC y DP (30 items).	120 min	Prueba individual pedagógica.
Semana 2	Aplicación del Pretest al grupo experimental. Orientación al ABP, presentación de la metodología, formación de equipos colaborativos	120 min	Prueba pedagógica individual-Grupal (Google Colab. Visual Studio Code).

Periodo	Contenido y problema ABP	Duración	Estrategia y herramientas
	e instalación y configuración del entorno de desarrollo Python.		
Semana 3	Sesión 1: Introducción a Python y pensamiento algorítmico. Problema ABP: diseño de un algoritmo para clasificar registros de notas de estudiantes mediante diagramas de flujo y pseudocódigo.	90 min	ABP Fases 1-2. Google Colab. Diagramación digital. Trabajo en equipo.
Semana 4	Sesión 2: Tipos de datos, variables y estructuras condicionales. Problema ABP: sistema de evaluación automática del rendimiento académico según rangos de calificación.	90 min	ABP Fases 2-3. Google Colab. Jupyter Notebook. Trabajo en equipo.
Semana 5	Sesión 3: Estructuras de control: bucles. Problema ABP: calculo iterativo de promedios y detección de estudiantes en riesgo académico mediante ciclos repetitivos.	90 min	ABP Fases 3-4. Google Colab. Visual Studio Code. Trabajo en equipo.
Semana 6	Sesión 4: Estructuras de datos: listas, tuplas, diccionarios y conjuntos. Problema ABP: sistema de gestión y consulta del registro de asistencia estudiantil usando colecciones de datos.	90 min	ABP Fase 4. Visual Studio Code. GitHub colaborativo. Trabajo en equipo.

Periodo	Contenido y problema ABP	Duración	Estrategia y herramientas
Semana 7	Sesión 5: Funciones en Python. Problema ABP: desarrollo de un módulo de cálculo de promedios ponderados y generación de reportes académicos con funciones reutilizables.	90 min	ABP Fases 3-4. Visual Studio Code. Programación en pares. Trabajo en equipo.
Semana 8	Sesión 6: Programación Orientada a Objetos I: clases y objetos. Problema ABP: modelamiento de un sistema de gestión de estudiantes universitarios con atributos y métodos propios.	90 min	ABP Fases 3-4. Visual Studio Code. GitHub. Trabajo en equipo.
Semana 9	Sesión 7: Programación Orientada a Objetos II: herencia y polimorfismo. Problema ABP: extensión del sistema de estudiantes para modelar diferentes tipos de usuarios de un campus universitario.	90 min	ABP Fase 4. Visual Studio Code. GitHub. Revisión de código entre pares. Trabajo en equipo.
Semana 10	Sesión 8: Manejo de errores y excepciones en Python. Problema ABP: diagnóstico y corrección de fallos en un sistema de registro de calificaciones que genera datos inválidos.	90 min	ABP Fases 2-3. Python Debugger. Análisis de trazas de error. Trabajo en equipo.

Periodo	Contenido y problema ABP	Duración	Estrategia y herramientas
Semana 11	Sesión 9: Depuración sistemática de código. Problema ABP: identificación y corrección de errores lógicos en un algoritmo de búsqueda de estudiantes con bajo rendimiento académico.	90 min	ABP Fases 3-4. Python Debugger. pylint. Trabajo en equipo.
Semana 12	Sesión 10: Pruebas de software en Python. Problema ABP: diseño y ejecución de casos de prueba para validar un módulo de cálculo de créditos académicos.	90 min	ABP Fase 4. pytest. Validación de resultados. Trabajo en equipo.
Semana 13	Sesión 11: Proyecto integrador. Problema ABP real: construcción de un sistema de consulta del historial académico de estudiantes en Python con validación de entradas y manejo de errores.	90 min	ABP Fases 1-5. GitHub colaborativo. Presentación técnica con rubrica. Trabajo en equipo.
Semana 14	Aplicación del Posttest al grupo de control. Medición final de las competencias de programación en las tres dimensiones: Pensamiento Lógico-Algorítmico, Habilidades de Codificación y Depuración y Prueba.	120 min	Prueba pedagógica individual.
Semana 15	Aplicación del posttest al grupo experimental. Medición final de competencias y reflexión	120 min	Prueba pedagógica individual-grupal.

Periodo	Contenido y problema ABP	Duración	Estrategia y herramientas
	metacognitiva sobre los aprendizajes construidos durante la intervención.		Cierre del portafolio digital.

Nota. Secuencia de sesiones.

5.7.3. Estrategias de evaluación

La evaluación del proceso se realizó de manera continua y formativa a lo largo de la intervención, complementando la evaluación sumativa del pretest y posttest. Los instrumentos formativos incluyen revisiones de código entre pares mediadas por GitHub, presentaciones técnicas evaluadas con rubricas de desempeño en la Fase 5 del ABP y reflexiones metacognitivas al cierre de cada sesión. La evaluación del impacto de la intervención se realizó mediante la aplicación del instrumento de prueba de pedagógica de desarrollo de 30 preguntas antes y después de la intervención experimental.

5.8. Impacto esperado y sostenibilidad

Se espera que la implementación de la propuesta produzca un efecto positivo en el proceso de aprendizaje mediante de solución de problemas en un curso de programación.

Asimismo, del impacto sobre el rendimiento académico de los estudiantes, la propuesta busca generar un cambio en la cultura de aprendizaje de la programación en la UNSCH, de una enseñanza centrada en la transmisión de sintaxis hacia una formación orientada a la construcción activa de soluciones tecnológicas. Los estudiantes del grupo experimental no solo han sido expuestos a una metodología pedagógica más eficaz, sino también a herramientas y prácticas profesionales como el control de versiones, la revisión

de código y el desarrollo colaborativo, que constituyen el estándar de la industria del software contemporánea.

Con relación a la parte de la sostenibilidad, la propuesta está hecha para que pueda ser replicada dentro de la Escuela Profesional de Ingeniería de Sistemas. Los materiales elaborados, guías de problemas ABP, rúbricas y repositorios de código constituyen un patrimonio pedagógico reutilizable que puede ser adaptado a asignaturas de mayor complejidad del plan curricular. Al adoptarse herramientas de libre acceso se eliminan barreras económicas para su implementación institucional y al estar alineadas con el perfil de egreso de la carrera garantizan su pertinencia curricular en el largo plazo.

VI. Conclusiones

1. La implementación del Aprendizaje Basado en Problemas mejoró significativamente las competencias de programación en Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga durante el semestre académico 2024-II, respondiendo al objetivo general de la investigación. El grupo experimental alcanzó una media de $\bar{x} = 15.7$ frente a $\bar{x} = 7.3$ del grupo control en el Promedio General, con una diferencia de 8.4 puntos estadísticamente significativa (U de Mann-Whitney, $p < 0.001$), confirmada en las tres dimensiones evaluadas: Pensamiento Lógico-Algorítmico, Habilidades de Codificación y Depuración y Prueba ($p < 0.001$ en todos los casos), lo que permite concluir que el ABP, aplicado mediante problemas auténticos e integración de herramientas tecnológicas contemporáneas, constituye una estrategia pedagógica eficaz para el desarrollo integral de las competencias de programación en la educación superior.
2. La aplicación del Aprendizaje Basado en Problemas mejoró significativamente el Pensamiento Lógico-Algorítmico en los estudiantes de Ingeniería de Sistemas de la UNSCH durante el semestre académico 2024-II, respondiendo al primer objetivo específico de la investigación. El grupo experimental alcanzó una media de $\bar{x} = 16.04$ ($Md = 18.00$) frente a $\bar{x} = 6.98$ ($Md = 6.00$) del grupo control en el posttest, con una diferencia estadísticamente significativa (U de Mann-Whitney, $p < 0.001$), lo que permite concluir que la resolución de problemas auténticos que demandaron análisis, descomposición y representación algorítmica progresiva favoreció el desarrollo del razonamiento lógico-secuencial en los estudiantes del grupo experimental.
3. La aplicación del Aprendizaje Basado en Problemas mejoró significativamente las Habilidades de Codificación en los estudiantes de Ingeniería de Sistemas de la

UNSCH durante el semestre académico 2024-II, respondiendo al segundo objetivo específico de la investigación. El grupo experimental alcanzó una media de $\bar{x} = 15.60$ ($Md = 18.00$) frente a $\bar{x} = 7.60$ ($Md = 6.00$) del grupo control en el posttest, con una diferencia estadísticamente significativa (U de Mann-Whitney, $p < 0.001$), lo que permite concluir que la construcción colaborativa de soluciones en Python, integrada con herramientas como Google Colab, Visual Studio Code y GitHub, promovió el desarrollo de habilidades de codificación funcionales y transferibles más allá del conocimiento sintáctico del lenguaje.

4. La aplicación del Aprendizaje Basado en Problemas mejoró significativamente la competencia de Depuración y Prueba en los estudiantes de Ingeniería de Sistemas de la UNSCH durante el semestre académico 2024-II, respondiendo al tercer objetivo específico de la investigación. Los alumnos del grupo experimental obtuvieron una media de $\bar{x} = 15.42$ ($Md = 18.00$) en comparación con $\bar{x} = 7.20$ ($Md = 4.00$) del grupo control en la prueba posterior, una diferencia estadísticamente significativa (Wilcoxon, $W = 60.00$; $p < 0.001$), lo cual permite concluir que las sesiones orientadas al manejo de errores, la depuración sistemática con Python Debugger y el diseño de pruebas unitarias con pytest proporcionaron al grupo experimental un entorno de práctica deliberada que la enseñanza expositiva tradicional no consigue desarrollar.
5. En términos metodológicos, el diseño cuasiexperimental con pretest y posttest empleado en la presente investigación demostró ser adecuado para evaluar el efecto del Aprendizaje Basado en Problemas en un contexto educativo donde la asignación aleatoria de participantes no es viable, asegurando la validez interna de los resultados mediante la verificación de la equivalencia inicial entre grupos (U de Mann-Whitney,

$p = 0.599$ en el pretest). La aplicación de la prueba de Shapiro-Wilk confirmó la no normalidad de los datos ($p < 0.05$) en la mayoría de las variables, justificando el uso consistente de pruebas no paramétricas U de Mann-Whitney para muestras independientes y Rangos con Signo de Wilcoxon para muestras relacionadas como estadígrafos de contraste, lo que permitió obtener conclusiones estadísticamente sólidas y metodológicamente coherentes con la naturaleza de los datos recolectados mediante el instrumento de 30 ítems validado por juicio de expertos y con un coeficiente KR-20 de 0.89.

Recomendaciones

Se recomienda a la Escuela Profesional de Ingeniería de Sistemas de la UNSCH promover la incorporación progresiva del ABP en los cursos de programación y otras asignaturas afines, debido a que esta metodología favorece significativamente el desarrollo de competencias profesionales, fortalece el aprendizaje activo y contribuye a una mejor comprensión de los contenidos mediante la resolución de situaciones reales y contextualizadas.

Se recomienda a los docentes implementar estrategias didácticas orientadas a la resolución de problemas reales utilizando herramientas tecnológicas como Python, Google Colab, GitHub y entornos de desarrollo integrados, con el propósito de fortalecer el pensamiento lógico-algorítmico, las habilidades de codificación y las capacidades de depuración y prueba. Asimismo, resulta importante fomentar el trabajo colaborativo y el aprendizaje autónomo como elementos esenciales para la formación de futuros profesionales en ingeniería.

Se recomienda a futuros investigadores desarrollar estudios similares en otros cursos, programas académicos o instituciones de educación superior, considerando muestras más amplias y periodos de intervención más prolongados. Ello permitirá contrastar los resultados obtenidos, ampliar la evidencia científica sobre la efectividad del Aprendizaje Basado en Problemas y contribuir al fortalecimiento de las metodologías activas en la enseñanza de la programación.

Referencias bibliográficas

- Aguilar, E. E. (2014). *El aprendizaje basado en problemas y el rendimiento académico de los estudiantes del C.E.P. Cibernet, Huamanga—Ayacucho*. 2012 [Universidad Nacional de San Cristóbal de Huamanga]. <http://repositorio.unsch.edu.pe/handle/UNSCH/778>
- Ahirrao, M., Borse, S., Deshmukh, A., & Mishra, V. (2024). Programming Skills Recommendation System with Generative AI. *International Journal For Multidisciplinary Research*, 6(2), 14528. <https://doi.org/10.36948/ijfmr.2024.v06i02.14528>
- Alshaye, I. A., Tasir, Z., & Jumaat, N. F. (2023). The Effectiveness of Online Problem-Based Learning Tasks on Riyadh's Secondary School Students' Problem-Solving Ability and Programming Skills. *Open Education Studies*, 5(1), 20220208. <https://doi.org/10.1515/edu-2022-0208>
- AlSideiri, A., Tawafak, R. M., Malik, S. I., Alfarsi, G., Khudayer, B. H., & Mathew, R. (2023). Examining the Impact of Software Engineering Problem-Based Learning on Student Performance. *2023 24th International Arab Conference on Information Technology (ACIT)*, 1-6. <https://doi.org/10.1109/ACIT58888.2023.10453857>
- Amanda, F. F., Sumitro, S. B., Lestari, S. R., & Ibrohim, I. (2024). Enhancing Critical Thinking And Problem Solving Skills By Complexity Science-Problem Based Learning Model. *REMIE: Multidisciplinary Journal of Educational Research*, 14(1), 96-114. <https://dialnet.unirioja.es/servlet/articulo?codigo=9315025>
- Araya Cabrera, M. P., & Martínez Clares, P. (2023). El Aprendizaje Basado en Problemas como estrategia de seguimiento de las competencias del perfil de egreso. Una experiencia en Ingeniería Civil en la Universidad de Valparaíso. *Educatio Siglo XXI*, 41(2), 11-38. <https://doi.org/10.6018/educatio.503551>
- Arias, F. G. (2012). *El Proyecto de Investigación. Introducción a la Metodología Científica*. 6ta. Edición. Fidas G. Arias Odón.
- Becker, B. A., & Quille, K. (2019). 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. *Proceedings of the 50th*

ACM Technical Symposium on Computer Science Education, 338-344.
<https://doi.org/10.1145/3287324.3287432>

Bennedsen, J., & Caspersen, M. E. (2019a). Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10(2), 30-36. <https://doi.org/10.1145/3324888>

Bennedsen, J., & Caspersen, M. E. (2019b). Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10(2), 30-36. <https://doi.org/10.1145/3324888>

Bernal, C. A. B. (2022). *Metodología de la investigación. Administración, Economía, Humanidades y Ciencias Sociales*.

Boud, D., & Falchikov, N. (2006). Aligning assessment with long-term learning. *Assessment & Evaluation in Higher Education*, 31(4), 399-413.
<https://doi.org/10.1080/02602930600679050>

Branda, L. A. (2009). El aprendizaje basado en problemas en la formación en ciencias de la salud. En *El aprendizaje basado en problemas: Una herramienta para toda la vida* (pp. 15-28). Agencia Láin Entralgo. (Obra original publicada en Bilbao, J.)

Bustamante Vega, R. M., & Mejía Mejía, E. J. (2023). El ABP como estrategia para mejorar la comprensión lectora en educación superior. *IGOVERNANZA*, 6(23), 86-114.
<https://doi.org/10.47865/igob.vol6.n23.2023.290>

Campbell, D. T., & Stanley, J. C. (1966). *Experimental and quasi-experimental designs for research*. Wadsworth.

Carrasco, S. (2008). *Metodología de la investigación científica: Pautas metodológicas para diseñar y elaborar el proyecto de investigación* (2a ed). San Marcos.

Carrasco, S. (2015). *Metodología de la investigación científica: Pautas metodológicas para diseñar y elaborar el proyecto de investigación* (Segunda edición, novena reimpresión 2015). San Marcos.

Chueh, H.-E., & Kao, C.-Y. (2024). Exploring the impact of integrating problem based learning and agile in the classroom on enhancing professional competence. *Heliyon*, 10(3), e24887. <https://doi.org/10.1016/j.heliyon.2024.e24887>

Córdova, M. (2003). *Estadística descriptiva e inferencial: Aplicaciones* (5a. ed). Moshera.

Corral, Y. (2022). Validez y confiabilidad en instrumentos de investigación: Una mirada teórica. *Revista ciencias de la educación*, (60), 562-586. <https://dialnet.unirioja.es/servlet/articulo?codigo=9710359>

Dewey, J. (1969). *EXPERIENCE AND EDUCATION: THE KAPPA DELTA PI LECTURE SERIES*.

Díaz Barriga, F. (2002). *Estrategias Docentes Para Un Aprendizaje Significativo - 2b: Edición*.

Field, A. (2018). *Discovering Statistics Using IBM SPSS Statistics*. SAGE Publications.

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 111(23), 8410-8415. <https://doi.org/10.1073/pnas.1319030111>

Freire, P. (1970). *Pedagogía del oprimido* (Tra). Siglo Veintiuno Editores.

Galicia, L. A., Balderrama, J. A., & Edel, R. (2017). Content validity by experts judgment: Proposal for a virtual tool. *Apertura*, 9(2), 42-53. <https://doi.org/10.32870/Ap.v9n2.993>

García, B. L., Oscanoa, R. B., & Rivera, F. (2021). *Influencia del aprendizaje basado en problemas en las competencias adquiridas en el curso de base de datos: Caso Universidad Peruana*. <https://repositorio.uide.edu.ec/handle/37000/4760>

García Ramos, J. M. (1989). *Bases pedagógicas de la evaluación: (Guía práctica para educadores)*. Síntesis. <https://dialnet.unirioja.es/servlet/libro?codigo=89624>

Garrison, D. R., & Vaughan, N. D. (2008). *Blended learning in higher education: Framework, principles, and guidelines* (1st ed). Jossey-Bass. <https://doi.org/10.1002/9781118269558>

George, D., & Mallery, P. (2003). *SPSS for Windows step by step: A simple guide and reference, 11.0 update* (4th ed). Allyn and Bacon.

- Guerrero-Rodríguez, O. A., Herrera-Navas, O. D., Guzmán-Hernández, R., & Baque-Arteaga, M. E. (2025). Metodología B-Learning en el desarrollo de competencias en Programación y Bases de Datos en estudiantes de primero bachillerato informática. *MQRInvestigar*, 9(1), e265. <https://doi.org/10.56048/MQR20225.9.1.2025.e265>
- Guillemin, M., & Gillam, L. (2004). Ethics, Reflexivity, and “Ethically Important Moments” in Research. *Qualitative Inquiry*, 10(2), 261-280. <https://doi.org/10.1177/1077800403262360>
- Hasbi, A. R. R., Arafah, K., & Khaeruddin. (2024). The Effect of Problem Based Learning Models and Interest to Learn Physics on Students Critical Thinking Skills of Class XI Students at SMA Negeri 1 Selayar. *International Journal of Social Science and Human Research*, 07(02). <https://doi.org/10.47191/ijsshr/v7-i02-74>
- Hernández-Sampieri, R., & Mendoza, C. P. (2018). *Metodología de la investigación: Las rutas cuantitativa, cualitativa y mixta* (First edition). McGraw-Hill Education.
- Hidajat, F. A. (2023). A comparison between problem-based conventional learning and creative problem-based learning on self-regulation skills: Experimental study. *Heliyon*, 9(9), e19512. <https://doi.org/10.1016/j.heliyon.2023.e19512>
- Hmelo-Silver, C. E. (2004). Problem-Based Learning: What and How Do Students Learn? *Educational Psychology Review*, 16(3), 235-266. <https://doi.org/10.1023/B:EDPR.0000034022.16470.f3>
- Holenko Dlab, M., & Hoic-Bozic, N. (2021). Effectiveness of game development-based learning for acquiring programming skills in lower secondary education in Croatia. *Education and Information Technologies*, 26(4), 4433-4456. <https://doi.org/10.1007/s10639-021-10471-w>
- INEI. (2023). *Perú: Estadísticas de las Tecnologías de Información y Comunicación en las Empresas*. https://www.inei.gob.pe/media/MenuRecursivo/publicaciones_digitales/Est/Lib1539/libro.pdf
- Kolb, D. A. (1984). *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall.

- Kuder, G. F., & Richardson, M. W. (1937). The Theory of the Estimation of Test Reliability. *Psychometrika*, 2(3), 151-160. <https://doi.org/10.1007/BF02288391>
- Lema, V. M., Mbondo, M., & Kamau, E. N. (2010). Informed consent for clinical trials: A review. *East African Medical Journal*, 86(3), 133-142. <https://doi.org/10.4314/eamj.v86i3.54968>
- Loyens, S. M. M., Kirschner, P. A., & Paas, F. (2012). Problem-based learning. En K. R. Harris, S. Graham, T. Urdan, A. G. Bus, S. Major, & H. L. Swanson (Eds.), *APA educational psychology handbook, Vol 3: Application to learning and teaching*. (pp. 403-425). American Psychological Association. <https://doi.org/10.1037/13275-016>
- Lu, C., Macdonald, R., Odell, B., Kokhan, V., Demmans Epp, C., & Cutumisu, M. (2022). A scoping review of computational thinking assessments in higher education. *Journal of Computing in Higher Education*, 34(2), 416-461. <https://doi.org/10.1007/s12528-021-09305-y>
- Luy-Montejo, C. (2019a). El Aprendizaje Basado en Problemas (ABP) en el desarrollo de la inteligencia emocional de estudiantes universitarios. *Propósitos y Representaciones*, 7(2), 353-383. <https://doi.org/10.20511/pyr2019.v7n2.288>
- Luy-Montejo, C. (2019b). El Aprendizaje Basado en Problemas (ABP) en el desarrollo de la inteligencia emocional de estudiantes universitarios. *Propósitos y Representaciones*, 7(2), 353-383. <https://doi.org/10.20511/pyr2019.v7n2.288>
- McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education*, 18(2), 67-92. <https://doi.org/10.1080/08993400802114581>
- Milková, E., & Hulková, A. (2013). Algorithmic and Logical Thinking Development: Base of Programming Skills. *WSEAS Transactions on Computers*, 12. <https://www.wseas.com/journals/articles.php?id=5736>
- Morin, E. (1999). *Los Siete saberes necesarios para la educación del futuro*. <https://repositorio.minedu.gob.pe/handle/20.500.12799/1448>

O'Grady, G., Yew, E. H. J., Goh, K. P. L., & Schmidt, H. G. (Eds.). (2012). *One-Day, One-Problem: An Approach to Problem-based Learning*. Springer Singapore. <https://doi.org/10.1007/978-981-4021-75-3>

Osmanović Zajić, J., & Maksimović, J. (2022). Quasi-Experimental Research as An Epistemological-Methodological Approach in Education Research. *International Journal of Cognitive Research in Science, Engineering and Education (IJCRSEE)*, 10(3), 177-183. <https://doi.org/10.23947/2334-8496-2022-10-3-177-183>

Palomino, J. T., & Osorio, V. G. (2023). El aprendizaje basado en problemas para el logro de competencias en educación superior. *Dilemas contemporáneos: Educación, Política y Valores*. <https://doi.org/10.46377/dilemas.v2i10.3484>

Peirce, C. (2016). How to Make our Ideas Clear, by Charles S. Peirce. *Popular Science Monthly* 12 (January 1878), 286-302. *Revista Filosofía UIS*, 15(2), 287-303. <https://dialnet.unirioja.es/servlet/articulo?codigo=8028952>

Perrenoud, P. (2004). *Diez nuevas competencias para enseñar: Invitación al viaje*. Graó. <https://dialnet.unirioja.es/servlet/libro?codigo=249036>

Rattanakha, R., & Chatwattana, P. (2023). The Problem-based Learning Model: PBL Model via Cloud Technology to Promote Programming Skills. *Higher Education Studies*, 13(4), 201. <https://doi.org/10.5539/hes.v13n4p201>

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2), 137-172. <https://doi.org/10.1076/csed.13.2.137.14200>

Santiago, V. H. (2022). *Python y la mejora del pensamiento algorítmico en los alumnos de la escuela profesional de Ingeniería de Sistemas, 2022* [Universidad Nacional Helario Valdizan]. <https://repositorio.unheval.edu.pe/item/42bf876f-25af-4152-a98a-72d649e7f54a>

Sari, Y. I., Sumarmi, Utomo, D. H., & Astina, I. K. (2021). The Effect of Problem Based Learning on Problem Solving and Scientific Writing Skills. *International Journal of Instruction*, 14(2), 11-26. <https://eric.ed.gov/?id=EJ1290959>

- Siegel, S., & Castellan, N. J. (1998). *Estadística no paramétrica aplicada a las ciencias de la conducta* (1a.ed). Trillas.
- Skjong, R., & Wentworth, B. H. (2001). *Expert Judgment and Risk Perception*. 537-544.
- TIOBE Software. (s. f.). TIOBE index for July 2025. *TIOBE*. Recuperado 19 de agosto de 2025, de <https://www.tiobe.com/tiobe-index/>
- Tobón, S. T. (2013). *Formación integral y competencias*. Ecoe Ediciones.
- Vinueza-Morales, M., Cárdenas-Cobo, J., Robles-Salguero, R., & Vidal-Silva, C. (2025). Impacto del lenguaje de programación ALICE en el desarrollo de competencias de programación de estudiantes universitarios. *Formación Universitaria*, 18(3), 167-178. <https://doi.org/10.4067/s0718-50062025000300167>
- Vinueza-Morales, M., Rodas-Silva, J., & Vidal-Silva, C. (2025a). Teaching Programming in Higher Education: Analyzing Trends, Technologies, and Pedagogical Approaches Through a Bibliometric Lens. *International Journal of Advanced Computer Science and Applications*, 16(1). <https://doi.org/10.14569/IJACSA.2025.0160106>
- Vinueza-Morales, M., Rodas-Silva, J., & Vidal-Silva, C. (2025b). Teaching Programming in Higher Education: Analyzing Trends, Technologies, and Pedagogical Approaches Through a Bibliometric Lens. *International Journal of Advanced Computer Science and Applications*, 16(1). <https://doi.org/10.14569/IJACSA.2025.0160106>
- Vygotsky, L. S. (1978). *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press.
- Xue, G., Chen, S., Xie, J., Yang, M., & Wang, Z. (2025). Application of problem-based learning combined with three-dimensional visualization reconstruction technology in trauma orthopedics teaching and its impact on teaching satisfaction. *BMC Medical Education*, 26(1), 53. <https://doi.org/10.1186/s12909-025-08364-4>

Anexos

Anexo 1: Matriz de coherencia

Aprendizaje Basado en Problemas para mejorar las competencias de programación en los estudiantes de Ingeniería de Sistemas, UNSCH 2024

PROBLEMAS	OBJETIVOS	HIPÓTESIS	VARIABLES, DIM. INDICADORES	METODOLOGÍA
General ¿De qué manera el Aprendizaje Basado en Problemas mejora las competencias de programación en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024? Específicos: 1) ¿De qué manera la aplicación del Aprendizaje Basado en Problemas mejora el Pensamiento Lógico-Algorítmico? 2) ¿Cómo la aplicación del Aprendizaje Basado en Problemas mejora las Habilidades de Codificación? 3) ¿De qué manera la aplicación del Aprendizaje Basado en	General Implementar el Aprendizaje Basado en Problemas para mejorar las competencias de programación utilizando el lenguaje Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024. Específicos: 1) Utilizar el Aprendizaje Basado en Problemas para mejorar el Pensamiento Lógico-Algorítmico en los estudiantes. 2) Utilizar el Aprendizaje Basado en Problemas para mejorar las Habilidades de Codificación en los estudiantes 3) Utilizar el Aprendizaje Basado en Problemas para mejorar la	General H1: La implementación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación utilizando el lenguaje Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024. Específicas: 1) La aplicación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación en relación al Pensamiento Lógico-Algorítmico en los estudiantes. 2) La aplicación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación en relación a las	VARIABLE 1 Aprendizaje Basado en Problemas <u>Dimensiones</u> - Planificación de actividades - Ejecución de estrategias - Evaluación del proceso VARIABLE 2 Competencias de programación <u>Dimensiones</u> - Pensamiento Lógico-Algorítmico - Habilidades de Codificación - Depuración y Prueba	Tipo de investigación Aplicada Nivel de investigación Experimental Diseño Cuasi Experimental $\text{GE: } O_1 \quad X \quad O_2$ $\text{GC: } O_3 \quad - \quad O_4$ Población La población de la presente investigación estuvo conformada por 140 estudiantes de la serie 100 de la Escuela Profesional de Ingeniería de Sistemas de la UNSCH, matriculados en el semestre académico 2024-II. Los estudiantes cursaban asignaturas del plan de estudios 2018. Muestra La muestra estuvo conformada por 100 estudiantes, distribuidos en dos grupos (Grupo A y B) intactos correspondientes a la asignatura de Algoritmos y Solución de Problemas de la

Problemas mejora la Depuración y Prueba?	Depuración y Prueba de codificación en los estudiantes.	habilidades de codificación en los estudiantes. 3) La aplicación del Aprendizaje Basado en Problemas mejora significativamente las competencias de programación en relación a la Depuración y Pruebas en códigos		Escuela Profesional de Ingeniería de Sistemas, semestre 2024-II. La Grupo A fue asignada como grupo experimental (n = 50) y la Grupo B como grupo control (n = 50) ● Técnicas Prueba pedagógica ● Instrumentos Prueba de desarrollo.
---	---	--	--	---

Investigador(a): PELAYO QUISPE BAUTISTA
Maestría en Docencia Universitaria UNSCH

Anexo 2: Constancia de autorización**UNSCH**Facultad de Ingeniería de
Minas, Geología y CivilEscuela Profesional
Ingeniería de Sistemas

LA QUE SUSCRIBE; DIRECTORA DE LA ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS DE LA UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA;

CONSTANCIA

Que, autoriza al Bach. Pelayo Quispe Bautista la ejecución de la investigación titulada: "Aprendizaje Basado en Problemas para mejorar las competencias de programación en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional de San Cristóbal de Huamanga, Ayacucho 2024", incluyendo la aplicación de los instrumentos de recolección de datos y demás actividades académicas requeridas para el desarrollo del estudio.

Se expide la presente a solicitud del interesado para los fines académicos correspondientes.

Ayacucho, 10 de junio del 2024

UNIVERSIDAD NACIONAL DE SAN
CRISTÓBAL DE HUAMANGA
Escuela Profesional de Ingeniería de Sistemas

Mg. Edith F. Guevara Morote
DIRECTORA

Anexo 3 Instrumentos (Prueba pedagógica)

Prueba Pedagógica

- ¿Qué tipo de dato se asigna automáticamente a la variable en el siguiente código?

```
x = 5.7
```

 - Integer (entero)
 - Float (decimal)
 - String (cadena)
 - Boolean (booleano)
- ¿Cuál es el resultado de la siguiente expresión en Python?

```
resultado = 10 % 3 + 4 ** 2
```

 - 10
 - 17
 - 19
 - 25
- ¿Qué valor tendrá la variable resultada después de ejecutar el siguiente código?

```
a = 10
b = 20
resultado = (a < b) and (a + b > 15)
```

 - 10
 - 30
 - True
 - False
- ¿Qué imprimirá el siguiente código?

```
edad = 15
if edad >= 18:
    print("Mayor de edad")
else:
    print("Menor de edad")
```

 - Mayor de edad
 - Menor de edad
 - 15
 - Nada
- ¿Qué imprimirá el siguiente código?

```
suma = 0
for i in range(1, 5):
    suma += i
print(suma)
```

 - 10
 - 15
 - 5
 - 4
- ¿Cuál será el valor de contador después de ejecutar el siguiente código?

```
contador = 0
numero = 20
while numero > 0:
    contador += 1
    numero = numero // 2
```

 - 5
 - 6
 - 4
 - 20
- ¿Qué imprimirá el siguiente código?

```
x = 10
y = 5
if x > y:
    if x % 2 == 0:
        print("A")
    else:
        print("B")
else:
    if y % 2 == 0:
        print("C")
    else:
        print("D")
```

 - A
 - B
 - C
 - D
- ¿Cuál es la salida del siguiente código?

```
for i in range(3):
    for j in range(2):
        print(i * j, end=" ")
    print()
```

 - 0 0 0 0 0 0
 - 0 1 0 1 0 1
 - 0 0 0 1 0 2
 - 0 0 0 1 2 0
- ¿Qué concepto algorítmico se implementa en la siguiente función?

```
def busqueda_binaria(arr, objetivo):
    izquierda, derecha = 0, len(arr) - 1
    while izquierda <= derecha:
        medio = izquierda + (derecha - izquierda) // 2
        if arr[medio] == objetivo:
            return medio
        elif arr[medio] < objetivo:
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return -1
```

 - Divide y vencerás
 - Programación dinámica
 - Búsqueda exhaustiva
 - Backtracking
- ¿Qué técnica de diseño algorítmico se utiliza en el siguiente código?

```
def fibonacci(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 1:
        return n
    memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo)
    return memo[n]
```

- a) Recursión simple
- b) Programación dinámica con memoización
- c) Algoritmo voraz
- d) Búsqueda binaria

11. ¿Cuál es el resultado de la siguiente expresión en Python?

```
resultado = 10 // 3
print(resultado)
```

- a) 3.333
- b) 3
- c) 3.0
- d) Error

12. ¿Qué hace el siguiente código en Python?

```
numeros = [10, 20, 30, 40, 50]
print(numeros[1:4])
```

- a) Imprime [10, 20, 30, 40]
- b) Imprime [20, 30, 40]
- c) Imprime [10, 20, 30]
- d) Imprime [20, 30, 40, 50]

13. ¿Cómo se crea una matriz 3x3 llena de ceros en Python?

```
# Opción A
matriz = [0, 0, 0, 0, 0, 0, 0, 0, 0]
# Opción B
matriz = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
# Opción C
matriz = [[0] * 3] * 3
# Opción D
matriz = []
for i in range(3):
    matriz.append([0, 0, 0])
```

- a) Opción A
- b) Opción B
- c) Opción C
- d) Opción D

14. ¿Qué operación realiza el siguiente código?

```
def operacion(lista):
    resultado = []
    for elemento in lista:
        resultado.append(elemento * 2)
    return resultado
numeros = [1, 2, 3, 4]
print(operacion(numeros))
```

- a) Suma todos los elementos de la lista
- b) Duplica la lista
- c) Multiplica cada elemento por 2
- d) Elimina números impares

15. ¿Cuál es la diferencia entre las siguientes dos funciones?

```
# Función A
def suma1(a, b):
    return a + b
# Función B
def suma2(a, b):
    resultado = a + b
    print(resultado)
```

- a) No hay diferencia entre ellas
- b) La función A retorna un valor, la función B imprime un valor
- c) La función A solo suma enteros, la función B suma cualquier tipo
- d) La función B es más eficiente que la función A

16. ¿Qué imprimirá el siguiente código?

```
def modificar(num, lista):
    num = num + 10
    lista.append(4)
x = 5
numeros = [1, 2, 3]
modificar(x, numeros)
print(x, numeros)
```

- a) 5 [1, 2, 3]
- b) 15 [1, 2, 3]
- c) 5 [1, 2, 3, 4]
- d) 15 [1, 2, 3, 4]

17. ¿Qué calcula la siguiente función recursiva?

```
def funcion(n):
    if n <= 1:
        return 1
    return n * funcion(n - 1)
```

- a) La suma de los primeros n números
- b) El factorial de n
- c) El n-ésimo número de Fibonacci
- d) La potencia de n elevado a n

18. ¿Qué resultado se obtiene al ejecutar el siguiente código en Python?

```
lista = [1, 2, 3]
tupla = (4, 5, 6)
resultado = lista + list(tupla)
print(resultado)
```

- a) Error, no se pueden combinar listas y tuplas
- b) [1, 2, 3, (4, 5, 6)]
- c) [1, 2, 3, 4, 5, 6]
- d) [(1, 2, 3), (4, 5, 6)]

19. ¿Qué resultado produce el siguiente código en Python?

```
diccionario = {"a": 1, "b": 2}
conjunto = {"b", "c"}
resultado = set(diccionario.keys()) & conjunto
print(resultado)
```

- a) {'a', 'b', 'c'}
- b) {'b'}
- c) {'a', 'c'}
- d) Error, no se pueden operar diccionarios y conjuntos

20. ¿Qué concepto de POO demuestra el siguiente código?

```
class Animal:
    def hacer_sonido(self):
        pass
class Perro(Animal):
    def hacer_sonido(self):
        return "Guau"
class Gato(Animal):
    def hacer_sonido(self):
        return "Miau"
def sonido_animal(animal):
    print(animal.hacer_sonido())
```

- ```
perro = Perro()
gato = Gato()
sonido_animal(perro)
sonido_animal(gato)
```
- a) Encapsulamiento
  - b) Herencia
  - c) Polimorfismo
  - d) Abstracción

21. ¿Qué tipo de error se producirá al ejecutar el siguiente código?

```
def dividir(a, b):
 return a / b
resultado = dividir(10, "2")
```

- a) TypeError
- b) ValueError
- c) ZeroDivisionError
- d) NameError

22. ¿Qué error se producirá al ejecutar el siguiente código?

```
numeros = [1, 2, 3]
print(numeros[3])
```

- a) TypeError
- b) ValueError
- c) IndexError
- d) NameError

23. ¿Qué error sintáctico hay en el siguiente código?

```
def saludar(nombre)
 print("Hola, " + nombre)
```

- a) Falta el operador + entre "Hola, " y nombre
- b) Falta la indentación en la segunda línea
- c) Falta el signo de dos puntos (:) después de los parámetros
- d) Falta el paréntesis de cierre en print

24. ¿Qué problema tiene el siguiente código para calcular la suma de los primeros n números?

```
def suma_numeros(n):
 total = 0
 for i in range(n):
 total += i
 return total
print(suma_numeros(5)) # Debería imprimir 15
(1+2+3+4+5)
```

- a) El bucle debería empezar en 1, no en 0
- b) Debería usar range(1, n+1) en lugar de range(n)
- c) La variable total debería inicializarse con 1
- d) La función debería retornar total+1

25. ¿Qué error ocurrirá al ejecutar el siguiente código?

```
def dividir(a, b):
 return a / b
resultado = dividir(10, 0)
print(resultado)
```

- a) TypeError
- b) ValueError
- c) ZeroDivisionError
- d) ArithmeticError

26. ¿Cómo se puede corregir el código anterior para evitar que el programa se detenga con un error?

```
def dividir(a, b):
 if b == 0:
 return "No se puede dividir por cero"
 return a / b
resultado = dividir(10, 0)
print(resultado)
```

- a) Usando un bloque try-except
- b) Verificando si b es cero antes de dividir
- c) Inicializando b con un valor por defecto
- d) Cambiando el tipo de retorno de la función



27. ¿Qué hará el siguiente código si el usuario ingresa "abc"?

```
try:
 numero = int(input("Ingrese un número: "))
 resultado = 100 / numero
 print(f"El resultado es {resultado}")
except ValueError:
 print("Debe ingresar un número válido")
except ZeroDivisionError:
 print("No se puede dividir por cero")
finally:
 print("Fin del programa")
```

a) Imprimirá "El resultado es..." y "Fin del programa"  
 b) Imprimirá "No se puede dividir por cero" y "Fin del programa"  
 c) Imprimirá "Debe ingresar un número válido" y "Fin del programa"  
 d) Imprimirá solo "Fin del programa"

28. ¿Cuál es el propósito principal de utilizar print() durante la depuración?

- a) Mejorar el rendimiento del código
- b) Documentar el código
- c) Visualizar valores de variables y flujo de ejecución
- d) Formatear la salida final del programa

29. ¿Qué está verificando el siguiente código de prueba?

```
def test_suma():
 assert suma(2, 3) == 5, "La suma de 2 y 3 debería ser 5"
 assert suma(-1, 1) == 0, "La suma de -1 y 1 debería ser 0"
 assert suma(0, 0) == 0, "La suma de 0 y 0 debería ser 0"
 print("Todas las pruebas pasaron")
```

a) Que la función suma sea rápida  
 b) Que la función suma maneje bien los errores  
 c) Que la función suma devuelva resultados correctos para diferentes entradas  
 d) Que la función suma esté bien documentada

30. ¿Qué técnica de manejo de errores se implementa en el siguiente código?

```
class ValorNegativoError(Exception):
 """Se lanza cuando un valor es negativo y no debería serlo."""
 pass

def calcular_raiz_cuadrada(numero):
 if numero < 0:
```

```
 raise ValorNegativoError(f"No se puede calcular la raíz de {numero}")
 return numero ** 0.5

try:
 resultado = calcular_raiz_cuadrada(-5)
except ValorNegativoError as e:
 print(f"Error: {e}")
```

a) Manejo simple de excepciones  
 b) Propagación de excepciones  
 c) Definición y uso de excepciones personalizadas  
 d) Supresión de excepciones

Nombres y Apellidos: \_\_\_\_\_

Código de estudiante: \_\_\_\_\_

## Anexo 4 Confiabilidad de instrumento

Para estimar la confiabilidad del instrumento se aplicó el coeficiente Kuder-Richardson 20 (KR-20), dado que los ítems de la prueba fueron calificados de forma dicotómica (correcto = 1, incorrecto = 0). Este coeficiente se seleccionó por ser el más adecuado para instrumentos con ese tipo de respuesta, siendo equivalente al Alfa de Cronbach pero aplicado a pruebas objetivas. La interpretación de KR-20 se efectuó de acuerdo a la siguiente tabla.

### *Interpretación del coeficiente de confiabilidad KR-20*

| Valor de KR-20 | Interpretación             |
|----------------|----------------------------|
| 0.90 - 1.00    | Excelente confiabilidad    |
| 0.80 - 0.89    | Buena confiabilidad        |
| 0.70 - 0.79    | Aceptable confiabilidad    |
| 0.60 - 0.69    | Cuestionable confiabilidad |
| < 0.60         | Inaceptable confiabilidad  |

*Nota.* Fuente: George y Mallery (2003)

```

Calcular KR-20 por instrumento

resultados = []

print("=" * 60)
print(" CÁLCULO DE CONFIABILIDAD KR-20")
print("=" * 60)

for nombre, archivo in INSTRUMENTOS.items():
 df = pd.read_excel("/content/drive/MyDrive/Posgrado/TESIS/BORRADORA/Datos/Piloto/"+archivo, header=0, index_col=0, usecols="B:L")
 df = df.apply(pd.to_numeric, errors="coerce").fillna(0)
 #display(df)
 kr20, sum_pq, vt, k = calcular_kr20(df)
 interp = interpretar(kr20)

 resultados.append({
 "Instrumento": nombre,
 "Nº de ítems": k,
 "Participantes": df.shape[0],
 "Σ(p×q)": sum_pq,
 "Varianza total": vt,
 "KR-20": kr20,
 "Interpretación": interp,
 })

print(f"\n {nombre}")
print(f" Ítems : {k}")
print(f" Participantes: {df.shape[0]}")
print(f" Σ(p×q) : {sum_pq}")
print(f" Varianza T : {vt}")
print(f" KR-20 : {kr20} → {interp}")
print(" " + "-" * 56)

```

*Nota.* Código Python para calcular prueba piloto





## Experto 01



**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA**

**Unidad de Progrado de la Facultad de Ciencias de la Educación**

**Maestría en Docencia Universitario**

### TABLA DE EVALUACIÓN DE EXPERTOS

**Apellidos y Nombres del experto** : GUEVARA MOROTE EDITH FELICITAS

**Grado académico** : Maestro en Ingeniería de Sistemas Mención Informática

**Título profesional** : Ingeniero de Computación y Sistemas

**Institución en el que labora** : Universidad Nacional de San Cristóbal de Huamanga

**Instrumento de evaluación** : Prueba de desarrollo

En la presente tabla de evaluación de expertos, usted tiene la facultad de evaluar cada uno de los ítems marcando con una equis (X) en las columnas de SÍ (1) o NO (0). Asimismo, se le exhorta registrar las observaciones en el casillero correspondiente con la finalidad de mejorar la pertinencia del instrumento en evaluación.

| Ítems | Preguntas o Ítems                                                                           | Aprecia |    | Observación |
|-------|---------------------------------------------------------------------------------------------|---------|----|-------------|
|       |                                                                                             | Sí      | No |             |
| 01    | ¿Qué tipo de dato se asigna automáticamente a la variable en el siguiente código? $x = 5.7$ | X       |    |             |
| 02    | ¿Cuál es el resultado de la siguiente expresión en Python? resultado = $10 \% 3 + 4 ** 2$   | X       |    |             |
| 03    | ¿Qué valor tendrá la variable resultada después de ejecutar el siguiente código?            | X       |    |             |
| 04    | ¿Qué imprimirá el siguiente código?                                                         |         | X  |             |
| 05    | ¿Qué imprimirá el siguiente código?                                                         | X       |    |             |
| 06    | ¿Cuál será el valor de contador después de ejecutar el siguiente código?                    | X       |    |             |
| 07    | ¿Qué imprimirá el siguiente código?                                                         | X       |    |             |
| 08    | ¿Cuál es la salida del siguiente código?                                                    | X       |    |             |
| 09    | ¿Qué concepto algorítmico se implementa en la siguiente función?                            | X       |    |             |
| 10    | ¿Qué técnica de diseño algorítmico se utiliza en el siguiente código?                       | X       |    |             |
| 11    | ¿Cuál es el resultado de la siguiente expresión en Python?                                  | X       |    |             |
| 12    | ¿Qué hace el siguiente código en Python?                                                    |         | X  |             |
| 13    | ¿Cómo se crea una matriz 3x3 llena de ceros en Python?                                      | X       |    |             |
| 14    | ¿Qué operación realiza el siguiente código?                                                 | X       |    |             |
| 15    | ¿Cuál es la diferencia entre las siguientes dos funciones?                                  | X       |    |             |
| 16    | ¿Qué imprimirá el siguiente código?                                                         | X       |    |             |
| 17    | ¿Qué calcula la siguiente función recursiva?                                                | X       |    |             |


**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA**
**Unidad de Progrado de la Facultad de Ciencias de la Educación**
**Maestría en Docencia Universitario**

|    |                                                                                                 |   |  |  |
|----|-------------------------------------------------------------------------------------------------|---|--|--|
| 18 | ¿Qué resultado se obtiene al ejecutar el siguiente código en Python?                            | x |  |  |
| 19 | ¿Qué resultado produce el siguiente código en Python?                                           | x |  |  |
| 20 | ¿Qué concepto de POO demuestra el siguiente código?                                             | x |  |  |
| 21 | ¿Qué tipo de error se producirá al ejecutar el siguiente código?                                | x |  |  |
| 22 | ¿Qué error se producirá al ejecutar el siguiente código?                                        | x |  |  |
| 23 | ¿Qué error sintáctico hay en el siguiente código?                                               | x |  |  |
| 24 | ¿Qué problema tiene el siguiente código para calcular la suma de los primeros n números?        | x |  |  |
| 25 | ¿Qué error ocurrirá al ejecutar el siguiente código?                                            | x |  |  |
| 26 | ¿Cómo se puede corregir el código anterior para evitar que el programa se detenga con un error? | x |  |  |
| 27 | ¿Qué hará el siguiente código si el usuario ingresa "abc"?                                      | x |  |  |
| 28 | ¿Cuál es el propósito principal de utilizar print() durante la depuración?                      | x |  |  |
| 29 | ¿Qué está verificando el siguiente código de prueba?                                            | x |  |  |
| 30 | ¿Qué técnica de manejo de errores se implementa en el siguiente código?                         | x |  |  |

**Sugerencias:**

.....

.....

.....

.....

**EDITH FELICITAS GUEVARA MOROTE**

## Experto 02



UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA

Unidad de Progrado de la Facultad de Ciencias de la Educación

Maestría en Docencia Universitario

### TABLA DE EVALUACIÓN DE EXPERTOS

Apellidos y Nombres del experto : PERALTA SOTOMAYOR KAREL

Grado académico : Maestro en Ingeniería Informática con Mención en Ingeniería De Software

Título profesional : Ingeniero de Sistemas

Institución en el que labora : Universidad Nacional de San Cristóbal de Huamanga

Instrumento de evaluación : Prueba de desarrollo

En la presente tabla de evaluación de expertos, usted tiene la facultad de evaluar cada uno de los ítems marcando con una equis (X) en las columnas de SÍ (1) o NO (0). Asimismo, se le exhorta registrar las observaciones en el casillero correspondiente con la finalidad de mejorar la pertinencia del instrumento en evaluación.

| Items | Preguntas o Items                                                                           | Aprecia |    | Observación |
|-------|---------------------------------------------------------------------------------------------|---------|----|-------------|
|       |                                                                                             | Sí      | No |             |
| 01    | ¿Qué tipo de dato se asigna automáticamente a la variable en el siguiente código? $x = 5.7$ | ✓       |    |             |
| 02    | ¿Cuál es el resultado de la siguiente expresión en Python? resultado = $10 \% 3 + 4 ** 2$   | ✓       |    |             |
| 03    | ¿Qué valor tendrá la variable resultada después de ejecutar el siguiente código?            | ✓       |    |             |
| 04    | ¿Qué imprimirá el siguiente código?                                                         | ✓       |    |             |
| 05    | ¿Qué imprimirá el siguiente código?                                                         | ✓       |    |             |
| 06    | ¿Cuál será el valor de contador después de ejecutar el siguiente código?                    | ✓       |    |             |
| 07    | ¿Qué imprimirá el siguiente código?                                                         | ✓       |    |             |
| 08    | ¿Cuál es la salida del siguiente código?                                                    | ✓       |    |             |
| 09    | ¿Qué concepto algorítmico se implementa en la siguiente función?                            | ✓       |    |             |
| 10    | ¿Qué técnica de diseño algorítmico se utiliza en el siguiente código?                       | ✓       |    |             |
| 11    | ¿Cuál es el resultado de la siguiente expresión en Python?                                  | ✓       |    |             |
| 12    | ¿Qué hace el siguiente código en Python?                                                    | ✓       |    |             |
| 13    | ¿Cómo se crea una matriz 3x3 llena de ceros en Python?                                      | ✓       |    |             |
| 14    | ¿Qué operación realiza el siguiente código?                                                 | ✓       |    |             |
| 15    | ¿Cuál es la diferencia entre las siguientes dos funciones?                                  | ✓       |    |             |
| 16    | ¿Qué imprimirá el siguiente código?                                                         | ✓       |    |             |
| 17    | ¿Qué calcula la siguiente función recursiva?                                                | ✓       |    |             |




**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA**
**Unidad de Progrado de la Facultad de Ciencias de la Educación**
**Maestría en Docencia Universitario**

|    |                                                                                                 |   |   |  |
|----|-------------------------------------------------------------------------------------------------|---|---|--|
| 18 | ¿Qué resultado se obtiene al ejecutar el siguiente código en Python?                            | ✓ |   |  |
| 19 | ¿Qué resultado produce el siguiente código en Python?                                           | ✗ |   |  |
| 20 | ¿Qué concepto de POO demuestra el siguiente código?                                             | ✗ |   |  |
| 21 | ¿Qué tipo de error se producirá al ejecutar el siguiente código?                                |   | ✗ |  |
| 22 | ¿Qué error se producirá al ejecutar el siguiente código?                                        | ✗ |   |  |
| 23 | ¿Qué error sintáctico hay en el siguiente código?                                               | ✓ |   |  |
| 24 | ¿Qué problema tiene el siguiente código para calcular la suma de los primeros n números?        | ✗ |   |  |
| 25 | ¿Qué error ocurrirá al ejecutar el siguiente código?                                            | ✓ |   |  |
| 26 | ¿Cómo se puede corregir el código anterior para evitar que el programa se detenga con un error? | ✗ |   |  |
| 27 | ¿Qué hará el siguiente código si el usuario ingresa "abc"?                                      | ✗ |   |  |
| 28 | ¿Cuál es el propósito principal de utilizar print() durante la depuración?                      | ✓ |   |  |
| 29 | ¿Qué está verificando el siguiente código de prueba?                                            | ✓ |   |  |
| 30 | ¿Qué técnica de manejo de errores se implementa en el siguiente código?                         | ✗ |   |  |

**Sugerencias:**

.....

.....

.....

.....

**KAREL PERALTA SOTOMAYOR**



### Experto 03



UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA

Unidad de Progrado de la Facultad de Ciencias de la Educación

Maestría en Docencia Universitario

#### TABLA DE EVALUACIÓN DE EXPERTOS

Apellidos y Nombres del experto : CARREÑO GAMARRA JUAN CARLOS

Grado académico : Maestro en Ingeniería de Sistemas Mención en Informática

Título profesional : Ingeniero de Sistemas y Computo

Institución en el que labora : Universidad Nacional de San Cristóbal de Huamanga

Instrumento de evaluación : Prueba de desarrollo

En la presente tabla de evaluación de expertos, usted tiene la facultad de evaluar cada uno de los ítems marcando con una equis (X) en las columnas de SÍ (1) o NO (0). Asimismo, se le exhorta registrar las observaciones en el casillero correspondiente con la finalidad de mejorar la pertinencia del instrumento en evaluación.

| Ítems | Preguntas o Ítems                                                                                | Aprecia |    | Observación |
|-------|--------------------------------------------------------------------------------------------------|---------|----|-------------|
|       |                                                                                                  | Si      | No |             |
| 01    | ¿Qué tipo de dato se asigna automáticamente a la variable en el siguiente código? $x = 5.7$      |         | X  |             |
| 02    | ¿Cuál es el resultado de la siguiente expresión en Python? $\text{resultado} = 10 \% 3 + 4 ** 2$ | X       |    |             |
| 03    | ¿Qué valor tendrá la variable resultada después de ejecutar el siguiente código?                 | X       |    |             |
| 04    | ¿Qué imprimirá el siguiente código?                                                              | X       |    |             |
| 05    | ¿Qué imprimirá el siguiente código?                                                              | X       |    |             |
| 06    | ¿Cuál será el valor de contador después de ejecutar el siguiente código?                         | X       |    |             |
| 07    | ¿Qué imprimirá el siguiente código?                                                              | X       |    |             |
| 08    | ¿Cuál es la salida del siguiente código?                                                         | X       |    |             |
| 09    | ¿Qué concepto algorítmico se implementa en la siguiente función?                                 | X       |    |             |
| 10    | ¿Qué técnica de diseño algorítmico se utiliza en el siguiente código?                            | X       |    |             |
| 11    | ¿Cuál es el resultado de la siguiente expresión en Python?                                       | X       |    |             |
| 12    | ¿Qué hace el siguiente código en Python?                                                         | X       |    |             |
| 13    | ¿Cómo se crea una matriz 3x3 llena de ceros en Python?                                           | X       |    |             |
| 14    | ¿Qué operación realiza el siguiente código?                                                      | X       |    |             |
| 15    | ¿Cuál es la diferencia entre las siguientes dos funciones?                                       | X       |    |             |
| 16    | ¿Qué imprimirá el siguiente código?                                                              | X       |    |             |
| 17    | ¿Qué calcula la siguiente función recursiva?                                                     | X       |    |             |


**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA**
**Unidad de Progrado de la Facultad de Ciencias de la Educación**
**Maestría en Docencia Universitario**

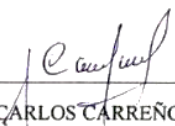
|    |                                                                                                 |   |  |  |
|----|-------------------------------------------------------------------------------------------------|---|--|--|
| 18 | ¿Qué resultado se obtiene al ejecutar el siguiente código en Python?                            | ✓ |  |  |
| 19 | ¿Qué resultado produce el siguiente código en Python?                                           | ✗ |  |  |
| 20 | ¿Qué concepto de POO demuestra el siguiente código?                                             | ✓ |  |  |
| 21 | ¿Qué tipo de error se producirá al ejecutar el siguiente código?                                | ✓ |  |  |
| 22 | ¿Qué error se producirá al ejecutar el siguiente código?                                        | ✓ |  |  |
| 23 | ¿Qué error sintáctico hay en el siguiente código?                                               | ✗ |  |  |
| 24 | ¿Qué problema tiene el siguiente código para calcular la suma de los primeros n números?        | ✓ |  |  |
| 25 | ¿Qué error ocurrirá al ejecutar el siguiente código?                                            | ✗ |  |  |
| 26 | ¿Cómo se puede corregir el código anterior para evitar que el programa se detenga con un error? | ✓ |  |  |
| 27 | ¿Qué hará el siguiente código si el usuario ingresa "abc"?                                      | ✗ |  |  |
| 28 | ¿Cuál es el propósito principal de utilizar print() durante la depuración?                      | ✗ |  |  |
| 29 | ¿Qué está verificando el siguiente código de prueba?                                            | ✗ |  |  |
| 30 | ¿Qué técnica de manejo de errores se implementa en el siguiente código?                         | ✓ |  |  |

**Sugerencias:**

.....

.....

.....

  
 JUAN CARLOS CARREÑO GAMARRA

## Experto 04



**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA**

Unidad de Posgrado de la Facultad de Ciencias de la Educación

Maestría en Docencia Universitaria

### TABLA DE EVALUACIÓN DE EXPERTOS

**Apellidos y Nombres del experto** : CHRISTIAN LEZAMA CUELLAR

**Grado académico** : Maestro en ciencias de ingeniería; Mención en Gestión de Tecnologías de la Información y Comunicación

**Título profesional** : Ingeniero de Electrónico

**Institución en el que labora** : Universidad Nacional de San Cristóbal de Huamanga

**Instrumento de evaluación** : Prueba de desarrollo

En la presente tabla de evaluación de expertos, usted tiene la facultad de evaluar cada uno de los ítems marcando con una equis (X) en las columnas de SÍ (1) o NO (0). Asimismo, se le exhorta registrar las observaciones en el casillero correspondiente con la finalidad de mejorar la pertinencia del instrumento en evaluación.

| Ítems | Preguntas o Ítems                                                                           | Aprecia |    | Observación              |
|-------|---------------------------------------------------------------------------------------------|---------|----|--------------------------|
|       |                                                                                             | Si      | No |                          |
| 01    | ¿Qué tipo de dato se asigna automáticamente a la variable en el siguiente código? $x = 5.7$ |         | X  | no es simple la pregunta |
| 02    | ¿Cuál es el resultado de la siguiente expresión en Python? resultado = $10 \% 3 + 4 ** 2$   | X       |    |                          |
| 03    | ¿Qué valor tendrá la variable resultada después de ejecutar el siguiente código?            | X       |    |                          |
| 04    | ¿Qué imprimirá el siguiente código?                                                         | X       |    |                          |
| 05    | ¿Qué imprimirá el siguiente código?                                                         | X       |    |                          |
| 06    | ¿Cuál será el valor de contador después de ejecutar el siguiente código?                    | X       |    |                          |
| 07    | ¿Qué imprimirá el siguiente código?                                                         | X       |    |                          |
| 08    | ¿Cuál es la salida del siguiente código?                                                    | X       |    |                          |
| 09    | ¿Qué concepto algorítmico se implementa en la siguiente función?                            | X       |    |                          |
| 10    | ¿Qué técnica de diseño algorítmico se utiliza en el siguiente código?                       | X       |    |                          |
| 11    | ¿Cuál es el resultado de la siguiente expresión en Python?                                  | X       |    |                          |
| 12    | ¿Qué hace el siguiente código en Python?                                                    | X       |    |                          |
| 13    | ¿Cómo se crea una matriz 3x3 llena de ceros en Python?                                      | X       |    |                          |
| 14    | ¿Qué operación realiza el siguiente código?                                                 | X       |    |                          |
| 15    | ¿Cuál es la diferencia entre las siguientes dos funciones?                                  | X       |    |                          |
| 16    | ¿Qué imprimirá el siguiente código?                                                         | X       |    |                          |


**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA**

Unidad de Posgrado de la Facultad de Ciencias de la Educación

Maestría en Docencia Universitaria

|    |                                                                                                 |   |  |  |
|----|-------------------------------------------------------------------------------------------------|---|--|--|
| 17 | ¿Qué calcula la siguiente función recursiva?                                                    | ✗ |  |  |
| 18 | ¿Qué resultado se obtiene al ejecutar el siguiente código en Python?                            | ✓ |  |  |
| 19 | ¿Qué resultado produce el siguiente código en Python?                                           | ✓ |  |  |
| 20 | ¿Qué concepto de POO demuestra el siguiente código?                                             | ✓ |  |  |
| 21 | ¿Qué tipo de error se producirá al ejecutar el siguiente código?                                | ✓ |  |  |
| 22 | ¿Qué error se producirá al ejecutar el siguiente código?                                        | ✓ |  |  |
| 23 | ¿Qué error sintáctico hay en el siguiente código?                                               | ✓ |  |  |
| 24 | ¿Qué problema tiene el siguiente código para calcular la suma de los primeros n números?        | ✓ |  |  |
| 25 | ¿Qué error ocurrirá al ejecutar el siguiente código?                                            | ✓ |  |  |
| 26 | ¿Cómo se puede corregir el código anterior para evitar que el programa se detenga con un error? | ✓ |  |  |
| 27 | ¿Qué hará el siguiente código si el usuario ingresa "abc"?                                      | ✓ |  |  |
| 28 | ¿Cuál es el propósito principal de utilizar print() durante la depuración?                      | ✓ |  |  |
| 29 | ¿Qué está verificando el siguiente código de prueba?                                            | ✓ |  |  |
| 30 | ¿Qué técnica de manejo de errores se implementa en el siguiente código?                         | ✓ |  |  |

Sugerencias:

.....

.....

.....

.....

Ing. CHRISTIAN LEZAMA CUELLAR

## Anexo 6 Procesamiento estadístico

### Resultados descriptivos pretest

*Estadísticos descriptivos del pretest por grupo y variable*

|           | Pretest Control |      |      |      | Pretest Experimental |      |      |      |
|-----------|-----------------|------|------|------|----------------------|------|------|------|
|           | PLA             | HC   | DP   | CP   | PLA                  | HC   | DP   | CP   |
| $\bar{x}$ | 6.09            | 6.08 | 6.58 | 6.25 | 6.31                 | 6.22 | 6.40 | 6.31 |
| Md        | 4               | 4    | 4    | 7.33 | 2                    | 6    | 4    | 6    |
| DE        | 6.33            | 6.38 | 6.05 | 3.50 | 6.94                 | 5.15 | 6.45 | 3.68 |
| R         | 18              | 18   | 18   | 14   | 20                   | 18   | 20   | 13.3 |

*Nota.*  $\bar{x}$  = media aritmética; Md = mediana; DE = desviación estándar; R = rango; PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General.

### Código Python

```
def calcular_estadisticas(data):
 return {
 'Media': round(float(np.mean(data)), 4),
 'Mediana': round(float(np.median(data)), 4),
 'DE': round(float(np.std(data, ddof=1)), 4),
 'Rango': round(float(np.max(data) - np.min(data)), 4),
 'Mínimo': round(float(np.min(data)), 4),
 'Máximo': round(float(np.max(data)), 4),
 }

filas = []
for var in VARIABLES:
 for grupo, df in [('Control', df_ctrl), ('Experimental', df_exp)]:
 stats = calcular_estadisticas(df[var].dropna().values)
 for stat, val in stats.items():
 filas.append({'Variable': var, 'Grupo': grupo, 'Estadístico': stat, 'Valor': val})

tabla = pd.DataFrame(filas).pivot_table(
 index=['Variable', 'Estadístico'], columns='Grupo', values='Valor'
)
display(tabla)
```

### Resultados descriptivos posttest

*Estadísticos descriptivos del posttest por grupo y variable*

|  | Postest Control |    |    |    | Postest Experimental |    |    |    |
|--|-----------------|----|----|----|----------------------|----|----|----|
|  | PLA             | HC | DP | CP | PLA                  | HC | DP | CP |

|           |     |     |     |      |     |     |       |      |
|-----------|-----|-----|-----|------|-----|-----|-------|------|
| $\bar{x}$ | 7   | 7   | 7.2 | 7.3  | 16  | 16  | 15.42 | 15.7 |
| <b>Md</b> | 6   | 6   | 4   | 6.7  | 18  | 18  | 18    | 16   |
| <b>DE</b> | 6.1 | 6.1 | 6.7 | 3    | 4.5 | 4.5 | 5.7   | 2.6  |
| <b>R</b>  | 18  | 20  | 20  | 10.7 | 20  | 18  | 18    | 9.3  |

*Nota.*  $\bar{x}$  = media aritmética; Md = mediana; DE = desviación estándar; R = rango; PLA = Pensamiento Lógico Algorítmico; HC = Habilidades de Codificación; DP = Depuración y Prueba; CP = Promedio General.

## Código Python

```
def calcular_estadisticas(data):
 return {
 'Media': round(float(np.mean(data)), 4),
 'Mediana': round(float(np.median(data)), 4),
 'DE': round(float(np.std(data, ddof=1)), 4),
 'Rango': round(float(np.max(data) - np.min(data)), 4),
 'Mínimo': round(float(np.min(data)), 4),
 'Máximo': round(float(np.max(data)), 4),
 }

filas = []
for var in VARIABLES:
 for grupo, df in [('Control', df_ctrl), ('Experimental', df_exp)]:
 stats = calcular_estadisticas(df[var].dropna().values)
 for stat, val in stats.items():
 filas.append({'Variable': var, 'Grupo': grupo, 'Estadístico': stat, 'Valor': val})

tabla = pd.DataFrame(filas).pivot_table(
 index=['Variable', 'Estadístico'], columns='Grupo', values='Valor'
)
display(tabla)
```

## Anexo 7 Artículo científico

### Artículo científico

#### **Aprendizaje basado en problemas para mejorar las competencias de programación en los estudiantes de Ingeniería de Sistemas, UNSCH 2024**

#### **Problem-Based Learning to Improve Programming Competencies in Systems Engineering Students, UNSCH 2024**

Universidad Nacional de San Cristóbal de Huamanga

Bach. Pelayo Quispe Bautista

Correo: pelayo.quispe@unsch.edu.pe

### **Resumen**

La presente investigación tuvo como objetivo implementar el Aprendizaje Basado en Problemas (ABP) para fortalecer las competencias de programación en Python en estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga, Ayacucho, durante el año 2024. El estudio se desarrolló bajo un enfoque cuantitativo, de tipo aplicada y con diseño cuasiexperimental de pretest y postest. La muestra estuvo integrada por 90 estudiantes del curso de Algoritmos y Solución de Problemas del semestre 2024-II, distribuidos en un grupo experimental y un grupo control, ambos con 45 participantes.

Para la recolección de datos se empleó una prueba de desarrollo de 30 ítems, validada mediante juicio de expertos y con un coeficiente de confiabilidad KR-20 promedio de 0.89. El instrumento evaluó tres dimensiones: pensamiento lógico-algorítmico, habilidades de codificación y procesos de depuración y prueba. La intervención comprendió 11 sesiones basadas en ABP, incorporando herramientas tecnológicas como Python, Google Colab, GitHub y PyCharm.

Debido a que los datos no presentaron distribución normal según la prueba de Shapiro-Wilk ( $p < 0.05$ ), se aplicaron las pruebas no paramétricas U de Mann-Whitney y Rangos con Signo de Wilcoxon. Los resultados del postest mostraron diferencias estadísticamente significativas ( $p < 0.001$ ) a favor del grupo experimental en todas las dimensiones evaluadas. Asimismo, se obtuvo una diferencia de 8.4 puntos en el promedio general entre el grupo experimental ( $\bar{x} = 15.7$ ) y el grupo control ( $\bar{x} = 7.3$ ). Se concluye que el ABP mejoró significativamente las competencias de programación en Python de los estudiantes de Ingeniería de Sistemas.

**Palabras clave:** aprendizaje basado en problemas, competencias de programación, pensamiento lógico-algorítmico, habilidades de codificación, depuración y prueba, Python.

### Abstract

This research aimed to implement Problem-Based Learning (PBL) to improve programming competencies in Python among Systems Engineering students at the Universidad Nacional San Cristóbal de Huamanga, Ayacucho 2024.

The study followed a quantitative approach, applied type, and quasi-experimental design with pretest and posttest. The sample consisted of 90 students from the Algorithms and Data Structures course during the 2024-II semester, distributed into an experimental group ( $n = 45$ ) and a control group ( $n = 45$ ). A 30-item development test was used, validated through expert judgment with an average KR-20 coefficient of 0.89, structured in three dimensions: Logical-Algorithmic Thinking, Coding Skills, and Debugging and Testing. The intervention comprised 11 PBL sessions integrating technological tools such as Python, Google Colab, GitHub, and PyCharm.

Since the data did not follow a normal distribution according to the Shapiro-Wilk test ( $p < 0.05$ ), the Mann-Whitney U test was used for between-group comparisons and the Wilcoxon Signed-Rank test for within-group comparisons. Initial equivalence between groups was confirmed in the pretest ( $p > 0.05$  for all variables).

Posttest results showed statistically significant differences ( $p < 0.001$ ) across all three dimensions, with an 8.4-point difference in the Overall Average between the experimental group ( $\bar{x} = 15.7$ ) and the control group ( $\bar{x} = 7.3$ ). It is concluded that PBL significantly improved programming competencies in Systems Engineering students at UNSCH.

**Keywords:** problem-based learning, programming competencies, logical-algorithmic thinking, coding skills, debugging and testing, Python.



## Introducción

La enseñanza de la programación en la educación superior constituye uno de los principales desafíos en la formación de profesionales vinculados a las ciencias de la computación e ingeniería de sistemas. En diversos contextos universitarios, los estudiantes presentan dificultades para comprender conceptos abstractos relacionados con algoritmos, estructuras de control, razonamiento lógico y resolución de problemas, lo que repercute directamente en su desempeño académico y en el desarrollo de competencias necesarias para el ámbito profesional. Bennedsen y Caspersen (2019) reportaron que los cursos introductorios de programación registran elevados índices de desaprobación a nivel internacional, evidenciando la persistencia de limitaciones asociadas a metodologías tradicionales centradas en la transmisión pasiva de contenidos.

En el contexto latinoamericano y peruano, esta problemática adquiere mayor relevancia debido a las exigencias actuales del mercado laboral y al acelerado crecimiento de la industria tecnológica, que demandas profesionales con capacidades para desarrollar soluciones informáticas eficientes, trabajar colaborativamente y adaptarse a entornos tecnológicos cambiantes. En las universidades peruanas, especialmente en carreras de Ingeniería de Sistemas, se observa que muchos estudiantes presentan dificultades para desarrollar competencias de programación relacionadas con el pensamiento lógico-algorítmico, la codificación y la depuración de programas, aspectos fundamentales para la construcción de software funcional y de calidad.

En la Universidad Nacional San Cristóbal de Huamanga (UNSCH), particularmente en el curso de Algoritmos y Solución de Problemas, se evidencian limitaciones en el aprendizaje de programación en Python, lenguaje ampliamente utilizado en el desarrollo de software, análisis de datos e inteligencia artificial debido a su sintaxis sencilla y versatilidad. Los estudiantes muestran dificultades para analizar problemas, diseñar algoritmos, traducir soluciones a código ejecutable y corregir errores durante el proceso de programación. Estas limitaciones generan bajo rendimiento académico y reducen las posibilidades de desarrollar competencias tecnológicas acordes con las demandas profesionales contemporáneas.

Frente a esta realidad, surge la necesidad de implementar estrategias metodológicas innovadoras que permitan fortalecer el aprendizaje activo y significativo de la programación. En este contexto, el Aprendizaje Basado en Problemas (ABP) se presenta como una metodología pedagógica orientada a la construcción del conocimiento mediante la resolución de problemas reales y contextualizados. Esta estrategia promueve la participación activa del estudiante, el trabajo colaborativo y el desarrollo de habilidades cognitivas superiores vinculadas con el análisis, la reflexión y la toma de decisiones.

De acuerdo con Luy-Montejo (2019), el ABP constituye una metodología centrada en el estudiante que favorece el aprendizaje autónomo y la adquisición de conocimientos a través de experiencias prácticas relacionadas con situaciones problemáticas reales. Asimismo, Sari et al. (2021) sostienen que esta metodología fortalece el pensamiento crítico y el razonamiento científico, permitiendo que los estudiantes desarrollen capacidades para resolver problemas de manera estructurada y sistemática. Por su parte,

Hidajat (2023) señala que el ABP impulsa la autorregulación del aprendizaje y la búsqueda autónoma de información, favoreciendo el desarrollo del pensamiento de orden superior.

Desde el punto de vista pedagógico, el ABP encuentra sustento en las teorías constructivistas y socioculturales del aprendizaje. Piaget sostiene que el conocimiento se construye activamente a partir de la interacción del sujeto con su entorno, mientras que Vygotsky enfatiza la importancia de la interacción social y del aprendizaje colaborativo en el desarrollo cognitivo. Del mismo modo, Ausubel plantea que el aprendizaje significativo ocurre cuando los nuevos conocimientos se relacionan con estructuras cognitivas previas del estudiante. Estas teorías coinciden en considerar al estudiante como protagonista de su propio proceso formativo, aspecto esencial en el desarrollo de competencias de programación.

Asimismo, Dewey (1969) argumentó que el aprendizaje resulta más efectivo cuando el estudiante enfrenta situaciones problemáticas auténticas que demandan reflexión y acción. Desde esta perspectiva, el aula debe convertirse en un espacio dinámico de experimentación y resolución de problemas, donde el conocimiento se construye mediante la práctica y la experiencia. Esta concepción pedagógica guarda estrecha relación con la enseñanza de la programación, debido a que el desarrollo de software implica procesos permanentes de análisis, diseño, prueba y solución de problemas.

Diversos estudios han demostrado la efectividad del Aprendizaje Basado en Problemas en la educación superior y particularmente en áreas relacionadas con ingeniería y programación. Freeman et al. (2014), en un metaanálisis realizado en cursos universitarios de ciencia, tecnología, ingeniería y matemáticas (STEM), concluyeron que las metodologías activas mejoran significativamente el rendimiento académico y reducen las tasas de desaprobación en comparación con la enseñanza tradicional. Del mismo modo, Hmelo-Silver (2004) afirma que el ABP favorece el aprendizaje autónomo, el trabajo colaborativo y el desarrollo de habilidades de resolución de problemas complejos.

En relación con las competencias de programación, diversos autores coinciden en que estas comprenden un conjunto de conocimientos, habilidades y capacidades orientadas al desarrollo de soluciones informáticas mediante el uso de lenguajes de programación. Vinuesa-Morales et al. (2025) definen las competencias de programación como habilidades necesarias para resolver problemas mediante la comprensión y escritura de código, incluyendo estructuras de control, funciones, algoritmos y manejo de datos. Asimismo, Ahirrao et al. (2024) destacan que dichas competencias permiten adaptarse a las demandas tecnológicas y resolver problemas de manera eficiente en contextos profesionales reales.

Dentro de estas competencias, el pensamiento lógico-algorítmico constituye una dimensión fundamental, ya que permite organizar secuencias lógicas y estructuradas para la solución de problemas computacionales. Lu et al. (2022) sostienen que esta habilidad forma parte esencial del pensamiento computacional y posibilita la comprensión de procesos secuenciales, estructuras repetitivas y toma de decisiones mediante operadores lógicos. Del mismo modo, Milková y Hulková (2013) consideran que el pensamiento

lógico-algorítmico fortalece la capacidad de razonamiento secuencial y resolución estructurada de problemas.

Otra dimensión relevante corresponde a las habilidades de codificación, entendidas como la capacidad de transformar soluciones algorítmicas en código funcional utilizando un lenguaje de programación. Robins et al. (2003) señalan que estas habilidades incluyen no solo el dominio sintáctico del lenguaje, sino también la comprensión semántica y la capacidad de estructurar programas eficientes y legibles. En el caso de Python, estas competencias abarcan el manejo de estructuras de control, funciones, listas, diccionarios y buenas prácticas de programación.

Asimismo, la depuración y prueba representan procesos esenciales para garantizar la calidad del software desarrollado. McCauley et al. (2008) describen la depuración como una actividad cognitiva compleja orientada a identificar, analizar y corregir errores en el código, mientras que la prueba implica verificar el funcionamiento adecuado de los programas bajo diferentes condiciones. Estas capacidades permiten desarrollar soluciones confiables y funcionales, indispensables en la formación de futuros ingenieros de sistemas.

En este contexto, el Aprendizaje Basado en Problemas constituye una alternativa pedagógica pertinente para fortalecer las competencias de programación en estudiantes universitarios, debido a que integra la resolución de problemas reales, el trabajo colaborativo y el aprendizaje autónomo mediante herramientas tecnológicas. La implementación de metodologías activas en cursos de programación favorece el desarrollo de capacidades analíticas, prácticas y reflexivas necesarias para responder a las exigencias del entorno tecnológico actual.

Por ello, la presente investigación tuvo como objetivo determinar la influencia del Aprendizaje Basado en Problemas en el desarrollo de competencias de programación en Python en estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga durante el año 2024. Para alcanzar dicho propósito, se implementó una propuesta pedagógica basada en sesiones ABP utilizando herramientas tecnológicas como Python, Google Colab, GitHub y PyCharm, orientadas a fortalecer el aprendizaje práctico, colaborativo y significativo de la programación.

## **Metodología**

La investigación se desarrolló bajo un enfoque cuantitativo y un nivel experimental, debido a que se orientó a determinar el efecto del Aprendizaje Basado en Problemas (ABP) sobre las competencias de programación en estudiantes de Ingeniería de Sistemas de la Universidad Nacional de San Cristóbal de Huamanga durante el semestre académico 2024-II. Para ello, se evaluaron cuantitativamente las dimensiones de pensamiento lógico-algorítmico, habilidades de codificación y depuración y prueba, mediante la aplicación de instrumentos antes y después de la intervención pedagógica. Asimismo, se emplearon técnicas de estadística descriptiva e inferencial para el procesamiento y análisis de los datos, permitiendo contrastar las hipótesis planteadas y determinar la significancia de los resultados obtenidos.

El estudio adoptó un diseño cuasiexperimental con pretest y posttest, conformado por un grupo experimental y un grupo control. La población estuvo integrada por 140 estudiantes de la serie 100 de la Escuela Profesional de Ingeniería de Sistemas de la UNSCH, mientras que la muestra estuvo conformada por 100 estudiantes distribuidos en ambos grupos. El método experimental permitió analizar el efecto de la aplicación del ABP mediante la comparación de resultados entre los grupos, mientras que el método estadístico facilitó la organización, análisis e interpretación de la información recolectada, garantizando rigurosidad metodológica y validez en los hallazgos de la investigación

### Resultados

Los resultados obtenidos evidencian que la aplicación del Aprendizaje Basado en Problemas (ABP) generó mejoras significativas en las competencias de programación de los estudiantes del grupo experimental. Para el análisis inferencial se empleó la prueba de rangos con signo de Wilcoxon, comparando los resultados del pretest y posttest en las dimensiones evaluadas.

**Tabla** **27**  
*Prueba de Wilcoxon - Pensamiento Lógico-Algorítmico, grupo experimental (pretest vs. posttest)*

| Variable | W     | p       | Decisión         |
|----------|-------|---------|------------------|
| PLA      | 53.00 | < 0.001 | Se rechaza $H_0$ |

*Nota.* W = estadístico de Wilcoxon (suma de rangos negativos); p = significancia unilateral.  $\alpha = 0.05$ . n = 45. PLA = Pensamiento Lógico Algorítmico.

**Prueba de hipótesis específica 1,** Dado que  $p < 0.001 < \alpha = 0.05$ , se rechaza la hipótesis nula y se acepta  $H_1$ . Se concluye que la aplicación del Aprendizaje Basado en Problemas mejoró significativamente las competencias de programación en relación al Pensamiento Lógico-Algorítmico en los estudiantes de Ingeniería de Sistemas de la UNSCH durante el semestre académico 2024-II.

**Tabla** **28**  
*Prueba de Wilcoxon - Habilidades de Codificación, grupo experimental (pretest vs. posttest)*

| Variable | W     | p       | Decisión         |
|----------|-------|---------|------------------|
| HC       | 15.00 | < 0.001 | Se rechaza $H_0$ |

*Nota.* W = estadístico de Wilcoxon (suma de rangos negativos); p = significancia unilateral.  $\alpha = 0.05$ . n = 45. HC = Habilidades de Codificación.

**Prueba de hipótesis específica.** Dado que  $p < 0.001 < \alpha = 0.05$ , se rechaza la hipótesis nula y se acepta  $H_1$ . Se concluye que la aplicación del Aprendizaje Basado en Problemas mejoró significativamente las competencias de programación en relación a las Habilidades de Codificación en los estudiantes de Ingeniería de Sistemas de la UNSCH durante el semestre académico 2024-II.

**Tabla** **29**  
***Prueba de Wilcoxon - Depuración y Prueba, grupo experimental (pretest vs. posttest)***

| Variable | W     | p       | Decisión      |
|----------|-------|---------|---------------|
| DP       | 60.00 | < 0.001 | Se rechaza H0 |

*Nota.* W = estadístico de Wilcoxon (suma de rangos negativos); p = significancia unilateral.  $\alpha = 0.05$ . n = 45. DP = Depuración y Prueba.

***Prueba de hipótesis específica 3.*** Dado que  $p < 0.001 < \alpha = 0.05$ , se rechaza la hipótesis nula y se acepta H1. Se concluye que la aplicación del Aprendizaje Basado en Problemas mejoró significativamente las competencias de programación en relación a la Depuración y Pruebas en códigos en los estudiantes de Ingeniería de Sistemas de la UNSCH durante el semestre académico 2024-II.

### **Discusión de resultados**

Los resultados obtenidos en la presente investigación confirman que la implementación del Aprendizaje Basado en Problemas mejoró significativamente las competencias de programación en Python en los estudiantes de Ingeniería de Sistemas de la Universidad Nacional San Cristóbal de Huamanga durante el semestre académico 2024-II. En efecto, la diferencia de 8.4 puntos en el Promedio General entre el grupo experimental ( $\bar{x} = 15.7$ ) y el grupo control ( $\bar{x} = 7.3$ ), confirmada mediante la prueba U de Mann-Whitney ( $p < 0.001$ ), constituye evidencia estadísticamente sólida del efecto positivo de la intervención pedagógica implementada. Estos hallazgos son convergentes con los reportados por García et al. (2021), quienes, en un estudio cuasiexperimental con estudiantes de Ingeniería de Sistemas de una universidad pública peruana, constataron que la aplicación del ABP en el curso de Base de Datos produjo diferencias significativas en el posttest entre el grupo experimental y el grupo control ( $\text{Sig} = 0.000 < 0.05$ ), confirmando que el ABP constituye una estrategia pedagógica eficaz para el desarrollo de competencias técnicas especializadas en contextos similares al de la presente investigación.

En lo que respecta a la dimensión de Pensamiento Lógico-Algorítmico, el grupo experimental registró un incremento de  $\bar{x} = 6.31$  en el pretest a  $\bar{x} = 16.04$  en el posttest, con una mediana de  $Md = 18.00$ , lo que indica que la mayoría de los participantes alcanzó niveles de logro próximos al puntaje máximo. Este resultado es coherente con lo reportado por Alshaye et al. (2023), quienes demostraron que las tareas de ABP mejoran significativamente la capacidad de resolución de problemas y las habilidades de programación en estudiantes, con diferencias favorables al grupo experimental en el

postest. En la misma línea, Lu et al. (2022) señalan que el pensamiento lógico-algorítmico constituye una habilidad esencial del pensamiento computacional que el ABP activa de manera estructural al situar al estudiante frente a problemas auténticos que demandan análisis, planificación y representación algorítmica antes de la implementación en código. Asimismo, Milková y Hulková (2013) sostienen que el pensamiento algorítmico se potencia cuando el estudiante organiza ideas y hechos en una progresión encadenada con sentido propio, proceso que las sesiones ABP de la presente propuesta promovieron sistemáticamente desde las primeras semanas de intervención.

En cuanto a las Habilidades de Codificación, el grupo experimental alcanzó en el postest una media de  $\bar{x} = 15.60$  y una mediana de  $Md = 18.00$ , frente a  $\bar{x} = 7.60$  y  $Md = 6.00$  del grupo control. Adicionalmente, la reducción de la desviación estándar en el grupo experimental evidencia una tendencia hacia la homogeneización del aprendizaje, con la mayoría de los estudiantes concentrándose en los niveles superiores de la escala. Este efecto es consistente con los hallazgos de AlSideiri et al. (2023), quienes reportaron mejoras significativas en el rendimiento académico y en el logro de competencias de los estudiantes que cursaron ingeniería de software bajo el enfoque ABP, en comparación con aquellos que recibieron instrucción tradicional. Del mismo modo, Rattanakha y Chatwattana (2023) demostraron que el modelo ABP mediado por tecnología en la nube promueve el pensamiento analítico sistemático y el uso efectivo de herramientas tecnológicas, competencias que la presente propuesta cultivó mediante una secuencia progresiva que integró Python, Google Colab, Visual Studio Code y GitHub. Al respecto, Robins et al. (2003) argumentan que las habilidades de codificación trascienden el conocimiento sintáctico del lenguaje para implicar comprensión semántica y capacidad de estructurar código de forma eficiente, precisamente el tipo de competencia que el ABP desarrolla al exigir al estudiante no solo escribir código, sino justificar sus decisiones de diseño, corregir errores y presentar soluciones funcionales ante sus pares.

Por su parte, la dimensión de Depuración y Prueba reveló las diferencias más pronunciadas en términos de tendencia central entre ambos grupos. El grupo experimental obtuvo  $\bar{x} = 15.42$  y  $Md = 18.00$ , mientras que el grupo control registró  $\bar{x} = 7.20$  y  $Md = 4.00$ , lo que indica que al menos la mitad de los integrantes de este último grupo no superó los 4 puntos sobre 20 en esta dimensión. Está marcada discrepancia pone de manifiesto las limitaciones de la enseñanza expositiva tradicional para el desarrollo de una

competencia cognitiva avanzada que, según McCauley et al. (2008), requiere comprensión profunda del programa, razonamiento lógico y aplicación sistemática de estrategias de diagnóstico. En contraste, las sesiones ABP orientadas al manejo de errores, la depuración sistemática con Python Debugger y el diseño de pruebas unitarias con pytest proporcionaron al grupo experimental un entorno de práctica deliberada y reflexiva que favoreció la adquisición de esta competencia, resultado confirmado mediante la prueba de Rangos con Signo de Wilcoxon ( $W = 60.00$ ;  $p < 0.001$ ).

Desde una perspectiva teórica, los resultados de la presente investigación hallan sustento en los fundamentos psicopedagógicos del ABP. La teoría sociocultural de Vygotsky (1978) ilumina el mecanismo subyacente al efecto diferenciado observado entre los grupos: el trabajo en equipos colaborativos con roles técnicos rotativos y problemas de complejidad creciente reprodujo la dinámica de la Zona de Desarrollo Próximo, permitiendo que los estudiantes abordaran desafíos que habrían resultado inaccesibles de manera individual. Complementariamente, el ciclo de aprendizaje experiencial de Kolb (1984) se manifiesta en la homogeneización de los puntajes del grupo experimental en el posttest, indicativa de que los estudiantes atravesaron iterativamente las fases de experiencia concreta, observación reflexiva, conceptualización abstracta y experimentación activa en cada sesión ABP, consolidando aprendizajes más sólidos y transferibles que los producidos por la exposición magistral. En concordancia con ello, Luy-Montejo (2019) identifica la pertinencia contextual de los problemas como condición determinante para la activación del aprendizaje genuino en el ABP universitario, condición que la presente propuesta operacionalizó mediante situaciones vinculadas a la gestión académica, el análisis de datos estudiantiles y el modelamiento de sistemas universitarios en Python.

En cuanto a los aportes de la presente investigación al campo educativo, cabe destacar que los hallazgos contribuyen a fortalecer la evidencia empírica sobre la eficacia del ABP en la enseñanza de la programación en el contexto de la educación superior tecnológica peruana, ámbito en el que la investigación experimental sigue siendo escasa. A diferencia de Santiago (2022), quien demostró mejoras en el pensamiento algorítmico atribuibles al uso del lenguaje Python sin una metodología activa articulada, la presente investigación demuestra que es la combinación de Python con el ABP como marco pedagógico la que genera incrementos sustantivamente superiores en el conjunto de las

competencias de programación. Asimismo, la propuesta pedagógica desarrollada, que integra herramientas como GitHub, Google Colab, pytest y Python Debugger en una secuencia de 11 sesiones con el modelo de cinco fases ABP, constituye un aporte metodológico concreto y replicable para docentes de cursos de programación que buscan superar las limitaciones del modelo transmisivo tradicional. En este sentido, la investigación ofrece tanto evidencia cuantitativa de impacto como un diseño didáctico documentado que puede adaptarse a distintas asignaturas del plan curricular de Ingeniería de Sistemas.

No obstante, los hallazgos anteriores, la presente investigación presenta algunas limitaciones que deben considerarse al momento de interpretar y generalizar sus resultados. En primer lugar, el diseño cuasiexperimental, si bien apropiado para el contexto educativo en que fue implementado, no permite la asignación aleatoria de participantes a los grupos, lo que limita el control de variables extrañas y restringe la posibilidad de establecer relaciones de causalidad con la misma solidez que un experimento puro. En segundo lugar, la muestra estuvo circunscrita a dos secciones de una única asignatura en un solo semestre académico de la UNSCH, lo que limita la validez externa de los hallazgos y su generalización a otros cursos, instituciones o niveles educativos. En tercer lugar, el instrumento de evaluación, aunque validado mediante juicio de expertos y con un coeficiente KR-20 de 0.89, mide competencias de programación en un momento puntual, sin dar cuenta del proceso de aprendizaje ni de la transferencia de las competencias adquiridas a contextos profesionales reales. Finalmente, la duración de la intervención, compuesta por 11 sesiones distribuidas en 13 semanas, podría no ser suficiente para consolidar competencias de mayor complejidad, aspecto que futuras investigaciones deberían abordar mediante diseños longitudinales con seguimiento posterior a la intervención

### **Conclusiones**

- ✓ La implementación del Aprendizaje Basado en Problemas (ABP) mejoró significativamente las competencias de programación en Python de los estudiantes de Ingeniería de Sistemas de la UNSCH durante el semestre 2024-II. El grupo experimental obtuvo una media de 15.7 frente a 7.3 del grupo control, con diferencias estadísticamente significativas ( $p < 0.001$ ) en el promedio general y en las



dimensiones de Pensamiento Lógico-Algorítmico, Habilidades de Codificación y Depuración y Prueba, evidenciando que el ABP es una estrategia pedagógica eficaz para el desarrollo integral de competencias de programación.

- ✓ La aplicación del ABP mejoró significativamente el Pensamiento Lógico-Algorítmico de los estudiantes, ya que el grupo experimental alcanzó una media de 16.04 frente a 6.98 del grupo control ( $p < 0.001$ ). Esto demuestra que la resolución de problemas auténticos favoreció el desarrollo del razonamiento lógico y la capacidad de análisis y representación algorítmica.
- ✓ El ABP también fortaleció significativamente las Habilidades de Codificación en Python. El grupo experimental obtuvo una media de 15.60 frente a 7.60 del grupo control ( $p < 0.001$ ), concluyéndose que el trabajo colaborativo y el uso de herramientas como Google Colab, Visual Studio Code y GitHub promovieron competencias de codificación funcionales y transferibles.
- ✓ Desde el punto de vista metodológico, el diseño cuasiexperimental con pretest y posttest resultó adecuado para evaluar el efecto del ABP, garantizando la validez de los hallazgos mediante la equivalencia inicial entre grupos y el uso de pruebas no paramétricas apropiadas. Asimismo, el instrumento aplicado mostró alta confiabilidad ( $KR-20 = 0.89$ ), respaldando la solidez estadística y metodológica de la investigación.

### Referencias

- Aguilar, E. E. (2014). *El aprendizaje basado en problemas y el rendimiento académico de los estudiantes del C.E.P. Cibernet, Huamanga—Ayacucho. 2012* [Universidad Nacional de San Cristóbal de Huamanga]. <http://repositorio.unsch.edu.pe/handle/UNSCH/778>
- Ahirrao, M., Borse, S., Deshmukh, A., & Mishra, V. (2024). Programming Skills Recommendation System with Generative AI. *International Journal For Multidisciplinary Research*, 6(2), 14528. <https://doi.org/10.36948/ijfmr.2024.v06i02.14528>
- Alshaye, I. A., Tasir, Z., & Jumaat, N. F. (2023). The Effectiveness of Online Problem-Based Learning Tasks on Riyadh's Secondary School Students' Problem-Solving Ability

and Programming Skills. *Open Education Studies*, 5(1), 20220208. <https://doi.org/10.1515/edu-2022-0208>

AlSideiri, A., Tawafak, R. M., Malik, S. I., Alfarsi, G., Khudayer, B. H., & Mathew, R. (2023). Examining the Impact of Software Engineering Problem-Based Learning on Student Performance. *2023 24th International Arab Conference on Information Technology (ACIT)*, 1-6. <https://doi.org/10.1109/ACIT58888.2023.10453857>

Amanda, F. F., Sumitro, S. B., Lestari, S. R., & Ibrohim, I. (2024). Enhancing Critical Thinking And Problem Solving Skills By Complexity Science-Problem Based Learning Model. *REMIE: Multidisciplinary Journal of Educational Research*, 14(1), 96-114. <https://dialnet.unirioja.es/servlet/articulo?codigo=9315025>

Araya Cabrera, M. P., & Martínez Clares, P. (2023). El Aprendizaje Basado en Problemas como estrategia de seguimiento de las competencias del perfil de egreso. Una experiencia en Ingeniería Civil en la Universidad de Valparaíso. *Educatio Siglo XXI*, 41(2), 11-38. <https://doi.org/10.6018/educatio.503551>

Arias, F. G. (2012). *El Proyecto de Investigación. Introducción a la Metodología Científica*. 6ta. Edición. Fidia G. Arias Odón.

Becker, B. A., & Quille, K. (2019). 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 338-344. <https://doi.org/10.1145/3287324.3287432>

Bennedsen, J., & Caspersen, M. E. (2019a). Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10(2), 30-36. <https://doi.org/10.1145/3324888>

Bennedsen, J., & Caspersen, M. E. (2019b). Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10(2), 30-36. <https://doi.org/10.1145/3324888>

Bernal, C. A. B. (2022). *Metodología de la investigación. Administración, Economía, Humanidades y Ciencias Sociales*.

Boud, D., & Falchikov, N. (2006). Aligning assessment with long-term learning. *Assessment & Evaluation in Higher Education*, 31(4), 399-413. <https://doi.org/10.1080/02602930600679050>

Branda, L. A. (2009). El aprendizaje basado en problemas en la formación en ciencias de la salud. En *El aprendizaje basado en problemas: Una herramienta para toda la vida* (pp. 15-28). Agencia Laín Entralgo. (Obra original publicada en Bilbao, J.)

Bustamante Vega, R. M., & Mejía Mejía, E. J. (2023). El ABP como estrategia para mejorar la comprensión lectora en educación superior. *IGOBERNANZA*, 6(23), 86-114. <https://doi.org/10.47865/igob.vol6.n23.2023.290>

Campbell, D. T., & Stanley, J. C. (1966). *Experimental and quasi-experimental designs for research*. Wadsworth.

Carrasco, S. (2008). *Metodología de la investigación científica: Pautas metodológicas para diseñar y elaborar el proyecto de investigación* (2a ed). San Marcos.

Carrasco, S. (2015). *Metodología de la investigación científica: Pautas metodológicas para diseñar y elaborar el proyecto de investigación* (Segunda edición, novena reimpresión 2015). San Marcos.

Chueh, H.-E., & Kao, C.-Y. (2024). Exploring the impact of integrating problem based learning and agile in the classroom on enhancing professional competence. *Heliyon*, 10(3), e24887. <https://doi.org/10.1016/j.heliyon.2024.e24887>

Córdova, M. (2003). *Estadística descriptiva e inferencial: Aplicaciones* (5a. ed). Moshera.

Corral, Y. (2022). Validez y confiabilidad en instrumentos de investigación: Una mirada teórica. *Revista ciencias de la educación*, (60), 562-586. <https://dialnet.unirioja.es/servlet/articulo?codigo=9710359>

Dewey, J. (1969). *EXPERIENCE AND EDUCATION: THE KAPPA DELTA PI LECTURE SERIES*.

Díaz Barriga, F. (2002). *Estrategias Docentes Para Un Aprendizaje Significativo - 2b: Edición*.

Field, A. (2018). *Discovering Statistics Using IBM SPSS Statistics*. SAGE Publications.

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science,

engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 111(23), 8410-8415. <https://doi.org/10.1073/pnas.1319030111>

Freire, P. (1970). *Pedagogía del oprimido* (Tra). Siglo Veintiuno Editores.

Galicia, L. A., Balderrama, J. A., & Edel, R. (2017). Content validity by experts judgment: Proposal for a virtual tool. *Apertura*, 9(2), 42-53. <https://doi.org/10.32870/Ap.v9n2.993>

García, B. L., Oscanoa, R. B., & Rivera, F. (2021). *Influencia del aprendizaje basado en problemas en las competencias adquiridas en el curso de base de datos: Caso Universidad Peruana*. <https://repositorio.uide.edu.ec/handle/37000/4760>

García Ramos, J. M. (1989). *Bases pedagógicas de la evaluación: (Guía práctica para educadores)*. Síntesis. <https://dialnet.unirioja.es/servlet/libro?codigo=89624>

Garrison, D. R., & Vaughan, N. D. (2008). *Blended learning in higher education: Framework, principles, and guidelines* (1st ed). Jossey-Bass. <https://doi.org/10.1002/9781118269558>

George, D., & Mallery, P. (2003). *SPSS for Windows step by step: A simple guide and reference, 11.0 update* (4th ed). Allyn and Bacon.

Guerrero-Rodríguez, O. A., Herrera-Navas, O. D., Guzmán-Hernández, R., & Baque-Arteaga, M. E. (2025). Metodología B-Learning en el desarrollo de competencias en Programación y Bases de Datos en estudiantes de primero bachillerato informática. *MQRInvestigar*, 9(1), e265. <https://doi.org/10.56048/MQR20225.9.1.2025.e265>

Guillemin, M., & Gillam, L. (2004). Ethics, Reflexivity, and “Ethically Important Moments” in Research. *Qualitative Inquiry*, 10(2), 261-280. <https://doi.org/10.1177/1077800403262360>

Hasbi, A. R. R., Arafah, K., & Khaeruddin. (2024). The Effect of Problem Based Learning Models and Interest to Learn Physics on Students Critical Thinking Skills of Class XI Students at SMA Negeri 1 Selayar. *International Journal of Social Science and Human Research*, 07(02). <https://doi.org/10.47191/ijsshr/v7-i02-74>

Hernández-Sampieri, R., & Mendoza, C. P. (2018). *Metodología de la investigación: Las rutas cuantitativa, cualitativa y mixta* (First edition). McGraw-Hill Education.

Hidajat, F. A. (2023). A comparison between problem-based conventional learning and creative problem-based learning on self-regulation skills: Experimental study. *Heliyon*, 9(9), e19512. <https://doi.org/10.1016/j.heliyon.2023.e19512>

Hmelo-Silver, C. E. (2004). Problem-Based Learning: What and How Do Students Learn? *Educational Psychology Review*, 16(3), 235-266. <https://doi.org/10.1023/B:EDPR.0000034022.16470.f3>

Holenko Dlab, M., & Hoic-Bozic, N. (2021). Effectiveness of game development-based learning for acquiring programming skills in lower secondary education in Croatia. *Education and Information Technologies*, 26(4), 4433-4456. <https://doi.org/10.1007/s10639-021-10471-w>

INEI. (2023). *Perú: Estadísticas de las Tecnologías de Información y Comunicación en las Empresas*. [https://www.inei.gob.pe/media/MenuRecursivo/publicaciones\\_digitales/Est/Lib1539/libro.pdf](https://www.inei.gob.pe/media/MenuRecursivo/publicaciones_digitales/Est/Lib1539/libro.pdf)

Kolb, D. A. (1984). *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall.

Kuder, G. F., & Richardson, M. W. (1937). The Theory of the Estimation of Test Reliability. *Psychometrika*, 2(3), 151-160. <https://doi.org/10.1007/BF02288391>

Lema, V. M., Mbondo, M., & Kamau, E. N. (2010). Informed consent for clinical trials: A review. *East African Medical Journal*, 86(3), 133-142. <https://doi.org/10.4314/eamj.v86i3.54968>

Loyens, S. M. M., Kirschner, P. A., & Paas, F. (2012). Problem-based learning. En K. R. Harris, S. Graham, T. Urdan, A. G. Bus, S. Major, & H. L. Swanson (Eds.), *APA educational psychology handbook, Vol 3: Application to learning and teaching*. (pp. 403-425). American Psychological Association. <https://doi.org/10.1037/13275-016>

Lu, C., Macdonald, R., Odell, B., Kokhan, V., Demmans Epp, C., & Cutumisu, M. (2022). A scoping review of computational thinking assessments in higher education. *Journal of*

*Computing in Higher Education*, 34(2), 416-461. <https://doi.org/10.1007/s12528-021-09305-y>

Luy-Montejo, C. (2019a). El Aprendizaje Basado en Problemas (ABP) en el desarrollo de la inteligencia emocional de estudiantes universitarios. *Propósitos y Representaciones*, 7(2), 353-383. <https://doi.org/10.20511/pyr2019.v7n2.288>

Luy-Montejo, C. (2019b). El Aprendizaje Basado en Problemas (ABP) en el desarrollo de la inteligencia emocional de estudiantes universitarios. *Propósitos y Representaciones*, 7(2), 353-383. <https://doi.org/10.20511/pyr2019.v7n2.288>

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: A review of the literature from an educational perspective. *Computer Science Education*, 18(2), 67-92. <https://doi.org/10.1080/08993400802114581>

Milková, E., & Hulková, A. (2013). Algorithmic and Logical Thinking Development: Base of Programming Skills. *WSEAS Transactions on Computers*, 12. <https://www.wseas.com/journals/articles.php?id=5736>

Morin, E. (1999). *Los Siete saberes necesarios para la educación del futuro*. <https://repositorio.minedu.gob.pe/handle/20.500.12799/1448>

O'Grady, G., Yew, E. H. J., Goh, K. P. L., & Schmidt, H. G. (Eds.). (2012). *One-Day, One-Problem: An Approach to Problem-based Learning*. Springer Singapore. <https://doi.org/10.1007/978-981-4021-75-3>

Osmanović Zajić, J., & Maksimović, J. (2022). Quasi-Experimental Research as An Epistemological-Methodological Approach in Education Research. *International Journal of Cognitive Research in Science, Engineering and Education (IJCRSEE)*, 10(3), 177-183. <https://doi.org/10.23947/2334-8496-2022-10-3-177-183>

Palomino, J. T., & Osorio, V. G. (2023). El aprendizaje basado en problemas para el logro de competencias en educación superior. *Dilemas contemporáneos: Educación, Política y Valores*. <https://doi.org/10.46377/dilemas.v2i10.3484>

Peirce, C. (2016). How to Make our Ideas Clear, by Charles S. Peirce. *Popular Science Monthly* 12 (January 1878), 286-302. *Revista Filosofía UIS*, 15(2), 287-303. <https://dialnet.unirioja.es/servlet/articulo?codigo=8028952>

Perrenoud, P. (2004). *Diez nuevas competencias para enseñar: Invitación al viaje*. Graó.  
<https://dialnet.unirioja.es/servlet/libro?codigo=249036>

Rattanakha, R., & Chatwattana, P. (2023). The Problem-based Learning Model: PBL Model via Cloud Technology to Promote Programming Skills. *Higher Education Studies*, 13(4), 201. <https://doi.org/10.5539/hes.v13n4p201>

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2), 137-172. <https://doi.org/10.1076/csed.13.2.137.14200>

Santiago, V. H. (2022). *Python y la mejora del pensamiento algorítmico en los alumnos de la escuela profesional de Ingeniería de Sistemas, 2022* [Universidad Nacional Helario Valdizan]. <https://repositorio.unheval.edu.pe/item/42bf876f-25af-4152-a98a-72d649e7f54a>

Sari, Y. I., Sumarmi, Utomo, D. H., & Astina, I. K. (2021). The Effect of Problem Based Learning on Problem Solving and Scientific Writing Skills. *International Journal of Instruction*, 14(2), 11-26. <https://eric.ed.gov/?id=EJ1290959>

Siegel, S., & Castellan, N. J. (1998). *Estadística no paramétrica aplicada a las ciencias de la conducta* (1a.ed). Trillas.

Skjong, R., & Wentworth, B. H. (2001). *Expert Judgment and Risk Perception*. 537-544.

TIOBE Software. (s. f.). TIOBE index for July 2025. *TIOBE*. Recuperado 19 de agosto de 2025, de <https://www.tiobe.com/tiobe-index/>

Tobón, S. T. (2013). *Formación integral y competencias*. Ecoe Ediciones.

Vinueza-Morales, M., Cárdenas-Cobo, J., Robles-Salguero, R., & Vidal-Silva, C. (2025). Impacto del lenguaje de programación ALICE en el desarrollo de competencias de programación de estudiantes universitarios. *Formación Universitaria*, 18(3), 167-178. <https://doi.org/10.4067/s0718-50062025000300167>

Vinueza-Morales, M., Rodas-Silva, J., & Vidal-Silva, C. (2025a). Teaching Programming in Higher Education: Analyzing Trends, Technologies, and Pedagogical Approaches

Through a Bibliometric Lens. *International Journal of Advanced Computer Science and Applications*, 16(1). <https://doi.org/10.14569/IJACSA.2025.0160106>

Vinueza-Morales, M., Rodas-Silva, J., & Vidal-Silva, C. (2025b). Teaching Programming in Higher Education: Analyzing Trends, Technologies, and Pedagogical Approaches Through a Bibliometric Lens. *International Journal of Advanced Computer Science and Applications*, 16(1). <https://doi.org/10.14569/IJACSA.2025.0160106>

Vygotsky, L. S. (1978). *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press.

Xue, G., Chen, S., Xie, J., Yang, M., & Wang, Z. (2025). Application of problem-based learning combined with three-dimensional visualization reconstruction technology in trauma orthopedics teaching and its impact on teaching satisfaction. *BMC Medical Education*, 26(1), 53. <https://doi.org/10.1186/s12909-025-08364-4>



## Anexo 8 Guía de sesiones

Introducción e Instalación python Sesión 01

| Introducción a Python y Pensamiento Algorítmico |                                       | SESIÓN 01<br>Sumaria 3 |                |
|-------------------------------------------------|---------------------------------------|------------------------|----------------|
| Asignatura:                                     | Algoritmos y Estructuras de Datos     | Semestre:              | 2024-II        |
| Docente:                                        | Ing. Pelayo Quispe Bautista           | Duración:              | 90 minutos     |
| Metodología:                                    | Aprendizaje Basado en Problemas (ABP) | Fases ABP:             | Fases 1 y 2    |
| Herramientas:                                   | Google Colab, Diagramación digital    | Semana:                | Semana 3 de 13 |

**1. FASE 1: PRESENTACIÓN DEL PROBLEMA**

**FASE 1: Presentación del Problema**

El docente presenta el escenario tecnológico real vinculado al contexto universitario UNSCH. Los estudiantes identifican el problema, formulan preguntas guía y activan sus conocimientos previos. Competencia desarrollada: PLA - Análisis y descomposición del problema.

**1.1 Escenario real del problema**

**Situación problemática:**  
La Oficina de Tecnologías de Información (OTI) de la UNSCH recibe cada fin de semestre un archivo Excel con más de 3.000 registros de notas del Sistema de Información e Integración de Gestión Educativa (SIGIE). El proceso actual es completamente manual: un operador revisa fila por fila, calcula el promedio de cada estudiante con una calculadora y escribe a mano la categoría de rendimiento. Este proceso tarda 3 días, genera errores frecuentes y debe repetirse cada semestre.

**El jefe de la OTI, el Ing. Hubner Janampa, le plantea el siguiente desafío:**  
"Necesitamos un algoritmo que reciba el código del estudiante, su nombre y sus tres notas (práctica 30%, investigación 30%, examen final 40%), calcule su promedio ponderado automáticamente y lo clasifique según la escala de rendimiento de la UNSCH: Sobresaliente (17-20), Bueno (14-16), Aprobado (10.5-13) o Desaprobado (0-10.4). El sistema debe ser claro, reproducible y adaptable a cualquier número de estudiantes."

**Tu misión como equipo (Fase 1):** Analizar el problema, formular las preguntas guía e identificar qué necesitan aprender para resolverlo.

**1.2 Preguntas guía: Lo que el equipo debe responder**  
En la metodología ABP, el equipo NO recibe la solución directamente. Primero identifica el problema y formula las preguntas que guiarán su aprendizaje. Responden juntos las siguientes preguntas antes de continuar:

**Actividad Fase 1: Análisis del problema**

Leen el escenario de la OTI y responden en equipo:

**Pregunta 1:** ¿Cuáles son exactamente las ENTRADAS del sistema que pide la OTI? ¿Qué datos necesita recibir el algoritmo?

**Pregunta 2:** ¿Cuál es el PROCESO que debe realizar el algoritmo? ¿Qué operaciones en qué orden?

**Pregunta 3:** ¿Cuáles son las SALIDAS que debe producir el sistema? ¿Qué resultado debe mostrar al operador de la OTI?

**Pregunta 4:** ¿Por qué Excel no es suficiente para resolver este problema con 3.000 registros?

**Pregunta 5:** ¿Qué herramienta de programación sería más adecuada y por qué?

**Pregunta 6:** ¿Qué conocimientos previos tiene el equipo sobre algoritmos? ¿Qué necesitan aprender hoy?

**1.3 Análisis de entradas, proceso y salidas**  
Todo algoritmo tiene tres componentes fundamentales que deben identificarse antes de diseñarlo. En el problema SIGIE:

| ENTRADAS                                                                                                                                                                                  | PROCESO                                                                                                                                                                                                                                                                                                                             | SALIDAS                                                                                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>código_estudiante (texto)</li> <li>nombre_estudiante (texto)</li> <li>nota_practica (0-20, 30%)</li> <li>nota_investigacion (0-20, 30%)</li> </ul> | <ol style="list-style-type: none"> <li>Calcular promedio ponderado:<br/> <math display="block">\text{promedio} = \frac{\text{practica} \times 0.30 + \text{inv} \times 0.30 + \text{ex} \times 0.40}{1}</math> </li> <li>Clasificar según escala UNSCH:<br/> <math display="block">\geq 17 \text{ Sobresaliente}</math> </li> </ol> | <ul style="list-style-type: none"> <li>promedio_final (decimal)</li> <li>categoría_rendimiento (texto)</li> <li>estado: Aprobado / Desaprobado</li> <li>Reporte para el operador OTI</li> </ul> |

Ing. Pelayo Quispe Bautista 1 | Página

Introducción e Instalación python Sesión 01

|                           |                                                                 |  |
|---------------------------|-----------------------------------------------------------------|--|
| • nota_examen (0-20, 40%) | $\geq 14$ Bueno<br>$\geq 10.5$ Aprobado<br>$< 10.5$ Desaprobado |  |
|---------------------------|-----------------------------------------------------------------|--|

**2. COMPETENCIAS A DESARROLLAR**

| Pensamiento Lógico-Algorítmico (PLA)                                                                                                                                        | Habilidades de Codificación (HC)                                                                                                     | Depuración y Prueba (DP)                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Analiza el problema SIGIE, descompone el proceso en pasos lógicos, identifica entradas/procesos/salidas y representa la solución mediante pseudocódigo y diagrama de flujo. | Escribe el primer programa Python en Google Colab traduciendo el pseudocódigo diseñado. Se introduce en esta sesión de forma guiada. | Verifica que el pseudocódigo y el diagrama de flujo cubren todos los casos del problema: nota mínima, nota máxima y casos límite. |

**3. CONTENIDO CONCEPTUAL**

El siguiente contenido ha sido seleccionado específicamente para que el equipo pueda responder las preguntas guía de la Fase 1 y planificar la solución en la Fase 2. Léanlo en equipo, discutan y concórralo con el problema de la OTI.

**3.1 Pensamiento algorítmico: La base de la programación**  
Antes de escribir una sola línea de código, todo programador profesional diseña el algoritmo. Un algoritmo es una secuencia finita, ordenada y precisa de pasos que resuelve un problema específico. En el ABP, diseñar el algoritmo primero es la diferencia entre un programador que entiende el problema y uno que solo copia código.

Propiedades que debe tener un buen algoritmo:

- Finitud:** Debe terminar en algún momento. Un algoritmo que ejecuta para siempre no sirve a la OTI.
- Precisión:** Cada paso debe ser exacto. No puede haber ambigüedad. ¿Promedio mayor o igual a 10.5? No es lo mismo que "¿mayor a 10.5?"
- Entrada definida:** Los datos deben estar claramente especificados. En el SIGIE: código, nombre y tres notas.
- Salida definida:** El resultado debe estar claramente especificado. En el SIGIE: categoría de rendimiento y promedio.
- Efectividad:** Cada paso debe ser realizable. No puede pedir al operador que adivine datos.

**3.2 Pseudocódigo: El borrador del programador**  
El pseudocódigo es una representación textual del algoritmo en lenguaje natural estructurado. No es Python ni ningún otro lenguaje: es un borrador que permite pensar la solución antes de preocuparse por la sintaxis. En el ABP, el pseudocódigo es la herramienta de la Fase 2 para planificar antes de codificar.

Convenciones del pseudocódigo usadas en esta guía:

- INICIO / FIN:** Delimitan el algoritmo completo.
- LEER variable:** Solicitar un dato de entrada.
- ESCRIBIR texto:** Mostrar un resultado.
- variable = valor:** Asignar un valor a una variable.
- SI condición ENTONCES ... SINO ... FIN SI:** Estructura condicional.
- PARA / MIENTRAS:** Estructuras de repetición (sesiones 3 y 4).

**3.3 Diagrama de flujo: La representación visual del algoritmo**  
Un diagrama de flujo es la representación gráfica del algoritmo mediante símbolos estandarizados conectados por flechas. Permite ver el flujo completo del proceso antes de codificarlo. En la Fase 2, el equipo debe elaborar el diagrama de flujo de la solución SIGIE antes de abrir Google Colab.

| Símbolo          | Nombre y forma                         | Uso en el algoritmo SIGIE                                                                     |
|------------------|----------------------------------------|-----------------------------------------------------------------------------------------------|
| INICIO / FIN     | Óvalo o elipse. Delimita el algoritmo. | INICIO: el sistema SIGIE arranca. FIN: el reporte fue generado.                               |
| PROCESO          | Rectángulo. Operación o cálculo.       | Calcular promedio = $(n1 \times 0.30 + n2 \times 0.30 + n3 \times 0.40)$ . Asignar categoría. |
| DECISION         | Rombo. Condición con salida SI / No.   | ¿promedio $\geq 17$ ? ¿promedio $\geq 14$ ? ¿promedio $\geq 10.5$ ? Ramifica el flujo.        |
| ENTRADA / SALIDA | Paralelogramo. Leer o mostrar datos.   | Entrada: código, nombre, notas. Salida: categoría y promedio.                                 |
| FLECHA           | Línea con punta. Dirección del flujo.  | Une todos los símbolos indicando el orden de ejecución del algoritmo.                         |

Ing. Pelayo Quispe Bautista 2 | Página

Introducción e Instalación python Sesión 01

**3.4 Por qué Python para el problema de la OTI**  
Python es un lenguaje de programación interpretado, de alto nivel y propósito general. Según el índice TIOBE (mayo 2025), Python ocupa el primer lugar mundial con un 28.98% de popularidad, impulsado por su protagonismo en inteligencia artificial, análisis de datos y automatización de procesos: exactamente lo que necesita la OTI.

| Característica     | Excel (actual en OTI)                                      | Python (propuesta)                                            |
|--------------------|------------------------------------------------------------|---------------------------------------------------------------|
| Volumen de datos   | Lento con más de 1.000 filas. Propenso a errores manuales. | Procesa 3.000+ registros en segundos con Pandas.              |
| Automatización     | Requiere acción manual cada semestre.                      | Un script se ejecuta automáticamente sin intervención humana. |
| Reproducibilidad   | Difícil de replicar exactamente.                           | El mismo código produce el mismo resultado siempre.           |
| Integración con BD | Limitada. Requiere exportar/importar manualmente.          | Conexión directa a la base de datos del SIGIE.                |
| Popularidad TIOBE  | No aplica (no es lenguaje de programación).                | 1er lugar mundial: 28.98% (mayo 2025).                        |

**3.5 Primer contacto con Python en Google Colab**  
Google Colab es un entorno de Jupyter Notebook en la nube de Google. Permite escribir y ejecutar Python directamente en el navegador sin instalar nada. El equipo puede trabajar colaborativamente en tiempo real compartiendo el notebook por enlace.

**Verificación del entorno**

# Celda 1: Verificar que Python está disponible

```
import sys
print(Python [sys.version])
print("Google Colab está listo para el proyecto SIGIE")
```

# Celda 2: Presentación del equipo

```
nombre_equipo = "Equipo Alpha"
integrantes = ["Ana Quispe (Líder)",
 "Carlos Palomino (Desarrollador)",
 "María Huamán (Tester)",
 "Jorge Cárdenas (Documentador)"]

print("\n")
print("PROYECTO SIGIE - OTI UNSCH | Sesión 1")
print("\n")
print(f"Equipo: {nombre_equipo}")
for integrante in integrantes:
 print(f"• {integrante}")
print("\n")
```

**4. FASE 2: ORGANIZACIÓN DEL APRENDIZAJE**

**FASE 2: Organización del Aprendizaje**

Los equipos discuten conocimientos previos, identifican brechas de aprendizaje y elaboran el plan de investigación. Cada integrante asume su rol técnico: líder, desarrollador, tester y documentador. El equipo diseña el pseudocódigo y el diagrama de flujo ANTES de codificar. Competencia desarrollada: PLA — Diseño de estrategias de solución.

**4.1 Ficha de organización del equipo**

**Actividad Fase 2: Organización del equipo**

Completen esta ficha en equipo antes de comenzar a diseñar el algoritmo:

Nombre del equipo: \_\_\_\_\_

Ing. Pelayo Quispe Bautista 3 | Página

Introducción e Instalación python Sesión 01

**Roles asignados:**

- Líder de equipo: \_\_\_\_\_  
(Coordina el trabajo y toma decisiones en caso de desacuerdo)
- Desarrollador: \_\_\_\_\_  
(Escribe el pseudocódigo y el código Python)
- Tester: \_\_\_\_\_  
(Verifica el algoritmo con casos de prueba límite)
- Documentador: \_\_\_\_\_  
(Registra el proceso, errores encontrados y decisiones tomadas)

Lo que ya sabemos sobre el problema: \_\_\_\_\_

Lo que necesitamos aprender para resolverlo: \_\_\_\_\_

Plan de trabajo para esta sesión:

- Diseñar el pseudocódigo del algoritmo clasificador
- Elaborar el diagrama de flujo
- Traducir a Python en Google Colab
- Verificar con casos de prueba

**4.2 Diseño del pseudocódigo**  
Con el análisis de la Fase 1 completo, el equipo diseña ahora el pseudocódigo. Este es el paso más importante: pensar antes de codificar. El desarrollador lleva estas actividades mientras el documentador registra las decisiones tomadas.

**SEUDOCÓDIGO - Clasificador de rendimiento académico SIGIE**

```
INICIO
// ENTRADAS: Datos del estudiante
LEER código_estudiante // Ejemplo: '27150415'
LEER nombre_estudiante // Ejemplo: 'Carmen Quispe Huamán'
LEER nota_practica // Valor entre 0 y 20, peso: 30%
LEER nota_investigacion // Valor entre 0 y 20, peso: 30%
LEER nota_examen // Valor entre 0 y 20, peso: 40%

// PROCESO 1: Cálculo del promedio ponderado
// Fórmula: suma de (nota * peso) para cada componente
promedio = (nota_practica * 0.30) +
 (nota_investigacion * 0.30) +
 (nota_examen * 0.40)

// PROCESO 2: Clasificación según escala vigeresimal UNSCH
// Regla: se evalúa de mayor a menor para evitar solapamientos
Si promedio >= 17 ENTONCES
 categoria = "Sobresaliente"
 estado = "Aprobado"
Sino Si promedio >= 14 ENTONCES
 categoria = "Bueno"
 estado = "Aprobado"
Sino Si promedio >= 10.5 ENTONCES
 categoria = "Aprobado"
Sino Si promedio < 10.5 ENTONCES
 categoria = "Desaprobado"
 estado = "Desaprobado"

FIN
```

Ing. Pelayo Quispe Bautista 4 | Página

Introducción e Instalación python Sesión 01

```

SINO Si promedio >= 10.5 ENTONCES
 categoria ← 'Aprobado'
 estado ← 'Aprobado'
SINO
 categoria ← 'Desaprobado'
 estado ← 'Desaprobado'
FIN_SI

// SALIDAS: Reporte para el operador de la OTI
ESCRIBIR '=====
ESCRIBIR ' REPORTE SIGE - OTI UNSCH
ESCRIBIR '=====
ESCRIBIR ' Código : ', codigo_estudiante
ESCRIBIR ' Estudiante : ', nombre_estudiante
ESCRIBIR ' Promedio : ', promedio
ESCRIBIR ' Categoría : ', categoria
ESCRIBIR ' Estado : ', estado
ESCRIBIR '=====
FIN

```

### 5. TRADUCCIÓN A PYTHON

Una vez que el tester ha verificado que el diagrama de flujo y el pseudocódigo son correctos, el desarrollador procede a traducirlo a Python en Google Colab. Cada línea del pseudocódigo tiene su equivalente exacto en Python.

#### 5.1 Tabla de equivalencias pseudocódigo

| Pseudocódigo                    | Python equivalente                              |
|---------------------------------|-------------------------------------------------|
| LEER codigo_estudiante          | codigo_estudiante = input('Ingrese código: ')   |
| LEER nota_practica              | nota_practica = float(input('Nota practica: ')) |
| promedio ← (n_prac*0.30)+...    | promedio = (nota_practica*0.30) + ...           |
| Si promedio >= 11 ENTONCES      | if promedio >= 11:                              |
| categoria ← 'Sobresaliente'     | categoria = 'Sobresaliente'                     |
| SINO Si promedio >= 14 ENTONCES | elif promedio >= 14:                            |
| SINO                            | else:                                           |
| ESCRIBIR 'Promedio: ', promedio | print('Promedio: ', promedio, 2f)               |

#### 5.2 Código Python completo — Problema central resuelto

El siguiente código traduce directamente el pseudocódigo diseñado en la Fase 2. El documentador debe copiar este código en el notebook de Google Colab y registrar cualquier diferencia entre lo que planeamos y lo que resulta.

```

=====
SISTEMA SIGE - OTI UNSCH | Sesión 1
Clasificador de rendimiento académico
Algoritmo: Promedio ponderado + clasificación vigesimal
Equipo: [nombre del equipo] | Fecha: 2024
=====
ENTRADAS: Datos del estudiante
LEER código estudiante → en Python:
codigo_estudiante = '27150415'
nombre_estudiante = 'Carmen Quispe Huamán'

```

Ing. Pelayo Quispe Bautista

5 | Página

Introducción e Instalación python Sesión 01

```

nota_practica = 15.0 # Peso: 30%
nota_investigacion = 14.5 # Peso: 30%
nota_examen = 16.0 # Peso: 40%

PROCESO 1: Cálculo del promedio ponderado
promedio ← (nota_practica * 0.30) + ... → en Python:
promedio = (nota_practica * 0.30) + 1
 (nota_investigacion * 0.30) + 1
 (nota_examen * 0.40)

PROCESO 2: Clasificación según escala vigesimal UNSCH
Si promedio >= 17 ENTONCES ... → en Python:
if promedio >= 17:
 categoria = 'Sobresaliente'
 estado = 'Aprobado'
elif promedio >= 14:
 categoria = 'Bueno'
 estado = 'Aprobado'
elif promedio >= 10.5:
 categoria = 'Aprobado'
 estado = 'Aprobado'
else:
 categoria = 'Desaprobado'
 estado = 'Desaprobado'

SALIDAS: Reporte para el operador de la OTI
ESCRIBIR ... → en Python: print()
print("=====")
print(" REPORTE DE RENDIMIENTO - SIGE OTI UNSCH")
print("=====")
print("Código : [codigo_estudiante]")
print("Estudiante : [nombre_estudiante]")
print("Práctica : [nota_practica, 1f] (peso 30%)")
print("Investigac.: [nota_investigacion, 1f] (peso 30%)")
print("Examen : [nota_examen, 1f] (peso 40%)")
print("Promedio : [promedio, 2f]")
print("Categoría : [categoria]")
print("Estado : [estado]")
print("-----")

```

Salida esperada en Google Colab:

```

=====
REPORTE DE RENDIMIENTO - SIGE OTI UNSCH
=====
Código : 27150415
Estudiante : Carmen Quispe Huamán
Práctica : 15.0 (peso 30%)
Investigac.: 14.5 (peso 30%)
Examen : 16.0 (peso 40%)

Promedio : 15.25
Categoría : Bueno
Estado : Aprobado
=====

```

Ing. Pelayo Quispe Bautista

6 | Página

Introducción e Instalación python Sesión 01

### 6. SEGUNDO PROBLEMA ABP: Sistema de elegibilidad de beca UNSCH

**Situación problemática 2:**  
La Unidad de Bienestar Universitario de la UNSCH otorga becas de apoyo económico a estudiantes que cumplan simultáneamente tres condiciones: (1) promedio semestral ≥ 14, (2) asistencia ≥ 80% y (3) no tener deuda en tesorería. El proceso manual actual tarda días en revisar cada expediente. Si la U de Bienestar le encargara diseñar el algoritmo que automatice esta verificación.

**Aplica el mismo proceso: análisis → pseudocódigo → diagrama de flujo → Python.**

#### 6.1 Análisis del problema 2

| ENTRADAS                                                                                                                                                                   | PROCESO                                                                                                                              | SALIDAS                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>nombre_estudiante</li> <li>promedio_semestral (0-20)</li> <li>porcentaje_asistencia (0-100)</li> <li>tiene_deuda (SI/NO)</li> </ul> | Verificar 3 condiciones:<br>cond1: promedio >= 14<br>cond2: asistencia >= 80<br>cond3: sin deuda<br>Elegible = cond1 Y cond2 Y cond3 | <ul style="list-style-type: none"> <li>Resultado: ELEGIBLE / NO ELEGIBLE</li> <li>Detalle de condición no cumplida</li> <li>Mensaje para área de Bienestar</li> </ul> |

#### 6.2 Pseudocódigo del problema 2

```

SEUDOCODIGO: Verificación de elegibilidad de beca UNSCH

INICIO
 LEER nombre_estudiante
 LEER promedio_semestral
 LEER porcentaje_asistencia
 LEER tiene_deuda // SI o NO

// Evaluar cada condición por separado
cond_promedio ← (promedio_semestral >= 14)
cond_asistencia ← (porcentaje_asistencia >= 80)
cond_sin_deuda ← (tiene_deuda = NO)

// Condición global: las tres deben ser verdaderas
Si cond_promedio Y cond_asistencia Y cond_sin_deuda ENTONCES
 resultado ← 'ELEGIBLE para beca universitaria'
SINO
 resultado ← 'NO ELEGIBLE para beca universitaria'
FIN_SI

// Mostrar resultado con detalle de condiciones
ESCRIBIR 'Nombre: ', nombre_estudiante
ESCRIBIR 'Promedio: ', promedio_semestral, ' cond_promedio'
ESCRIBIR 'Asistencia: ', porcentaje_asistencia, ' cond_asistencia'
ESCRIBIR 'Sin deuda: ', tiene_deuda, ' cond_sin_deuda'
ESCRIBIR 'Resultado: ', resultado

// Si no es elegible, mostrar motivos
Si NO cond_promedio ENTONCES
 ESCRIBIR 'Motivo: promedio menor a 14'
FIN_SI
Si NO cond_asistencia ENTONCES
 ESCRIBIR 'Motivo: asistencia menor al 80%'
FIN_SI
Si cond_sin_deuda = FALSO ENTONCES
 ESCRIBIR 'Motivo: tiene deuda en Tesorería UNSCH'
FIN_SI
FIN

```

Ing. Pelayo Quispe Bautista

7 | Página

Introducción e Instalación python Sesión 01

### 6.3 Código Python del problema 2

```

CODIGO PYTHON: Verificación de elegibilidad de beca

=====
SISTEMA DE BECAS - BIENESTAR UNIVERSITARIO UNSCH
Verificación automática de elegibilidad
=====
ENTRADAS
nombre_estudiante = 'Pedro Cárdenas López'
promedio_semestral = 15.8 # Promedio del semestre 2024-I
porcentaje_asistencia = 75.0 # Porcentaje de clases asistidas
tiene_deuda = False # False = sin deuda | True = con deuda

PROCESO: Evaluar las tres condiciones
cond_promedio = promedio_semestral >= 14
cond_asistencia = porcentaje_asistencia >= 80
cond_sin_deuda = not tiene_deuda # True si NO tiene deuda

Condición global: las tres deben ser True simultáneamente
es_elegible = cond_promedio and cond_asistencia and cond_sin_deuda

SALIDAS
print("=====")
print(" SISTEMA DE BECAS - BIENESTAR UNIVERSITARIO UNSCH")
print("-----")
print("Estudiante : [nombre_estudiante]")
print("Promedio ≥ 14 : [cond_promedio] ([promedio_semestral])")
print("Asistencia ≥ 80% : [cond_asistencia] ([porcentaje_asistencia]%)")
print("Sin deuda : [cond_sin_deuda] ([cond_sin_deuda])")
print("-----")

if es_elegible:
 print('RESULTADO: ELEGIBLE para beca universitaria')
else:
 print('RESULTADO: NO ELEGIBLE para beca universitaria')
 print('Motivos:')
 if not cond_promedio:
 print('→ Promedio [promedio_semestral] no alcanza el mínimo de 14.0')
 if not cond_asistencia:
 print('→ Asistencia ([porcentaje_asistencia]%) no alcanza el 80%')
 if not cond_sin_deuda:
 print('→ Tiene deuda pendiente en Tesorería UNSCH')
 print("-----")

PRUEBA DE CASOS LÍMITE (el tester ejecuta estos)
print()
print("PRUEBA DE CASOS LÍMITE (====)")
casos = [
 ('Ana García', 17.0, 90.0, False), # ELEGIBLE
 ('Luis Torres', 13.5, 85.0, False), # NO: promedio bajo
 ('Rosa Huamán', 15.0, 70.0, False), # NO: asistencia baja
 ('Juan Contreras', 14.0, 80.0, True), # NO: tiene deuda
 ('Eva Palomino', 14.0, 80.0, False), # ELEGIBLE (valores exactos)
]

for nombre, prom, asist, deuda in casos:
 c1 = prom >= 14
 c2 = asist >= 80
 c3 = not deuda
 resultado = 'ELEGIBLE' if (c1 and c2 and c3) else 'NO ELEGIBLE'

```

Ing. Pelayo Quispe Bautista

8 | Página

Introducción e Instalación



Sesión 01

```
print(f'nombre:<16> | Prom:<prom> | Asist:<asist>% | Deuda:<deuda> --> {resultado}')
```

### 7. EJERCICIOS PROPUESTOS

Resuelven cada ejercicio siguiendo el proceso completo que indica el ASP: primero el análisis de entradas/procesos/salidas, luego el pseudocódigo, después el diagrama de flujo y finalmente el código Python en Google Colab. El documentador registra el proceso y el tester verifica con los casos indicados.

**Ejercicio 1: Clasificador con mensaje motivacional (Nivel básico)**

La OTI necesita que el sistema SIGE, además de clasificar el rendimiento, muestre un mensaje motivacional personalizado para cada categoría. Modifica el algoritmo del Problema Central para agregar los mensajes:

- Sobresaliente: '¡Excelente desempeño! Estás en el tercio superior de tu escuela.'
- Bueno: '¡Buen trabajo! Mantén este ritmo para alcanzar el tercio superior.'
- Aprobado: '¡Aprobado! Dedica más tiempo al estudio para mejorar tu promedio.'
- Desaprobado: '¡Ajá! Adelante a tu docente para recibir apoyo académico.'

**Diseña primero:** Modifica el pseudocódigo y el diagrama de flujo agregando la variable 'mensaje'. Luego tradúcelo a Python.

**Cases de prueba (el tester verifica):** Notas 18/18/18 → Sobresaliente. Notas 14/15/16 → Bueno. Notas 10/10/5/10/5 → Aprobado (caso límite). Notas 9/9/7 → Desaprobado.

**Ejercicio 2: Sistema de alertas tempranas de riesgo académico (Nivel intermedio)**

La Dirección de Escuela quiere detectar a los estudiantes en riesgo académico para derivarlos a tutoría. Un estudiante está en riesgo si cumple al menos UNA de estas condiciones: promedio < 12, asistencia < 70%, o tiene más de 2 cursos desaprobados. Diseña el algoritmo completo (seudocódigo + diagrama + Python) que:

- Lee el nombre, promedio, porcentaje de asistencia y número de cursos desaprobados.
- Evalúa cada condición de riesgo por separado y de forma independiente.
- Determine si el estudiante está en riesgo (al menos una condición verdadera).
- Clasifique el nivel de riesgo: ALTO (2 o más condiciones), MEDIO (1 condición), SIN RIESGO.

**Cases de prueba:** Prom 11, asist 65%, 3 desaprobados → Riesgo ALTO. Prom 13, asist 72%, 1 desaprobado → Riesgo MEDIO. Prom 15, asist 85%, 0 desaprobados → SIN RIESGO.

**Ejercicio 3: Calculadora de carga académica y verificación de matrícula (Nivel avanzado)**

El sistema de matrícula de la UNSCH necesita verificar si un estudiante puede matricularse en la cantidad de créditos seleccionados. Reglas: mínimo 12 créditos por semestre, máximo 22 créditos. Si el promedio anterior es < 12, el máximo se reduce a 18 créditos. Diseña el algoritmo completo que:

- Lee nombre, créditos seleccionados y promedio del semestre anterior.
- Determine el límite máximo según el rendimiento previo.
- Verifique si los créditos están dentro del rango permitido.
- Muestre el estado de la matrícula (APROBADA / RECHAZADA) con el motivo específico.

**Nota adicional:** ¿Cómo extenderías el algoritmo para manejar automáticamente cualquier número de estudiantes sin repetir el código? Documenta tu respuesta en el notebook y concéntala con el problema original de la OTI con 3,000+ registros. Esta reflexión anticipa el contenido de la Sesión 3 (bucles).

Ing. Pelayo Quispe Bautista

9 | Página

Tipos de Datos, Variables y Estructuras de Control Condicionales



Sesión 02

| Tipos de Datos, Variables y Estructuras Condicionales |                                       | SESIÓN 02<br>Semana 4 |                |
|-------------------------------------------------------|---------------------------------------|-----------------------|----------------|
| Asignatura:                                           | Algoritmos y Estructuras de Datos     | Semestre:             | 2024-II        |
| Docente:                                              | Ing. Pelayo Quispe Bautista           | Duración:             | 90 minutos     |
| Metodología:                                          | Aprendizaje Basado en Problemas (ABP) | Fases ABP:            | Fases 2 y 3    |
| Herramientas:                                         | Google Colab, Jupyter Notebook        | Semana:               | Semana 4 de 13 |

### 1. FASE 2: ORGANIZACIÓN DEL APRENDIZAJE

| FASE 2<br>Organización del Aprendizaje | Los equipos retoman el proyecto SIGE iniciado en la Sesión 1 y reciben el nuevo requerimiento de la OTI. Discuten conocimientos previos sobre tipos de datos, identifican brechas y elaboran el plan de investigación. Asignan roles técnicos rotativos: líder, desarrollador, tester y documentador. Diseñan el pseudocódigo del sistema de evaluación ANTES de abrir Google Colab. Competencia desarrollada: PLA — diseño de la estrategia de solución con tipos de datos correctos. |
|----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**1.1 Nuevo requerimiento del proyecto SIGE**

**Situación problemática (Semana 4):**

El módulo de la Sesión 1 ya clasifica el rendimiento de un estudiante. Ahora el jefe de la OTI plantea un requerimiento más exigente: el sistema debe procesar las notas de cualquier estudiante ingresadas desde el teclado; calcular su promedio ponderado (prácticas 30% + investigación 30% + examen final 40%) y clasificar automáticamente su rendimiento en cuatro categorías. Además, el sistema debe convertir la nota a escala internacional de letras (A, B, C, D, F) para los convenios de intercambio académico de la UNSCH.

El operador ingresará las notas como texto desde el formulario web. El sistema debe manejar correctamente este tipo de entrada y evitar que un error de tipo detenga todo el proceso.

**🔗 Tu misión como equipo (Fase 2): Seleccionar los tipos de datos correctos, diseñar el pseudocódigo y planificar la solución antes de codificar.**

### 1.2 Actividad de organización

**Actividad Fase 2**

Antes de investigar el contenido, responden estas preguntas en equipo:

**Pregunta 1:** ¿Qué tipo de dato debe tener la nota de un estudiante? ¿int o float? ¿Por qué?

(Pensar: ¿puede una nota ser 14.75 en el SIGE? ¿qué pasa si usan int?)

**Pregunta 2:** Las notas llegan como texto desde el formulario web (ej: '15.5').

¿Qué ocurrirá si el sistema intenta multiplicar '15.5' \* 0.30 directamente?

**Pregunta 3:** ¿Cuántas condiciones necesita el if-elif-else para clasificar en Sobresaliente / Bueno / Aprobado / Desaprobado?

¿Imports el orden en que se evalúan?

**Pregunta 4:** ¿Qué conoce el equipo sobre operadores aritméticos en Python?

¿Qué diferencia hay entre / y //?

Brechas de aprendizaje identificadas (lo que necesitamos aprender hoy):

.....

Plan del equipo para esta sesión:

- Estudiar tipos de datos y conversión de tipos
- Diseñar el pseudocódigo del sistema de evaluación
- Investigar operadores y condicionales en Python

Ing. Pelayo Quispe Bautista1 | Página

Tipos de Datos, Variables y Estructuras de Control Condicionales



Sesión 02

4. Implementar y verificar en Google Colab

### 2. COMPETENCIAS A DESARROLLAR

| Pensamiento Lógico-Algorítmico (PLA)                                                                                                                                  | Habilidades de Codificación (HC)                                                                                                                                              | Depuración y Prueba (DP)                                                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Diseña el algoritmo de evaluación automática seleccionando los tipos de datos adecuados (int, float, str, bool) y representándolo en pseudocódigo antes de codificar. | Implementa en Python el sistema de evaluación con variables, operadores aritméticos, condicionales if-elif-else y operador ternario para clasificar el rendimiento académico. | Detecta y corrige errores de tipo (TypeError, ValueError) al operar con notas ingresadas como texto; verifica con casos de prueba en los valores límite (10.5, 14, 17). |

### 3. FASE 3: INVESTIGACIÓN Y CONSTRUCCIÓN

| FASE 3<br>Investigación y Construcción | Los equipos investigan de forma autónoma los tipos de datos, operadores y condicionales en Python. Construyen el pseudocódigo y lo traducen a código Python en Google Colab. El tester verifica el sistema con casos de prueba en valores límite. Competencia desarrollada: PLA — implementación del sistema de evaluación automática. |
|----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|----------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 3.1 Tipos de datos en Python

Antes de escribir una sola línea de código, el equipo debe decidir qué tipo de dato usará para representar cada variable del sistema. Elegir el tipo incorrecto genera errores que detienen el programa cuando el operador ingresa una nota. En el SIGE, esta decisión es crítica porque las notas llegan como texto desde el formulario web.

| Tipo  | Descripción                                       | Ejemplo en contexto SIGE                                   | Operaciones clave                                                 |
|-------|---------------------------------------------------|------------------------------------------------------------|-------------------------------------------------------------------|
| int   | Enteros sin decimal. Precisión exacta.            | creditos_aprobados = 80<br>estudiantes_matriculados = 1247 | +, -, *, /, %, ** type() devuelve <class 'int'>                   |
| float | Números con decimal. 15-17 dígitos de precisión.  | promedio_final = 14.75<br>nota_ponderada = 15.5            | +, -, *, /, round(), is_integer() type() devuelve <class 'float'> |
| str   | Texto inmutable entre comillas simples o dobles.  | codigo = '27150419'<br>nombre = 'Carmen Quispe'            | +, *, upper(), split(), len() startswith(), strip()               |
| bool  | Valores True o False. Resultado de comparaciones. | es_aprobado = True<br>tiene_deuda = False                  | and, or, not ==, !=, >, <, >=, <= type() devuelve <class 'bool'>  |

### 3.2 Conversión de tipos (casting)

Cuando el operador ingresa una nota en el formulario web del SIGE, Python la recibe como texto (str). Si el sistema intenta multiplicar ese texto por 0.30 sin convertirlo primero, el programa se detiene con un TypeError. La conversión de tipos (casting) es el mecanismo que resuelve este problema.

**Problema de tipos y su solución**

```
=====
PROBLEMA: Las notas llegan como texto desde el formulario web
=====
Simulando lo que el operador escribe en el formulario
nota_como_texto = '15.5' # Python lo recibe como str

SIN conversión el sistema se detiene con TypeError
resultado = nota_como_texto * 0.30
TypeError: can't multiply sequence by non-int of type 'float'

CON conversión correcto
nota_como_numero = float(nota_como_texto) # str '15.5' → float 15.5
resultado = nota_como_numero * 0.30 # 4.65
```

Ing. Pelayo Quispe Bautista

2 | Página

Tipos de Datos, Variables y Estructuras de Control Condicionales



Sesión 02

```
print(f'Texto original : {nota_como_texto} | tipo: {type(nota_como_texto)}')
print(f'Numero : {nota_como_numero} | tipo: {type(nota_como_numero)}')
print(f'Resultado : {resultado}')
```

# Conversiones útiles en el sistema SIGE

```
creditos_int = 4
creditos_float = int(creditos_int) # str '4' → int 4
promedio_float = 14.75
promedio_str = str(promedio_float) # float 14.75 → str '14.75'
codigo_int = 27150415
codigo_str = str(codigo_int) # int → str (para concatenar)

print(f'Creditos como entero : {creditos_int}')
print(f'Promedio como texto : {promedio_str}')
print(f'Código como texto : {codigo_str}')
```

### 3.3 Variables y operadores aritméticos

Una variable es un contenedor con nombre que almacena un valor. Python usa tipado dinámico: el tipo de la variable se determina automáticamente según el valor asignado y puede cambiar. Para el cálculo del promedio ponderado del SIGE, los operadores aritméticos y el orden de evaluación (PEMDAS) son fundamentales.

**Variables y operadores**

```
=====
VARIABLES DEL SISTEMA SIGE : Tipado dinámico en Python
=====
codigo_estudiante = '27150419' # str — código universitario
nombre_estudiante = 'Carmen Quispe Huamani' # str
nota_practica = 15.0 # float — peso: 30%
nota_investigacion = 14.5 # float — peso: 30%
nota_examen = 18.0 # float — peso: 40%
creditos_curso = 4 # int — créditos del curso IS-402
es_estudiante_activo = True # bool — estado en el SIGE

Operadores aritméticos y su precedencia (PEMDAS)
** (potencia) > * / (multiplicación y división) > + - (suma/resta)

Cálculo correcto del promedio ponderado
Python evalúa primero las multiplicaciones, luego las sumas
promedio = (nota_practica * 0.30) + (\n (nota_investigacion * 0.30) + (\n (nota_examen * 0.40)

Operadores de comparación — devuelven bool
es_aprobado = promedio >= 10.5 # True o False
en_tercio_sup = promedio >= 14.0 # True o False
tiene_nota_maxima = promedio == 20.0 # True o False

Operadores de división
total_notas = 3
promedio_simple = (15.0 + 14.5 + 16.0) / total_notas # 15.17 (float)
division_entera = (15 + 14 + 16) // total_notas # 15 (int)
residuo = 15 % total_notas # 0

print(f'Promedio ponderado: {promedio:2f}')
print(f'Aprobado : {es_aprobado}')
print(f'División float : {promedio_simple:2f}')
```

Ing. Pelayo Quispe Bautista

3 | Página

```
print('Division entera : {division_entera}')
```

### 3.4 Estructuras condicionales

Las estructuras condicionales permiten que el algoritmo tome decisiones basadas en el valor del promedio. En el SIGE necesitamos cuatro categorías de rendimiento, lo que requiere un if-elif-else con tres condiciones intermedias. El orden en que se evalúan las condiciones es crítico: siempre de mayor a menor para evitar clasificaciones incorrectas.

| Estructura        | Cuando usarla                                                            | Aplicación en el SIGE                                                         |
|-------------------|--------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| if                | Cuando solo existe una condición posible y si no se cumple no hay acción | Si la nota es mayor a 0 se registra. Si no, se ignora.                        |
| if-else           | Cuando hay exactamente dos caminos posibles: se cumple o no se cumple.   | Aprobado / Desaprobado. Solo dos posibilidades para el SIGE.                  |
| if-elif-else      | Cuando hay tres o más categorías mutuamente excluyentes.                 | Sobresaliente / Bueno / Aprobado / Desaprobado. Cuatro categorías en el SIGE. |
| Operador ternario | Para asignaciones simples en una sola línea. Solo dos opciones.          | estado = 'Aprobado' if promedio >= 10.5 else 'Desaprobado'                    |

### if-elif-else

```
=====
CLASIFICACION POR RENDIMIENTO — Escala vigesimal UNSCH
Importante: evaluar de MAYOR a MENOR para evitar errores
=====
promedio = 14.75 # resultado del calculo ponderado
Clasificación con 4 categorías (if-elif-else)
categoria = 'Sobresaliente'
estado = 'Aprobado'
elif promedio >= 14:
 categoria = 'Bueno'
 estado = 'Aprobado'
elif promedio >= 10.5:
 categoria = 'Aprobado'
 estado = 'Aprobado'
else:
 categoria = 'Desaprobado'
 estado = 'Desaprobado'
```

```
Por que el orden importa:
Si evaluáramos promedio >= 10.5 PRIMERO, un promedio de 17
sería clasificado como 'Aprobado' en lugar de 'Sobresaliente'
porque 17 >= 10.5 es True y el if-elif se detiene ahí.
```

```
Operador ternario para asignaciones simples
estado_compacto = 'Aprobado' if promedio >= 10.5 else 'Desaprobado'
```

```
Escala internacional de letras para convenios
if promedio >= 17:
 letra = 'A'
```

```
elif promedio >= 14:
 letra = 'B'
elif promedio >= 11:
 letra = 'C'
elif promedio >= 10.5:
```

Ing. Pelayo Quique Bautista

4 | Página

```
letra = 'D'
else:
 letra = 'F'

print(Promedio : {promedio})
print(Categoria : {categoria})
print(Estado : {estado})
print(Letra : {letra})
```

### 3.5 Operadores lógicos

Los operadores lógicos (and, or, not) permiten combinar múltiples condiciones en una sola expresión. En el SIGE son útiles para verificar si un estudiante cumple varios criterios simultáneamente, como en el sistema de alertas tempranas o la elegibilidad para becas.

#### Operadores lógicos

```
=====
OPERADORES LOGICOS — and, or, not
=====
promedio = 14.75
porcentaje_asistencia = 85.0
tiene_deuda = False

AND - todas las condiciones deben ser True
elegible_beca = (promedio >= 14) and (porcentaje_asistencia >= 80) and (not tiene_deuda)
print('Elegible para beca : {elegible_beca}')

OR - al menos una condición debe ser True
requiere_atencion = (promedio < 12) or (porcentaje_asistencia < 70)
print('Requiere atención : {requiere_atencion}')

NOT - invierte el valor booleano
puede_matricularse = not tiene_deuda
print('Puede matricularse : {puede_matricularse}')

Condiciones anidadas — nivel de riesgo académico
cond_promedio = promedio < 12
cond_asistencia = porcentaje_asistencia < 70

if cond_promedio and cond_asistencia:
 nivel_riesgo = 'ALTO' — derivar a tutoría urgente'
elif cond_promedio or cond_asistencia:
 nivel_riesgo = 'MEDIO' — seguimiento semanal'
else:
 nivel_riesgo = 'SIN RIESGO'

print(Nivel de riesgo : {nivel_riesgo})
```

### 4. DISEÑO DEL ALGORITMO

Con el contenido de la Fase 3 comprendido, el equipo retoma el pseudocódigo iniciado en la Fase 2 y lo completa. El desarrollador escribe el pseudocódigo; el tester identifica los casos de prueba que deben funcionar; el documentador registra las decisiones de diseño.

#### SEUDOCÓDIGO: Sistema de evaluación académica SIGE (Fase 2 completada)

```
INICIO
// ENTRADAS: con conversión de tipos
LEER codigo_estudiante // texto: '27150415'
LEER nombre_estudiante // texto: 'Carmen Quique'
```

Ing. Pelayo Quique Bautista

5 | Página

```
LEER nota_practica_bt // texto del formulario: '15.0'
LEER nota_investigacion_bt // texto del formulario: '14.5'
LEER nota_examen_bt // texto del formulario: '16.0'

// Convertir texto a numero (casting)
nota_practica = CONVERTIR_FLOAT(nota_practica_bt)
nota_investigacion = CONVERTIR_FLOAT(nota_investigacion_bt)
nota_examen = CONVERTIR_FLOAT(nota_examen_bt)
```

```
// PROCESO 1: Promedio ponderado
promedio = (nota_practica * 0.30) +
(nota_investigacion * 0.30) +
(nota_examen * 0.40)
```

```
// PROCESO 2: Clasificación por rendimiento
Si promedio >= 17 ENTONCES
 categoria <- 'Sobresaliente' | letra <- 'A'
SINO Si promedio >= 14 ENTONCES
 categoria <- 'Bueno' | letra <- 'B'
SINO Si promedio >= 10.5 ENTONCES
 categoria <- 'Aprobado' | letra <- 'C' o 'D'
SINO
 categoria <- 'Desaprobado' | letra <- 'F'
FIN_SI
```

```
// PROCESO 3: Estado académico
Si promedio >= 10.5 ENTONCES
 estado <- 'Aprobado'
SINO
 estado <- 'Desaprobado'
FIN_SI
```

```
// SALIDAS: Reporte completo
ESCRIBIR reporte con: código, nombre, notas, promedio, categoría, letra, estado
FIN
```

### 5. PROBLEMA ABP

#### Situación problemática:

La Oficina de Bienestar Universitario administra las plazas de residencia estudiantil. El puntaje de admisión se calcula con la fórmula: Puntaje = (promedio semestral x 3) + (50 puntos si el distrito de origen no es Huamanga) - (ingreso familiar / 100 x 5). El estudiante accede a la residencia si su puntaje es mayor o igual a 70. El proceso manual genera errores al calcular puntajes con decimales. Tu equipo debe diseñar y codificar el algoritmo que automatice este proceso.

#### Seudocódigo del Problema

##### SEUDOCÓDIGO: Sistema de residencia estudiantil UNSCH

```
INICIO
LEER nombre_estudiante
LEER promedio_semestral // float: promedio del semestre anterior
LEER distrito_origen // texto: 'Huamanga' u otro
LEER ingreso_familiar_mensual // float: ingresos en soles
// Determinar bono por origen
Si distrito_origen != 'Huamanga' ENTONCES
 bono_origen <- 50
SINO
```

Ing. Pelayo Quique Bautista

6 | Página

```
bono_origen <- 0
FIN_SI

// Calcular descuento por ingreso familiar
descuento_ingreso = (ingreso_familiar_mensual / 100) * 5
// Calcular puntaje total
puntaje = (promedio_semestral * 3) + bono_origen - descuento_ingreso
// Determinar elegibilidad
Si puntaje >= 70 ENTONCES
 resultado <- 'ADMITIDO a residencia'
SINO
 resultado <- 'NO ADMITIDO'
 diferencia = 70 - puntaje
 ESCRIBIR 'Le faltan', diferencia, 'puntos para la admisión'
FIN_SI

ESCRIBIR reporte completo del puntaje y resultado
FIN
```

#### Código Python del Problema

##### CÓDIGO PYTHON: Sistema de residencia estudiantil UNSCH (Problema 2 resuelto)

```
SISTEMA DE RESIDENCIA ESTUDIANTIL BIENESTAR UNSCH
// Puntaje = (promedio x 3) + bono_origen - descuento_ingreso
// Admitido si puntaje >= 70
=====

ENTRADAS
nombre_estudiante = 'Roberto Quique Palomino'
promedio_semestral = 15.8 # promedio del semestre anterior
distrito_origen = 'Cangallo' # no es Huamanga — bono 50
ingreso_familiar_mensual = 800.0 # ingresos en soles

PROCESO: Cálculos del puntaje
Bono por origen provincial
bono_origen = 50 if distrito_origen != 'Huamanga' else 0

Descuento según ingreso familiar
descuento_ingreso = (ingreso_familiar_mensual / 100) * 5

// Puntaje total
puntaje = (promedio_semestral * 3) + bono_origen - descuento_ingreso

PROCESO: Determinación de elegibilidad
PUNTAJE_MINIMO = 70

if puntaje >= PUNTAJE_MINIMO:
 resultado = 'ADMITIDO a residencia universitaria'
 diferencia = puntaje - PUNTAJE_MINIMO
 detalle = f'Supera el mínimo por {diferencia:.2f} puntos'
else:
 resultado = 'NO ADMITIDO a residencia universitaria'
 diferencia = PUNTAJE_MINIMO - puntaje
 detalle = f'Faltan {diferencia:.2f} puntos para la admisión'

SALIDAS
```

Ing. Pelayo Quique Bautista

7 | Página

Tipos de Datos, Variables y Estructuras de Control Condicionales

Sesión 02

```
print("==" * 55)
print(" SISTEMA DE RESIDENCIA — BIENESTAR UNSCH")
print("==" * 55)
print(f"Estudiante : {nombre_estudiante}")
print(f"Promedio semestral : {promedio_semestral}")
print(f"Distrito origen : {distrito_origen}")
print(f"Ingreso familiar : $ {ingreso_familiar_mensual:2f}")
print("==" * 55)
print(f"Componente academica : {promedio_semestral * 3:2f} (promedio x 3)")
print(f"Bono origen : {bono_origen:2f}")
print(f"Descuento ingreso : {descuento_ingreso:2f}")
print(f"PUNTAJE TOTAL : {puntaje:2f}")
print("==" * 55)
print(f"Resultado : {resultado}")
print(f"Detalle : {detalle}")
print("==" * 55)

PRUEBA DE CASOS DEL TESTER
print()
print("==== CASOS DE PRUEBA ===")
casos_residencia = [
 ("Mara Flores", 17.0, "Vicos", 400), # Alto puntaje
 ("Pablo Castro", 12.0, "Huamanga", 600), # Sin bono, ingreso medio
 ("Luis Coarale", 10.5, "Pauca", 200), # Prom bajo, bono + ingreso bajo
 ("Ana Sulca", 15.0, "Huamanga", 500), # Sin bono, ingreso alto
]

for nom, prom, dist, ing in casos_residencia:
 b = 50 if dist != "Huamanga" else 0
 d = (ing / 1000) * 5
 p = (prom * 3) + b - d
 res = "ADMITIDO" if p >= 70 else "NO ADMITIDO"
 print(f"nom=<14, puntaje=<8,2f) -> {res}")
```

6. EJERCICIOS PROPUESTOS

Resuelvan cada ejercicio siguiendo el proceso ABP: primero el análisis de entradas/procesos/salidas, luego el pseudocódigo con los tipos de datos correctos y finalmente el código Python en Google Colab. El tester verifica con los casos indicados.

**Ejercicio 1: Calculadora de nota final con retroalimentación (Nivel básico)**

Un docente de la UNSCH necesita un sistema que calcule la nota final de un estudiante con los siguientes pesos: exámenes parciales 40% (promedio de dos parciales), prácticas 30% y proyecto final 30%. El sistema debe mostrar que componente le bajo más la nota y sugerir en cuanto debe mejorar el examen final para aprobar.

- **Diseña primero:** Pseudocódigo con los tipos de datos de cada variable y la lógica para identificar el componente más bajo.
- **Implementa:** Código Python con if-elif-else para el mensaje de retroalimentación.

**Caso del tester:** Parcial1=12, Parcial2=10, Practicas=15, Proyecto=14. Resultado esperado: promedios parciales=11, nota final=12.5, estado=Aprobado. Probar también con parciales bajos (8 y 9) para verificar el mensaje de mejora.

**Ejercicio 2 : Sistema de alertas de créditos académicos UNSCH (Nivel intermedio)**

La oficina de registros académicos necesita verificar el avance curricular de los estudiantes. Un estudiante está en alerta si cumple alguna de estas condiciones: créditos aprobados menores al 60% del total del plan, promedio acumulado menor a 11, o tiene más de 3 cursos desaprobados en el historial. Diseña y codifica el sistema que:

1. Lea el nombre, créditos aprobados, créditos totales del plan, promedio acumulado y cursos desaprobados.
2. Calcule el porcentaje de avance curricular (creditos\_aprobados / creditos\_totales \* 100).
3. Evalúa cada condición de alerta por separado usando operadores lógicos.
4. Clasifique el nivel de alerta: CRITICO (0 o más condiciones), MODERADO (1 condición), NORMAL.
5. Muestre un reporte con el porcentaje de avance y el nivel de alerta.

Ing. Pelayo Quispe Bautista

8 | Página

Tipos de Datos, Variables y Estructuras de Control Condicionales

Sesión 02

**Casos del tester:** Créditos 80/200, promedio 10.5, 4 desaprobados → CRITICO. Créditos 120/200, promedio 12, 2 desaprobados → MODERADO. Créditos 160/200, promedio 14, 1 desaprobado → NORMAL.

**Ejercicio 3: Simulador de matrícula con restricciones múltiples (Nivel avanzado)**

El sistema de matrícula de la UNSCH aplica reglas múltiples que deben verificarse en orden. Un estudiante puede matricularse si: (1) no tiene deuda en la escuela, (2) tiene un promedio acumulado de al menos 11, y (3) los créditos que desea matricular están en el rango permitido (mín 12, máx 22, o máx 18 si su promedio acumulado es menor a 11). El sistema debe procesar estas reglas en orden de prioridad y detenerse en la primera que falle. Diseña y codifica:

6. Verificación de deuda (primera prioridad — si tiene deuda, no se evalúa nada más).
7. Verificación del promedio mínimo (segunda prioridad).
8. Cálculo del máximo de créditos según rendimiento.
9. Verificación del rango de créditos solicitados.
10. Reporte con el estado de matrícula, el motivo en caso de rechazo y el rango de créditos disponible.

**Reto del tester:** Diseña al menos 6 casos de prueba que cubran todas las combinaciones posibles de reglas. Documenta cada caso con el resultado esperado antes de ejecutarlo. Esta práctica de diseño de casos de prueba es la base de la Fase 4 del ABP y de la competencia DP que desarrollarán en sesiones posteriores.

Estructuras de Control: Bucles

Sesión 03

| Estructuras de Control: Bucles                    |                                       | SESION 03  |                |
|---------------------------------------------------|---------------------------------------|------------|----------------|
| Cálculo Iterativo y Detección de Riesgo Académico |                                       | Semana 3   |                |
| Asignatura:                                       | Algoritmos y Estructuras de Datos     | Semestre:  | 2024-II        |
| Docente:                                          | Ing. Pelayo Quispe Bautista           | Duración:  | 90 minutos     |
| Metodología:                                      | Aprendizaje Basado en Problemas (ABP) | Fases ABP: | Fases 3 y 4    |
| Herramientas:                                     | Google Colab, Visual Studio Code      | Semana:    | Semana 5 de 13 |

**1. FASES 3 Y 4: INVESTIGACIÓN, CONSTRUCCIÓN Y PRESENTACIÓN**

|                                               |                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FASE 3</b><br>Investigación y Construcción | Los equipos investigan autónomamente los bucles for, while, range() y funciones de control (break, continue). Construyen el módulo de cálculo iterativo de promedios del SIGE consultando la documentación oficial de Python. El desarrollador lidera la codificación; el tester diseña los casos de prueba antes de ejecutar. Competencia desarrollada: HC - implementación del sistema iterativo de promedios SIGE.    |
| <b>FASE 4</b><br>Desarrollo y Presentación    | Los equipos codifican la solución completa en VS Code o Google Colab y verifican con casos de prueba. El tester valida el sistema con casos límite: lista vacía, un estudiante, nota exactamente 10.5, nota 0 y nota 20. El equipo presenta el módulo funcional ante sus pares y documenta los errores encontrados. Competencia desarrollada: DP - detección de bucles infinitos, IndexError y validación de resultados. |

**1.1 Nuevo requerimiento del proyecto SIGE**

**Situación problemática:**

El sistema SIGE ya puede procesar las notas de un estudiante individualmente (Sesión 2). Sin embargo, el jefe de la OTI señala que cada sección de Ingeniería de Sistemas tiene entre 40 y 50 estudiantes. Procesar cada uno manualmente es inviable y contradice el objetivo de automatización. El nuevo requerimiento es claro:

"El sistema debe recorrer automáticamente la lista de estudiantes de una sección, calcular el promedio ponderado de cada uno, detectar quienes están en riesgo académico (promedio < 10.5), generar estadísticas de la sección (promedio general, porcentaje de aprobados y desaprobados) y mostrar un reporte completo. El proceso no puede detenerse aunque encuentre una nota faltante".

Esta sesión además conecta con un hallazgo de la Sesión 2: la limitación de procesar solo un estudiante. Los bucles son la respuesta a ese problema identificado por el equipo.

🔗 Tu misión como equipo (Fases 3 y 4): investigar for y while, construir el módulo iterativo y validarlo con la sección completa IS-101.

**Actividad Fases 3-4: Preguntas de investigación del equipo**

Antes de investigar el contenido, el líder plantea estas preguntas al equipo:

**Pregunta 1:** ¿Que estructura de repetición usarían para recorrer una lista de 45 estudiantes? ¿for o while? ¿Por qué?

**Pregunta 2:** ¿Que pasaría si intentan acceder a notas[50] en una lista de 45 elementos? ¿Qué tipo de error generaría?

**Pregunta 3:** ¿Como determinarían el bucle si encuentran un código de estudiante inválido?

**Pregunta 4:** ¿Como calcularían el promedio general de toda la sección usando un bucle? ¿Qué variables necesitarían acumular?

**Pregunta 5:** Si la lista de estudiantes estuviera vacía, ¿qué errores podrían ocurrir al intentar calcular el promedio general? ¿Como los previenen?

**2. COMPETENCIAS A DESARROLLAR**

|                                      |                                  |                          |
|--------------------------------------|----------------------------------|--------------------------|
| Pensamiento Lógico-Algorítmico (PLA) | Habilidades de Codificación (HC) | Depuración y Prueba (DP) |
|--------------------------------------|----------------------------------|--------------------------|

Ing. Pelayo Quispe Bautista

1 | Página

Estructuras de Control: Bucles

Sesión 03

|                                                                                                                                                   |                                                                                                                                                                     |                                                                                                                                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Diseña algoritmos iterativos para procesar colecciones de datos del SIGE, seleccionando for o while según el tipo de problema antes de codificar. | Implementa en Python bucles for con range() y enumerate(), bucle while con condición de parada, break, continue y bucles anidados para el sistema de reportes SIGE. | Detecta y corrige bucles infinitos, IndexError y condiciones de parada incorrectas, verifica el sistema con casos límite: lista vacía, un elemento, notas 0 y 20. |
|---------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**3. CONTENIDO CONCEPTUAL: Bucles en Python**

**3.1 ¿Por qué los bucles son la solución al problema del SIGE?**

En la Sesión 2 el equipo identificó una limitación crítica: el sistema solo procesa un estudiante a la vez. Para automatizar el SIGE con 3,000+ registros, se necesita una estructura que repita automáticamente el mismo proceso para cada elemento de una colección. Esa estructura es el bucle (loop).

Un bucle es una estructura de control que ejecuta un bloque de código repetidamente mientras se cumple una condición o hasta que se hayan procesado todos los elementos de una colección. Sin bucles, el programa tendría que copiar el mismo código 3,000 veces - exactamente el problema que el jefe de la OTI quiere resolver.

| Característica     | Bucle for                                                                                             | Bucle while                                                                                                 |
|--------------------|-------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| Cuando usarlo      | Cuando se conoce de antemano la cantidad de iteraciones o se recorre una colección de elementos.      | Cuando la cantidad de iteraciones depende de una condición que puede cambiar durante la ejecución.          |
| Aplicación en SIGE | Recorrer lista de 45 estudiantes, calcular promedios semestrales, analizar tendencias entre periodos. | Validar código universitario ingresado, controlar menú de atención, solicitar datos hasta que sean válidos. |
| Riesgo principal   | IndexError si se accede a una posición fuera del rango de la lista.                                   | Bucle infinito si la condición de parada nunca se cumple o se actualiza mal.                                |
| Herramientas clave | range(), enumerate(), zip(), len()                                                                    | Flag booleano, break, contador de intentos                                                                  |

**3.2 Bucle for: Recorrido de colecciones del SIGE**

El bucle for de Python recorre cada elemento de una secuencia (lista, string, range, dict) ejecutando el bloque de código para cada uno. No requiere gestionar manualmente el índice, lo que reduce errores. Es la estructura principal para procesar los registros del SIGE.

**Bucle for - Fundamentos con colecciones del SIGE**

```
=====
BUCLE FOR: Recorrer colecciones del SIGE
=====
Ejemplo 1: Recorrer lista de estudiantes
estudiantes = ["Carmen Quispe", "Pietro Cardenas", "Marta Huaman", "Jorge Coarale", "Ana Palomino"]

print(f"Lista de estudiantes de la sección IS-101:
for estudiante in estudiantes:
 print(f" - {estudiante}")

Ejemplo 2: enumerate() — obtener índice y valor simultáneamente
Usá para mostrar números de orden en el reporte del SIGE
print()
print(f"Número con número de orden:
for numero, estudiante in enumerate(estudiantes, start=1):
 print(f" {numero} - {estudiante}")

Ejemplo 3: range() — generar secuencias numéricas
print()
print(f"Semestres analizadas:
for semestre in range(1, 11): # del 1 al 10 inclusive
 print(f" Semestre {semestre}, end="")
print() # salto de línea al final
```

Ing. Pelayo Quispe Bautista

2 | Página



```
Ejemplo 4: range() con paso — semestres pares únicamente
print(Semestres pares)
for semestre in range(2, 11, 2): # inicio=2, fin=10, paso=2
 print(Semestre {semestre}, end=" ")
print()
```

### 3.3 Función range():

La función range(inicio, fin, paso) genera una secuencia de enteros que no incluye el valor final. Es fundamental para acceder a listas por índice, generar números de registro y analizar tendencias entre periodos académicos del SIGE.

#### range() — Análisis de tendencias semestrales UNSCH

```
=====
range() — Análisis de tendencias de promedios semestrales
=====
Promedios de la sección IS-101 en los últimos 10 semestres
promedios_historicos = [13.2, 13.8, 14.1, 13.5, 14.7,
 15.0, 14.8, 15.3, 15.1, 15.6]

print("=== TENDENCIA DE PROMEDIOS — IS-101 UNSCH ===")
print("Periodos analizados: {len(promedios_historicos)} semestres")
print()

Comparar cada semestre con el anterior usando range()
for i in range(1, len(promedios_historicos)):
 actual = promedios_historicos[i]
 anterior = promedios_historicos[i - 1]
 diff = actual - anterior
 if diff > 0:
 tendencia = f"subio (diff: {diff}) pts"
 elif diff < 0:
 tendencia = f"bajo (abs(diff): {diff}) pts"
 else:
 tendencia = "sin cambio"
 print(f"Semestre {i+1}: {actual:.1f} {tendencia}")

Calcular el promedio general del historial
suma_total = 0
for promedio in promedios_historicos:
 suma_total += promedio
promedio_general = suma_total / len(promedios_historicos)
print(f"Promedio historico: {promedio_general:.2f}")
```

### 3.4 Bucle while

El bucle while ejecuta su bloque mientras una condición sea verdadera. Es la elección correcta cuando el número de iteraciones depende de la respuesta del usuario o de una condición dinámica. En el SIGE se usa para validar entradas: el sistema no avanza hasta que el operador ingrese un código de estudiante con el formato correcto.

#### Bucle while: Validación de código universitario UNSCH

```
=====
BUCLE WHILE: Validación de entradas en el SIGE
El sistema no avanza hasta recibir un código válido
=====
Formato válido UNSCH empieza con '27' y tiene 8 dígitos
print("=== SISTEMA SIGE — INGRESO DE CODIGO UNIVERSITARIO ===")

codigo_valido = False # flag de control del bucle
```

Ing. Pelayo Quispe Bautista

3 | Página

```
while not codigo_valido:
 codigo = input("Ingrese su código UNSCH (formato: 27XXXXXX): ")

 # Validar formato: 8 dígitos, empieza con '27'
 if len(codigo) == 8 and codigo.startswith('27') and codigo[2:].isdigit():
 codigo_valido = True
 print("Código {codigo} registrado correctamente.")
 print("¡Bienvenido al SIGE — UNSCH!")
 else:
 print("Formato incorrecto: {codigo}")
 print("El código debe tener 8 dígitos y comenzar con 27.")
 print("Ejemplo: 27150415")

While con contador de intentos (para seguridad)
print()
print("=== VERSION CON LIMITE DE INTENTOS ===")
MAX_INTENTOS = 3
intentos = 0
codigo_valido = False

while not codigo_valido and intentos < MAX_INTENTOS:
 codigo = input("Código UNSCH (Intento {intentos}/{MAX_INTENTOS}): ")
 intentos += 1
 if len(codigo) == 8 and codigo.startswith('27') and codigo[2:].isdigit():
 codigo_valido = True
 print("Acceso concedido al SIGE.")
 else:
 print("Código incorrecto.")

if not codigo_valido:
 print("Acceso bloqueado. Contacte al administrador del SIGE.")
```

### 3.5 Instrucciones de control: break, continue, pass

Dentro de un bucle, estas instrucciones modifican el flujo de ejecución. En el SIGE son especialmente útiles para manejar datos irregulares: cursos convalidados que no deben sumarse al promedio, o registros incompletos que deben omitirse sin detener el proceso.

| Instrucción     | Que hace                                                         | Uso en el SIGE                                                              |
|-----------------|------------------------------------------------------------------|-----------------------------------------------------------------------------|
| <b>break</b>    | Salte inmediatamente del bucle al cumplirse la condición.        | Detener la atención en caja al alcanzar el límite de estudiantes del turno. |
| <b>continue</b> | Salta a la siguiente iteración sin ejecutar el resto del bloque. | Omitir cursos con estado 'Libre' o 'Convalidado' al procesar el kardex.     |
| <b>pass</b>     | No hace nada. Marcador de posición para código futuro.           | Reservar espacio para módulos del SIGE aun no implementados.                |

#### break / continue / pass: Procesamiento de datos irregulares en el SIGE

```
=====
CONTROL DE FLUJO EN BUCLES — Datos irregulares del SIGE
=====
Lista de cursos del kardex con diferentes estados
cursos_kardex = [
 (nombre: 'Algoritmos', nota: 15.0, estado: 'cursado'),
 (nombre: 'Matemáticas', nota: 13.5, estado: 'cursado'),
 (nombre: 'Libre 1', nota: 0, estado: 'libre'),
 (nombre: 'Base de Datos', nota: 16.0, estado: 'cursado'),
]
```

Ing. Pelayo Quispe Bautista

4 | Página

```
{(nombre: 'Riesgos', nota: 0, estado: 'pendiente'),
 (nombre: 'POO', nota: 14.5, estado: 'cursado'),
}

CONTINUE: omitir cursos que no cuentan para el promedio
suma_notas = 0
total_cursos = 0

print("Cursos incluidos en el cálculo del promedio:")
for curso in cursos_kardex:
 if curso['estado'] != 'cursado':
 continue # Salta al siguiente curso sin procesar este
 suma_notas += curso['nota']
 total_cursos += 1
 print(f" {curso['nombre']}: {curso['nota']}")

if total_cursos > 0:
 promedio = suma_notas / total_cursos
 print(f"Promedio calculado: {promedio:.2f} ({total_cursos} cursos)")

BREAK: detener proceso al encontrar curso reprobado crítico
print()
print("Verificación de prerequisitos para matricular:")
prerequisito_bloqueante = "Matemáticas"

for curso in cursos_kardex:
 if curso['nombre'] == prerequisito_bloqueante and curso['nota'] < 10.5:
 print("BLOQUEADO: {curso['nombre']} no aprobado ({curso['nota']})")
 print("No puede matricularse en cursos de ciclos superiores.")
 break # detiene la verificación
 print(f" OK: {curso['nombre']} ({curso['nota']})")
```

### 3.6 Bucles anidados: Reportes multi-sección del SIGE

Los bucles anidados (un bucle dentro de otro) permiten procesar datos con dos dimensiones: por cada sección (bucle exterior), recorrer cada estudiante (bucle interior). Cada iteración del bucle exterior ejecuta el interior completo. Son fundamentales para generar reportes comparativos entre secciones del SIGE.

#### Bucles anidados: Reporte multi-sección SIGE UNSCH

```
=====
BUCLES ANIDADOS: Reporte multi-sección del SIGE
=====
Datos de dos secciones de Algoritmos y Estructuras de Datos
secciones = [
 ('IS-101 (maternal)', [15.0, 12.5, 17.0, 9.0, 14.5, 11.0, 16.5, 8.5]),
 ('IS-102 (bardo)', [13.0, 14.5, 11.5, 16.0, 10.0, 15.5, 12.0, 17.5])
]

print("=== REPORTE COMPARATIVO POR SECCIÓN — SIGE UNSCH ===")
print("===")

Bucle exterior: recorre cada sección
for nombre_seccion, notas in secciones.items():
 suma = 0
 aprobados = 0
 riesgo = []

 # Bucle interior: procesa cada estudiante de la sección
```

Ing. Pelayo Quispe Bautista

5 | Página

```
for i, nota in enumerate(notas, start=1):
 suma += nota
 if nota == 10.5:
 aprobados += 1
 else:
 riesgo.append(f"Est. {i} ({nota})")

Estadísticas de la sección
total = len(notas)
promedio = suma / total if total > 0 else 0
pct_apr = (aprobados / total) * 100 if total > 0 else 0

print(f"Sección: {nombre_seccion}")
print(f"Promedio general: {promedio:.2f}")
print(f" Aprobados: {aprobados}/{total} ({pct_apr:.1f}%)")
if riesgo:
 print(f" En riesgo: {len(riesgo)}")
else:
 print(f" En riesgo: {ninguno}")
print(f"{'-' * 50}")
```

### 4. ERRORES COMUNES EN BUCLES

La Fase 4 exige que el tester verifique el sistema con casos extremos. Antes de ejecutar las pruebas, el equipo debe conocer los errores más frecuentes en bucles para poder identificarlos rápidamente al aparecer en la consola.

#### ERROR 1: IndexError: list index out of range

```
INCORRECTO: acceder por índice fuera del rango
notas = [15, 12, 17, 9, 14]
for i in range(6): # range(6) genera 0,1,2,3,4,5 — pero notas solo tiene índices 0-4
 print(notas[i]) # cuando i=5 → IndexError

CORRECTO: usar len() para limitar al tamaño real de la lista
for i in range(len(notas)): # range(5) genera 0,1,2,3,4
 print(notas[i])
```

```
AUN MEJOR: iterar directamente sobre la lista
for nota in notas:
 print(nota)
```

#### ERROR 2: condición de parada que nunca se cumple

```
INCORRECTO: la variable 'intentos' nunca cambia dentro del bucle
intentos = 0
while intentos < 3:
 codigo = input("Ingrese código: ")
 # Se olvidó incrementar intentos — bucle infinito

CORRECTO: actualizar la variable de control en cada iteración
intentos = 0
while intentos < 3:
 codigo = input("Ingrese código: ")
 intentos += 1 # fundamental: actualizar el contador
```

#### ERROR 3: ZeroDivisionError al calcular promedio de lista vacía

Ing. Pelayo Quispe Bautista

6 | Página

Estructuras de Control: Bucles



Sesión 03

```
INCORRECTO: no verificar si la lista tiene elementos
notas = [] # lista vacía — puede ocurrir si el grupo no tiene estudiantes
promedio = sum(notas) / len(notas) # ZeroDivisionError: division by zero

CORRECTO: verificar antes de dividir
if len(notas) > 0:
 promedio = sum(notas) / len(notas)
 print(Promedio: {promedio: 2f})
else:
 print(No hay notas registradas para esta sección.)
```

5. PROBLEMA ABP RESUELTO

**Situación problemática 2:**  
El docente de la sección IS-101 necesita registrar la asistencia de los estudiantes clase por clase. El sistema debe mostrar la lista de estudiantes, permitir marcar la asistencia de cada uno (Presente, Ausente, Tardanza) y al final calcular el porcentaje de asistencia de cada estudiante. Si un estudiante tiene menos del 70% de asistencia, debe aparecer en la lista de alerta. El operador puede seguir ingresando datos hasta que escriba 'fin'.

**CÓDIGO PYTHON: Sistema de control de asistencia con while**

```
=====
SISTEMA DE ASISTENCIA — SIGE UNSCH | Sesión 3
Control de asistencia con bucle while + estadísticas
=====

Lista de la sección IS-101
estudiantes = ['Carmen Quispe', 'Pedro Cordonas', 'Maria Huaman',
 'Jorge Cente', 'Ana Palomino']
asistencias = {} # {'P': 0, 'A': 0, 'T': 0} for est in estudiantes]
total_clases = 0

print("""===== SISTEMA DE ASISTENCIA — IS-101 SIGE UNSCH =====
print(Ingresa la asistencia de cada clase.)
print(Para terminar, escribe fin en el nombre del estudiante.)
print()

BUCLE WHILE: continuar hasta que el docente escriba 'fin' —
continuar = True
while continuar:
 print(--- Clase {total_clases + 1} ---)
 asistencia_clase = {}

 for estudiante in estudiantes:
 marca = input(f'{estudiante} (P/A/T o fin): ').strip().upper()
 if marca == 'FIN':
 continuar = False
 break
 if marca not in 'P', 'A', 'T':
 marca = 'X' # dato inválido se registra como ausencia
 asistencia_clase[estudiante] = marca

 # Solo registrar si la clase fue completada
 if continuar or len(asistencia_clase) > 0:
 total_clases += 1
```

Ing. Pelayo Quispe Bautista

7 | Página

Estructuras de Control: Bucles



Sesión 03

```
for est, marca in asistencia_clase.items():
 asistencias[est][marca] += 1

REPORTE FINAL DE ASISTENCIA
print()
print(f"""===== REPORTE DE ASISTENCIA — {total_clases} CLASES REGISTRADAS =====
print(f"Estudiante: {est} (Pres: {pres}<7) (Ausent: {ausent}<5) (Tard: {tard}<7) (Asist: {asist}<10) Estado
print("-----")

alerta_asistencia = []
for estudiante, datos in asistencias.items():
 presentes = datos['P']
 ausentes = datos['A']
 tardanzas = datos['T']
 # Las tardanzas cuentan como media asistencia
 asistencia_efectiva = presentes + (tardanzas * 0.5)
 pct = (asistencia_efectiva / total_clases * 100) if total_clases > 0 else 0
 estado_asist = 'Normal' if pct >= 70 else 'ALERTA'
 if pct < 70:
 alerta_asistencia.append(f'{estudiante} (pct: {pct}<10.1f) (estado: {estado_asist})
 print(f'Estudiante: {est} (presentes: {pres}<7) (ausentes: {ausent}<5) (tardanzas: {tard}<7) (estado: {estado_asist})

if alerta_asistencia:
 print()
 print(ALERTA — Estudiantes con asistencia menor al 70%.)
 for est in alerta_asistencia:
 print(f" {est}")
```

6. EJERCICIOS PROPUESTOS

Resuelvan cada ejercicio siguiendo el proceso ABP: pseudocódigo → código Python → verificación del tester con casos límite. El documentador registra los errores encontrados y como los resuelven. Al finalizar, el equipo presenta su módulo funcional ante los demás grupos.

**Ejercicio 1: Calculadora de promedio con tendencia por ciclo (Nivel básico)**  
El área académica de Ingeniería de Sistemas necesita analizar la evolución del promedio de cada ciclo entre dos semestres. Diseña un programa que:

- Defina dos listas con los promedios de los 10 ciclos de Ingeniería de Sistemas en los semestres 2024-I y 2024-II.
- Use range() para comparar cada ciclo entre los dos semestres e identificar si subió, bajó o se mantuvo.
- Calcule el promedio general de cada semestre y la diferencia.
- Muestre un reporte con la tendencia de cada ciclo y las estadísticas generales.

**Datos de prueba:** Semestre 2024-I: [12.5, 13.0, 14.2, 13.8, 15.0, 14.5, 13.5, 14.0, 15.5, 14.8], Semestre 2024-II: [13.0, 13.5, 13.8, 14.5, 15.3, 14.8, 14.0, 14.5, 15.8, 15.2].

**Casos del tester:** Listas de igual longitud (correcto). Listas vacías (debe manejar el error). Lista de un elemento (debe funcionar). Dos promedios idénticos (tendencia sin cambio).

**Ejercicio 2: Sistema de búsqueda de estudiantes con bajo rendimiento (Nivel intermedio)**  
El tutor académico necesita un sistema que busque rápidamente a todos los estudiantes con promedio menor a un umbral definido por el usuario. El sistema debe:

- Leer el umbral de búsqueda desde el teclado (con validación: debe ser un float entre 0 y 20).
- Reconocer la sección IS-101 calculando el promedio ponderado de cada estudiante.
- Usar continue para omitir a los estudiantes que ya estén en riesgo conocido.
- Construir la lista de estudiantes que estén por debajo del umbral.
- Calcular cuántos créditos adicionales de tutoría se necesitan (2 horas por estudiante en riesgo).
- Mostrar el reporte con la lista de estudiantes identificados y el total de horas de tutoría requeridas.

**Casos del tester:** Umbral: 10.5 (aprobados mínimos). Umbral 14 (estudiantes buenos). Umbral 20 (todos). Umbral 0 (ninguno). Umbral -1 (inválido, debe rechazarse con while).

Ing. Pelayo Quispe Bautista

8 | Página

Estructuras de Control: Bucles



Sesión 03

**Ejercicio 3: Reporte multi-semestre con bucles anidados (Nivel avanzado)**  
El jefe de la OTI necesita un reporte comparativo de los tres últimos semestres (2023-II, 2024-I, 2024-II) para la sección IS-101. El sistema debe:

- Organizar los datos en una lista de diccionarios donde cada elemento tiene el semestre y la lista de promedios ponderados de los estudiantes.
- Usar bucles anidados: el exterior recorre cada semestre, el interior procesa cada estudiante.
- Para cada semestre calcular: promedio general, aprobados, desaprobados, mejor nota y peor nota.
- Identificar el semestre con mayor promedio general y el con mayor porcentaje de aprobación.
- Generar un reporte consolidado con todos los indicadores y resaltar con asterisco (\*) el mejor semestre en cada categoría.

**Reto del tester:** Diseña al menos 5 casos de prueba que incluyan: semestre con todos aprobados, semestre con todos desaprobados, semestre con lista vacía, semestre con un solo estudiante, y el caso en que dos semestres empaten en el mejor promedio. Documenta el comportamiento esperado antes de ejecutar cada caso. Esta práctica sistemática de diseño de pruebas es la base de la competencia DP que desarrollaras plenamente en las Sesiones 8, 9 y 10.

Ing. Pelayo Quispe Bautista

9 | Página

Estructuras de datos en Python (Listas, Tuplas, Diccionarios y Conjuntos)



SESIÓN 04  
Semana 6

|               |                                         |           |                                   |
|---------------|-----------------------------------------|-----------|-----------------------------------|
| Asignatura:   | Algoritmos y Estructuras de Datos       | Semestre: | 2024-II                           |
| Docente:      | Ing. Pelayo Quispe Bautista             | Duración: | 90 minutos                        |
| Metodología:  | Aprendizaje Basado en Problemas (ABP)   | Fase ABP: | Fase 4: Desarrollo y Presentación |
| Herramientas: | Visual Studio Code, GitHub colaborativo | Semana:   | Semana 6 de 13                    |

1. FASE 4: DESARROLLO, PRESENTACION Y VALIDACION CON GITHUB

**FASE 4: Desarrollo y Presentación**

Esta es la primera sesión que implementa únicamente la Fase 4 del modelo ABP. El equipo ya trae el diseño y el pseudocódigo elaborados — ahora codifica, valida y presenta. El desarrollador lidera la codificación en VS Code, el tester diseña y ejecuta los casos de prueba. El documentador registra los commits en GitHub y los errores encontrados con sus soluciones. Al finalizar, cada equipo presenta su módulo funcional ante los demás grupos. Competencias desarrolladas: HC (implementación) + DP (validación, corrección y presentación).

1.1 Nuevo requerimiento del proyecto SIGE

**Situación problemática:**  
El sistema SIGE ya procesa notas y detecta estudiantes en riesgo (Sesión 3). Ahora el jefe de la OTI plantea el siguiente requerimiento de mayor complejidad:  
"El sistema necesita un módulo de gestión completa del registro de asistencia estudiantil. Cada estudiante tiene un conjunto de fechas en que asistió a clases. El sistema debe: (1) registrar nuevas asistencias sin duplicar fechas, (2) calcular el porcentaje de asistencia de cada estudiante usando las fechas programadas, (3) identificar que fechas faltó cada uno, (4) consultar cuantos estudiantes tienen asistencia menor al 70% para alertas tempranas, y (5) generar reportes de asistencia cruzando información entre listas y conjuntos."  
La clave de este requerimiento: los diccionarios agrupan los atributos de cada estudiante, los conjuntos (sets) garantizan que no se dupliquen fechas de asistencia, y las listas de diccionarios forman la base de datos del sistema.  
**Tu misión como equipo (Fase 4): Codificar el módulo completo en VS Code, hacer commit a GitHub y presentar el producto funcional verificado con casos de prueba.**

**Actividad Fase 4: Plan de trabajo del equipo**

Antes de abrir VS Code, el líder coordina la división de tareas:  
Desarrollador: Implementa los módulos A, B y C del sistema de asistencia.  
Usa .git() para acceso seguro a diccionarios.  
Hace commit a GitHub al terminar cada módulo.  
Tester: Diseña los casos de prueba ANTES de ejecutar el código:  
Caso 1: estudiante con asistencia perfecta (100%)  
Caso 2: estudiante con asistencia exactamente 70%  
Caso 3: estudiante con 0% de asistencia  
Caso 4: buscar una clave que no existe en el diccionario  
Caso 5: conjunto de fechas vacío  
Documentador: Registro de los notebooks: estructura de datos elegida para cada componente, errores de KeyError/IndexError encontrados y como se corrigieron.  
Líder: Coordina los tiempos, resuelve bloqueos y prepara la presentación del módulo.

Ing. Pelayo Quispe Bautista

1 | Página

## 2. COMPETENCIAS A DESARROLLAR

| Pensamiento Lógico-Algorítmico (PLA)                                                                                                                              | Habilidades de Codificación (HC)                                                                                                                                                      | Depuración y Prueba (DP)                                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Selecciona la estructura de datos adecuada (lista, tupla, dict, set) para cada componente del sistema de asistencia, justificando la elección antes de codificar. | Implementa en VS Code el sistema de gestión de asistencia usando listas de diccionarios, tuplas para datos fijos, sets para fechas únicas y dicts anidados con acceso seguro [get()]. | Detecta y corrige KeyError, IndexError y TypeError al acceder a estructuras anidadas; verifica con casos límite: estudiante sin asistencia, clave inexistente, set vacío. |

## 3. CONTENIDO CONCEPTUAL: Estructuras de datos en Python

## 3.1 Guía de selección de estructuras

La elección incorrecta de estructura de datos genera código lento, propenso a errores y difícil de mantener. En el sistema de asistencia del SIGE, cada tipo de dato tiene un rol específico que no puede intercambiarse sin consecuencias. El equipo debe tomar esta decisión ANTES de codificar.

| Estructura | Ordenada  | Mutable | Duplicada  | Uso principal en SIGE                                                                 |
|------------|-----------|---------|------------|---------------------------------------------------------------------------------------|
| list []    | Si        | Si      | Si         | Lista de estudiantes, notas de evaluaciones, historial de cursos matriculados.        |
| tuple ()   | Si        | No      | Si         | Horarios de clases (inmutables), coordenadas UNSCH, pesos porcentuales fijos.         |
| dict {}    | Inserción | Si      | Claves: No | Ficha de estudiante (atributos), catálogo de cursos, registro de notas por código.    |
| set {}     | No        | Si      | No         | Fechas de asistencia (sin duplicados), cursos activos únicos, intersección de grupos. |

## 3.2 Listas

Las listas son la estructura más versátil de Python. Almacenan elementos en orden, permiten duplicados y son mutables. En el SIGE se usan para mantener el registro de todos los estudiantes de una sección, donde el orden de inscripción se preserva.

```

LISTAS: Registro de estudiantes de la sección IS-101
Creación y acceso básico
estudiantes = ['Carmen Quispe', 'Pedro Cardenas', 'Maria Humari',
 'Jorge Coente', 'Ana Palomino', 'Luis Torres']

primer = estudiantes[0] # 'Carmen Quispe' (índice positivo)
ultimo = estudiantes[-1] # 'Luis Torres' (índice negativo)
sub_lista = estudiantes[1:4] # ['Pedro Cardenas', 'Maria Humari', 'Jorge Coente']

print(f'Total estudiantes: {len(estudiantes)}')
print(f'Primer inscrito : {primer}')
print(f'Ultimo inscrito : {ultimo}')

Modificación — las listas son mutables
estudiantes.append('Rosa Sulca') # agregar al final
estudiantes.insert(2, 'Eva Palomino') # insertar en posición 2
estudiantes.remove('Jorge Coente') # eliminar por valor
retirado = estudiantes.pop(0) # eliminar por índice y retornar

print(f'Estudiante retirado: {retirado}')
print(f'Lista actualizada : {estudiantes}')

Ordenar la lista alfabéticamente para el reporte
estudiantes_ordenados = sorted(estudiantes) # nueva lista ordenada
estudiantes.sort() # ordena in-place

```

Ing. Pelayo Quispe Bautista

2 | Página

```

Verificar pertenencia
print('Carmen Quispe' in estudiantes) # True o False

List comprehension — crear lista de códigos universitarios
codigos = [f'271504' + str(i).zfill(2) for i in range(1, len(estudiantes)+1)]
print(f'Códigos generados: {codigos[3:]}')

```

## 3.3 Tuplas

Las tuplas son idénticas a las listas pero inmutables: una vez creadas, no pueden modificarse. Son la elección correcta para datos que no deben cambiar durante la ejecución: horarios de clases, coordenadas geográficas de la UNSCH, pesos porcentuales de evaluación. Su inmutabilidad las hace más rápidas que las listas para lectura y permite usarlas como claves de diccionario.

```

Tuplas: Datos fijos e inmutables del SIGE
Horarios del curso IS-402 (día, hora_inicio, hora_fin) — no deben cambiar
horarios_is402 = [
 ('Lunes', '08:00', '10:00'),
 ('Miércoles', '10:00', '12:00'),
 ('Viernes', '14:00', '16:00')
]

Coordenadas geográficas de la UNSCH (inmutables por definición)
ubicacion_unsch = (-13.1639, -74.2236) # (latitud, longitud)

Pesos de evaluación del SIGE (definidos en el reglamento académico)
pesos_evaluacion = (0.30, 0.30, 0.40) # practicas, investigacion, examen

Desempaquetado de tuplas
peso_prac, peso_inv, peso_exam = pesos_evaluacion
lat, lon = ubicacion_unsch
print(f'Peso examen: {peso_exam} | {peso_exam*100.0}%')
print(f'Ubicación UNSCH: {lat}, {lon}')

Reconocer horarios con desempaquetado
print('Horarios IS-402:')
for dia, inicio, fin in horarios_is402:
 print(f'{dia}: {inicio} - {fin}')

Las tuplas NO pueden modificarse (inmutabilidad)
pesos_evaluacion[0] = 0.40 # TypeError: tuple object does not support item assignment

Pero sí pueden usarse como claves de diccionario (las listas no)
ubicaciones = {(-13.1639, -74.2236): 'UNSCH Sede Central',
 (-13.1500, -74.2000): 'UNSCH Sede Norte'}
print(f'ubicaciones: {ubicaciones}')

```

## 3.4 Diccionarios

Los diccionarios son la estructura central del sistema de gestión del SIGE. Permiten acceder a los atributos de un estudiante por nombre (clave) en lugar de por posición numérica, haciendo el código mucho más legible y resistente a errores. La lista de diccionarios es el patrón para representar una tabla de base de datos en Python.

```

Diccionarios
DICCIONARIOS: Ficha de estudiante y base de datos SIGE

```

Ing. Pelayo Quispe Bautista

3 | Página

```

Ficha completa de un estudiante (equivalente a una fila de la BD)
estudiante = {
 'codigo': '27150415',
 'nombre': 'Carmen Quispe Humari',
 'carrera': 'Ingeniería de Sistemas',
 'ciclo': 6,
 'cursos': ['IS-401', 'IS-402', 'IS-403'],
 'promedio': 15.8
}

Acceso DIRECTO — lanza KeyError si la clave no existe
nombre = estudiante['nombre'] # 'Carmen Quispe Humari'

Acceso SEGURO con .get() — devuelve None o un valor por defecto
telefono = estudiante.get('telefono') # None (no lanza error)
dni = estudiante.get('dni', 'No registrado') # 'No registrado'

print(f'Nombre : {nombre}')
print(f'Telefono : {telefono}')
print(f'DNI : {dni}')

Modificación y edición de claves
estudiante['promedio'] = 16.2 # actualizar clave existente
estudiante['email'] = 'cquispe@unsch.edu.pe' # agregar nueva clave
estudiante['cursos'].append('IS-404') # modificar lista interna
Eliminar claves
del estudiante['ciclo'] # KeyError si no existe
retirado = estudiante.pop('email', None) # seguro con valor por defecto

Iterar sobre el diccionario
print(f'Ficha actualizada:')
for clave, valor in estudiante.items():
 print(f'{clave}<12>: {valor}')

Fusionar dos diccionarios (Python 3.9+)
datos_extra = {'dni': '71764255', 'sede': 'Huamanga'}
estudiante.update(datos_extra) # agrega/reescribe claves

Lista de diccionarios — patrón de base de datos
bd_estudiantes = [
 {'codigo': '27150415', 'nombre': 'Carmen Quispe', 'promedio': 15.8},
 {'codigo': '27150416', 'nombre': 'Pedro Cardenas', 'promedio': 12.3},
 {'codigo': '27150417', 'nombre': 'Maria Humari', 'promedio': 17.5},
]

Buscar un estudiante por código
codigo_buscar = '27150416'
resultado = None
for est in bd_estudiantes:
 if est['codigo'] == codigo_buscar:
 resultado = est
 break
resultado:
print(f'Encontrado: {resultado["nombre"]} — promedio {resultado["promedio"]}')
else:
 print(f'Estudiante {codigo_buscar} no encontrado.')

```

## 3.5 Conjuntos (sets)

Los conjuntos son colecciones desordenadas de elementos únicos. Son la estructura ideal para el registro de asistencia del SIGE, porque garantizan automáticamente que una fecha no se registre dos veces, aunque el operador la ingrese por error. Además, soportan operaciones matemáticas de teoría de conjuntos (unión, intersección, diferencia) que permiten consultas poderosas sobre las asistencias.

Ing. Pelayo Quispe Bautista

4 | Página

| Operación      | Símbolo / Método           | Aplicación en el sistema de asistencia SIGE                                          |
|----------------|----------------------------|--------------------------------------------------------------------------------------|
| Unión          | o union()                  | Todas las fechas programadas de dos secciones del mismo curso.                       |
| Intersección   | & o intersection()         | Fechas en que un estudiante asistió Y que estaban programadas (asistencias válidas). |
| Diferencia     | - o difference()           | Fechas programadas - fechas asistidas = fechas de inasistencia del estudiante.       |
| Dif. simétrica | ^ o symmetric_difference() | Fechas que tiene uno u otro estudiante, pero no ambos (para comparar asistencias).   |
| Subconjunto    | <= o issubset()            | Verificar si todas las fechas obligatorias están en las asistencias del estudiante.  |

## Conjuntos

```

SETS: Control de asistencia sin duplicados - SIGE UNSCH
Fechas programadas del curso IS-402 (semana 1 a la semana 4)
fechas_programadas = {
 '2024-08-12', '2024-08-14', '2024-08-19', '2024-08-21',
 '2024-08-26', '2024-08-28', '2024-09-02', '2024-09-04'
}

Asistencias de dos estudiantes (los sets eliminan duplicados automáticamente)
asist_carmen = {
 '2024-08-12', '2024-08-14', '2024-08-19', # semana 1 y 2
 '2024-08-26', '2024-08-28', '2024-09-02', '2024-09-04' # semana 3 y 4
}
asist_pedro = {
 '2024-08-12', '2024-08-19', '2024-08-26',
 '2024-09-02', '2024-09-04'
}

Registrar una nueva asistencia (add) - no duplica si ya existe
asist_carmen.add('2024-09-04') # ya existe - no pasa nada
asist_carmen.add('2024-09-05') # nueva fecha - se agrega
print(f'Carmen — fechas registradas: {len(asist_carmen)}')

Inasistencias = fechas programadas MENOS fechas asistidas
inasist_carmen = fechas_programadas - asist_carmen
inasist_pedro = fechas_programadas - asist_pedro

Calcular porcentaje de asistencia
Solo contar asistencias que coinciden con fechas programadas
asist_validas_carmen = asist_carmen & fechas_programadas # interseccion
asist_validas_pedro = asist_pedro & fechas_programadas

pct_carmen = len(asist_validas_carmen) / len(fechas_programadas) * 100
pct_pedro = len(asist_validas_pedro) / len(fechas_programadas) * 100

print(f'Carmen: {pct_carmen:.1f}% ({len(asist_validas_carmen)} / {len(fechas_programadas)}) clases')
print(f'Pedro: {pct_pedro:.1f}% ({len(asist_validas_pedro)} / {len(fechas_programadas)}) clases')

print(f'Inasistencias Carmen: {sorted(inasist_carmen)}')
print(f'Inasistencias Pedro: {sorted(inasist_pedro)}')

Fechas en que ambos asistieron (para reportes de clase)
ambos_presentes = asist_validas_carmen & asist_validas_pedro
print(f'Clases con ambos presentes: {len(ambos_presentes)}')

Verificar si algun estudiante tiene asistencia perfecta

```

Ing. Pelayo Quispe Bautista

5 | Página



asistencia\_perfecta = fechas\_programadas <= asist\_carmen # subconjunto  
print(Carmen tiene asistencia perfecta: (asistencia\_perfecta))

### 3.6 Errores comunes en estructuras de datos

La Fase 4 exige que el tester ejecute casos de prueba que fueren estos errores para verificar que el código los maneja correctamente con get() y validaciones previas.

#### ERROR 1: KeyError: clave inexistente en diccionario

```
#INCORRECTO: acceso directo sin verificar existencia
fecha = {'nombre': 'Carmen', 'promedio': 15.8}
print(fecha['telefono']) # KeyError: 'telefono'

#CORRECTO opción 1: usar get() con valor por defecto
print(fecha.get('telefono', 'No registrado')) # 'No registrado'

#CORRECTO opción 2: verificar antes de acceder
if 'telefono' in fecha:
 print(fecha['telefono'])
else:
 print('Telefono no registrado en el SIGE')
```

#### ERROR 2: TypeError: operación invalida con tipos incompatibles

```
#INCORRECTO: intentar sumar str y float (entrada sin convertir)
nota_md = 15.5 # viene del formulario como texto
promedio = nota_md * 0.30 # TypeError: can't multiply sequence by non-int

#CORRECTO: convertir antes de operar
promedio = float(nota_md) * 0.30 # 4.65

#INCORRECTO: intentar agregar una lista a un set
fechas = {'2024-08-12', '2024-08-14'}
fechas.add('2024-08-19', '2024-08-21') # TypeError: unhashable type: 'list'

#CORRECTO: agregar elemento por elemento o usar update()
fechas.update('2024-08-19', '2024-08-21') # agrega multiples elementos
```

#### ERROR 3: Modificar diccionario mientras se itera (RuntimeError)

```
#INCORRECTO: eliminar claves durante la iteración
estudiante = {'nombre': 'Carmen', 'telefono': None, 'promedio': 15.8}
for clave in estudiante:
if estudiante[clave] is None:
del estudiante[clave] # RuntimeError: dictionary changed size during iteration

#CORRECTO: iterar sobre una copia de las claves
for clave in list(estudiante.keys()): # list() crea una copia
 if estudiante[clave] is None:
 del estudiante[clave]
print('Ficha limpia: (estudiante)')
```

### 4. SEUDOCODIGO DEL SISTEMA

Aunque esta sesión es de Fase 4, el equipo debe tener su pseudocódigo listo antes de abrir VS Code. Este pseudocódigo conecta las tres estructuras de datos en un sistema cohesivo.

Ing. Pelayo Quispe Bautista

6 | Página

### SEUDOCODIGO: Sistema de gestión de asistencia estudiantil SIGE

```
INICIO
BASE DE DATOS: Lista de diccionarios
bd_asistencia <- [
 {codigo, nombre, carrera, cursos_matriculados (lista),
 fechas_asistidas (set)}
 ... para cada estudiante ...
]

FUNCION: Registrar asistencia
FUNCION registrar_asistencia(codigo, fecha):
 Si codigo NO ESTÁ en bd_asistencia ENTONCES
 ESCRIBIR 'Estudiante no encontrado'
 RETORNAR
FIN SI

AGREGAR fecha AL SET fechas_asistidas del estudiante // no duplica
FIN_FUNCION

FUNCION: Calcular porcentaje de asistencia
FUNCION calcular_asistencia(codigo, fechas_programadas):
 Si codigo NO ESTÁ en bd_asistencia ENTONCES
 RETORNAR 0
 FIN SI
 asistencias_validas <- fechas_asistidas & fechas_programadas // interseccion
 Si longitud(fechas_programadas) = 0 ENTONCES
 RETORNAR 0
 FIN SI
 RETORNAR longitud(asistencias_validas) / longitud(fechas_programadas) * 100
FIN_FUNCION

// --- REPORTE: Identificar estudiantes con alerta de asistencia ---
PARA CADA estudiante EN bd_asistencia:
 pct <- calcular_asistencia(estudiante.codigo, fechas_programadas)
 Si pct < 70 ENTONCES
 AGREGAR estudiante A lista_alerta
 FIN SI
FIN_PARA

ESCRIBIR reporte con porcentajes e inasistencias por estudiante
FIN
```

### 5. PROBLEMA ABP RESUELTO

Implementación completa del pseudocódigo anterior. Este es el módulo central del requerimiento de la OTI para la Semana 6.

#### CODIGO PYTHON COMPLETO

```
#####
SISTEMA SIGE - OTI UNSCH | Sesión 4
Módulos: Gestión de asistencia estudiantil
Estructuras: list + dict + set + tuple
#####
DATOS: Lista de diccionarios - base de datos del SIGE
bd_asistencia = [
 {
 'codigo': '27150418',
 'nombre': 'Carmen Quispe Huamán',
 'carrera': 'Ing. de Sistemas',
 'cursos': ['IS-401', 'IS-402', 'IS-403', 'IS-404']
 },
 {
 'codigo': '27150417',
 'nombre': 'María Huamán Quispe',
 'carrera': 'Ing. de Sistemas',
 'cursos': ['IS-401', 'IS-402', 'IS-403', 'IS-404']
 },
 {
 'codigo': '27150419',
 'nombre': 'Jorge Coente Palomino',
 'carrera': 'Ing. de Sistemas',
 'cursos': ['IS-401', 'IS-402']
 }
]
```

Ing. Pelayo Quispe Bautista

7 | Página

```
{
 'fechas_asistidas': {
 '2024-08-12', '2024-08-14', '2024-08-19',
 '2024-08-26', '2024-08-28', '2024-09-04'
 },
 {
 'codigo': '27150418',
 'nombre': 'Carmen Quispe Huamán',
 'carrera': 'Ing. de Sistemas',
 'cursos': ['IS-401', 'IS-402', 'IS-403', 'IS-404'],
 'fechas_asistidas': {
 '2024-08-12', '2024-08-19', '2024-08-26', '2024-09-02'
 },
 {
 'codigo': '27150417',
 'nombre': 'María Huamán Quispe',
 'carrera': 'Ing. de Sistemas',
 'cursos': ['IS-401', 'IS-402', 'IS-403', 'IS-404'],
 'fechas_asistidas': {
 '2024-08-12', '2024-08-14', '2024-08-19', '2024-08-21',
 '2024-08-26', '2024-08-28', '2024-09-02', '2024-09-04'
 },
 {
 'codigo': '27150419',
 'nombre': 'Jorge Coente Palomino',
 'carrera': 'Ing. de Sistemas',
 'cursos': ['IS-401', 'IS-402'],
 'fechas_asistidas': {'2024-08-12', '2024-08-26'}
 }
]
 }

Fechas programadas del curso IS-402 (tupla - no debe cambiar)
fechas_programadas = (
 '2024-08-12', '2024-08-14', '2024-08-19', '2024-08-21',
 '2024-08-26', '2024-08-28', '2024-09-02', '2024-09-04'
)

set_fechas_prog = set(fechas_programadas) # convertir a set para operaciones

MODULO A: Registrar nueva asistencia
def registrar_asistencia(codigo, fecha):
 for est in bd_asistencia:
 if est.get('codigo') == codigo:
 est['fechas_asistidas'].add(fecha) # set no duplica
 print(f'Asistencia {fecha} registrada para {est["nombre"]}')
 return True
 print(f'Código {codigo} no encontrado en el SIGE.')
 return False

MODULO B: Calcular porcentaje de asistencia
def calcular_asistencia(codigo):
 for est in bd_asistencia:
 if est.get('codigo') == codigo:
 asist_validas = est['fechas_asistidas'] & set_fechas_prog
 if len(set_fechas_prog) == 0:
 return 0.0
 pct = len(asist_validas) / len(set_fechas_prog) * 100
 return round(pct, 1)
```

Ing. Pelayo Quispe Bautista

8 | Página

```
return 0.0 # código no encontrado

MODULO C: Inasistencias de un estudiante
def obtener_inasistencias(codigo):
 for est in bd_asistencia:
 if est.get('codigo') == codigo:
 return sorted(set_fechas_prog - est['fechas_asistidas'])
 return []

REPORTE COMPLETO DE ASISTENCIA
print("\n")
print("REPORTE DE ASISTENCIA - IS-402 | SIGE UNSCH")
print("\n")
print(f"Estudiante: {codigo} (Código: {codigo}) (Alerta: {pct < 70}) (Inasistencias: {len(inasistencias)})")
print("\n")

alerta_asistencia = []

for estudiante in bd_asistencia:
 codigo = estudiante.get('codigo')
 nombre = estudiante.get('nombre')
 pct = calcular_asistencia(codigo)
 inasistencias = obtener_inasistencias(codigo)
 estado = "ALERTA" if pct < 70 else "Normal"

 if pct < 70:
 alerta_asistencia.append(f"{nombre} ({pct}%)")

Mostrar solo las dos primeras inasistencias para brevedad
inasist_str = str(inasistencias[0:2]) + ("..." if len(inasistencias) > 2 else "")
print(f"Nombre: {codigo} (Código: {codigo}) (Alerta: {pct < 70}) (Inasistencias: {len(inasistencias)})")

print("\n")
print(f"Total estudiantes: {len(bd_asistencia)}")
print(f"Con alerta asistencia: {len(alerta_asistencia)}")

if alerta_asistencia:
 print("ALERTA - Asistencia menor al 70%")
 for a in alerta_asistencia:
 print(f" {a}")

PRUEBA DEL TESTER: Casos límite
print("\n")
print("=== CASOS DE PRUEBA DEL TESTER ===")
print(f"Asistencia 27150418 (Carmen): {calcular_asistencia(27150418)}%")
print(f"Asistencia 27150419 (Jorge): {calcular_asistencia(27150419)}%")
print(f"Código inexistente 99999999: {calcular_asistencia(99999999)}%")

Registrar nueva asistencia y verificar que no duplica
registrar_asistencia(27150418, '2024-08-12') # ya existe
registrar_asistencia(27150418, '2024-09-05') # nueva fecha
```

### 6. EJERCICIOS PROPUESTOS

Implementen cada ejercicio en VS Code, hagan commit a GitHub al terminar cada módulo y el tester verifica con los casos indicados. Al finalizar, el equipo presenta su producto funcional ante los demás grupos.

Ejercicio 1: Sistema de matrícula con listas y diccionarios (Nivel básico)

La Oficina de Registros necesita un sistema que gestione la matrícula de estudiantes en cursos. Usando listas de diccionarios, implementa:

Ing. Pelayo Quispe Bautista

9 | Página

Estructuras de datos en Python (Listas, Tuplas, Diccionarios y Conjuntos)

Sesión 04

1. Un catálogo de cursos disponibles (código, nombre, créditos, vacantes, horario como tupla).

2. Una función agregar\_matricula(codigo\_est, codigo\_curso) que verifique vacantes disponibles y registre la matrícula.

3. Una función consultar\_carga(codigo\_est) que devuelva los cursos matriculados y los créditos totales.

4. Una función verificar\_prerequisito(codigo\_est, codigo\_curso, prerequisito) que use sets para verificar si el estudiante ya aprobó el curso prerequisito.

**Casos del tester:** Matricular en curso con vacantes (correcto). Matricular en curso sin vacantes (debe rechazar). Consultar carga de estudiante sin matriculas (0 créditos). Verificar prerequisito aprobado (debe permitir). Verificar prerequisito no aprobado (debe bloquear).

**Ejercicio 2: Analizador de rendimiento por escuela profesional (Nivel intermedio)**

La Dirección Académica necesita comparar el rendimiento entre las escuelas profesionales de la UNSCH. Usando diccionarios y sets, implemente:

5. Estructura: {escuela: {ciclo: [promedios de estudiantes]}}

6. Función calcular\_estadisticas(escuela) que devuelva: promedio general, ciclo con mejor rendimiento, ciclo con mayor riesgo y porcentaje de estudiantes con promedio >= 14.

7. Función comparar\_escuelas(escuela1, escuela2) que muestre lado a lado las estadísticas de ambas usando diccionarios.

8. Función identificar\_ciclos\_criticos() que use sets para listar todos los ciclos con promedio < 11 en cualquier escuela.

**Casos del tester:** Escuela con todos los ciclos con promedio > 14. Escuela con todos en riesgo. Ciclo con un solo estudiante. Escuela sin datos (dict vacío).

**Ejercicio 3: Sistema completo de gestión académica con GitHub (Nivel avanzado)**

Integrando todo lo aprendido en la Sesión 4, el equipo construye el módulo más completo del proyecto SIGE hasta la fecha. El sistema debe:

9. Combinar la base de datos de asistencia (Problema 1) con el historial de notas (Problema 2) en una estructura unificada de lista de diccionarios con todos los atributos del estudiante.

10. Implementar una función generar\_fecha\_completa(codigo) que consulte ambas bases de datos y devuelva un diccionario con: nombre, carrera, promedio ponderado, porcentaje de asistencia, lista de inasistencias, cursos en riesgo y estado académico general.

11. Implementar una función exportar\_reporte\_seccion() que genere un reporte de toda la sección IS-101 con los datos de cada estudiante ordenados por promedio descendente.

12. Hacer al menos 3 commits a GitHub durante el desarrollo: uno al terminar la estructura de datos, uno al terminar las funciones y uno al terminar las pruebas del tester.

**Reto de presentaciones:** Al terminar, el equipo tiene 5 minutos para presentar su módulo ante los demás grupos explicando: (1) que estructura de datos eligieron para cada componente y por qué; (2) que errores encontraron durante las pruebas y como los corrigieron; (3) que caso límite fue el más difícil de manejar.

Ing. Pelayo Quispe Bautista

10 | Página

Funciones en Python

Sesión 05

Funciones en Python: Módulo de Promedios Ponderados y Reportes Académicos Reutilizables - SIGE UNSCH

SESION 05  
Semana 7

|               |                                        |            |                |
|---------------|----------------------------------------|------------|----------------|
| Asignatura:   | Algoritmos y Estructuras de Datos      | Semestre:  | 2024-II        |
| Docente:      | Ing. Pelayo Quispe Bautista            | Duración:  | 90 minutos     |
| Metodología:  | Aprendizaje Basado en Problemas (ABP)  | Fases ABP: | Fases 3 y 4    |
| Herramientas: | VS Code, Programación en pares, GitHub | Semana:    | Semana 7 de 13 |

1. FASES 3 Y 4: INVESTIGACION, CONSTRUCCION Y PROGRAMACION EN PARES

FASE 3  
Investigación y Construcción

Los equipos investigan autónomamente los conceptos de funciones: definición (def), parámetros, return y scope. Identifican los bloques de código repetidos en las sesiones anteriores que merecen convertirse en funciones. El par desarrollador-revisor trabaja de forma colaborativa: uno escribe, el otro revisa en tiempo real. Competencia desarrollada: HC - implementación de funciones reutilizables para el sistema SIGE.

FASE 4  
Desarrollo y Presentación

Los equipos ensamblan el módulo completo de funciones del SIGE en VS Code con programación en pares. El tester diseña y ejecuta casos de prueba para cada función: valores válidos, límites e inválidos. El equipo hace commit a GitHub al terminar cada función y al final presenta el módulo ante sus pares. Competencia desarrollada: DP - detección de funciones que retornan None, errores de scope y parámetros incorrectos.

1.1 Nuevo requerimiento del proyecto SIGE

**Situación problemática:**

El sistema SIGE de las sesiones anteriores funciona, pero el jefe de la OTI detecta un problema grave de mantenimiento: el cálculo del promedio ponderado aparece escrito 4 veces en el código (sesiones 2, 3 y 4), y la lógica de clasificación del rendimiento se repite en 5 lugares distintos. Cuando la UNSCH decide cambiar la nota mínima de aprobación de 10.5 a 11.0 para el siguiente semestre, el programador tiene que buscar y editar cada ocurrencia manualmente, con alto riesgo de error.

El requerimiento del jefe de la OTI es claro: "El sistema debe estar modularizado. Cada operación debe estar encapsulada en una función con nombre descriptivo, que reciba los datos como parámetros y devuelva el resultado con return para que otros módulos puedan usarlo. El código no debe repetirse".

Esta sesión usa programación en pares: el par (driver + navegador) rota cada 15 minutos. El driver escribe el código; el navegador revisa, detecta errores y propone mejoras en tiempo real.

**🔗 Tu misión:** Identificar los bloques repetidos en el código de las sesiones anteriores, encapsularlos en funciones y construir el módulo reutilizable del SIGE.

Ing. Pelayo Quispe Bautista

1 | Página

Funciones en Python

Sesión 05

• Detecta errores lógicos, casos no considerados y código repetido.

• Propone mejoras, pero NO toma el teclado, comunica verbalmente.

**Rotación cada 15 minutos:** los roles se intercambian al terminar cada función.

**Preguntas de Investigación antes de codificar:**

1. ¿Qué bloques de código de las sesiones 2, 3 y 4 se reemplen exactamente?

2. ¿Qué parámetros necesita recibir cada función para ser verdaderamente reutilizable?

3. ¿Qué diferencia hay entre usar print() y usar return en una función del SIGE?

4. ¿Qué pasa si la función calcular\_promedio() recibe una lista vacía?

2. COMPETENCIAS A DESARROLLAR

| Pensamiento Lógico-Algorítmico (PLA)                                                                                                                                           | Habilidades de Codificación (HC)                                                                                                                                                                  | Depuración y Prueba (DP)                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Descompone el sistema SIGE en funciones especializadas, identificando claramente las entradas (parámetros), el proceso y la salida (return) de cada módulo antes de codificar. | Implementa en VS Code funciones con parámetros posicionales, nombrados y por defecto; usa return para devolver valores reutilizables; aplica programación en pares para la revisión colaborativa. | Detecta y corrige funciones que retornan None inesperadamente, errores de scope, parámetros en orden incorrecto y funciones con efectos secundarios ocultos; verifica con casos límite. |

1. CONTENIDO CONCEPTUAL

1.1 ¿Por qué funciones? El problema del código repetido en el SIGE

El código sin funciones viola el principio DRY (Don't Repeat Yourself - No te repitas). Cada vez que se copia y pega un bloque de código, se crea una nueva fuente potencial de error: si el original tiene un bug o necesita actualizarse, hay que recordar arreglar cada copia manualmente. En el SIGE esto es crítico porque el cálculo del promedio y la clasificación del rendimiento se usan en docenas de lugares.

1.2 Anatomía de una función en Python

Una función en Python se define con el palabra reservada def, seguida del nombre, paréntesis con parámetros y dos puntos. El cuerpo va indentado. La instrucción return devuelve un valor y termina la función. Si no hay return, la función devuelve None automáticamente.

Anatomía completa de una función

```
=====
ANATOMIA DE UNA FUNCION: Partes y su propósito
=====
def calcular_promedio_ponderado(
 nota_practica, # parametro 1 (posicional)
 nota_investigacion, # parametro 2 (posicional)
 nota_examen, # parametro 3 (posicional)
 nota_minima_aprobacion=10.5, # parametro 4 (con valor por defecto)
):
 """Calcula el promedio ponderado segun el reglamento academico UNSCH.
 Pesos: practicas 30%, investigacion 30%, examen final 40%."""

 # Parametros:
 # nota_practica (float) : Nota de practicas (0-20)
 # nota_investigacion (float) : Nota de investigacion (0-20)
 # nota_examen (float) : Nota del examen final (0-20)
 # nota_minima_aprobacion (float): Nota minima para aprobar (default: 10.5)

 # Retorna:
 # tupla: (promedio: float, estado: str, categoria: str)
 """
 # Cuerpo de la función (indentado)
 """
```

Ing. Pelayo Quispe Bautista

2 | Página

Funciones en Python

Sesión 05

```
promedio = (nota_practica * 0.30) + (nota_investigacion * 0.30) + (nota_examen * 0.40)

Clasificación por rendimiento
if promedio >= 17:
 categoria = "Sobresaliente"
elif promedio >= 14:
 categoria = "Bueno"
elif promedio >= nota_minima_aprobacion:
 categoria = "Aprobado"
else:
 categoria = "Desaprobado"

estado = "Aprobado" if promedio >= nota_minima_aprobacion else "Desaprobado"

return round(promedio, 2), estado, categoria # return de tupla — reusable

Llamadas con diferentes argumentos
prom_est, cat = calcular_promedio_ponderado(15.0, 14.5, 16.0) # usa default 10.5
print(f'Promedio: {prom} Estado: {est} Categoría: {cat}')

prom2, est2, cat2 = calcular_promedio_ponderado(15.0, 14.5, 16.0, 14.0) # posgrado: minimo 14
print(f'Posgrado: {prom2} Estado: {est2}')
```

3.3 Tipos de parámetros

| Tipo de parámetro     | Descripción                                                            | Ejemplo en el SIGE                                                             |
|-----------------------|------------------------------------------------------------------------|--------------------------------------------------------------------------------|
| Posicional            | Se pasan en el mismo orden que se definen. El orden es obligatorio.    | calcular_promedio(27150415, 15, 14, 16) código; nota1, nota2, nota3            |
| Nombrado (keyword)    | Se pasan con nombre=valor. El orden no importa. Mas legible.           | calcular_promedio(nota3=16, codigo="27150415", nota1=15, nota2=14)             |
| Por defecto (default) | Tiene valor predeterminado. Si no se pasa, usa el default.             | calcular_promedio(notas, nota_minima=10.5) - la nota minima UNSCH ya definida. |
| *args (variable)      | Acepta cualquier número de argumentos posicionales. Llegan como tupla. | calcular_promedio_variable(27150415, 14, 16, 15, 17, 13) — n notas.            |

Tipos de parámetros

```
=====
TIPOS DE PARAMETROS: Ejemplos con funciones del SIGE
=====

Posicionales - orden obligatorio
def registrar_estudiante(codigo, nombre, carrera, ciclo):
 return (codigo, codigo, 'nombre', nombre,
 'carrera', carrera, 'ciclo', ciclo)

Correcto: en orden
est = registrar_estudiante("27150415", "Carmen Quispe", "Ing. Sistemas", 6)
Incorrecto: orden equivocado - ciclo en lugar de carrera
est = registrar_estudiante("27150415", "Carmen Quispe", 6, "Ing. Sistemas")

Nombrados - orden libre, mas legible
```

Ing. Pelayo Quispe Bautista

3 | Página

Funciones en Python

```

es2 = registrar_estudiante(
 cidei=,
 nombre="Pedro Cardenas",
 codigo="27150416",
 carrera="Ing. Sistemas"
)
Por defecto - simplifica las llamadas mas frecuentes
del buscar_en_sige(codigo, base_datos, retornar_completo=False)
for est in base_datos:
 if est.get('codigo') == codigo:
 if retornar_completo:
 return est
 else:
 return est.get('nombre') # devuelve solo el nombre
return None
bd = [est, es2]
nombre = buscar_en_sige(27150416, bd) # solo nombre
completo = buscar_en_sige(27150416, bd, retornar_completo=True) # todo
print(f'Nombre : {nombre}')
print(f'Completo {completo}')

*args - numero variable de notas para promediar
def calcular_promedio_simple(*notas):
 """Acepta cualquier numero de notas como argumentos"""
 if not notas: # tupla vacia
 return 0.0
 return round(sum(notas) / len(notas), 2)
print(calcular_promedio_simple(14, 16, 15)) # 3 notas
print(calcular_promedio_simple(14, 16, 15, 17, 13)) # 5 notas
print(calcular_promedio_simple()) # sin notas --> 0.0

```

**3.4 return vs print**  
 Esta es el error más frecuente al diseñar funciones para el SIGE. Una función con `print()` muestra el resultado en consola pero no lo devuelve; si se intera guardar ese resultado en una variable o usado en otro calculo, se obtiene None. En el SIGE, donde los resultados de una función alimentan a otras funciones, siempre se debe usar `return`.

|                         | return                                                                                                | print()                                                                 |
|-------------------------|-------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------|
| Que hace                | Devuelve un valor al código que llamo la función. Termina la función.                                 | Muestra texto en la consola. La función continua ejecutándose.          |
| ¿Se puede usar después? | Si. El valor queda disponible para variables, listas, operaciones o nuevas funciones.                 | No. El resultado se imprime pero no puede almacenarse ni reutilizarse.  |
| En el SIGE              | calcular_promedio() devuelve el float: puede compararse, guardarse en dict e incluirse en el reporte. | mostrar_promedio() solo imprime: si se asigna a variable, obtiene None. |

Ing. Pelayo Quique Bautista

4 | Página

Funciones en Python

```

resultado_mai = mostrar_promedio_MAI(14, 16, 15)
print(f'Resultado guardado: {resultado_mai}') # None -- no sirve para nada
print(f'Estado: {Aprobado} if resultado_mai >= 10.5 else "Desaprobado"')
TypeError: >= not supported between NoneType and float
print()

FUNCION CON RETURN - devuelve valor reutilizable
def calcular_promedio_BIEN(notas1, notas2, notas3):
 promedio = (nota1 + nota2 + nota3) / 3
 return round(promedio, 2) # devuelve el float
resultado_bien = calcular_promedio_BIEN(14, 16, 15)
print(f'Resultado guardado: {resultado_bien}') # 15.0 -- útil
print(f'Estado: {Aprobado} if resultado_bien >= 10.5 else "Desaprobado"')

El resultado puede pasarse a otra función
def generar_alerta(promedio, seccion):
 if promedio_seccion < 12:
 return ALERTA_CRITICA -- toda la seccion en riesgo
 elif promedio_seccion < 14:
 return Advertencia -- redimiento por debajo del optimo
 return Normal
alerta = generar_alerta(resultado_bien) # Usa el return de la función anterior
print(f'Alerta: {alerta}')

```

**3.5 Scope (Variables locales y globales)**  
 El scope define desde donde es accesable una variable. Las variables definidas dentro de una función son locales: solo existen durante la ejecución de esa función. Las variables definidas fuera son globales. En el SIGE, las constantes de configuración (nota mínima, pesos porcentuales) se definen globalmente para que todas las funciones las usen sin que sea necesario pasarlas como parámetro.

Scope

```

=====
SCOPE: Ambito de variables en el SIGE
=====
Variables GLOBALES - configuración centralizada del SIGE
Cambiar aqui afecta todo el sistema en un solo lugar
NOTA_MINIMA_APROBACION = 10.5 # reglamento académico UNSCH
PESO_PRACTICAS = 0.30
PESO_INVESTIGACION = 0.30
PESO_EXAMEN = 0.40
SEMESTRE_ACTIVO = "2024-I"

def calcular_costo_matricula(cantidad_cursos):
 TASA_MATRICULA = 45.0 # variable LOCAL -- solo existe dentro de esta función
 costo_base = cantidad_cursos * TASA_MATRICULA
 descuento = costo_base * 0.10 # variable LOCAL
 return costo_base - descuento

print(TASA_MATRICULA) # NameError -- TASA_MATRICULA es local, no existe aquí
print(descuento) # NameError -- descuento es local

Las funciones SI pueden leer variables globales
def verificar_aprobacion(promedio):
 # Usa NOTA_MINIMA_APROBACION global sin necesidad de pasarla como parámetro
 return promedio >= NOTA_MINIMA_APROBACION

def calcular_promedio_ponderado(rp, ni, ne):
 # Usa los pesos globales -- si cambian, esta función se actualiza automáticamente

```

Ing. Pelayo Quique Bautista

5 | Página

Funciones en Python

```

return (np * PESO_PRACTICAS) + (ni * PESO_INVESTIGACION) + (ne * PESO_EXAMEN)

costo = calcular_costo_matricula(4)
print(f'Matricula por 4 cursos: S/ {costo:2f}')

prom = calcular_promedio_ponderado(15, 14, 16)
print(f'Promedio {prom} -- {Aprobado} if verificar_aprobacion(prom) else "Desaprobado"')

```

**3.6 Composición de funciones**  
 La composición de funciones es el patrón donde una función llama a otras funciones para construir operaciones más complejas. En el SIGE, la función `generar_reporte_seccion()` llama a `calcular_promedio_ponderado()` para cada estudiante, que a su vez usa `verificar_aprobacion()` y `calcular_rendimiento()`. Cada función hace UNA cosa y la hace bien.

Composicion de funciones

```

=====
COMPOSICION DE FUNCIONES: Construcción modular del SIGE
=====
Nivel 1: Funciones base (operaciones atómicas)
def calcular_promedio_ponderado(rp, ni, ne):
 """Calcula el promedio ponderado con pesos configurables"""
 if not all(0 <= nota <= 20 for nota in [rp, ni, ne]):
 raise ValueError("Notas fuera del rango 0-20: (rp, ni, ne)")
 return round((rp * pp) + (ni * pi) + (ne * pe), 2)

def clasificar_rendimiento(promedio, nota_minima=10.5):
 """Clasifica el rendimiento según la escala vigesimal UNSCH"""
 if promedio >= 17: return (Sobresaliente, 'A', Aprobado)
 elif promedio >= 14: return (Bueno, 'B', Aprobado)
 elif promedio >= nota_minima: return (Aprobado, 'C', Aprobado)
 else: return (Desaprobado, 'F', Desaprobado)

def calcular_asistencia_por(fechas_asistidas, fechas_programadas):
 """Calcula el porcentaje de asistencia usando sets"""
 if not fechas_programadas:
 return 0.0
 validas = set(fechas_asistidas) & set(fechas_programadas)
 return round(len(validas) / len(fechas_programadas) * 100, 1)

Nivel 2: Función intermedia (usa funciones base)
def procesar_estudiante(est_dict, fechas_programadas):
 """Procesa un estudiante llamando a las funciones base"""
 notas = est_dict.get('notas', (0, 0, 0))
 promedio = calcular_promedio_ponderado(*notas) # desempaqueta tupla
 cal, lei, estado = clasificar_rendimiento(promedio)
 asistencias = calcular_asistencia_por(
 est_dict.get('fechas_asistidas', set()),
 fechas_programadas
)
 return {
 'nombre': est_dict.get('nombre'),
 'promedio': promedio,
 'categoria': cal,
 'nota': lei,
 'estado': estado,
 'asistencia': asistencias
 }

```

Ing. Pelayo Quique Bautista

6 | Página

Funciones en Python

```

Nivel 3: Función principal (usa funciones intermedias)
def generar_reporte_seccion(seccion, fechas_programadas):
 """Genera el reporte completo de la sección SIGE"""
 resultados = []
 for est in seccion:
 resultado = procesar_estudiante(est, fechas_programadas)
 resultados.append(resultado)
 resultados.sort(key=lambda x: x['promedio'], reverse=True) # ordenar por promedio
 return resultados

```

#### 4. ERRORES COMUNES EN FUNCIONES

Durante la programación en pares, el navegador debe detectar estos errores antes de que el driver los cometa o inmediatamente después de que aparezcan.

**ERROR 1: Función que retorna None inesperadamente**

```

INCORRECTO: la función no tiene return en todas las ramas
def clasificar_rendimiento_MAI(promedio):
 if promedio >= 17:
 return Sobresaliente
 elif promedio >= 14:
 return Bueno
 elif promedio >= 10.5:
 return Aprobado
 # Si promedio < 10.5, la función termina sin return -- devuelve None

resultado = clasificar_rendimiento_MAI(8.0)
print(resultado) # None -- error silencioso
print(resultado.upper()) # AttributeError: 'NoneType' has no 'upper'

CORRECTO: cubrir todas las ramas con return o usar else
def clasificar_rendimiento_BIEN(promedio):
 if promedio >= 17: return Sobresaliente
 elif promedio >= 14: return Bueno
 elif promedio >= 10.5: return Aprobado
 else: return Desaprobado # cubre todos los casos

```

**ERROR 2: Parámetros posicionales en orden incorrecto**

```

def registrar_notas(codigo, curso, nota):
 print(f'Registrado: {codigo} en {curso} = {nota}')

INCORRECTO: argumentos en orden equivocado
registrar_notas('IS-402', 'Carmen', 15.0)
Salida: Registrado: IS-402 en Carmen = 15.0 y curso invertido

CORRECTO opción 1: respetar el orden
registrar_notas('notas/27150415', 'IS-402', 15.0)

CORRECTO opción 2: usar argumentos nombrados (elimina ambigüedad)
registrar_notas(nota=15.0, curso='IS-402', codigo='27150415')

```

**ERROR 3: Mutabilidad de parámetros (efecto secundario oculto)**

```

INCORRECTO: modificar una lista pasada como parámetro afecta la original

```

Ing. Pelayo Quique Bautista

7 | Página

Funciones en Python

python

Sesión 05

```
def agregar_nota_MAL(lista_notas, nueva_nota):
 lista_notas.append(nueva_nota) # modifica la lista ORIGINAL
 return lista_notas

notas_carmen = [14, 16, 15]
resultado = agregar_nota_MAL(notas_carmen, 17)
print(notas_carmen) # [14, 16, 15, 17] — la lista original fue modificada

CORRECTO: trabajar sobre una copia si no se quiere modificar el original
def agregar_nota_BIEN(lista_notas, nueva_nota):
 copia = lista_notas.copy() # trabaja sobre copia
 copia.append(nueva_nota)
 return copia

notas_pedro = [12, 11, 13]
resultado2 = agregar_nota_BIEN(notas_pedro, 15)
print(notas_pedro) # [12, 11, 13] — original sin cambios
print(resultado2) # [12, 11, 13, 15] — nueva lista con la nota agregada
```

5. SEUDOCODIGO DEL MODULO

SEUDOCODIGO

```
// MODULO: slgs_funciones.py
// Nivel 1: Funciones base (una sola responsabilidad)

FUNCION calcular_promedio_ponderado(nota_practica, nota_investigacion, nota_examen,
 nota_minima = 10.5):
 Si alguna nota < 0 O > 20 ENTONCES
 LANZAR error 'Nota fuera del rango valido 0-20'
 FIN SI
 promedio <- (nota_practica * 0.30) + (nota_investigacion * 0.30) + (nota_examen * 0.40)
 RETORNAR round(promedio, 2)
FIN FUNCION

FUNCION clasificar_rendimiento(promedio, nota_minima = 10.5):
 Si promedio >= 17 ENTONCES RETORNAR (Sobresaliente, 'A', 'Aprobado')
 Si promedio >= 14 ENTONCES RETORNAR (Bueno, 'B', 'Aprobado')
 Si promedio >= nota_minima ENTONCES RETORNAR ('Aprobado', 'C', 'Aprobado')
 SINO RETORNAR ('Desaprobado', 'F', 'Desaprobado')
FIN FUNCION

// Nivel 2: Función intermedia (llama a funciones base)
FUNCION procesar_estudiante(nombre, n1, n2, n3):
 promedio <- calcular_promedio_ponderado(n1, n2, n3)
 categoria, letra, estado <- clasificar_rendimiento(promedio)
 RETORNAR (nombre, promedio, categoria, letra, estado)
FIN FUNCION

// Nivel 3: Función principal (llama a funciones intermedias)
FUNCION generar_reporte_seccion(lista_estudiantes, notas):
 resultados <- []
 PARA CADA (nombre, n1, n2, n3) EN lista_estudiantes, notas:
 resultado <- procesar_estudiante(nombre, n1, n2, n3)
 AGREGAR resultado A resultados
 FIN PARA
 ORDENAR resultados POR promedio (descendente)
```

Ing. Pelayo Quique Bautista

8 | Página

Funciones en Python

python

Sesión 05

ESCRIBIR reporte completo  
RETORNAR resultados  
FIN FUNCION

6. PROBLEMA ABP RESUELTO 1

Implementación completa del pseudocódigo. Este es el módulo central del requerimiento de la OTI para la Semana 7. El par driver-navigator implementa primero las funciones base, luego las intermedias y finalmente la principal.

CODIGO PYTHON COMPLETO

```
=====
SISTEMA SIGE - OTI UNSCH | Sesión 5
Módulo: Funciones realizables para promedios y reportes
Desarrollado con: Programación en pares (VS Code + GitHub)
=====

CONSTANTES GLOBALES
NOTA_MINIMA = 10.5
PESO_PRAC = 0.30
PESO_INV = 0.30
PESO_EXAM = 0.40

NIVEL 1: Funciones base
def calcular_promedio_ponderado(nota_practica, nota_investigacion, nota_examen):
 """
 Calcula el promedio ponderado según reglamento UNSCH.
 Retorna: float — promedio redondeado a 2 decimales
 Lanza: ValueError si alguna nota está fuera del rango 0-20.
 """
 notas = [nota_practica, nota_investigacion, nota_examen]
 if not all(0 <= n <= 20 for n in notas):
 raise ValueError('Notas invalidas: (notas). Rango permitido: 0-20')
 return round(
 (nota_practica * PESO_PRAC) +
 (nota_investigacion * PESO_INV) +
 (nota_examen * PESO_EXAM), 2
)

def clasificar_rendimiento(promedio, nota_minima=NOTA_MINIMA):
 """
 Clasifica el rendimiento según escala vigesimal UNSCH.
 Retorna: tuple (categoria, letra, estado, estado)
 """
 if promedio >= 17: return ('Sobresaliente', 'A', 'Aprobado')
 elif promedio >= 14: return ('Bueno', 'B', 'Aprobado')
 elif promedio >= nota_minima: return ('Aprobado', 'C', 'Aprobado')
 else: return ('Desaprobado', 'F', 'Desaprobado')

def calcular_promedio_lista(lista_notas):
 """
 Calcula el promedio simple de una lista de notas.
 Retorna: float — promedio, o 0.0 si la lista está vacía.
 """
 if not lista_notas:
 return 0.0
 return round(sum(lista_notas) / len(lista_notas), 2)

Ejemplo de uso:
datos = [14, 16, 15, 17]
promedio = calcular_promedio_lista(datos)
print(promedio) # 15.5
categoria, letra, estado = clasificar_rendimiento(promedio)
print(categoria, letra, estado) # Aprobado, C, Aprobado
```

Ing. Pelayo Quique Bautista

9 | Página

Funciones en Python

python

Sesión 05

```
def calcular_promedio_ponderado_creditos(cursos_notas_creditos):
 """
 Calcula el promedio ponderado por créditos (para orden de merito).
 Parametro: lista de tuplas (nombre_curso, nota, creditos)
 Retorna: float — promedio ponderado.
 """
 suma_cred = sum(nota * cred for _, nota, cred in cursos_notas_creditos)
 total_cred = sum(cred for _, _, cred in cursos_notas_creditos)
 return round(suma_cred / total_cred, 2) if total_cred > 0 else 0.0

NIVEL 2: Función intermedia
def procesar_estudiante(nombre, nota_prac, nota_inv, nota_exam, codigo=""):
 """
 Procesa un estudiante llamando a las funciones base.
 Retorna: dict con todos los datos calculados.
 """
 promedio = calcular_promedio_ponderado(nota_prac, nota_inv, nota_exam)
 categoria, letra, estado = clasificar_rendimiento(promedio)
 return {
 'codigo': codigo,
 'nombre': nombre,
 'prac': nota_prac,
 'inv': nota_inv,
 'exam': nota_exam,
 'promedio': promedio,
 'categoria': categoria,
 'letra': letra,
 'estado': estado
 }

NIVEL 3: Función principal
def generar_reporte_seccion(lista_estudiantes, nombre_seccion="IS-101"):
 """
 Genera el reporte completo de una sección.
 Parametro: lista de tuplas (nombre, nota_prac, nota_inv, nota_exam)
 Retorna: lista de dicts con resultados, ordenados por promedio.
 """
 resultados = []
 for nombre, rp, ri, re in lista_estudiantes:
 resultado = procesar_estudiante(nombre, rp, ri, re)
 resultados.append(resultado)

 # Ordenar por promedio descendente (lambda — funcion anonima)
 resultados.sort(key=lambda x: x['promedio'], reverse=True)

 # Encabezado del reporte
 print("\n")
 print(f" REPORTE DE RENDIMIENTO — (nombre, seccion) | SIGE UNSCH")
 print(f"{'-' * 65}")
 print(f"{'ID' * 4} {'Nombre' * 40} {'Promedio' * 10} {'Categoría' * 10} {'Letra' * 10} {'Estado' * 10}")
 print(f"{'-' * 65}")

 aprobados = 0
 for i, r in enumerate(resultados, 1):
 print(f"{i:4} {r['nombre'] * 36} {r['promedio']:10.2f} {r['categoria']:10} {r['letra']:10} {r['estado']}")
 if (i % 5) == 0:
 aprobados += 1
 print(f"{'-' * 65}")
```

Ing. Pelayo Quique Bautista

10 | Página

Funciones en Python

python

Sesión 05

```
Estadísticas de sección
total = len(resultados)
promedios = [r['promedio'] for r in resultados]
print("\n")
if promedios:
 print(Promedio sección : [calcular_promedio_lista(promedios):2f])
 print(Aprobados : [aprobados/total] (aprobados/total*100, 10%))
 print(Mejor estudiante : [resultados[0]['nombre']] (resultados[0]['promedio']))
 return resultados

EJECUCION PRINCIPAL: Datos de la sección IS-101
datos_is101 = [
 ('Carmen Quispe', 15.0, 14.5, 16.0),
 ('Pedro Cardenas', 12.0, 11.5, 10.0),
 ('Maria Huaman', 18.0, 17.5, 19.0),
 ('Jorge Cuente', 8.5, 9.0, 7.5),
 ('Ana Palomino', 14.0, 15.0, 16.5),
 ('Luis Quispe', 10.5, 10.5, 10.5),
 ('Rosa Sultar', 17.0, 16.0, 18.0),
]

resultados = generar_reporte_seccion(datos_is101)

PRUEBA DEL TESTER: Casos limite
print()
print("==== CASOS DE PRUEBA DEL TESTER ===")
print(Promedio (10.5,10.5,10.5): [calcular_promedio_ponderado(10.5,10.5,10.5)])
print(Clasificacion 10.5 : [clasificar_rendimiento(10.5)])
print(Clasificacion 10.4 : [clasificar_rendimiento(10.4)])
print(Lista vacia : [calcular_promedio_lista([])])

try:
 calcular_promedio_ponderado(-1, 15, 16) # nota invalida
except ValueError as e:
 print(f"ValueError correctamente capturado: {e}")
```

8. EJERCICIOS PROPUESTOS

Trabajen en parejas con rotación de roles cada 15 minutos. Al terminar cada función hagan commit a GitHub. El tester diseña los casos de prueba antes de ejecutar. Al finalizar, presenten el módulo ante los demás grupos.

Ejercicio 1: Módulo de validación de datos del SIGE (Nivel básico)

El sistema SIGE recibe datos del formulario web que deben validarse antes de procesarse. Implementa las siguientes funciones de validación:

- validar\_codigo\_unsch(codigo): Retorna True si el código tiene 8 dígitos y empieza con '27'.
- validar\_notas(notas): Retorna True si la nota es un float entre 0 y 20 inclusive.
- validar\_porcentaje\_asistencia(pct): Retorna True si el porcentaje es float entre 0 y 100.
- validar\_ficha\_estudiante(ficha\_dict): Recibe un dict con código, nombre, notas y llama a las funciones anteriores. Retorna (True, ") si todo es válido, o (False, mensaje de error) si algo falla.

Casos del tester: Código 27150415 (válido), Código 12345678 (inválido — no empieza con 27), Nota 10.5 (válido), Nota 25 (inválido), Nota -1 (inválido), Ficha completa válida, Ficha con nota inválida.

Ejercicio 2: Módulo de generación de reportes multiformato (Nivel intermedio)

La OTI necesita exportar los reportes del SIGE en diferentes formatos según el destinatario. Implementa:

- formatear\_reporte\_texto(resultados): Recibe la lista de dicts de generar\_reporte\_seccion() y retorna un string formateado para consola.
- formatear\_reporte\_csv(resultados): Retorna un string en formato CSV con encabezados: Código,Nombre,Promedio,Categoría,Estado.

Ing. Pelayo Quique Bautista

11 | Página

python

Sesión 05

Funciones en Python

- 7. `formatear_reporte_resumen(resultado)`: Retorna un dict con `total_estudiantes`, `promedio_seccion`, `aprobados`, `desaprobados`, `mejor_promedio`, `peor_promedio`, `estudiantes_sobresalientes`.
- 8. `generar_reporte_completo(datos_seccion)`: Función principal que llama a `generar_reporte_seccion()` y luego a las tres funciones de formato, devolviendo los tres formatos.

**Cases del tester:** Sección con todos aprobados. Sección con todos desaprobados. Sección con un solo estudiante. Lista vacía (sin estudiantes).

**Ejercicio 3. Módulo integrador con todas las sesiones (Nivel avanzado)**

Integrando funciones de todas las sesiones anteriores, el par construye el módulo más completo del proyecto SIIGE. El sistema debe:

- 9. Implementar la función `generar_ficha_completa(codigo, bd_notas, bd_asistencia, fechas_prog)` que llame a `calcular_prom_ponderado_credito()`, `calcular_asistencia_pct()` y `clasificar_rendimiento()` para generar la ficha integral del estudiante.
- 10. Implementar `buscar_estudiantes_resgo(secciones, dni)` que recorra múltiples secciones y devuelva una lista de estudiantes con promedio < 10.5 O asistencia < 70%, incluyendo ambos criterios.
- 11. Implementar `comparar_secciones(seccion1, seccion2)` que use las funciones de estadísticas para comparar dos secciones y determinar cual tiene mejor rendimiento general.
- 12. Hacer al menos 4 commits a GitHub: funciones base, funciones intermedias, función principal y pruebas del tester.

**Reto de programación en pares:** Al finalizar, el par debe demostrar a los evaluadores que: (1) cada función tiene exactamente una responsabilidad, (2) ninguna función tiene efectos secundarios ocultos (listas o diccionarios externos), (3) todas las funciones usan `return` en lugar de `solo print()`, y (4) los casos límite están cubiertos con validaciones explícitas.

python

Sesión 06

Programación Orientada a Objetos I (POO) en Python

| Programación Orientada a Objetos I: Clases y Objetos  |                                         | SESIÓN 06  |                |
|-------------------------------------------------------|-----------------------------------------|------------|----------------|
| Modelamiento del Sistema de Gestión Estudiantil UNSCH |                                         |            |                |
| Asignatura:                                           | Algoritmos y Estructuras de Datos       | Semestre:  | 2024-II        |
| Docente:                                              | Ing. Pelayo Quirope Bautista            | Duración:  | 90 minutos     |
| Metodología:                                          | Aprendizaje Basado en Problemas (ABP)   | Fases ABP: | Fases 3 y 4    |
| Herramientas:                                         | Visual Studio Code, GitHub colaborativo | Semana:    | Semana 8 de 13 |

**1. FASES 3 Y 4 : INVESTIGACIÓN, CONSTRUCCIÓN Y PRESENTACIÓN**

|                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FASE 3</b><br>Investigación y Construcción | Los equipos investigan autónomamente los conceptos de POO: clase, objeto, atributo, método y herencia. Diseñan el diagrama de clases del sistema de gestión estudiantil UNSCH antes de codificar. Identifican que atributos y métodos de las funciones de la Sesión 5 pueden convertirse en clases. Competencia desarrollada: HC - implementación del sistema de gestión con clases en Python.                                                                                         |
| <b>FASE 4</b><br>Desarrollo y Presentación    | Los equipos codifican la jerarquía de clases en VS Code y hacen commit a GitHub al terminar cada clase. El tester verifica con casos límite: objeto sin atributos opcionales, clave inexistente, herencia incorrecta. Al finalizar, el equipo presenta su sistema de gestión ante los demás grupos con revisión de código entre pares. Competencia desarrollada: DP - detección de <code>AttributeError</code> , <code>TypeError</code> y errores de <code>super().__init__()</code> . |

**1.1 Nuevo requerimiento del proyecto SIIGE**

**Situación problemática:**

Las funciones del módulo SIIGE (Sesión 5) funcionan bien de forma individual, pero el jefe de la OTI presenta un nuevo requerimiento de mayor escala: el sistema debe manejar simultáneamente a cientos de estudiantes, decenas de docentes y múltiples cursos. Con funciones sueltas que operan sobre diccionarios, es difícil garantizar que los datos de un estudiante no interfieran con los de otro cuando el sistema crece.

El requerimiento es claro: "Cada entidad real del sistema - Estudiante, Docente, Curso - debe ser un objeto independiente que agrupe sus propios datos y comportamientos. Un objeto Estudiante debe saber calcular su propio promedio, registrar sus propias notas y reportar su propio estado académico. No debe depender de funciones externas para hacer lo que le corresponde."

Esta es la transición del pensamiento procedimental (funciones) al pensamiento orientado a objetos (clases), el paradigma dominante en el desarrollo de software profesional actual.

**🎯 Tu misión:** Modelar el sistema universitario UNSCH con clases, implementar la jerarquía `Persona -> Estudiante/Docente`, y conectar los objetos en un sistema funcional con GitHub.

python

Sesión 06

Programación Orientada a Objetos I (POO) en Python

Docente: id\_doc, cursos | asignar\_curso(), calificar\_estudiante()  
Curso: nombre, codigo, creditos, estudiantes | agregar\_estudiante(), generar\_reporte()

**2. COMPETENCIAS A DESARROLLAR**

| Pensamiento Logico-Algorítmico (PLA)                                                                                                                             | Habilidades de Codificación (HC)                                                                                                                                                                                                                                        | Depuración y Prueba (DP)                                                                                                                                                                                                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Modela las entidades del sistema universitario UNSCH (Estudiante, Docente, Curso) como clases, identificando atributos, métodos y relaciones antes de codificar. | Implementa en VS Code las clases <code>Persona</code> , <code>Estudiante</code> , <code>Curso</code> y <code>Docente</code> con constructor <code>__init__()</code> , self, métodos de instancia y herencia usando <code>super().__init__()</code> con commit a GitHub. | Detecta y corrige <code>AttributeError</code> al acceder a atributos inexistentes. <code>TypeError</code> por falta de self, y errores de <code>super().__init__()</code> en la cadena de herencia; verifica con casos límite. |

**3. CONTENIDO CONCEPTUAL**

**1.1 ¿Por qué POO? La limitación de las funciones con diccionarios**

En la Sesión 5 el sistema SIIGE usa funciones que reciben diccionarios como parámetros. Este enfoque funciona para problemas pequeños, pero cuando el sistema crece, sus datos y las funciones que los procesan viven en lugares distintos del código y es fácil mezclarlos. La POO resuelve este problema agrupando datos (atributos) y comportamientos (métodos) en una sola unidad: el objeto.

**Funciones vs. Clases**

```
=====
ENFOQUE PROCEDIMENTAL (Sesión 5): datos y funciones separados
=====
Datos del estudiante en un diccionario
est_dict = {'codigo': '27150415', 'nombre': 'Carmen Quispe', 'notas': []}

Función que opera sobre el diccionario — separada de los datos
def registrar_nota_proces(est_dict, curso, nota):
 if curso not in est_dict['notas']:
 est_dict['notas'][curso] = []
 est_dict['notas'][curso].append(nota)

Para saber que operaciones existen, hay que buscar en TODO el código
registrar_nota_proces(est_dict, 'IS-402', 15)

print()

=====
ENFOQUE POO (Sesión 6): datos y métodos juntos en el objeto
=====
class Estudiante:
 def __init__(self, codigo, nombre, carrera):
 self.codigo = codigo
 self.nombre = nombre
 self.carrera = carrera
 self.notas = {} # El objeto guarda sus propios datos

 def registrar_nota(self, curso, nota): # El objeto conoce su operación
 if curso not in self.notas:
 self.notas[curso] = []
 self.notas[curso].append(nota)

Uso limpio: el objeto sabe lo que hace
est_carmen = Estudiante('27150415', 'Carmen Quispe', 'Ing. Sistemas')
```

python

Sesión 06

Programación Orientada a Objetos I (POO) en Python

```
est_carmen.registrar_nota('IS-402', 15) # el objeto ejecuta su propio metodo
print(est_carmen.notas)
```

**3.2 Conceptos clave de POO aplicados al sistema UNSCH**

| Concepto                | Definición                                                                                       | Analogía en el sistema UNSCH                                                                                                            |
|-------------------------|--------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Clase                   | Plantilla o molde que define atributos y métodos comunes para un tipo de objeto.                 | Estudiante define qué todo alumno tiene: código, nombre, carrera, notas y puede calcular su promedio.                                   |
| Objeto                  | Instancia concreta de una clase con sus propios valores en los atributos.                        | <code>est_carmen = Estudiante('27150415', 'Carmen Quispe', 'Ing. Sistemas')</code> es un objeto real del SIIGE.                         |
| Atributo                | Variable que pertenece a un objeto. Se accede con <code>self.atributo</code> dentro de la clase. | <code>self.nombre</code> , <code>self.carrera</code> , <code>self.notas</code> — datos propios de cada estudiante del SIIGE.            |
| Método                  | Función definida dentro de una clase que opera sobre los datos del objeto (self).                | <code>calcular_promedio()</code> , <code>registrar_nota()</code> , <code>verificar_aprobacion()</code> — acciones del Estudiante.       |
| Herencia                | Una clase hija reutiliza atributos y métodos de una clase padre sin repetir código.              | Estudiante y Docente heredan de <code>Persona</code> : reutilizan <code>nombre</code> , <code>dni</code> y <code>presentarse()</code> . |
| <code>super()</code>    | Llama al constructor o método de la clase padre desde la clase hija.                             | <code>super().__init__(nombre, dni)</code> inicializa los atributos de <code>Persona</code> desde <code>Estudiante</code> .             |
| <code>__str__()</code>  | Método especial que define la representación en texto del objeto al usar <code>print()</code> .  | <code>print(est_carmen)</code> muestra automáticamente la ficha del estudiante.                                                         |
| <code>__init__()</code> | Constructor: se ejecuta automáticamente al crear un objeto con Clase().                          | Estudiante('27150415', 'Carmen', 'Ing. Sistemas') llama automáticamente a <code>__init__</code> .                                       |

**3.3 Clase Persona (Base del sistema universitario UNSCH)**

La clase `Persona` encapsula los atributos comunes a todos los miembros de la comunidad universitaria: `nombre` y `DNI`. Al crear esta clase base, `Estudiante` y `Docente` pueden heredar estos atributos sin repetir código, siguiendo el principio DRY de la Sesión 5.

**Clase Persona**

```
=====
CLASE PERSONA: Base de la jerarquía de clases UNSCH
=====
class Persona:
 """
 Clase base para todos los miembros de la comunidad UNSCH.
 Atributos: nombre (str), dni (int)
 Métodos: presentarse(), __str__()
 """
 institucion = "UNSCH" # atributo de CLASE — compartido por todos los objetos

 def __init__(self, nombre, dni):
 """Constructor — inicializa los atributos de instancia."""
 self.nombre = nombre # atributo de INSTANCIA — propio de cada objeto
 self.dni = dni

 def presentarse(self):
 """Método de instancia — usa self para acceder a sus atributos."""
 print(f'Hola, soy {self.nombre} | DNI: {self.dni} | {self.institucion}')

 def __str__(self):
 """Representación en texto del objeto — se llama con print(objeto)"""
```



```

 return f'Persona: {self.nombre}'

Crear objetos de Persona
persona1 = Persona('Luis Paz Quijse', 12345678)
persona2 = Persona('Ana Flores Coate', 37654321)

persona1.presentarse() # Usa self.nombre y self.institucion
persona2.presentarse() # llama a __str__ automáticamente
print(persona1)

Atributo de clase vs. atributo de instancia
print(Persona.institucion) # acceso por la clase
print(persona1.institucion) # acceso por el objeto (hereda del de clase)
print(persona1.nombre) # atributo de instancia - solo de persona1

```

#### 3.4 Clase Estudiante (Hereda de Persona, agrega lógica del SIGE)

La clase Estudiante hereda de Persona usando la sintaxis class Estudiante(Persona). El constructor de Estudiante llama a super().\_\_init\_\_(nombre, dni) para inicializar los atributos del padre sin repetirlos, y luego inicializa los atributos propios del estudiante: código, carrera y el diccionario de notas.

```

class Estudiante:
 # CLASE ESTUDIANTE - Hereda de Persona, lógica académica SIGE
 # hereda todos los atributos y métodos de Persona
 def __init__(self, nombre, dni, id_estudiante, carrera):
 super().__init__(nombre, dni) # inicializa nombre y dni de Persona
 self.id_estudiante = id_estudiante
 self.carrera = carrera
 self.notas = {} # {codigo, curso: [n1, n2, n3, ...]}
 self.fecha_asist = {} # set de fechas asistidas (Semana 4)

 def registrar_notas(self, codigo, curso, nota):
 """Agrega una nota al historial académico del estudiante"""
 if not (0 <= nota <= 20):
 raise ValueError(f'Nota {nota} fuera del rango 0-20 (UNSCH)')
 if codigo, curso not in self.notas:
 self.notas[codigo, curso] = []
 self.notas[codigo, curso].append(nota)

 def calcular_promedio(self, codigo, curso):
 """Calcula el promedio de un curso específico. Retorna 0.0 si no hay notas"""
 notas, curso = self.notas.get((codigo, curso), [])
 if not notas:
 return 0.0
 return round(sum(notas) / len(notas), 2)

 def calcular_promedio_ponderado(self, pesos=(0.30, 0.30, 0.40)):
 """Calcula el promedio ponderado de todos los cursos"""

```

Ing. Pelayo Quijse Bautista

4 | Página

```

Usa las funciones base de la Sesión 5.

def verificar_aprobacion(self, codigo, curso):
 """Retorna True si el estudiante aprobó el curso"""
 return self.calcular_promedio(codigo, curso) >= self.NOTA_MINIMA

def registrar_asistencia(self, fecha):
 """Registra una fecha de asistencia (set - no duplica)"""
 self.fecha_asist.add(fecha)

def calcular_pct_asistencia(self, fechas_programadas):
 """Calcula el porcentaje de asistencia"""
 if not fechas_programadas:
 return 0.0
 validas = self.fecha_asist & set(fechas_programadas)
 return round(len(validas) / len(fechas_programadas) * 100, 1)

def __str__(self):
 """Ficha completa del estudiante para imprimir"""
 lines = [
 f'===== FICHA SIGE ----- {self.id_estudiante} =====',
 f'Nombre: {self.nombre}',
 f'Carrera: {self.carrera}',
 f'DNI: {self.dni}',
]
 for curso, notas in self.notas.items():
 prom = self.calcular_promedio(curso)
 estado = 'Aprobado' if self.verificar_aprobacion(curso) else 'Desaprobado'
 lines.append(f'Curso: {codigo} - {curso} - {prom} ({estado})')
 return '\n'.join(lines)

Uso del objeto Estudiante
est_carmen = Estudiante('Carmen Quijse Human', 71764255, 27150415, 'Ing. Sistemas')

Método heredado de Persona
est_carmen.presentarse()

Métodos propios de Estudiante
est_carmen.registrar_notas('S-402', 15.0)
est_carmen.registrar_notas('S-402', 17.0)
est_carmen.registrar_notas('S-403', 14.5)
est_carmen.registrar_asistencia('2024-08-12')
est_carmen.registrar_asistencia('2024-08-14')

print(est_carmen) # llama a __str__ automáticamente
print(f'Aprobó S-402: {est_carmen.verificar_aprobacion("S-402")}')

fechas_prog = ['2024-08-12', '2024-08-14', '2024-08-19', '2024-08-21']
print(f'Asistencia: {est_carmen.calcular_pct_asistencia(fechas_prog)}%')

```

#### 3.5 Clase Docente (Hereda de Persona, gestiona cursos)

Ing. Pelayo Quijse Bautista 5 | Página

```

CLASE DOCENTE: Hereda de Persona, gestiona cursos
class Docente(Persona):
 # hereda todos los atributos y métodos de Persona
 def __init__(self, nombre, dni, id_docente, especialidad):
 super().__init__(nombre, dni) # hereda nombre y dni
 self.id_docente = id_docente
 self.especialidad = especialidad
 self.cursos_asignados = [] # lista de objetos Curso

 def asignar_curso(self, curso):
 """Asigna un objeto Curso al docente"""
 if curso not in self.cursos_asignados:
 self.cursos_asignados.append(curso)
 print(f'Curso {curso.nombre} asignado a {self.nombre}')

 def calificar_estudiante(self, estudiante, codigo, curso, nota):
 """Registra una nota a un estudiante en un curso específico"""
 estudiante.registrar_notas(codigo, curso, nota)
 print(f'{self.nombre} registró {nota} a {estudiante.nombre} en {codigo, curso}')

 def generar_reporte_cursos(self):
 """Muestra el listado de cursos asignados"""
 print(f'Docente: {self.nombre} ({self.id_docente})')
 if self.cursos_asignados:
 for curso in self.cursos_asignados:
 print(f' - {curso.nombre} (curso código) - {len(curso.estudiantes)} estudiantes')
 else:
 print('Sin cursos asignados')

 def __str__(self):
 cursos = [c.nombre for c in self.cursos_asignados]
 return f'Docente: {self.nombre} | ID: {self.id_docente} | Cursos: {cursos}'

```

#### 3.6 Clase Curso (Entidad independiente del SIGE)

```

class Curso:
 # CLASE CURSO - Entidad independiente, no hereda de Persona
 def __init__(self, nombre, codigo, credits, semestre='2024-I'):
 self.nombre = nombre
 self.codigo = codigo
 self.credits = credits
 self.semestre = semestre
 self.docente = None # objeto Docente (se asigna después)

```

Ing. Pelayo Quijse Bautista

6 | Página

```

self.estudiantes = [] # lista de objetos Estudiante
self.fecha_programada = self.fecha

def agregar_estudiante(self, estudiante):
 """Agrega un objeto Estudiante al curso"""
 if estudiante not in self.estudiantes:
 self.estudiantes.append(estudiante)
 print(f'{estudiante.nombre} inscrito en {self.nombre}')

def generar_reporte_seccion(self):
 """Genera el reporte de la sección llamando a métodos de cada Estudiante"""
 print(f'===== REPORTE - {self.nombre} ({self.codigo}) =====')
 if self.estudiantes:
 print(f'Docente: {self.docente.nombre}')
 print(f'({self.nombre} <22) ({self.codigo} <12) Estado')
 print(f'({self.codigo} <55)')

 aprobados = 0
 for est in self.estudiantes:
 prom = est.calcular_promedio(self.codigo)
 asist = est.calcular_pct_asistencia(self.fecha_programada)
 estado = 'Aprobado' if self.verificar_aprobacion(self.codigo) else 'Desaprobado'
 if estado == 'Aprobado':
 aprobados += 1
 print(f'{est.nombre} <22) {prom} <10) {asist} <12) {estado}')

 total = len(self.estudiantes)
 print(f'({self.codigo} <55)')
 print(f'Total: {total} | Aprobados: {aprobados} ({aprobados/total*100:.1f}%)')

 def __str__(self):
 return f'Curso: {self.nombre} ({self.codigo}) | {self.credits} credits | {len(self.estudiantes)} estudiantes'

```

#### 3.7 Errores comunes en POO

##### ERROR 1: TypeError: falta self en la definición del método

```

INCORRECTO: olvidar self - Python lo interpreta como función sin objeto
class EstudianteMal:
 def calcular_promedio(): # falta self
 return sum(self.notas) / len(self.notas) # también falta: self no definido

est = EstudianteMal()
est.calcular_promedio() -> TypeError: calcular_promedio() takes 0 positional arguments but 1 was given

CORRECTO: self siempre es el primer parámetro
class EstudianteBien:
 def __init__(self):
 self.notas = []
 def calcular_promedio(self): # self obligatorio
 if not self.notas: return 0.0
 return sum(self.notas) / len(self.notas)

```

##### ERROR 2 - AttributeError: acceder a atributo no definido en ...\_\_init\_\_

Ing. Pelayo Quijse Bautista

7 | Página

python

Sesión 06

```
INCORRECTO: atributo no inicializado en __init__
class EstudianteMal:
 def __init__(self, nombre):
 self.nombre = nombre
 # No se define self.notas

 def registrar_notas(self, curso, nota):
 self.notas[curso] = nota # AttributeError: 'EstudianteMal' has no attribute 'notas'

CORRECTO: inicializar todos los atributos en __init__
class EstudianteBien:
 def __init__(self, nombre):
 self.nombre = nombre
 self.notas = {} # siempre inicializar con valor vacío
```

ERROR 3: Error al olvidar super().\_\_init\_\_() en la clase hija

```
INCORRECTO: no llamar a super().__init__()
class EstudianteSinSuper(Persona):
 def __init__(self, nombre, dni, id_est):
 # No llama a super().__init__() -> self.nombre y self.dni NO existen
 self.id_est = id_est

est = EstudianteSinSuper('Ana', '123', '27001')
est.presentarse() -> AttributeError: 'EstudianteSinSuper' has no attribute 'nombre'

CORRECTO: siempre llamar a super().__init__() con los parámetros del padre
class EstudianteConSuper(Persona):
 def __init__(self, nombre, dni, id_est):
 super().__init__(nombre, dni) # inicializa atributos de Persona
 self.id_est = id_est

est2 = EstudianteConSuper('Ana', '123', '27001')
est2.presentarse() # funciona correctamente
```

4. PROBLEMA ABP RESUELTO 1

Implementación completa de la jerarquía de clases del SIGE. El equipo construye las cuatro clases y las conecta en un sistema funcional que integra toda la lógica de las sesiones anteriores.

CODIGO PYTHON COMPLETO

```
=====
SISTEMA SIGE - OTI UNSCH | Sesión 6
Módulo: Sistema de gestión con POO
=====
Jerarquía: Persona -> Estudiante / Docente / Curso
=====
[Las clases Persona, Estudiante, Docente y Curso van aquí]
(definidas en las sesiones 3 a 3.6)
EJECUCION: Construir el sistema del SIGE con objetos
print("==== CONSTRUYENDO EL SISTEMA SIGE CON POO =====")
print()

Crear objetos Estudiante
est_carmen = Estudiante('Carmen Quispe Huanan', '71764255', '27150415', 'Ing. Sistemas')
est_pedro = Estudiante('Pedro Caceres Lopez', '31654321', '27150416', 'Ing. Sistemas')
est_maria = Estudiante('Maria Huanan Quispe', '11223344', '27150417', 'Ing. Sistemas')
```

Ing. Pelayo Quispe Bautista

8 | Página

python

Sesión 06

```
est_jorge = Estudiante('Jorge Coente Palomino', '55667788', '27150418', 'Ing. Sistemas')

Crear objeto Docente
doc_pelayo = Docente('Pelayo Quispe Bautista', '12345678', '0001', 'Algoritmos y Estructuras')

Crear objeto Curso
curso_algo = Curso('Algoritmos y Estructuras de Datos', 'IS-402', 4)

Inscribir estudiantes al curso
for est in [est_carmen, est_pedro, est_maria, est_jorge]:
 curso.algo.agregar_estudiante(est)

Asignar docente al curso
doc_pelayo.asignar_curso(curso.algo)
curso.algo.docente = doc_pelayo

Registrar fechas de clase programadas
for fecha in ['2024-08-12', '2024-08-14', '2024-08-19', '2024-08-21']:
 curso.algo.agregar_fecha_clase(fecha)

Docente califica a los estudiantes
print("\n--- Registro de notas ---")
doc_pelayo.calificar_estudiante(est_carmen, 'IS-402', 15.0)
doc_pelayo.calificar_estudiante(est_carmen, 'IS-402', 17.0)
doc_pelayo.calificar_estudiante(est_pedro, 'IS-402', 12.0)
doc_pelayo.calificar_estudiante(est_pedro, 'IS-402', 9.0)
doc_pelayo.calificar_estudiante(est_maria, 'IS-402', 19.0)
doc_pelayo.calificar_estudiante(est_maria, 'IS-402', 18.5)
doc_pelayo.calificar_estudiante(est_jorge, 'IS-402', 8.0)
doc_pelayo.calificar_estudiante(est_jorge, 'IS-402', 7.5)

Registrar asistencias
for fecha in ['2024-08-12', '2024-08-14', '2024-08-19']:
 est_carmen.registrar_asistencia(fecha)
for fecha in ['2024-08-12', '2024-08-19']:
 est_pedro.registrar_asistencia(fecha)
for fecha in ['2024-08-12', '2024-08-14', '2024-08-19', '2024-08-21']:
 est_maria.registrar_asistencia(fecha)
est_jorge.registrar_asistencia(est_jorge, '2024-08-12')

Fichas individuales
print()
print(est_carmen) # llama a __str__

Reporte de la sección
curso_algo.generar_reporte_seccion()

Reporte del docente
print()
doc_pelayo.generar_reporte_cursos()

PRUEBA DEL TESTER: Casos límite
print("Y==== CASOS DE PRUEBA =====")
Promedio de curso sin notas
print("Promedio sin notas : (est_carmen.calcular_promedio('IS-402'))")
Asistencia con set vacío
print("Asistencia sin datos: (est_jorge.calcular_pct_asistencia(set()))")
```

Ing. Pelayo Quispe Bautista

9 | Página

python

Sesión 06

```
Nota invalida
try:
 est_carmen.registrar_notas('IS-402', 25) # fuera de rango
except ValueError as e:
 print(f'ValorError capturado: {e}')
```

6. EJERCICIOS PROPUESTOS

Implementar cada ejercicio en VS Code con commits a GitHub al terminar cada clase. El tester verifica con los casos indicados y al finalizar el equipo presenta con revisión de código entre pares.

**Ejercicio 1: Clase Biblioteca con composición (Nivel básico)**  
La Biblioteca Central de la UNSCH necesita un sistema de préstamos. Diseña e implementa:

- Clase Libro (título, autor, código, cantidad, total, cantidad disponible, Métodos: prestar() que reduce disponibles, devolver() que los aumenta, esta\_disponible() que retorna True/False, \_\_str\_\_())
- Clase Biblioteca: nombre, libros (list(código: Libro)). Métodos: agregar\_libro(), buscar\_libro(código), prestar\_libro(código, nombre\_usuario), devolver\_libro(código), generar\_reporte() que muestra disponibilidad.
- Clase HistorialPrestamo: lista de tuplas (fecha, libro, usuario). Método: agregar\_registro() y mostrar\_historial().

**Cases del tester:** Prestar libro disponible. Prestar libro sin stock (debe rechazar). Devolver libro no prestado. Buscar código inexistente. Biblioteca sin libros.

**Ejercicio 2: Jerarquía de personal universitario (Nivel intermedio)**  
La UNSCH tiene diferentes tipos de personal académico. Ampliar la jerarquía Persona con:

- PersonalAdministrativo (hereda Persona): id\_adm, departamento, cargo. Método: generar\_informe\_laboral() que muestra años de servicio.
- Investigador (hereda Docente): linea\_investigacion, proyectos (list). Métodos: agregar\_proyecto(), calcular\_indice\_produccion() que retorna proyectos / años\_servicio.
- EstudiantePosgrado (hereda Estudiante): nivel ('Maestría/Doctorado'), director, tesis (objeto Docente), título, tesis. Método: verificar\_aprobacion() que sobrescribe el de Estudiante usando NOTA\_MINIMA = 14.0 (más alta que aprobado).

**Cases del tester:** EstudiantePosgrado con promedio 13.5 debe ser Desaprobado (mínimo es 14). EstudiantePosgrado con 13.5 debe ser Aprobado. Investigador sin proyectos.

**Ejercicio 3: Sistema integrador: todas las clases del SIGE (Nivel avanzado)**  
Construyen el sistema SIGE completo integrando todas las clases de esta sesión con las funciones de la Sesión 5:

- Implementar los métodos \_\_eq\_\_, \_\_y\_\_ y \_\_hash\_\_ en Estudiante para que dos objetos con el mismo id\_estudiante sean iguales y puedan usarse en sets.
- Implementar el método to\_dict() en Estudiante, Docente y Curso que devuelva un diccionario con todos los atributos del objeto (para exportar a JSON).
- Implementar el método from\_dicts(datos) como método de clase que cree un objeto a partir de un diccionario (patrón de diseño Factory).
- Construir un sistema completo con al menos 5 estudiantes, 2 docentes y 3 cursos, generar todos los reportes y exportar los datos al formato dict.
- Hacer al menos 5 commits a GitHub: clase Persona, clase Estudiante, clase Docente, clase Curso y sistema completo con pruebas.

**Nota de presentación con revisión de código entre pares:** Otro equipo revisará el código en GitHub y señalará: (1) si algún método tiene más de una responsabilidad; (2) si hay atributos no inicializados en \_\_init\_\_; (3) si todas las clases hijas llaman a super().\_\_init\_\_() y (4) si los casos límite están cubiertos. El equipo tiene 3 minutos para responder a los comentarios de revisión.

Ing. Pelayo Quispe Bautista

10 | Página

python

Sesión 07

POO II: Herencia y Polimorfismo

Extensión del Sistema de Usuarios del Campus UNSCH

SESION 07

Semana 9

|               |                                                 |           |                                    |
|---------------|-------------------------------------------------|-----------|------------------------------------|
| Asignatura:   | Algoritmos y Estructuras de Datos               | Semestre: | 2024-II                            |
| Docente:      | Ing. Pelayo Quispe Bautista                     | Duración: | 90 minutos                         |
| Metodología:  | Aprendizaje Basado en Problemas (ABP)           | Fase ABP: | Fase 4 (Desarrollo y Presentación) |
| Herramientas: | VS Code, GitHub, Revisión de código entre pares | Semana:   | Semana 9 de 13                     |

1. FASE 4: DESARROLLO, PRESENTACIÓN Y REVISIÓN DE CÓDIGO ENTRE PARES

FASE 4  
Desarrollo y  
Presentación

Esta sesión implementa únicamente la Fase 4. El equipo ya trae el diseño de la jerarquía extendida. El desarrollador codifica las nuevas clases en VS Code; el tester diseña casos de prueba antes de ejecutar. El documentador registra cada commit en GitHub con mensajes descriptivos. Al finalizar, otro equipo revisa el código en GitHub y señala mejoras en una sesión de code review. Competencias: HC (implementación con herencia y polimorfismo) + DP (verificación con casos límite).

1.1 Nuevo requerimiento del sistema del campus UNSCH

**Situación problemática (Semana 9):**  
El sistema de gestión universitario de la Sesión 6 tiene las clases Persona, Estudiante, Docente y Curso. La OTI recibe nuevos requerimientos de la Dirección General:  
(1) La UNSCH tiene dos tipos de estudiantes con reglas académicas distintas: EstudiantePosgrado (nota mínima de aprobación: 10.5) y EstudiantePosgrado (nota mínima: 14.0 para maestría, 16.0 para doctorado). Un solo cálculo de promedio ya no es suficiente.  
(2) El campus tiene otros usuarios además de estudiantes y docentes: personal administrativo y autoridades. Todos deben poder generar su credencial universitaria de forma diferente según su rol.  
(3) El sistema debe poder procesar una lista mixta de cualquier tipo de usuario del campus y llamar al mismo método (presentarse(), generar\_credencial()) en cada uno, sin preguntar de qué tipo es cada objeto. Esto es polimorfismo.  
**Tu misión: Extender la jerarquía de clases con herencia especializada y polimorfismo para cubrir todos los tipos de usuario del campus UNSCH.**

Actividad Fase 4 (Plan de trabajo antes de codificar)

El líder coordina la extensión de la jerarquía de clases en papel antes de abrir VS Code.  
Jerarquía a implementar:  
UsuarioCampus (clase base abstracta)  
Persona (hereda UsuarioCampus)  
Estudiante (hereda Persona)  
EstudiantePosgrado (hereda Estudiante, NOTA\_MINIMA = 10.5)  
EstudiantePosgrado (hereda Estudiante, NOTA\_MINIMA = 14.0 o 16.0)  
Docente (hereda Persona)  
Administrativo (hereda UsuarioCampus directamente)  
Preguntas de análisis antes de codificar:  
1. ¿Qué método sobrescribe EstudiantePosgrado de Estudiante?  
2. ¿Cómo verificar el tipo exacto de un objeto en una lista mixta?  
3. ¿Qué pasa si una lista tiene EstudiantePosgrado y EstudiantePosgrado y se llama a verificar\_aprobacion() en cada uno con la misma nota?  
4. ¿Qué es issubclass() y para qué sirve en este contexto?

Ing. Pelayo Quispe Bautista

11 | Página

## 2 COMPETENCIAS A DESARROLLAR

| Pensamiento Lógico-Algorítmico (PLA)                                                                                                                                                        | Habilidades de Codificación (HC)                                                                                                                                                           | Depuración y Prueba (DP)                                                                                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Extiende la jerarquía de clases del campus UNSCH modelando diferentes tipos de usuarios (EstudiantePregrado, EstudiantePosgrado, Administrativo) con herencia especializada y polimorfismo. | Implementa herencia múltiple y simple, sobreeschura de métodos (override), polimorfismo en listas heterogéneas, y @property con validación; hace commit a GitHub con revisión entre pares. | Detecta y corrige errores de MRO (Method Resolution Order) en herencia múltiple, sobreeschuras incorrectas que rompan el contrato de la clase padre, y uso incorrecto de super(). |

### 3. CONTENIDO CONCEPTUAL

#### 3.1 Polimorfismo (El mismo mensaje, diferentes comportamientos)

El polimorfismo es uno de los pilares de la POO. Permite que objetos de diferentes clases respondan al mismo mensaje (mismo nombre de método) de maneras distintas, según su propia implementación. En el sistema UNSCH, verificar\_aprobacion() debe funcionar diferente para pregrado y posgrado sin que el código que lo llama necesite saber de qué tipo es el objeto.

| Concepto                       | Definición                                                                                        | Ejemplo en el campus UNSCH                                                                   |
|--------------------------------|---------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| <b>Polimorfismo</b>            | El mismo nombre de método produce resultados diferentes según el tipo de objeto que lo ejecuta.   | verificar_aprobacion() retorna True con nota >= 10.5 para pregrado y >= 14.0 para posgrado.  |
| <b>Override (sobreeschura)</b> | La clase hija redefine un método de la clase padre con comportamiento diferente.                  | EstudiantePosgrado sobrescribe verificar_aprobacion() de Estudiante.                         |
| <b>Clase abstracta</b>         | Clase que no puede instanciarse directamente; define métodos que las subclases deben implementar. | UsuarioCampus define presentarse() y generar_credencial() que cada subclase implementa.      |
| <b>Listas heterogéneas</b>     | Lista con objetos de diferentes clases que responden al mismo mensaje polimórficamente.           | Lista con EstudiantePregrado, EstudiantePosgrado y Docente: todos responden a presentarse(). |
| <b>isinstance()</b>            | Función que verifica si un objeto es instancia de una clase o de sus subclases.                   | isinstance(usuario, Estudiante) retorna True para EstudiantePregrado y EstudiantePosgrado.   |

#### Polimorfismo básico

```
Clase Estudiante de la Sesión 6 (clase padre)
class Estudiante(Personal):
 NOTA_MINIMA = 10.5 # regla por defecto

 def __init__(self, nombre, dni, id_est, carrera):
 super().__init__(nombre, dni)
 self.id_est = id_est
 self.carrera = carrera
 self.notas = []

 def calcular_promedio(self, codigo, curso):
 notas = self.notas.get(codigo, [])
 return round(sum(notas)/len(notas), 2) if notas else 0.0

 def verificar_aprobacion(self, codigo, curso):
 return self.calcular_promedio(codigo, curso) >= self.NOTA_MINIMA

 def registrar_nota(self, codigo, curso, nota):
 self.notas.setdefault(codigo, []).append(nota)
```

Ing. Pelayo Quirope Bautista

2 | Página

```
EstudiantePregrado hereda todo de Estudiante sin cambios
class EstudiantePregrado(Estudiante):
 NOTA_MINIMA = 10.5 # igual a padre, pero explícito

 def generar_credencial(self):
 return FCREDENCIAL_PREGRADO | (self.nombre) | (self.id_est) | (self.carrera)

 def __str__(self):
 return f'Pregrado: {self.nombre} | {self.id_est} | {self.id_est} | {self.carrera}'

EstudiantePosgrado SOBREScribe verificar_aprobacion()
class EstudiantePosgrado(Estudiante):
 NOTA_MINIMA = 14.0 # regla diferente para posgrado

 def __init__(self, nombre, dni, id_est, carrera, nivel='Maestría'):
 super().__init__(nombre, dni, id_est, carrera)
 self.nivel = nivel
 if nivel == 'Doctorado':
 self.NOTA_MINIMA = 16.0 # doctorado exige mas

 def generar_credencial(self):
 return FCREDENCIAL_POSGRADO | (self.nombre) | (self.id_est) | (self.nivel)

 def __str__(self):
 return f'Posgrado: {self.nombre} | {self.id_est} | {self.nivel}'

POLIMORFISMO EN ACCION
est_pre = EstudiantePregrado('Carmen Quirope', '71764250', '27150415', 'Ing. Sistemas')
est_pos = EstudiantePosgrado('Pablo Torres', '97654321', 'PG001', 'Maestría TI')
est_doc = EstudiantePosgrado('Maria Flores', '11223344', 'PG002', 'Doctorado', 'Doctorado')

for est in [est_pre, est_pos, est_doc]:
 est.registrar_nota('IS-101', 13.5)

print('Resultado con nota 13.5 en cada tipo de estudiante:')
for est in [est_pre, est_pos, est_doc]:
 aprobada = est.verificar_aprobacion('IS-101')
 print(f'{est} -> {aprobada} {aprobada}')

Mismo método, diferente resultado según el tipo de objeto
```

#### 3.2 Clase abstracta y contrato de interfaz

Una clase abstracta define métodos que las subclases DEBEN implementar. No puede instanciarse directamente. En Python se usa el módulo abc (Abstract Base Classes). En el campus UNSCH, UsuarioCampus define el contrato: todo usuario DEBE poder presentarse() y generar su credencial universitaria.

```
class abstracta UsuarioCampus:
 # Contrato para todos los usuarios UNSCH
 from abc import ABC, abstractmethod

 class UsuarioCampus(ABC):
 """
 Clase base abstracta para todos los usuarios del campus UNSCH. Define el contrato: todo usuario DEBE implementar estos métodos.
 """

 @abstractmethod
 def presentarse(self):
```

Ing. Pelayo Quirope Bautista

3 | Página

```
"""Cada tipo de usuario se presenta de forma diferente."""
pass

@abstractmethod
def generar_credencial(self):
 """Genera la credencial universitaria según el rol."""
 pass

def acceder_campus(self):
 """Método común a todos los usuarios"""
 print('Acceso concedido al campus UNSCH.')

Intentar instanciar UsuarioCampus directamente produce error
usuario = UsuarioCampus() -> TypeError: Can't instantiate abstract class

Clase Personal ahora hereda de UsuarioCampus
class Personal(UsuarioCampus):
 institucion = 'UNSCH'

 def __init__(self, nombre, dni):
 self.nombre = nombre
 self.dni = dni

 def presentarse(self): # implementa el método abstracto
 print(f'Hola, soy {self.nombre} | DNI: {self.dni} | {self.institucion}')

 def generar_credencial(self): # implementación base
 return f'CREDENCIAL | {self.nombre} | {self.dni} | {self.institucion}'

 def __str__(self):
 return f'Personal: {self.nombre} | {self.dni}'
```

#### 3.3 Clase Administrativa — Herencia directa de UsuarioCampus

El personal administrativo no es ni estudiante ni docente, pero sí es usuario del campus. Hereda directamente de UsuarioCampus e implementa sus propios métodos polimórficos.

```
class Administrativa(UsuarioCampus):
 """Representa al personal administrativo del campus UNSCH. Hereda directamente de UsuarioCampus (no de Personal)."""

 def __init__(self, nombre, dni, id_admin, departamento, cargo):
 self.nombre = nombre
 self.dni = dni
 self.id_admin = id_admin
 self.departamento = departamento
 self.cargo = cargo

 def presentarse(self): # implementación propia del abstracto
 print(f'Soy {self.nombre}, {self.cargo} en {self.departamento} | UNSCH')

 def generar_credencial(self): # credencial diferente a estudiante o docente
 return f'CREDENCIAL ADMINISTRATIVO | {self.nombre} | {self.id_admin} | {self.departamento}'

 def procesar_tramite(self, tipo, tramite):
 print(f'{self.nombre} proceso el tramite: {tipo, tramite}')
```

Ing. Pelayo Quirope Bautista

4 | Página

```
def __str__(self):
 return f'Admin: {self.nombre} | {self.cargo} | {self.departamento}'
```

#### 3.4 Polimorfismo con listas heterogéneas

La potencia del polimorfismo se muestra cuando una lista contiene objetos de diferentes clases y se puede llamar al mismo método en todos sin verificar el tipo. Esto permite escribir código que funciona con cualquier tipo de usuario futuro sin modificar la lógica existente.

#### Polimorfismo en listas heterogéneas

```
Sistema de control de acceso UNSCH
Lista con diferentes tipos de usuarios del campus UNSCH
usuarios_campus = [
 EstudiantePregrado('Carmen Quirope', '71764250', '27150415', 'Ing. Sistemas'),
 EstudiantePosgrado('Pablo Torres', '97654321', 'PG001', 'Maestría TI'),
 EstudiantePosgrado('Maria Flores', '11223344', 'PG002', 'Doctorado', 'Doctorado'),
 Administrativo('Luis Paz', '9567789', 'ADM001', 'OTI', 'Jefe de TI'),
]

POLIMORFISMO misma llamada, diferente comportamiento según el tipo
print('=== CONTROL DE ACCESO AL CAMPUS UNSCH ===')
print()

for usuario in usuarios_campus:
 usuario.presentarse() # cada tipo tiene su propia implementación
 cred = usuario.generar_credencial()
 print(f'Credencial: {cred}')
 usuario.acceder_campus() # método común heredado de UsuarioCampus
 print()

isinstance() para identificar tipos en la lista
print('=== CONTEO POR TIPO ===')
pregrado = [u for u in usuarios_campus if isinstance(u, EstudiantePregrado)]
posgrado = [u for u in usuarios_campus if isinstance(u, EstudiantePosgrado)]
admins = [u for u in usuarios_campus if isinstance(u, Administrativo)]
estudiantes = [u for u in usuarios_campus if isinstance(u, Estudiante)]

print(f'Estudiantes pregrado: {len(pregrado)}')
print(f'Estudiantes posgrado: {len(posgrado)}')
print(f'Personal administrativo: {len(admins)}')
print(f'Total estudiantes: {len(estudiantes)} (pregrado + posgrado)')

isinstance() con clase padre incluye subclases
print()
print(f'Verificación de herencia con isinstance():')
est = EstudiantePregrado('Ana Torres', '98857766', '27150420', 'Ing. Sistemas')
print(f'es EstudiantePregrado: {isinstance(est, EstudiantePregrado)}')
print(f'es Estudiante: {isinstance(est, Estudiante)}')
print(f'es Persona: {isinstance(est, Personal)}')
print(f'es UsuarioCampus: {isinstance(est, UsuarioCampus)}')
print(f'es Administrativo: {isinstance(est, Administrativa)}')
```

#### 3.5 @property (Control de acceso a atributos sensibles)

El decorador @property permite acceder a un atributo como si fuera una variable simple, pero en realidad ejecuta un método getter. Combinado con @tributo.setter permite validar los datos antes de asignarlos. En el sistema UNSCH, la nota mínima de aprobación de un EstudiantePosgrado no debería poder cambiarse a un valor inválido.

```
@property y setter
Control de la nota mínima con validación
```

Ing. Pelayo Quirope Bautista

5 | Página



Programación Orientada a Objetos II (POO) en Python  Sesión 07

```

class EstudiantePosgradoValidado(Estudiante):
 """ Estudiante de posgrado con nota minima protegida por @property. """

 def __init__(self, nombre, dni, id_est, carrera, nivel="Maestría"):
 super().__init__(nombre, dni, id_est, carrera)
 self.nivel = nivel
 self._nota_minima = 14.0 if nivel != "Doctorado" else 16.0

 @property
 def nota_minima(self):
 """Getter: retorna la nota minima actual."""
 return self._nota_minima

 @nota_minima.setter
 def nota_minima(self, valor):
 """Setter: valida que la nota minima este en rango valido (10.5 a 20)."""
 if not (10.5 <= valor <= 20):
 raise ValueError("Nota minima (valor) invalida. Debe estar entre 10.5 y 20.")
 self._nota_minima = valor
 print(f"Nota minima actualizada a {valor} para {self.nombre}")

 def verificar_aprobacion(self, codigo, curso):
 return self.calcular_promedio(codigo, curso) >= self._nota_minima

 def generar_credencial(self):
 return f"CREDENCIAL POSGRADO | {self.nombre} | {self.id_est} | {self.nivel}"

Uso del @property
est_pg = EstudiantePosgradoValidado("Rosa Canto", "12312312", "PG003", "Maestría Ed.", "Maestría")

print(f"Nota minima actual: {est_pg.nota_minima}") # usa el getter

est_pg.nota_minima = 15.0 # usa el setter con validacion
print(f"Nueva nota minima: {est_pg.nota_minima}")

try:
 est_pg.nota_minima = 5.0 # valor invalido
except ValueError as e:
 print(f"Error capturado: {e}")

```

### 3.6 Errores comunes en herencia avanzada

#### ERROR 1: Sobrescritura que rompe el contrato de la clase padre

# La clase padre define que verificar\_aprobacion() retorna bool  
# La clase hija sobrescribe y retorna string -> rompe el contrato  
class EstudianteMAL(Estudiante):  
 def verificar\_aprobacion(self, codigo, curso):  
 prom = self.calcular\_promedio(codigo, curso)  
 if prom >= 10.5:  
 return "Aprobado" # ERROR: retorna str en lugar de bool  
 return "Desaprobado"

est = EstudianteMAL("Juan", "123", "PG001", "Sistemas")  
est.registrar\_nota("IS-101", 10)  
resultado = est.verificar\_aprobacion("IS-101")  
# if resultado: -> siempre True porque "Aprobado" es truthy  
# if not resultado: -> siempre False aunque este desaprobado

Ing. Pelayo Quique Bautista

6 | Página

Programación Orientada a Objetos II (POO) en Python  Sesión 07

```

CORRECTO: respetar el tipo de retorno del padre (bool)
class EstudianteBIEN(Estudiante):
 NOTA_MINIMA = 10.5
 def verificar_aprobacion(self, codigo, curso):
 return self.calcular_promedio(codigo, curso) >= self.NOTA_MINIMA # bool

```

**ERROR 2: Olvidar llamar a super().\_\_init\_\_ en subclases profundas**

```

Con jerarquía de 3 niveles, hay que llamar super() en cada nivel
class EstudiantePosgradoMAL(Estudiante):
 def __init__(self, nombre, dni, id_est, carrera, nivel):
 # No llama a super() -> self.nombre, self.nivel NO existen
 self.nivel = nivel

est = EstudiantePosgradoMAL("Ana", "123", "PG001", "T", "Maestría")
est.registrar_nota("IS-101", 15) -> AttributeError: no attribute 'nota'

```

**CORRECTO:** llamar super() en cada nivel de la cadena

```

class EstudiantePosgradoBIEN(Estudiante):
 def __init__(self, nombre, dni, id_est, carrera, nivel):
 super().__init__(nombre, dni, id_est, carrera) # inicializa toda la cadena
 self.nivel = nivel

```

**ERROR 3: Instancias() vs type() para verificación de tipos**

```

est = EstudiantePosgrado("Ana", "123", "PG001", "Sistemas")

type() solo verifica la clase exacta, no la jerarquía
print(type(est) == Estudiante) # False (vs EstudiantePosgrado exactamente)
print(type(est) == EstudiantePosgrado) # True

isinstance() verifica la jerarquía completa -> preferido en POO
print(isinstance(est, Estudiante)) # True (es subclase de Estudiante)
print(isinstance(est, EstudiantePosgrado)) # True
print(isinstance(est, Persona)) # True (toda la cadena)
print(isinstance(est, UsuarioCampus)) # True

CONCLUSION: usar isinstance() en el SIGLE para verificación de tipos

```

### 4. PROBLEMA ABP RESUELTO

Implementación completa que extiende el sistema de la Sesión 6 con los nuevos tipos de usuario, polimorfismo y @property. El equipo hace commit a GitHub al terminar cada clase y al finalizar el sistema completo.

#### CODIGO COMPLETO

```

=====
SISTEMA CAMPUS UNSCH | Sesión 7
POO II: Herencia avanzada y polimorfismo
Jerarquía: UsuarioCampus -> Persona -> Estudiante ->
EstudiantePosgrado / EstudiantePosgrado
Administrativo
=====
from abc import ABC, abstractmethod

Las clases UsuarioCampus, Persona, Estudiante, Docente

```

Ing. Pelayo Quique Bautista

7 | Página

Programación Orientada a Objetos II (POO) en Python  Sesión 07

```

se importan del modulo sigle_sesion6.py (definidas en Sesión 6)
Aquí solo se muestran las nuevas clases y la integración
=====
NUEVAS CLASES: EstudiantePosgrado, EstudiantePosgrado, Administrativo
=====
class EstudiantePosgrado(Estudiante):
 NOTA_MINIMA = 10.5

 def generar_credencial(self):
 return f"CREDENCIAL PREGRADO | {self.nombre} | "
 f"{self.id_est} | {self.carrera}"

 def __str__(self):
 return f"[PREGRADO] {self.nombre} | {self.id_est} | "
 f"{self.carrera} | Mínimo: {self.NOTA_MINIMA}"

class EstudiantePosgrado(Estudiante):
 def __init__(self, nombre, dni, id_est, carrera, nivel="Maestría"):
 super().__init__(nombre, dni, id_est, carrera)
 self.nivel = nivel
 self._nota_minima = 16.0 if nivel == "Doctorado" else 14.0

 @property
 def nota_minima(self):
 return self._nota_minima

 @nota_minima.setter
 def nota_minima(self, valor):
 if not (10.5 <= valor <= 20):
 raise ValueError("Nota minima (valor) invalida")
 self._nota_minima = valor

 def verificar_aprobacion(self, codigo, curso):
 return self.calcular_promedio(codigo, curso) >= self._nota_minima

 def generar_credencial(self):
 return f"CREDENCIAL POSGRADO | {self.nombre} | "
 f"{self.id_est} | {self.nivel}"

 def __str__(self):
 return f"[POSGRADO {self.nivel}] {self.nombre} | {self.id_est} | "
 f"Mínimo: {self._nota_minima}"

class Administrativo(UsuarioCampus):
 def __init__(self, nombre, dni, id_admin, departamento, cargo):
 self.nombre = nombre
 self.dni = dni
 self.id_admin = id_admin
 self.departamento = departamento
 self.cargo = cargo

 def presentarse(self):
 print(f"Soy {self.nombre}, {self.cargo} | {self.departamento} | UNSCH")

 def generar_credencial(self):

```

Ing. Pelayo Quique Bautista

8 | Página

Programación Orientada a Objetos II (POO) en Python  Sesión 07

```

return f"CREDENCIAL ADMIN | {self.nombre} | "
f"{self.id_admin} | {self.departamento}"

def __str__(self):
 return f"[ADMIN] {self.nombre} | {self.cargo} | {self.departamento}"

=====
SISTEMA INTEGRADO: Registro y gestión de usuarios del campus
=====
class SistemaCampusUNSCH:
 """Gestiona todos los usuarios del campus con polimorfismo."""

 def __init__(self):
 self.usuarios = []

 def registrar_usuario(self, usuario):
 self.usuarios.append(usuario)
 print(f'Registrado: {usuario}')

 def control_acceso(self):
 print(f"CONTROL DE ACCESO AL CAMPUS UNSCH =====")
 for usuario in self.usuarios:
 usuario.presentarse()
 cred = usuario.generar_credencial()
 print(f'Credencial: {cred}')
 print()

 def generar_estadisticas(self):
 pregrado = [u for u in self.usuarios if isinstance(u, EstudiantePosgrado)]
 posgrado = [u for u in self.usuarios if isinstance(u, EstudiantePosgrado)]
 admin = [u for u in self.usuarios if isinstance(u, Administrativo)]
 print(f"===== ESTADÍSTICAS DEL CAMPUS =====")
 print(f"Estudiantes pregrado : {len(pregrado)}")
 print(f"Estudiantes posgrado : {len(posgrado)}")
 print(f"Personal administrativo: {len(admin)}")
 print(f"Total usuarios : {len(self.usuarios)}")

 def reporte_aprobacion(self, codigo, curso):
 """Solo aplica a estudiantes. Los administrativos no tienen notas."""
 print(f"===== REPORTE APROBACION (codigo, curso) =====")
 for usuario in self.usuarios:
 if isinstance(usuario, Estudiante):
 prom = usuario.calcular_promedio(codigo, curso)
 aprueba = usuario.verificar_aprobacion(codigo, curso)
 print(f" {usuario.nombre} | Prom: {prom} | Aprueba: {aprueba}")

=====
EJECUCION: Construir el sistema con polimorfismo
=====
sistema = SistemaCampusUNSCH()

est1 = EstudiantePosgrado("Carmen Quique", "71764250", "21150419", "Ing. Sistemas")
est2 = EstudiantePosgrado("Pedro Torres", "31664321", "PG001", "Maestría TI")
est3 = EstudiantePosgrado("Maria Flores", "11223344", "PG002", "Maestría Ed.")
adm1 = Administrativo("Luis Paz", "55667788", "ADM001", "OTT", "Jefe de TI")

for u in [est1, est2, est3, adm1]:

```

Ing. Pelayo Quique Bautista

9 | Página

Programación Orientada a Objetos II (POO) en Python

Sesión 07

```
sistema.registrar_usuario(u)

est1.registrar_nota("IS-402", 13.5)
est2.registrar_nota("IS-402", 13.5)
est3.registrar_nota("IS-402", 15.0)
print()
sistema.control_acceso()
sistema.generar_estadisticas()
sistema.reporte_aprobacion("IS-402")

Prueba del tester: misma nota, diferente resultado por polimorfismo
print()
prn("==== PRUEBA POLIMORFICA CON NOTAS 13.5 ===")
prn("Carmen (Pregado, min=10.5) (est1.verificar_aprobacion("IS-402"))")
prn("Pedro (Posgrado, min=14.0) (est2.verificar_aprobacion("IS-402"))")
```

5. EJERCICIOS PROPUESTOS

Implementen en VS Code con commits a GitHub. Al finalizar, otro equipo revisa el código y señala mejoras en una sesión de code review de 5 minutos.

Ejercicio 1: Jerarquía de becas con polimorfismo (Nivel básico)

La Oficina de Bienestar de la UNSCH otorga diferentes tipos de beca. Diseña e implementa:

- Clase Beca (abstracta): nombre, beca, monto, mensual, beneficiario. Método abstracto: verificar elegibilidad(estudiante). Método concreto: generar\_resolucion().
- Clase BecaAcademica(Beca): requiere promedio >= 15.0 y asistencia >= 80%.
- Clase BecaSocioeconomica(Beca): requiere ingreso\_familiar <= 500 soles y no tener deuda.
- Clase BecaDeportiva(Beca): requiere promedio >= 11.0 y ser deportista federado.
- Probar con una lista de becas y un estudiante: verificar\_elegibilidad() con polimorfismo.

Casos del tester: Estudiante elegible para beca académica. Estudiante elegible para beca socioeconómica. Estudiante elegible para ninguna. Estudiante elegible para las tres a la vez.

Ejercicio 2: Sistema de roles con permisos diferenciados (Nivel intermedio)

El sistema SIGE tiene roles con diferentes permisos. Extiende la jerarquía:

- Clase RolSIGE (abstracta): nombre, rol, permisos (list). Método abstracto: puede\_realizar(accion) que verifica si el rol tiene permiso para una acción específica.
- Clase RolEstudiante(RolSIGE): permisos = [ver\_notas, ver\_asistencia, ver\_horario].
- Clase RolDocente(RolSIGE): permisos = [ver\_notas, registrar\_notas, ver\_asistencia, generar\_reporte].
- Clase RolAdministrador(RolSIGE): permisos = [todos].
- Integrar los roles con las clases de usuarios del campus para que cada objeto tenga su rol asignado.

Casos del tester: Estudiante intentando registrar notas (debe negarse). Docente viendo horario (debe permitirse). Administrador haciendo cualquier acción (debe permitirse siempre).

Ejercicio 3: Sistema integrador con herencia múltiple niveles (Nivel avanzado)

Extiende el sistema completo del campus UNSCH integrando todo lo aprendido en POO I y II:

- Implementar InvestigadorDocente que hereda tanto de Docente como de una nueva clase Investigador (herencia múltiple). Investigador tiene linea\_investigacion y lista de proyectos.
- Implementar el método resolver\_conflicto\_herencia() que usa super() para demostrar el MRO (Method Resolution Order) de Python ante métodos con el mismo nombre en las dos clases padre.
- Agregar al SistemaCampusUNSCH un método generar\_directorio\_completo() que use polimorfismo para mostrar la ficha de cada tipo de usuario con su formato específico.
- Implementar to\_dict() y from\_dict() en todas las clases para serializar y deserializar objetos.
- Hacer al menos 6 commits a GitHub con mensajes descriptivos. El code review entre equipos evalúa: herencia correcta, polimorfismo real (no if/instance), @property con validación, y cobertura de casos límite.

Ing. Pelayo Quijpe Bautista

10 | Página

Manejo de Errores y Excepciones en Python

Sesión 08

Manejo de Errores y Excepciones en Python

Diagnostico y Corrección de Fallos en el Sistema SIGE

SESION 08

Semana 10

|               |                                              |            |                 |
|---------------|----------------------------------------------|------------|-----------------|
| Asignatura:   | Algoritmos y Estructuras de Datos            | Semestre:  | 2024-II         |
| Docente:      | Ing. Pelayo Quijpe Bautista                  | Duración:  | 90 minutos      |
| Metodología:  | Aprendizaje Basado en Problemas (ABP)        | Fases ABP: | Fases 2 y 3     |
| Herramientas: | Python Debugger, Análisis de trazas de error | Semana:    | Semana 10 de 13 |

1. FASES 2 Y 3: ORGANIZACIÓN E INVESTIGACIÓN

**FASE 2**  
**Organización del Aprendizaje**

El equipo recibe el sistema SIGE con fallos y organiza el diagnóstico antes de corregir. Identifica los tipos de excepción presentes en las trazas de error del sistema. Asigna roles: el tester reproduce los errores, el desarrollador investiga soluciones, el documentador registra cada fallo. Competencia: PLA, análisis de trazas de error y categorización de excepciones.

**FASE 3**  
**Investigación y Corrección**

El equipo investiga try-except, excepciones personalizadas y el Python Debugger. Aplica el manejo de errores para proteger cada módulo del SIGE que genera fallos. El tester verifica que el sistema ya no se detiene ante datos maliciosos del formulario web. Competencia: HC (implementación de try-except) + DP (uso del Python Debugger).

1.1 El sistema SIGE con fallos: situación real

**Situación problemática:**  
El sistema SIGE lleva tres semanas en producción en la OTI. Los operadores ingresan datos de los formularios web y el sistema se detiene abruptamente con mensajes como ValueError: could not convert string to float o KeyError: 27999999. Cada vez que esto ocurre, se pierden los datos que no habían sido guardados y el operador debe reiniciar el sistema.  
El jefe de la OTI presenta el registro de los cinco errores más frecuentes de la semana pasada:  
Error 1: Un operador escribió 'N/A' en el campo de nota. El sistema se detuvo.  
Error 2: Se bausó el código 27999999 que no existe en la base de datos.  
Error 3: Se intentó calcular el promedio de un estudiante sin notas registradas.  
Error 4: El archivo de exportación 'notas\_octubre.csv' no estaba en la carpeta.  
Error 5: Se accedió al atributo 'telefono' de un objeto Estudiante que no lo tiene.  
**Tu misión: analizar cada traza de error, clasificar el tipo de excepción y aplicar try-except para que el sistema continúe funcionando aunque ocurran estos errores.**

Actividad Fase 2: Análisis de trazas de error antes de codificar

El equipo analiza cada traza de error y responde:

**Traza 1:** ValueError: could not convert string to float: 'N/A'  
**Pregunta:** En qué línea del módulo registrar\_nota() ocurre este error?  
**Pregunta:** ¿Qué tipo de dato se intentó convertir y por qué fallo?

**Traza 2:** KeyError: 27999999  
**Pregunta:** ¿Qué diferencia hay entre dict['clave'] y dict.get('clave')?  
**Pregunta:** ¿Cómo evitar este error sin usar try-except?

**Traza 3:** ZeroDivisionError: division by zero  
**Pregunta:** ¿En qué condición del SIGE puede ocurrir división por cero?

Ing. Pelayo Quijpe Bautista

1 | Página

Manejo de Errores y Excepciones en Python

Sesión 08

Pregunta: ¿Que verificación previa evitaría este error?

**Traza 4:** FileNotFoundError: [Errno 2] No such file or directory: 'notas\_octubre.csv'  
**Pregunta:** ¿Para qué sirve el bloque finally en este caso?

**Traza 5:** AttributeError: 'Estudiante' object has no attribute 'telefono'  
**Pregunta:** ¿Cómo leer un atributo de forma segura si puede no existir?

2. COMPETENCIAS A DESARROLLAR

| Pensamiento Lógico-Algorítmico (PLA)                                                                                                             | Habilidades de Codificación (HC)                                                                                                                       | Depuración y Prueba (DP)                                                                                                                                               |
|--------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Análiza las trazas de error del sistema SIGE, categoriza los tipos de excepción y diseña la estrategia de manejo antes de codificar la solución. | Implementa estructura try-except-else-finally para proteger los módulos del SIGE; crea excepciones personalizadas para el dominio universitario UNSCH. | Lee e interpreta trazas de error completas (traceback); usa el Python Debugger para inspeccionar variables en el punto exacto del fallo; verifica con casos inválidos. |

3. CONTENIDO CONCEPTUAL: Manejo de errores y excepciones

3.1 Por qué ocurren las excepciones en el SIGE

Una excepción es un evento que interrumpe el flujo normal de un programa cuando ocurre una condición inesperada durante la ejecución. A diferencia de los errores de sintaxis que se detectan antes de ejecutar, las excepciones ocurren mientras el programa corre. En el SIGE, la mayoría provienen de datos ingresados por el operador que no tienen el formato esperado.

Sin manejo de excepciones, el primer error detiene todo el proceso. Con try-except, el programa captura el error, informa al operador y continúa con el siguiente registro. Esta diferencia es la que separa un sistema de producción robusto de un prototipo de laboratorio.

| Excepción             | Cuando ocurre en el SIGE                        | Ejemplo concreto                                          |
|-----------------------|-------------------------------------------------|-----------------------------------------------------------|
| ValueError            | Al convertir texto no numérico a float o int.   | float("dieciocho") al procesar nota desde formulario web. |
| KeyError              | Al acceder a clave inexistente en diccionario.  | base_datos["27999999"] cuando el código no existe.        |
| IndexError            | Al acceder a posición fuera del rango de lista. | notas[5] cuando la lista tiene solo 3 elementos.          |
| ZeroDivisionError     | Al calcular promedio de lista vacía.            | sum([]) / len([]) al no verificar lista vacía.            |
| TypeError             | Al operar tipos incompatibles sin conversión.   | "15.9" * 0.30 sin convertir el texto a float.             |
| AttributeError        | Al acceder a atributo inexistente en un objeto. | est.promedio cuando la clase no define ese atributo.      |
| FileNotFoundException | Al abrir archivo que no existe en disco.        | open("notas_2024.csv") si el archivo no está.             |

3.2 Como leer una traza de error (traceback)

El traceback es el mensaje de error completo que Python muestra cuando ocurre una excepción. Leerlo correctamente permite identificar en segundos dónde y por qué falló el programa, sin necesidad de revisar todo el código.

Analizando de una traza de error del SIGE

```
Ejemplo de traza que genera el SIGE al recibir 'N/A' como nota:

Traceback (most recent call last):
File "sign.py", line 45, in procesar_nota_estudiante
nota_float = float(nota_texto)
File "sign.py", line 12, in registrar_calificacion
procesar_nota_estudiante(codigo, nota_texto)
ValueError: could not convert string to float: 'N/A'

Como leer el traceback:
1. Línea de ABAJO hacia ARRIBA
2. La última línea dice QUÉ tipo de error es: ValueError
```

Ing. Pelayo Quijpe Bautista

2 | Página

Manejo de Errores y Excepciones en Python

Sesión 08

# 3. La línea 'File' más cercana al error dice DÓNDE ocurrió

# 4. El texto después de los ":" dice POR QUÉ ocurrió

# Estructura:

```
Traceback (most recent call last):
File "archivo.py", line N, in function_N:
< código que fallo >
TypeError: mensaje descriptivo < que y por que >
```

3.3 Estructura try-except-else-finally

| Bloque                | Descripción                                              | Uso en el SIGE                                              |
|-----------------------|----------------------------------------------------------|-------------------------------------------------------------|
| try                   | Contiene el código que puede lanzar una excepción.       | Conversión de nota de texto a float, acceso al diccionario. |
| except Tipo           | Captura un tipo específico de excepción.                 | except ValueError: cuando la nota no es numérica.           |
| except Exception as e | Captura cualquier excepción. Se usa como último recurso. | Para errores inesperados que no se anticiparon.             |
| else                  | Se ejecuta solo si el try terminó sin excepción.         | Registrar la nota exitosa en la base de datos del SIGE.     |
| finally               | Se ejecuta siempre, con o sin excepción.                 | Cerrar el archivo CSV aunque el procesamiento falle.        |

try-except-else-finally

```
proteger el registro de notas del SIGE
def registrar_nota_segura(codigo_est, nota_texto, base_datos):
 """
 Registra una nota en el SIGE con manejo completo de excepciones.
 Nunca detiene el programa ante datos inválidos.
 """
 try:
 # Puede lanzar ValueError si nota_texto no es numero
 nota_float = float(nota_texto)

 # Puede lanzar KeyError si el código no existe
 estudiante = base_datos[codigo_est]

 # Puede lanzar ValueError si nota fuera de rango
 if not (0 <= nota_float <= 20):
 raise ValueError(f"Nota {nota_float} fuera del rango 0-20")

 estudiante.registrar_nota("IS-402", nota_float)

 except ValueError as e:
 print(f'Datos inválidos: {e}')
 print(f'Nota recibida: {nota_texto}')
 print(f'La nota no fue registrada.')
 return False

 except KeyError:
 print(f'Código {codigo_est} no encontrado en el SIGE.')
 print(f'Verifique el código e intente nuevamente.')
 return False

 except Exception as e:
 print(f'Error inesperado: {type(e).__name__}: {e}')
```

try-except-else-finally

```
proteger el registro de notas del SIGE
def registrar_nota_segura(codigo_est, nota_texto, base_datos):
 """
 Registra una nota en el SIGE con manejo completo de excepciones.
 Nunca detiene el programa ante datos inválidos.
 """
 try:
 # Puede lanzar ValueError si nota_texto no es numero
 nota_float = float(nota_texto)

 # Puede lanzar KeyError si el código no existe
 estudiante = base_datos[codigo_est]

 # Puede lanzar ValueError si nota fuera de rango
 if not (0 <= nota_float <= 20):
 raise ValueError(f"Nota {nota_float} fuera del rango 0-20")

 estudiante.registrar_nota("IS-402", nota_float)

 except ValueError as e:
 print(f'Datos inválidos: {e}')
 print(f'Nota recibida: {nota_texto}')
 print(f'La nota no fue registrada.')
 return False

 except KeyError:
 print(f'Código {codigo_est} no encontrado en el SIGE.')
 print(f'Verifique el código e intente nuevamente.')
 return False

 except Exception as e:
 print(f'Error inesperado: {type(e).__name__}: {e}')
```

Ing. Pelayo Quijpe Bautista

3 | Página

Manejo de Errores y Excepciones en Python

python

Sesión 06

```

 return False

else:
 # Solo se ejecuta si NO hubo excepción
 print(f'Nota {nota_float} registrada correctamente para {codigo_est}')
 return True

finally:
 # Siempre se ejecuta: ideal para cerrar recursos
 print(f'Intento de registro para {codigo_est} finalizado.')

```

**3.4 Excepciones personalizadas para el SIGE**

Las excepciones personalizadas permiten crear errores específicos del dominio universitario. En lugar de un genérico ValueError, el sistema puede lanzar Notainvalida o EstudianteNoEncontrado, lo que hace el código más legible y el manejo de errores más preciso.

**Excepciones personalizadas para el dominio UNSCH**

# Jerarquía de excepciones del SIGE

```

class ErrorSistemaUNSCH(Exception):
 """Excepción base para todos los errores del SIGE."""
 pass

class Notainvalida(ErrorSistemaUNSCH):
 def __init__(self, codigo):
 self.codigo = codigo
 self.rango = rango
 self.mensaje = f'Nota {nota} inválida. Debe estar entre {rango[0]} y {rango[1]}.'
 super().__init__(self.mensaje)

class EstudianteNoEncontrado(ErrorSistemaUNSCH):
 def __init__(self, codigo):
 self.codigo = codigo
 super().__init__(f'Estudiante con código {codigo} no encontrado en el SIGE.')

class AsistenciaInvalida(ErrorSistemaUNSCH):
 def __init__(self, fecha):
 self.fecha = fecha
 super().__init__(f'Fecha de asistencia {fecha} con formato inválido.')

```

# Uso en el sistema SIGE

```

def registrar_nota_v2(codigo, nota, base_datos):
 # código not in base_datos
 raise EstudianteNoEncontrado(codigo) # excepción específica
 if not (0 <= nota <= 20):
 raise Notainvalida(nota) # excepción específica
 base_datos[codigo].registrar_nota(f'S-402', nota)
 return True

```

# Manejo con excepciones específicas

```

bd_demo = {'27150415': None} # simplificado para el ejemplo

try:
 registrar_nota_v2(27150415, 15, bd_demo) # código inexistente
except EstudianteNoEncontrado as e:
 print(f'Error específico: {e}')
 print(f' Código buscado: {e.codigo}')
except ErrorSistemaUNSCH as e:
 print(f'Error del sistema: {e}')

```

Ing. Pelayo Quirope Bautista

4 | Página

Manejo de Errores y Excepciones en Python

python

Sesión 06

```

try:
 registrar_nota_v2(27150415, 25, bd_demo) # nota inválida
except Notainvalida as e:
 print(f'Error específico: {e}')
 print(f' Nota ingresada: {e.nota}')

```

**3.5 Python Debugger**

El Python Debugger (pdb) permite pausar la ejecución del programa en cualquier punto e inspeccionar el valor de las variables en ese momento exacto. En VS Code se usa con puntos de interrupción visuales (breakpoints). En cualquier entorno Python se puede insertar la función breakpoint() directamente en el código.

**Python Debugger**

# uso de breakpoint() para diagnosticar fallos del calculador, promedio, con\_debuglista\_notas:

Version con breakpoint para diagnosticar un fallo de división por cero.

Ejecutar en terminal: python sige\_debug.py

breakpoint() # el programa se pausa aquí, abre el prompt (Pdb)

# Comandos útiles en el prompt (Pdb)

# n -> next: ejecuta la siguiente línea

# s -> step: entra dentro de una función

# c -> continue: continúa hasta el próximo breakpoint

# p variable -> print: muestra el valor de una variable

# p lista\_notas -> muestra el contenido de lista\_notas

# q -> quit: sale del debugger

total = sum(lista\_notas) # p total para ver el valor

n = len(lista\_notas) # p n para ver si es 0

return total / n # falla si n == 0

# En VS Code: clic en el número de línea para poner un breakpoint visual

# El programa se pausa en el breakpoint y se puede inspeccionar cada variable

# Alternativa: usar print debugging cuando no se tiene IDE

```

def calculador_promedio_print_debuglista_notas():
 print(f'DEBUG: lista_notas = {lista_notas}')
 print(f'DEBUG: len = {len(lista_notas)}')
 if not lista_notas:
 print(f'DEBUG: lista vacía, retornando 0.0')
 return 0.0
resultado = calculador_promedio_print_debuglista_notas()
print(f'DEBUG: resultado = {resultado}')
return round(resultado, 2)

```

**3.6 Raise: lanzar excepciones manualmente**

La instrucción raise permite lanzar una excepción desde dentro del código cuando se detecta una condición inválida. Es la forma correcta de señalar errores de lógica de negocio, como una nota fuera del rango vigesimal o un código universitario con formato incorrecto.

**raise**

# Lanzar excepciones en el módulo de validación SIGE

```

def validar_codigo_unsch(codigo):
 """Valida que el código tenga el formato correcto de la UNSCH."""
 if not isinstance(codigo, str):
 raise TypeError(f'El código debe ser texto, no {type(codigo).__name__}.')
 if len(codigo) != 8:

```

Ing. Pelayo Quirope Bautista

5 | Página

Manejo de Errores y Excepciones en Python

python

Sesión 06

```

 raise ValueError(f'El código debe tener 8 dígitos. Recibido: {codigo}')
 if not codigo.startswith('27'):
 raise ValueError(f'El código UNSCH debe comenzar con 27. Recibido: {codigo}')
 if not codigo[2:].isdigit():
 raise ValueError(f'Los últimos 6 caracteres deben ser numéricos. Recibido: {codigo}')
 return True

def validar_nota(nota):
 """Valida que la nota este en el rango vigesimal de la UNSCH."""
 try:
 nota_num = float(nota) # convierte si es texto
 except (ValueError, TypeError):
 raise Notainvalida(nota) from None # encadena con excepción personalizada
 if not (0 <= nota_num <= 20):
 raise Notainvalida(nota_num)
 return nota_num

Pruebas del tester
print("=== Pruebas de validación ===")

casos = [
 ('27150415', 'Código válido'),
 ('12345678', 'No empieza con 27'),
 ('2715041', 'Solo 7 dígitos'),
 ('27150415', 'Es int no str'),
]

for codigo, desc in casos:
 try:
 validar_codigo_unsch(codigo)
 print(f'OK: {codigo}')
 except (ValueError, TypeError) as e:
 print(f'ERROR: {codigo} {desc}: {e}')

```

**4. PROBLEMA ABP RESUELTO**

El equipo recibe el módulo de registro de calificaciones del SIGE con cinco fallos conocidos. Aplica manejo de excepciones para que el sistema sea robusto ante cualquier dato inválido del operador.

**CODIGO COMPLETO**

```

=====
SISTEMA SIGE - OTI UNSCH | Sesión 8
Módulo: Registro de calificaciones con manejo de excepciones
Objetivo: sistema que no se detiene ante datos inválidos
=====
import os

Excepciones personalizadas del SIGE
class ErrorSistemaUNSCH(Exception):
 pass
class Notainvalida(ErrorSistemaUNSCH):
 def __init__(self, nota):
 super().__init__(f'Nota {nota} inválida. Rango permitido: 0 a 20.')
 self.nota = nota
class EstudianteNoEncontrado(ErrorSistemaUNSCH):
 def __init__(self, codigo):
 super().__init__(f'Código {codigo} no encontrado en el SIGE.')
 self.codigo = codigo

Base de datos simplificada

```

Ing. Pelayo Quirope Bautista

6 | Página

Manejo de Errores y Excepciones en Python

python

Sesión 06

```

class EstudianteSimple:
 def __init__(self, nombre, codigo):
 self.nombre = nombre
 self.codigo = codigo
 self.nota = 0
 def registrar_nota(self, curso, nota):
 self.nota.setdefault(curso, []).append(nota)
 def calculador_promedio(self, curso):
 lista = self.nota.get(curso, [])
 return round(sum(lista)/len(lista), 2) if lista else 0.0

base_datos = {
 '27150415': EstudianteSimple('Carmen Quispe', '27150415'),
 '27150416': EstudianteSimple('Pietro Cardenas', '27150416'),
 '27150417': EstudianteSimple('Maria Huaman', '27150417'),
}

FALLO 1: Texto no numérico en campo de nota
=====
def registrar_nota(codigo, nota_texto, curso='S-402'):
 """Registra nota con manejo de dato no numérico y fuera de rango."""
 try:
 nota_num = float(nota_texto) # puede lanzar ValueError
 if not (0 <= nota_num <= 20):
 raise Notainvalida(nota_num)
 if codigo not in base_datos:
 raise EstudianteNoEncontrado(codigo)
 base_datos[codigo].registrar_nota(curso, nota_num)
 except ValueError:
 print(f'FALLO 1: Nota {nota_texto} no es numérica. No registrada.')
 return False
 except Notainvalida as e:
 print(f'FALLO 1: {e}')
 return False
 except EstudianteNoEncontrado as e:
 print(f'FALLO 2: {e}')
 return False
 else:
 print(f'OK: {nota_num} registrada para {codigo} en {curso}.')
 return True
 finally:
 print(f'Proceso de registro finalizado para {codigo}.')

FALLO 3: Promedio de estudiante sin notas (división por cero)
=====
def obtener_promedio(codigo, curso='S-402'):
 """Calcula promedio con manejo de lista vacía."""
 try:
 if codigo not in base_datos:
 raise EstudianteNoEncontrado(codigo)
 prom = base_datos[codigo].calculador_promedio(curso)
 return prom
 except EstudianteNoEncontrado as e:
 print(f'FALLO 3: {e}')
 return 0.0

FALLO 4: Archivo CSV inexistente

```

Ing. Pelayo Quirope Bautista

7 | Página

Manejo de Errores y Excepciones en Python



Sesión 08

```
=====
del exportar_reporte_csv(nombre_archivo)
"Exporta reporte con manejo de archivo inexistente."
archivo = None
try:
 archivo = open(nombre_archivo, 'r')
 contenido = archivo.read()
 print(' OK: Archivo {nombre_archivo} leído ({len(contenido)} chars.)')
 return contenido
except FileNotFoundError:
 print(' FALLO 4: Archivo {nombre_archivo} no encontrado.')
 print(' Verifique que el archivo esté en la carpeta del sistema.')
 return None
except PermissionError:
 print(' FALLO 4: Sin permisos para leer {nombre_archivo}.')
 return None
finally:
 if archivo: # se ejecuta siempre, cierra el archivo si fue abierto
 archivo.close()
 print(' Archivo cerrado correctamente.')

=====
FALLO 5: Atributo inexistente en objeto Estudiante
=====
def obtener_datos_estudiante(codigo, atributo):
 """Lee un atributo de forma segura con getitem."""
 try:
 if codigo not in base_datos:
 raise EstudianteNoEncontrado(codigo)
 est = base_datos[codigo]
 valor = getattr(est, atributo, None)
 if valor is None:
 raise AttributeError('El estudiante no tiene el atributo {atributo}.')
 return valor
 except (EstudianteNoEncontrado, AttributeError) as e:
 print(' FALLO 5: {e}')
 return None
=====
EJECUCION: Reproducir los 5 fallos y verificar el manejo
=====
print('== PRUEBA DE LOS 5 FALLOS DEL SIGE ==')
print()
FALLO 1 y 2: Nota inválida y código inexistente
registrar_nota(27150415, '15.0') # correcto
registrar_nota(27150415, 'NA') # texto no numérico
registrar_nota(27150415, '25') # fuera de rango
registrar_nota(27999999, '15.0') # código inexistente
print()
FALLO 3: Promedio sin notas
prom = obtener_promedio(27150415) # sin notas registradas
print(' Promedio: {prom}' (0.0 porque no hay notas))
print()
FALLO 4: Archivo CSV inexistente
exportar_reporte_csv('notas_octubre.csv') # no existe
print()
```

Ing. Pelayo Quique Bautista

6 | Página

Manejo de Errores y Excepciones en Python



Sesión 08

```
print('Fallo 5: Atributo inexistente')
dato = obtener_datos_estudiante(27150415, 'telefono') # no existe
print('! Dato obtenido: {dato}')
```

### 5. EJERCICIOS PROPUESTOS

El tester diseña los casos de prueba antes de que el desarrollador escriba el código. El documentador registra cada excepción encontrada y la solución aplicada.

**Ejercicio 1: Módulo de validación robusta para el formulario web (Nivel básico)**  
El formulario web del SIGE tiene cinco campos que pueden contener datos inválidos. Implementa una función validar\_formulario(datos\_dict) que reciba un diccionario con los campos del formulario y aplique validación con try-except para cada uno:

- codigo: está: debe ser str de 8 dígitos comenzando con '27'. Lanza ValueError si no.
- nombre: debe ser str con al menos 3 caracteres y sin números. Lanza ValueError si no.
- nota\_practica, nota\_investigacion, nota\_examen: deben ser convertibles a float en rango 0-20. Lanza ValueError si no.
- La función retorna (True, []) si todo es válido o (False, (campo: mensaje, error)) con todos los errores encontrados sin detenerse en el primero.

**Casos del tester:** Todos los campos válidos. Nota como texto 'quince'. Código de 7 dígitos. Nota -5. Nombre con solo números. Todos los campos inválidos a la vez.

**Ejercicio 2: Sistema de importación de notas desde CSV con finally (Nivel intermedio)**  
El SIGE necesita importar notas desde archivos CSV generados por otros sistemas. Implementa importar\_notas\_csv(nombre\_archivo) que:

- Abra el archivo con try-except-finally para garantizar que siempre se cierre.
- Lee líneas por línea. Si una línea tiene formato incorrecto, la omite con un mensaje y continúa con la siguiente usando continue dentro del try.
- Por cada línea válida, llama a registrar\_nota(). Si la nota lanza ValueError, la registra en una lista de rechazados y continúa.
- Al final, en el bloque else (si el archivo se procesó sin error fatal), muestra un resumen: notas importadas, notas rechazadas, líneas con formato incorrecto.
- En el finally, cierra el archivo y registra en un log la hora y el resultado del proceso.

**Casos del tester:** Archivo que no exista. Archivo con todas las notas válidas. Archivo con mezcla de notas válidas e inválidas. Archivo completamente vacío. Archivo con solo encabezado.

**Ejercicio 3: Módulo de diagnóstico automático del SIGE (Nivel avanzado)**  
Implementa una clase DiagnosticoSIGE que analice automáticamente el sistema y genere un reporte de salud. La clase debe:

- Implementar verificar\_integridad\_db(base\_datos) que recorra todos los objetos Estudiante y verifique que tienen los atributos requeridos. Por cada atributo faltante lanza AttributeError personalizado que captura y registra.
- Implementar verificar\_consistencia\_notas() que detecte notas fuera del rango 0-20 ya registradas en el sistema y las reporte como inconsistencias.
- Implementar generar\_reporte\_diagnostico() que ejecute todas las verificaciones envolviendo cada una en try-except para que un fallo en una verificación no impida ejecutar las demás.
- Todo el proceso debe registrar cada excepción encontrada en un log interno (lista de dicts con tipo, error, descripcion, timestamp).

**Reto extra:** Usa el módulo logging de Python en lugar de print() para registrar los errores en un archivo sige\_errores.log con nivel WARNING o ERROR si la gravedad.

Ing. Pelayo Quique Bautista

9 | Página

Depuración y Pruebas en Python



Sesión 09

| Depuración Sistemática de Código<br>Corrección de Errores Lógicos con Python Debugger y pylint |                                       | SESIÓN 09<br>Semana 11  |
|------------------------------------------------------------------------------------------------|---------------------------------------|-------------------------|
| Asignatura:                                                                                    | Algoritmos y Estructuras de Datos     | Semestre: 2024-II       |
| Docente:                                                                                       | Ing. Pelayo Quique Bautista           | Duración: 90 minutos    |
| Metodología:                                                                                   | Aprendizaje Basado en Problemas (ABP) | Fases ABP: Fases 3 y 4  |
| Herramientas:                                                                                  | Python Debugger, pylint, VS Code      | Semana: Semana 11 de 13 |

### 1. FASES 3 Y 4. INVESTIGACIÓN, CONSTRUCCIÓN Y PRESENTACIÓN

|                                                |                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FASE 3<br/>Investigación y Construcción</b> | El equipo recibe el algoritmo de búsqueda del SIGE con errores lógicos conocidos. Investiga las diferencias entre tipos de error y las estrategias de depuración sistemática. Aplica print debugging y Python Debugger para localizar exactamente dónde el algoritmo falla. Competencia: HC, aplicación de herramientas de depuración para localizar errores lógicos.                     |
| <b>FASE 4<br/>Desarrollo y Presentación</b>    | El equipo corrige los errores lógicos y verifica con los casos de prueba que el algoritmo ahora es correcto. Ejecuta pylint sobre el módulo corregido y mejora el score hasta llegar a 8.0 o más. Hace commit a GitHub con el mensaje descriptivo del bug corregido y presenta el resultado. Competencia: DP, corrección de errores lógicos, análisis estático con pylint y verificación. |

#### 1.1 El algoritmo de búsqueda del SIGE con errores lógicos

**Situación problemática (Semana 11):**  
El equipo de desarrollo de la OTI entrega el algoritmo de búsqueda de estudiantes con bajo rendimiento académico. El código compila y ejecuta sin lanzar ninguna excepción, pero el jefe de la OTI detecta que los resultados son incorrectos.

**Error reportado 1:** El sistema identifica como 'en riesgo' a estudiantes con promedio 14.0, que no deberían estar en esa categoría (el umbral correcto es promedio menor a 10.5).

**Error reportado 2:** Al calcular el promedio general de la sección, el resultado siempre es 0.0 aunque haya notas registradas.

**Error reportado 3:** La lista de estudiantes en riesgo aparece duplicada: cada estudiante aparece dos veces.

**Error reportado 4:** El control de estudiantes aprobados es siempre 0, aunque haya estudiantes con promedios altos.

Ninguno de estos errores lanza una excepción. El programa termina normalmente pero produce datos incorrectos. Son errores lógicos.

**Tu misión:** usar Python Debugger y pylint para localizar la causa exacta de cada error y corregirla sin romper el comportamiento correcto del sistema.

**Actividad Fase 3: Análisis del algoritmo con errores antes de depurar**  
El equipo lee el código con errores y responde estas preguntas antes de ejecutar el debugger:  
**Error 1:** La condición 'promedio > 10.5' en lugar de 'promedio < 10.5'.  
Pregunta: ¿Qué estudiantes quedan fuera de la lista cuando el operador es '>'?  
Pregunta: ¿El programa lanza alguna excepción con esta condición? ¿Por qué no?  
**Error 2:** El acumulador de suma está inicializado dentro del bucle en lugar de fuera.  
Pregunta: ¿Qué valor tiene suma, total al calcular el promedio si se inicializa dentro del bucle?  
Pregunta: ¿Cómo el print debugging revela este error?  
**Error 3:** La lista de riesgo se añade (extend) en lugar de append.  
Pregunta: ¿Qué diferencia hay entre lista.append(elemento) y lista.extend(elemento)?  
**Error 4:** El contador de aprobados no se incrementa dentro del if correcto.

Ing. Pelayo Quique Bautista

1 | Página

Depuración y Pruebas en Python



Sesión 09

Pregunta: ¿Cómo usar un breakpoint en VS Code para inspeccionar el valor del contador?

### 2. COMPETENCIAS A DESARROLLAR

| Pensamiento Lógico-Algorítmico (PLA)                                                                                                                                                       | Habilidades de Codificación (HC)                                                                                                                                                   | Depuración y Prueba (DP)                                                                                                                                                  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Distingue entre errores de sintaxis, errores en tiempo de ejecución y errores lógicos; diseña casos de prueba que exponen el comportamiento incorrecto del algoritmo de búsqueda del SIGE. | Aplica estrategias de depuración: print debugging, Python Debugger con breakpoints y análisis estático con pylint para localizar y corregir errores lógicos en el módulo del SIGE. | Corrige errores lógicos (condiciones incorrectas, acumuladores mal inicializados, índices erróneos) sin alterar el comportamiento correcto de los casos que sí funcionan. |

### 3. CONTENIDO CONCEPTUAL: Depuración sistemática

#### 3.1 Los tres tipos de error: la diferencia crítica

El mayor desafío en depuración no son los errores que detienen el programa (los vemos inmediatamente), sino los errores lógicos que permiten que el programa termine normalmente pero con resultados incorrectos. Un sistema de gestión académica que reporta promedios equivocados es más peligroso que uno que lanza una excepción, porque el error pasa desapercibido.

| Tipo de error                | Cuando se detecta                                                | Ejemplo en el SIGE                                                                                                    |
|------------------------------|------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| Error de sintaxis            | Antes de ejecutar. Python no puede interpretar el código.        | if promedio >= 10.5 (falta el '=' al final). Python lo señala inmediatamente.                                         |
| Error en tiempo de ejecución | Durante la ejecución. El programa se detiene con una excepción.  | float('NA') lanza ValueError cuando el operador ingresa texto en lugar de nota.                                       |
| Error lógico                 | El programa corre sin error pero produce resultados incorrectos. | El algoritmo reporta que un estudiante con promedio 14.0 está en riesgo cuando no debería. El programa no se detiene. |

**Los tres tipos de error demostrados con el SIGE**

```
ERROR DE SINTAXIS: Python lo detecta antes de ejecutar
del calcular_promedio(notas) <- falta '='
return sum(notas) / len(notas)
SyntaxError: expected '='

ERROR EN TIEMPO DE EJECUCION: ocurre al ejecutar, lanza excepción
notas_texto = ['15', '12', 'NA', '10']
try:
 notas_num = [float(n) for n in notas_texto]
except ValueError as e:
 print('Error en tiempo de ejecución: {e}')
El programa se detiene en la línea con error -> viable

ERROR LOGICO: el programa corre, pero el resultado es incorrecto
def buscar_en_riesgo_CON_ERROR(estudiantes):
 en_riesgo = []
 for nombre, promedio in estudiantes:
 if promedio > 10.5: # ERROR LOGICO: debería ser '<'
 en_riesgo.append(nombre)
 return en_riesgo

datos = [['Carmen', 15.0], ['Pedro', 8.5], ['María', 17.0], ['Jorge', 9.0]]
resultado = buscar_en_riesgo_CON_ERROR(datos)
print('En riesgo (resultado incorrecto): {resultado}')
```

Ing. Pelayo Quique Bautista

2 | Página

```
Muestra: ['Carmen', 'Maria'] -> INCORRECTO
Debería mostrar: ['Pedro', 'Jorge']
No hay excepcion. El error es invisible sin casos de prueba.
```

- 3.2 Estrategia de depuración sistemática: el proceso de 5 pasos**
- La depuración efectiva no es aleatoria. McCauley et al. (2008) describen la depuración como un proceso cognitivo que requiere una estrategia sistemática. El equipo debe seguir estos cinco pasos en orden antes de modificar una sola línea de código:
- **Paso 1:** Reproducir el error: ejecutar el programa con los datos que producen el resultado incorrecto y verificar que el error ocurre consistentemente.
  - **Paso 2:** Aislar el error: identificar cuál función o bloque específico produce el resultado incorrecto usando casos de prueba mínimos.
  - **Paso 3:** Localizar la causa: insertar print debugging o usar el Python Debugger para inspeccionar el valor de las variables en el punto del error.
  - **Paso 4:** Corregir y verificar: aplicar la corrección mínima necesaria y ejecutar todos los casos de prueba (no solo el que falló).
  - **Paso 5:** Analizar con pylint: ejecutar pylint sobre el código corregido para detectar problemas adicionales de calidad y convención.

**3.3 Print debugging: la técnica más directa**

El print debugging consiste en insertar sentencias print() estratégicas para mostrar el valor de variables en puntos clave del código. Es la técnica más rápida para verificar hipótesis sobre qué está pasando dentro de un algoritmo. La clave es insertar los prints ANTES del punto donde el error es visible, no después.

```
Print debugging: localizar el error del acumulador en el SIGE

CODIGO CON ERROR LOGICO: acumulador inicializado dentro del bucle
def calcular_promedio_seccion_CON_ERROR(lista_estudiantes):
 for nombre, notas in lista_estudiantes:
 suma_total = 0 # ERROR: se reinicia en cada iteracion
 for nota in notas:
 suma_total += nota
 promedio = suma_total / len(lista_estudiantes)
 return promedio

PASO 1: Reproducir
datos = [['Carmen', [15, 17, 18]], ['Pedro', [12, 10, 11]], ['Maria', [18, 19, 20]]]
resultado = calcular_promedio_seccion_CON_ERROR(datos)
print(f'Promedio seccion: {resultado}') # probablemente incorrecto

PASO 3: Localizar con print debugging
def calcular_promedio_seccion_DEBUG(lista_estudiantes):
 print(DEBUG: entrando con {len(lista_estudiantes)} estudiantes)
 for i, (nombre, notas) in enumerate(lista_estudiantes):
 suma_total = 0
 print(f'DEBUG iteracion {i}: suma_total reiniciada a {suma_total}')
 for nota in notas:
 suma_total += nota
 print(f'DEBUG: {nombre} suma_parcial={suma_total}')
 print(DEBUG: suma_total al terminar el bucle externo = {suma_total})
 return suma_total / len(lista_estudiantes)

PASO 4: Corregir
def calcular_promedio_seccion_CORRECTO(lista_estudiantes):
 if not lista_estudiantes:
 return 0.0
 suma_total = 0 # CORRECTO: inicializar FUERA del bucle
 for nombre, notas in lista_estudiantes:
 for nota in notas:
 suma_total += nota
 total_notas = sum(len(notas) for _, notas in lista_estudiantes)
 return round(suma_total / total_notas, 2) if total_notas > 0 else 0.0
```

**3.4 Python Debugger (pdb) en VS Code: Inspección Interactiva**

El Python Debugger permite pausar la ejecución en cualquier línea e inspeccionar el valor exacto de todas las variables en ese momento. En VS Code se usa con breakpoints visuales. Es especialmente útil cuando el error ocurre dentro de un bucle o en condiciones específicas que el print debugging no logra aislar fácilmente.

```
Python Debugger: comandos esenciales para depurar el SIGE

Uso en VS Code:
1. Clic en el número de línea para agregar un breakpoint (punto rojo)
2. Presionar F5 o Run -> Start Debugging
3. El programa se pausa en el breakpoint
4. En el panel de depuración se ven los valores de todas las variables

Uso en terminal con breakpoint():
def buscar_en_riesgo_DEBUG(estudiantes, umbral=10.5):
 en_riesgo = []
 aprobados = 0
 breakpoint() # pausa aquí al ejecutar en terminal
 # Comandos en el prompt (Pdb):
 # n -> ejecutar la siguiente línea (next)
 # s -> entrar dentro de la función llamada (step)
 # c -> continuar hasta el próximo breakpoint (continue)
 # p variable -> mostrar valor: p estudiantes, p umbral, p en_riesgo
 # pp variable -> pretty print para listas y diccs
 # l -> mostrar el código alrededor de la línea actual (list)
 # q -> salir del debugger (quit)
 # w -> mostrar la pila de llamadas (where)
 for nombre, promedio in estudiantes:
 # Poner breakpoint AQUÍ para ver promedio en cada iteracion
 # promedio = umbral
 en_riesgo.append(nombre)
 else:
 aprobados += 1
 return en_riesgo, aprobados

Casos de prueba para el debugger
datos_prueba = [
 ('Carmen', 15.0),
 ('Pedro', 8.5),
 ('Maria', 17.0),
 ('Jorge', 9.0),
 ('Ana', 10.5), # caso límite: exactamente en el umbral
]

Al inspeccionar con el debugger en cada iteracion:
Verificar que promedio < 10.5 identifica a Pedro (8.5) y Jorge (9.0)
Verificar que Ana (10.5) NO entra en en_riesgo (10.5 no es < 10.5)
```

**3.5 pylint: análisis estático de calidad del código**

pylint analiza el código Python sin ejecutarlo y detecta errores potenciales, violaciones de convención (PEP 8) y código de baja calidad. En el SIGE, se debe ejecutar pylint sobre cada módulo antes de hacer commit a GitHub, apuntando a un score mayor o igual a 8.0.

| Código pylint | Categoría   | Descripción y ejemplo en el SIGE                                                            |
|---------------|-------------|---------------------------------------------------------------------------------------------|
| C0103         | Convención  | Nombre de variable demasiado corto o no sigue snake_case. Ejemplo: en lugar de total_notas. |
| C0301         | Convención  | Línea demasiado larga (más de 79 caracteres). Recomendación PEP 8.                          |
| W0611         | Advertencia | Módulo importado pero no utilizado. Ejemplo: import math sin usar ninguna función de math.  |
| W0612         | Advertencia | Variable asignada pero nunca usada. Indica código muerto o error lógico.                    |

|       |                 |                                                                                       |
|-------|-----------------|---------------------------------------------------------------------------------------|
| E1101 | Error           | Acceso a atributo inexistente en un objeto. pylint detecta antes de ejecutar.         |
| R0201 | Refactorización | Método de clase que no usa self: debería ser función estática (@staticmethod).        |
| Score | Catificación    | Puntuación de 0 a 10. El SIGE debe apuntar a un score >= 8.0 antes de subir a GitHub. |

```
pylint

Instalacion (ejecutar en terminal una sola vez):
pip install pylint

Uso basico:
pylint sigse_busqueda.py

Ejemplo de salida de pylint sobre codigo con problemas:
***** Module sigse_busqueda
sigse_busqueda.py:3:0: C0103: Variable name 'i' doesn't conform to...
sigse_busqueda.py:8:4: W0612: Unused variable 'i' (unused-variable)
sigse_busqueda.py:15:0: C0301: Line too long (85/79 chars) (line-too-long)
sigse_busqueda.py:22:8: E1101: Module 'math' has no 'sqrt' member...
#
Your code has been rated at 4.50/10
Codigo con problemas que pylint detecta:
def buscarEnRiesgo(s, umbral=10.5): # C0103: nombre no es snake_case
 x = 'sin usar' # W0612: variable no usada
 r = []
 for i, p in s: # C0103: 'i' y 'p' muy cortos
 if p >= umbral:
 r.append(i)
 return r
Codigo corregido segun pylint (score 10/10):
def buscar_estudiantes_riesgo(estudiantes, umbral=10.5):
 """Busca estudiantes con promedio por debajo del umbral."""
 en_riesgo = []
 for nombre, promedio in estudiantes:
 if promedio < umbral:
 en_riesgo.append(nombre)
 return en_riesgo
```

**4. PROBLEMA ABP RESUELTO**

El equipo recibe el módulo completo con los cuatro errores lógicos reportados por la OTI. Aplica el proceso de 5 pasos de depuración sistemática para identificar y corregir cada error, luego ejecuta pylint para mejorar la calidad del código.

**Paso 1 y 2: Código original con los cuatro errores lógicos**

```
CODIGO ORIGINAL CON ERRORES LOGICOS

MODULO SIGE - BUSQUEDA DE RIESGO ACADEMICO
Version con 4 errores logicos (para depuracion en clase)

def buscarEstudiantesRiesgo(lista): # C0103: no es snake_case
 aprobados = 0
 for nombre, promedio in lista:
 if promedio > 10.5: # ERROR 1: > en lugar de <
 EnRiesgo.extend([nombre]) # ERROR 3: extend en lugar de append
 EnRiesgo.extend([nombre]) # ERROR 3: linea duplicada
```

```
promedio >= 10.5: # ERROR 4: nunca incrementa aprobados
pass # ERROR 4: falta: aprobados += 1
return EnRiesgo, aprobados

def calcularPromedioSeccion(lista): # C0103: no es snake_case
 for nombre, notas in lista:
 sumaTotal = 0 # ERROR 2: inicializado dentro del bucle
 for nota in notas:
 sumaTotal += nota
 n = len(lista) # C0103: 'n' muy corto
 return sumaTotal / n if n > 0 else 0
```

**Pasos 3, 4 y 5: Depuración, corrección y verificación con pylint**

```
CODIGO CORREGIDO Y VERIFICADO

MODULO SIGE - BUSQUEDA DE RIESGO ACADEMICO
Version corregida. 4 errores logicos solucionados
pylint score: 9.5/10

def buscar_estudiantes_riesgo(lista_estudiantes, umbral=10.5):
 """
 Busca estudiantes con promedio por debajo del umbral.
 Retorna: tuple (en_riesgo: list, aprobados: int)
 """
 Correcciones aplicadas:
 - Error 1: corregido ">" por "<" en la condicion de riesgo
 - Error 3: corregido extend() por append() para evitar duplicados
 - Error 4: agregado aprobados += 1 dentro del bloque else
 """
 en_riesgo = []
 aprobados = 0
 for nombre, promedio in lista_estudiantes:
 if promedio < umbral: # CORRECCION 1: "<" en lugar de ">"
 en_riesgo.append(nombre) # CORRECCION 3: append() no extend()
 else:
 aprobados += 1 # CORRECCION 4: incrementar aquí
 return en_riesgo, aprobados

def calcular_promedio_seccion(lista_notas_por_est):
 """
 Calcula el promedio general de la seccion.
 Retorna: float, promedio de todos los estudiantes.
 """
 Correccion: suma_total inicializada FUERA del bucle.
 if not lista_notas_por_est:
 return 0.0
 suma_total = 0 # CORRECCION 2: fuera del bucle externo
 total_notas = 0
 for nombre, notas in lista_notas_por_est:
 for nota in notas:
 suma_total += nota
 total_notas += 1
 return round(suma_total / total_notas, 2) if total_notas > 0 else 0.0

def generar_reporte_riesgo(estudiantes, promedios, notas_por_est, umbral=10.5):
 """Genera el reporte completo de riesgo academico para la OTI."""
```



Depuración y Pruebas en Python



Sesión 09

```
en_riesgo, aprobados = buscar_estudiantes_riesgo(estudiantes_promedios, umbral)
promedio_gral = calcular_promedio_seccion(notas_por_sesi)
total = len(estudiantes_promedios)

print("==== REPORTE DE RIESGO ACADEMICO- SIGE UNSCH ====")
print(f'Total estudiantes : {total}')
print(f'Aprobados : {aprobados} ({aprobados/total*100:1f}%)')
print(f'En riesgo : {len(en_riesgo)} ({len(en_riesgo)/total*100:1f}%)')
print(f'Promedio general : {promedio_gral}')
if en_riesgo:
 print('Estudiantes en riesgo (derivar a tutoría):')
 for nombre in en_riesgo:
 print(f' * {nombre}')
 return [en_riesgo, en_riesgo, 'aprobados', aprobados, 'promedio', promedio_gral]

=====
VERIFICACION: casos de prueba del tester
=====
promedios = [
 ('Carmen Calupe', 15.25),
 ('Pedro Cardenas', 10.17),
 ('Maria Humari', 18.50),
 ('Jorge Canter', 8.30),
 ('Ana Palomino', 10.50), # caso limite: exactamente en el umbral
 ('Luis Torres', 14.00),
]

notas = [
 ('Carmen', [15, 17, 14]),
 ('Pedro', [10, 11, 9]),
 ('Maria', [18, 20, 17]),
 ('Jorge', [8, 6, 9]),
 ('Ana', [11, 10, 10]),
 ('Luis', [14, 15, 13]),
]

resultado = generar_reporte_riesgo(promedios, notas)

Verificación de los 4 errores corregidos:
print("==== VERIFICACION DE CORRECCIONES ====")
er_ap = buscar_estudiantes_riesgo(promedios)
print(f'En riesgo correctos : {er} # debe ser: {\"Pedro\", \"Jorge\"}')
print(f'Aprobados correctos : {ap} # debe ser: 4')
print(f'Sin duplicados : {len(er)} == len(set(er)) # debe ser: True')
prom = calcular_promedio_seccion(notas)
print(f'Promedio correcto : {prom} # debe ser aprox: 13.0, no 0.0
```

5. EJERCICIOS PROPUESTOS

El tester diseña los casos de prueba antes de que el desarrollador ejecute el código. Al finalizar cada ejercicio se ejecuta pylint y se corrigen los avisos hasta alcanzar un score mayor o igual a 8.0.

**Ejercicio 1: Localizar y corregir errores lógicos en el módulo de asistencia (Nivel básico)**

El módulo de cálculo de asistencia del SIGE tiene tres errores lógicos. El código ejecuta sin excepción pero los resultados son incorrectos. Usando print debugging, localiza y corrige cada uno.

- Error 1: El porcentaje de asistencia siempre da 100% aunque el estudiante haya faltado varias clases. Pista: revisar la operación de conjuntos usada.

Ing. Pelayo Quique Bautista

7 | Página

Depuración y Pruebas en Python



Sesión 09

- Error 2: La lista de estudiantes con alerta aparece con los nombres al revés (último primero). Pista: revisar si hay algún reverse() o slicing incorrecto.
- Error 3: El conteo de fechas de inasistencia devuelve siempre 0. Pista: revisar la diferencia de conjuntos.

**Casos del tester:** Estudiante con 0% de asistencia. Estudiante con 100%. Estudiante con exactamente 70% (límite de alerta). Conjunto de fechas vacío.

**Ejercicio 2: Depuración con Python Debugger en el módulo de orden de mérito (Nivel intermedio)**

El módulo de cálculo de orden de mérito (Sesión 5) tiene dos errores lógicos que solo se manifiestan cuando hay más de 3 estudiantes. Usando el Python Debugger en VS Code:

- Coloca un breakpoint en la línea donde se calcula el promedio ponderado y verifica que los pesos sumen exactamente 1.0 en cada iteración.
- Coloca un breakpoint en la ordenación y verifica que el sort use el criterio correcto (descendente, no ascendente).
- Ejecuta el módulo con los datos de la sección IS-101 completa (8 estudiantes) y verifica que el primero en el ranking sea el que tiene mayor promedio ponderado.
- Ejecuta pylint sobre el módulo y corrige todos los avisos C y W hasta llegar a score mayor a 8.5.

**Ejercicio 3: Pipeline de calidad para el SIGE con pylint y corrección sistemática (Nivel avanzado)**

Implementa un pipeline de calidad que el equipo ejecutará antes de cada commit a GitHub:

- Crea un script verificar\_calidad.py que ejecute pylint sobre todos los módulos del SIGE y muestre un resumen: score de cada módulo, total de errores E, advertencias W y conversiones C.
- Define un estándar de calidad: el commit solo puede realizarse si todos los módulos tienen score mayor o igual a 8.0 y cero errores de tipo E.
- Aplica el pipeline sobre los módulos de las sesiones 5, 6 y 8. Corrige todos los problemas detectados hasta que el pipeline apruebe.
- Documenta en el README.md del repositorio GitHub qué errores lógicos fueron encontrados, qué herramienta los detectó (print debug, breakpoint, pylint) y qué corrección se aplicó.

**Reflexión del equipo:** ¿Cuál tipo de error fue más difícil de encontrar: el de sintaxis, el de ejecución o el lógico? ¿Por qué el error lógico es el más peligroso en un sistema de gestión académica real como el SIGE?

Ing. Pelayo Quique Bautista

8 | Página

Pruebas de Software en Python con pytest



Sesión 10

|                                                   |  |           |
|---------------------------------------------------|--|-----------|
| Pruebas de Software en Python con pytest          |  | SESIÓN 10 |
| Validación del Módulo de Créditos Académicos SIGE |  | Semana 12 |

|               |                                       |           |                                    |
|---------------|---------------------------------------|-----------|------------------------------------|
| Asignatura:   | Algoritmos y Estructuras de Datos     | Semestre: | 2024-II                            |
| Docente:      | Ing. Pelayo Quique Bautista           | Duración: | 90 minutos                         |
| Metodología:  | Aprendizaje Basado en Problemas (ABP) | Fase ABP: | Fase 4 (Desarrollo y Presentación) |
| Herramientas: | pytest, VS Code, GitHub colaborativo  | Semana:   | Semana 12 de 13                    |

1. FASE 4. DESARROLLO, VALIDACIÓN Y PRESENTACIÓN CON PYTEST

FASE 4  
Desarrollo y Presentación

Esta sesión implementa únicamente la Fase 4. El equipo ya tiene el módulo de créditos académicos funcionando. La misión es escribir la suite de pruebas completa con pytest que valide cada función del módulo.

El tester lidera el diseño de casos de prueba; el desarrollador implementa las pruebas en código.

Al finalizar, el equipo ejecuta pytest-cov, revisa el reporte de cobertura y cubre los casos faltantes.

Competencia: DP, diseño y ejecución de pruebas unitarias con cobertura mayor o igual al 90%.

1.1 El módulo de créditos académicos del SIGE

**Situación problemática:**

El módulo de cálculo de créditos académicos del SIGE ha sido desarrollado por el equipo durante las sesiones anteriores. Antes de subirlo a producción y que las operadoras de la OTI lo usen con datos reales de los 3,000+ estudiantes, el jefe de la OTI exige una batería de pruebas automatizadas que garanticen que el módulo funciona correctamente en todos los escenarios posibles.

Los requisitos del módulo son:

Regla 1: El avance curricular = (créditos aprobados / créditos totales del plan) x 100.

Regla 2: Un estudiante entra en alerta si su avance es menor al 60%.

Regla 3: Un estudiante puede graduarse si aprobó al menos el 95% de los créditos del plan.

Regla 4: Los créditos aprobados no pueden ser negativos ni mayores al total del plan.

Regla 5: Si la lista de cursos está vacía, el avance es 0% sin lanzar excepción.

**Tu misión: diseñar los casos de prueba para TODAS las reglas, implementarlos con pytest y lograr una cobertura mayor o igual al 90% usando pytest-cov.**

Actividad Fase 4: Diseño de casos de prueba antes de escribir pytest

El tester lidera el diseño de casos de prueba en papel antes de escribir código:

**Para la Regla 1 (calcular\_avance):**

Caso 1: 180 créditos de 200 -> avance = 90.0%

Caso 2: 0 créditos de 200 -> avance = 0.0%

Caso 3: 200 créditos de 200 -> avance = 100.0%

Caso 4: créditos totales = 0 -> ¿qué debe retornar? ¿o lanzar excepción?

**Para la Regla 2 (verificar\_alerta):**

Caso 5: avance = 58.5% -> alerta = True

Caso 6: avance = 60.0% -> alerta = False (justo en el umbral)

Caso 7: avance = 60.1% -> alerta = False

**Para la Regla 4 (validar\_creditos):**

Caso 8: créditos = -10 -> debe lanzar ValueError

Ing. Pelayo Quique Bautista

1 | Página

Pruebas de Software en Python con pytest



Sesión 10

Caso 9: créditos ap > total -> debe lanzar ValueError

Caso 10: créditos = 0 -> válido (estudiante nuevo)

Pregunta al equipo: ¿Cuántos casos de prueba se necesitan para cubrir el 90% del código?

Pregunta al equipo: ¿Cuál pasa si dos funciones tienen exactamente el mismo caso de prueba?

2. COMPETENCIAS A DESARROLLAR

| Pensamiento Lógico-Algorítmico (PLA)                                                                                                                             | Habilidades de Codificación (HC)                                                                                                                                      | Depuración y Prueba (DP)                                                                                                                                           |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Diseña casos de prueba que cubren los valores límite del módulo de créditos académicos: valores mínimos, máximos, exactamente en el umbral y entradas inválidas. | Implementa suites de prueba con pytest: funciones de prueba, fixtures reutilizables, parámetros para múltiples casos y verificación de excepciones con pytest.raises. | Logra cobertura de código mayor o igual al 90% con pytest-cov; identifica los casos de prueba faltantes según el reporte de cobertura e implementa los que faltan. |

3. CONTENIDO CONCEPTUAL: Pruebas de software con pytest

**3.1 Por qué las pruebas automatizadas son esenciales en el SIGE**

Una prueba unitaria es una función que verifica automáticamente que otra función produce el resultado correcto ante una entrada específica. Sin pruebas automatizadas, el único mecanismo de verificación es ejecutar el programa manualmente y revisar el resultado visualmente: un proceso que no escala cuando el sistema maneja 3,000 registros y crece con cada nueva función.

Según tu tesis, la competencia de Depuración y Prueba es la que muestra la mayor diferencia entre el grupo experimental y el grupo control (15.42 vs 7.20). Las pruebas unitarias con pytest son la herramienta que más contribuye a desarrollar esta competencia, porque obligan al estudiante a pensar explícitamente en todos los casos posibles antes de que el software llegue a producción.

Por qué las pruebas automatizadas detectan lo que el ojo no ve.

```
SIN pruebas: el módulo parece correcto al probarlo manualmente
con el caso 'normal' (180 de 200 créditos)
def calcular_avance(aprobados, total):
 return aprobados / total * 100

Prueba manual: calcular_avance(180, 200) > 90.0 OK
Parece correcto... pero hay un bug oculto
CON pytest: el caso total=0 revela el bug inmediatamente
test_creditos.py
def test_creditos():
 # del test_avance, sin_creditos, totales)
 # resultado = calcular_avance(0, 0) # ZeroDivisionError!
 # assert resultado == 0.0
 # FAILED test_creditos.py: test_avance, sin_creditos, totales
 # ZeroDivisionError: division by zero
 # CORRECCION: verificar total > 0 antes de dividir
 def calcular_avance_corregido(aprobados, total):
 if total <= 0:
 return 0.0
 return round(aprobados / total * 100, 2)
```

3.2 Instalación y estructura de un proyecto con pytest

**Instalación de pytest y pytest-cov**

```
Instalación (ejecutar en terminal una sola vez):
pip install pytest pytest-cov
Estructura del proyecto SIGE con pruebas:
sige/
|__ __init__.py
|__ __credits.py <- módulo que queremos probar
|__ __test_creditos.py <- suite de pruebas
|__ __pytest.ini <- configuración de pytest (opcional)
```

Ing. Pelayo Quique Bautista

2 | Página

```
Ejecutar todas las pruebas:
pytest
Ejecutar con reporte detallado:
pytest -v
Ejecutar con cobertura:
pytest --cov=creditos --cov-report=term-missing
Salida esperada con cobertura:
===== test session starts =====
test_creditos.py [100%]
===== 10 passed in 0.12s =====
Name Stmts Miss Cover Missing
credits.py 28 2 93% 45-46
Las líneas 45-46 no están cubiertas: diseñar un caso que las ejecute
```

### 3.3 Anatomía de una prueba pytest

Una prueba en pytest es simplemente una función cuyo nombre comienza con `test_`. No necesita herencia de clases. Pytest la descubre automáticamente y ejecuta todas las aserciones. Si todas pasan, la prueba es verde. Si alguna falla, pytest muestra exactamente qué valor se esperaba y qué valor se obtuvo.

#### Astoria de una prueba pytest para el módulo de créditos SIGE

```
test_creditos.py
import pytest
from credits import calcular_avance, verificar_alerta, puede_graduarse

Prueba basica: caso feliz
def test_calcular_avance_caso_normal():
 "Prueba el calculo de avance con datos validos tipicos."
 resultado = calcular_avance(160, 200)
 assert resultado == 80.0

Prueba de valor limite: caso borde inferior
def test_calcular_avance_cero_aprobados():
 "Estudiante nuevo sin creditos aprobados."
 assert calcular_avance(0, 200) == 0.0

Prueba de valor limite: plan completo
def test_calcular_avance_plan_completo():
 "Estudiante que aprobo todos los creditos."
 assert calcular_avance(200, 200) == 100.0

Prueba de umbral exacto: el punto donde cambia el comportamiento
def test_verificar_alerta_acertadamente_en_umbral():
 "60.0% exacto: NO debe estar en alerta (umbral es < 60)."
 assert verificar_alerta(60.0) is False

def test_verificar_alerta_bajo_el_umbral():
 "59.9%: SI debe estar en alerta."
 assert verificar_alerta(59.9) is True

Prueba de excepcion: entrada invalida
def test_calcular_avance_creditos_negativos():
 "Creditos negativos deben lanzar ValueError."
 with pytest.raises(ValueError):
 calcular_avance(-10, 200)

def test_calcular_avance_aprobados_mayores_al_total():
 "Creditos aprobados > total: logicamente imposible."
```

Ing. Pelayo Quique Bautista

3 | Página

```
with pytest.raises(ValueError):
 calcular_avance(250, 200)
```

```
Prueba de caso vicio: sin creditos totales
def test_calcular_avance_sin_creditos_totales():
 "Plan con 0 creditos: retornar 0.0 sin excepcion."
 assert calcular_avance(0, 0) == 0.0
```

### 3.4 Fixtures: reutilizar datos de prueba

Un fixture en pytest es una función que prepara datos de prueba y los inyecta automáticamente en las pruebas que los necesitan. Evita repetir la misma inicialización en cada función de prueba y garantiza que cada prueba empiece desde el mismo estado inicial.

#### Fixtures: preparar el estado inicial para pruebas del SIGE

```
test_creditos_fixtures.py
import pytest
from credits import GestorCreditos

Fixture: crea un GestorCreditos listo para usar
@pytest.fixture
def gestor_vacio():
 "GestorCreditos sin ninguna matricula registrada."
 return GestorCreditos(creditos_total_plan=200)

@pytest.fixture
def gestor_con_estudiantes():
 "GestorCreditos con tres estudiantes ya registrados."
 gestor = GestorCreditos(creditos_total_plan=200)
 gestor.registrar_estudiante(27150415, 'Carmen Quispe', 160)
 gestor.registrar_estudiante(27150416, 'Pedro Cardenas', 80)
 gestor.registrar_estudiante(27150417, 'Maria Humant', 195)
 return gestor

Las pruebas reciben el fixture como parametro automaticamente
def test_gestor_vacio_sin_estudiantes(gestor_vacio):
 assert gestor_vacio.total_estudiantes() == 0

def test_avance_carmen(gestor_con_estudiantes):
 avance = gestor_con_estudiantes.calcular_avance(27150415)
 assert avance == 80.0

def test_alerta_pedro(gestor_con_estudiantes):
 "Pedro tiene 80 de 200 = 40%: debe estar en alerta"
 assert gestor_con_estudiantes.esta_en_alerta(27150416) is True

def test_maria_puede_graduarse(gestor_con_estudiantes):
 "Maria tiene 195 de 200 = 97.5% > 95%: puede graduarse"
 assert gestor_con_estudiantes.puede_graduarse(27150417) is True
```

### 3.5 @pytest.mark.parametrize: probar múltiples casos con una sola función

El decorador `@pytest.mark.parametrize` permite ejecutar la misma prueba con diferentes valores de entrada de forma automática. Es ideal para probar los valores límite del módulo de créditos sin repetir código.

#### @pytest.mark.parametrize: probar todos los límites del modulo de creditos

```
import pytest
from credits import calcular_avance, verificar_alerta, puede_graduarse
```

Ing. Pelayo Quique Bautista

4 | Página

```
Probar calcular_avance con múltiples entradas
@pytest.mark.parametrize('aprobados,total,esperado', [
 (160, 200, 80.0), # caso normal
 (0, 200, 0.0), # limite inferior
 (200, 200, 100.0), # plan completo
 (1, 200, 0.5), # minimo posible > 0
 (0, 0, 0.0), # sin creditos totales
 (100, 100, 100.0), # plan de 100 creditos completo
])
def test_calcular_avance_parametrizado(aprobados, total, esperado):
 assert calcular_avance(aprobados, total) == esperado

Probar verificar_alerta con los valores criticos del umbral
@pytest.mark.parametrize('avance,en_alerta', [
 (0.0, True), # sin avance
 (59.9, True), # justo bajo el umbral
 (60.0, False), # exactamente en el umbral: NO alerta
 (60.1, False), # sobre el umbral
 (100.0, False), # plan completo
])
def test_verificar_alerta_parametrizado(avance, en_alerta):
 assert verificar_alerta(avance) is en_alerta

Probar puede_graduarse con los valores criticos
@pytest.mark.parametrize('aprobados,total,puede', [
 (189, 200, False), # 94.5% exacto: NO puede (umbral es > 95%)
 (191, 200, True), # 95.5%: SI puede
 (200, 200, True), # 100%: SI puede
 (189, 200, False), # 94.5%: NO puede
])
def test_puede_graduarse_parametrizado(aprobados, total, puede):
 assert puede_graduarse(aprobados, total) is puede
```

### 3.6 pytest-cov: reporte de cobertura de código

La cobertura de código (code coverage) indica qué porcentaje de las líneas del módulo fueron ejecutadas durante las pruebas. Un módulo con 90% de cobertura tiene el 10% de sus líneas sin probar, lo que significa que hay escenarios no verificados. `pytest-cov` genera el reporte automáticamente.

#### pytest-cov: interpretar el reporte de cobertura del SIGE

```
Ejecutar cobertura:
pytest --cov=creditos --cov-report=term-missing -v

Ejemplo de salida del reporte:
Name Stmts Miss Cover Missing
credits.py 35 4 89% 52, 67-69

Interpretación:
Stmts = total de líneas ejecutables en credits.py
Miss = líneas que ninguna prueba ejecuto
Cover = porcentaje de cobertura (89% significa que falta 11%)
Missing = numeros de líneas no cubiertos: 52 y 67-69

Acción: revisar las líneas 52 y 67-69 del modulo y diseñar
un caso de prueba que las ejecute

Si la línea 52 es:
if creditos_aprobados > creditos_totales:
raise ValueError('Creditos invalidos')
```

Ing. Pelayo Quique Bautista

5 | Página

```
El caso faltante es: creditos_aprobados > creditos_totales
Agregar a la suite
def test_creditos_aprobados_mayores_total():
with pytest.raises(ValueError):
calcular_avance(250, 200) # 250 > 200: invalido

Si la línea 67-69 es:
if total == 0:
return 0.0

El caso faltante es: total == 0
def test_total_creditos_cero():
assert calcular_avance(0, 0) == 0.0
```

### 4. PROBLEMA ABP RESUELTO

El módulo `credits.py` contiene las cinco reglas del sistema de créditos académicos. A continuación se presenta el módulo completo y la suite de pruebas que alcanza una cobertura del 95%.

#### El módulo de créditos académicos: `credits.py`

##### credits.py: módulo de créditos académicos del SIGE (a probar)

```
credits.py
Módulo de calculo de creditos academicos - SIGE UNSCH

UMBRAL_ALERTA = 60.0 # porcentaje minimo de avance sin alerta
UMBRAL_GRADUACION = 95.0 # porcentaje minimo para graduarse
def calcular_avance(creditos_aprobados, creditos_totales):
 "Calcula el porcentaje de avance curricular."
 if creditos_aprobados < 0 or creditos_totales < 0:
 raise ValueError("Los creditos no pueden ser negativos.")
 if creditos_aprobados > creditos_totales and creditos_totales > 0:
 raise ValueError("Creditos aprobados no pueden superar el total.")
 if creditos_totales == 0:
 return 0.0
 return round(creditos_aprobados / creditos_totales * 100, 2)

def verificar_alerta(porcentaje_avance):
 "Retorna True si el estudiante esta en alerta academica."
 return porcentaje_avance < UMBRAL_ALERTA

def puede_graduarse(creditos_aprobados, creditos_totales):
 "Verifica si el estudiante cumple el requisito de graduación."
 avance = calcular_avance(creditos_aprobados, creditos_totales)
 return avance > UMBRAL_GRADUACION

def calcular_creditos_pendientes(creditos_aprobados, creditos_totales):
 "Calcula los creditos que faltan para completar el plan."
 if creditos_aprobados < 0 or creditos_totales < 0:
 raise ValueError("Los creditos no pueden ser negativos.")
 return max(0, creditos_totales - creditos_aprobados)

def generar_estado_academico(codigo, nombre, aprobados, totales):
 "Genera el estado academico completo de un estudiante."
 avance = calcular_avance(aprobados, totales)
 alerta = verificar_alerta(avance)
```

Ing. Pelayo Quique Bautista

6 | Página

```

graduable = puede_graduarse(aprobados, totales)
pendientes = calcular_creditos_pendientes(aprobados, totales)
return {
 'codigo': codigo,
 'nombre': nombre,
 'aprobados': aprobados,
 'totales': totales,
 'avance': avance,
 'alerta': alerta,
 'graduable': graduable,
 'pendientes': pendientes
}

```

La suite de pruebas: test\_creditos.py con cobertura del 95%

test\_creditos.py: suite completa de pruebas con pytest (Problema ASP resuelto)

```

test_creditos.py
Suite de pruebas para el modulo creditos.py del SIGE
Cobertura objetivo: >= 90%
Ejecutar: pytest test_creditos.py -v --cov=creditos --cov-report=term-missing
import pytest
from creditos import (
 calcular_avance,
 verificar_alerta,
 puede_graduarse,
 calcular_creditos_pendientes,
 generar_estado_academico
)
=====
PRUEBAS: calcular_avance()
=====
@pytest.mark.parametrize('aprobados,totales,esperado', [
 (160, 200, 80.0),
 (0, 200, 0.0),
 (200, 200, 100.0),
 (1, 200, 0.5),
 (0, 0, 0.0),
 (100, 100, 100.0),
])
def test_calcular_avance_casos_validos(aprobados, totales, esperado):
 assert calcular_avance(aprobados, totales) == esperado

def test_calcular_avance_creditos_negativos():
 with pytest.raises(ValueError):
 calcular_avance(-1, 200)

def test_calcular_avance_total_negativo():
 with pytest.raises(ValueError):
 calcular_avance(100, -200)

def test_calcular_avance_aprobados_superan_totales():
 with pytest.raises(ValueError):
 calcular_avance(201, 200)
=====
PRUEBAS: verificar_alerta()
=====
@pytest.mark.parametrize('avance,alerta,esperado', [
 (0.0, True),
 (55.9, True),
 (80.0, False),
 (80.1, False),
 (80.0, False),
 (100.0, False),
])
def test_verificar_alerta_umbrales(avance, alerta, esperado):
 assert verificar_alerta(avance) is alerta_esperado
=====
PRUEBAS: puede_graduarse()
=====
@pytest.mark.parametrize('aprobados,totales,puede', [
 (160, 200, False),
 (191, 200, True),
 (200, 200, True),
 (0, 200, False),
])
def test_puede_graduarse_casos(aprobados, totales, puede):
 assert puede_graduarse(aprobados, totales) is puede
=====
PRUEBAS: calcular_creditos_pendientes()
=====
def test_creditos_pendientes_con_avance():
 assert calcular_creditos_pendientes(160, 200) == 40

def test_creditos_pendientes_plan_completo():
 assert calcular_creditos_pendientes(200, 200) == 0

def test_creditos_pendientes_sin_aprobados():
 assert calcular_creditos_pendientes(0, 200) == 200
=====
PRUEBAS: generar_estado_academico()
=====
@pytest.fixture
def estado_carmen():
 return generar_estado_academico('27150415', 'Carmen Quispe', 160, 200)

def test_estado_academico_estructura(estado_carmen):
 claves = ['codigo', 'nombre', 'aprobados', 'totales', 'avance', 'alerta', 'graduable', 'pendientes']
 assert set(estado_carmen.keys()) == claves

def test_estado_academico_avance(estado_carmen):
 assert estado_carmen['avance'] == 80.0

def test_estado_academico_sin_alerta(estado_carmen):
 assert estado_carmen['alerta'] is False

def test_estado_academico_no_graduable(estado_carmen):
 assert estado_carmen['graduable'] is False

def test_estado_academico_pendientes(estado_carmen):
 assert estado_carmen['pendientes'] == 40

```

Ing. Pelayo Quispe Bautista

7 | Página

```

(0.0, True),
(55.9, True),
(80.0, False),
(80.1, False),
(80.0, False),
(100.0, False),
])
def test_verificar_alerta_umbrales(avance, alerta, esperado):
 assert verificar_alerta(avance) is alerta_esperado
=====
PRUEBAS: puede_graduarse()
=====
@pytest.mark.parametrize('aprobados,totales,puede', [
 (160, 200, False),
 (191, 200, True),
 (200, 200, True),
 (0, 200, False),
])
def test_puede_graduarse_casos(aprobados, totales, puede):
 assert puede_graduarse(aprobados, totales) is puede
=====
PRUEBAS: calcular_creditos_pendientes()
=====
def test_creditos_pendientes_con_avance():
 assert calcular_creditos_pendientes(160, 200) == 40

def test_creditos_pendientes_plan_completo():
 assert calcular_creditos_pendientes(200, 200) == 0

def test_creditos_pendientes_sin_aprobados():
 assert calcular_creditos_pendientes(0, 200) == 200
=====
PRUEBAS: generar_estado_academico()
=====
@pytest.fixture
def estado_carmen():
 return generar_estado_academico('27150415', 'Carmen Quispe', 160, 200)

def test_estado_academico_estructura(estado_carmen):
 claves = ['codigo', 'nombre', 'aprobados', 'totales', 'avance', 'alerta', 'graduable', 'pendientes']
 assert set(estado_carmen.keys()) == claves

def test_estado_academico_avance(estado_carmen):
 assert estado_carmen['avance'] == 80.0

def test_estado_academico_sin_alerta(estado_carmen):
 assert estado_carmen['alerta'] is False

def test_estado_academico_no_graduable(estado_carmen):
 assert estado_carmen['graduable'] is False

def test_estado_academico_pendientes(estado_carmen):
 assert estado_carmen['pendientes'] == 40

```

5. EJERCICIOS PROPUESTOS: Práctica en equipo

Ing. Pelayo Quispe Bautista

8 | Página

\*Implementen la suite de pruebas en VS Code, ejecuten pytest y revisen el reporte de cobertura. El objetivo es alcanzar una cobertura mayor o igual al 90% antes de hacer commit a GitHub.

Ejercicio 1: Suite de pruebas para el módulo de asistencia (Nivel básico)

El módulo de cálculo de asistencia de la Sesión 4 necesita su suite de pruebas. Diseña e implementa con pytest:

1. calcular\_porcentaje\_asistencia(fechas\_asistidas, fechas\_programadas): prueba con asistencia perfecta, asistencia 0%, asistencia exactamente 70%, conjunto vacío y fechas asistidas que no están en las programadas.
2. verificar\_alerta\_asistencia(porcentaje): prueba los valores límite 69.9% (alerta), 70.0% (no alerta) y 70.1% (no alerta).
3. registrar\_asistencia(codigo, fecha, base\_datos): prueba registro exitoso, código inexistente (debe retornar False) y fecha duplicada (si se no debe duplicar).

**Meta:** cobertura mayor o igual al 80% con pytest-cov. Reportar las líneas no cubiertas y añadir los casos de prueba faltantes.

Ejercicio 2: Suite TDD para una nueva función del SIGE (Nivel intermedio)

En el enfoque TDD (Test-Driven Development), primero se escriben las pruebas y luego se implementa la función. Practica TDD con la nueva función `calcular_indice_rendimiento()`:

4. Primero: escribe 6 pruebas con pytest que definen exactamente el comportamiento esperado de la función. Las pruebas deben FALLAR porque la función no existe aún.
5. Segundo: implementa `calcular_indice_rendimiento(promedio_avance, asistencia)` que retorne un índice entre 0 y 100 usando la fórmula:  $(\text{promedio}20 * 0.5) + (\text{avance}/100 * 0.3) + (\text{asistencia}/100 * 0.2)$  multiplicado por 100.
6. Tercero: ejecuta pytest hasta que todas las pruebas sean verdes. Ejecuta `pytest-cov` y alcanza 100% de cobertura.

**Reflexión del equipo:** ¿En qué momento del desarrollo fue más útil tener las pruebas escritas primero? ¿Cambió alguna decisión de diseño al ver los casos de prueba antes del código?

Ejercicio 3: Suite de integración para el sistema SIGE completo (Nivel avanzado)

Las pruebas de integración verifican que múltiples módulos funcionen juntos correctamente. Diseña e implementa:

7. Una suite de pruebas que use fixtures compartidos entre los módulos de notas, asistencia y créditos para crear un estudiante completo con todos sus datos.
8. Prueba que `generar_estado_academico()` produzca resultados coherentes con `calcular_promedio_ponderado()` y `calcular_porcentaje_asistencia()` cuando se usan juntos.
9. Prueba que un estudiante en alerta por notas Y en alerta por asistencia Y por debajo del 60% de créditos es correctamente identificado como 'riesgo alto' por el módulo de diagnóstico.
10. Ejecuta la suite completa y genera el reporte de cobertura. El objetivo es cobertura mayor o igual al 90% en todos los módulos. Sube los resultados a GitHub con el mensaje de commit que incluya el porcentaje de cobertura alcanzado.

**Reto final:** ¿Cuántas pruebas adicionales se necesitan para llegar del 90% al 100% de cobertura? ¿Vale la pena el esfuerzo? Documenta tu respuesta como comentario en el README.md del repositorio.

Ing. Pelayo Quispe Bautista

9 | Página





ESCUELA DE

**POSGRADO**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL  
DE HUAMANGA**CONSTANCIA DE ORIGINALIDAD N°0093-2026-UNSCH-EPG/KBA**

El que suscribe; responsable verificador de originalidad de trabajo de tesis de Posgrado en segunda instancia para la **Escuela de Posgrado – UNSCH**; en cumplimiento a la Resolución Directoral N°002-2026-UNSCH-EPG/D, Reglamento de Originalidad de trabajos de Investigación de la UNSCH, otorga lo siguiente:

**CONSTANCIA DE ORIGINALIDAD**

|                                              |                                                                                                                                        |
|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <b>AUTOR</b>                                 | Bach. PELAYO QUISPE BAUTISTA                                                                                                           |
| <b>DENOMINACIÓN DEL PROGRAMA DE ESTUDIOS</b> | MAESTRÍA EN DOCENCIA UNIVERSITARIA                                                                                                     |
| <b>GRADO ACADÉMICO QUE OTORGA</b>            | MAESTRO                                                                                                                                |
| <b>DENOMINACIÓN DEL GRADO ACADÉMICO</b>      | MAESTRO(A) EN DOCENCIA UNIVERSITARIA                                                                                                   |
| <b>TÍTULO DE TESIS</b>                       | Aprendizaje basado en problemas para mejorar las competencias de programación en los estudiantes de Ingeniería de Sistemas, UNSCH 2024 |
| <b>EVALUACIÓN DE ORIGINALIDAD</b>            | 15% de similitud                                                                                                                       |
| <b>N° DE TRABAJO</b>                         | 3012537369                                                                                                                             |
| <b>FECHA</b>                                 | 14 de agosto de 2026                                                                                                                   |

Por tanto, según los artículos 12, 13 y 17 del Reglamento de Originalidad de Trabajos de Investigación, es procedente otorgar la constancia de originalidad con depósito.

Se expide la presente constancia a solicitud del interesado para los fines que crea conveniente.

14 de agosto de 2026.

  
UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA  
ESCUELA DE POSGRADO  
Mg. Kelvis BERTOCAL ARGUMEDO  
Sub Director de Investigación

CC.  
Archivo  
KBA/

# Aprendizaje basado en problemas para mejorar las competencias de programación en los estudiantes de Ingeniería de Sistemas, UNSCH 2024

*por* Pelayo QUISPE BAUTISTA

---

**Fecha de entrega:** 14-ago-2026 03:01p. m. (UTC-0500)

**Identificador de la entrega:** 3012537369

**Nombre del archivo:** Pelayo\_QUISPE\_BAUTISTA\_T.docx (1.57M)

**Total de palabras:** 27528

**Total de caracteres:** 159743

Aprendizaje basado en problemas para mejorar las competencias de programación en los estudiantes de Ingeniería de Sistemas, UNSCH 2024

INFORME DE ORIGINALIDAD

|                     |                     |               |                         |
|---------------------|---------------------|---------------|-------------------------|
| 15%                 | 15%                 | 13%           | 9%                      |
| INDICE DE SIMILITUD | FUENTES DE INTERNET | PUBLICACIONES | TRABAJOS DEL ESTUDIANTE |

FUENTES PRIMARIAS

|    |                                                                |     |
|----|----------------------------------------------------------------|-----|
| 1  | Submitted to Universidad Nacional de San Cristóbal de Huamanga | 3%  |
|    | Trabajo del estudiante                                         |     |
| 2  | repositorio.unsch.edu.pe                                       | 1%  |
|    | Fuente de Internet                                             |     |
| 3  | repositorio.une.edu.pe                                         | 1%  |
|    | Fuente de Internet                                             |     |
| 4  | Submitted to uncedu                                            | 1%  |
|    | Trabajo del estudiante                                         |     |
| 5  | repositorio.unheval.edu.pe                                     | 1%  |
|    | Fuente de Internet                                             |     |
| 6  | hdl.handle.net                                                 | <1% |
|    | Fuente de Internet                                             |     |
| 7  | repositorio.uide.edu.ec                                        | <1% |
|    | Fuente de Internet                                             |     |
| 8  | repositorio.uwiener.edu.pe                                     | <1% |
|    | Fuente de Internet                                             |     |
| 9  | repositorio.upeu.edu.pe                                        | <1% |
|    | Fuente de Internet                                             |     |
| 10 | repositorio.ucv.edu.pe                                         | <1% |
|    | Fuente de Internet                                             |     |
| 11 | repositorio.undac.edu.pe                                       | <1% |
|    | Fuente de Internet                                             |     |
| 12 | repositorio.unach.edu.pe                                       | <1% |
|    | Fuente de Internet                                             |     |
| 13 | repositorio.uns.edu.pe                                         | <1% |
|    | Fuente de Internet                                             |     |
| 14 | repositorio.unjbg.edu.pe                                       | <1% |
|    | Fuente de Internet                                             |     |

repositorio.unfv.edu.pe

|    |                                                                                                                                                                                                                                                 |      |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| 15 | Fuente de Internet                                                                                                                                                                                                                              | <1 % |
| 16 | repositorio.continental.edu.pe<br>Fuente de Internet                                                                                                                                                                                            | <1 % |
| 17 | alicia.concytec.gob.pe<br>Fuente de Internet                                                                                                                                                                                                    | <1 % |
| 18 | repositorio.unc.edu.pe<br>Fuente de Internet                                                                                                                                                                                                    | <1 % |
| 19 | apirepositorio.unh.edu.pe<br>Fuente de Internet                                                                                                                                                                                                 | <1 % |
| 20 | gredos.usal.es<br>Fuente de Internet                                                                                                                                                                                                            | <1 % |
| 21 | editorial.edulearn.ec<br>Fuente de Internet                                                                                                                                                                                                     | <1 % |
| 22 | Submitted to Colegio Sebastián de Benalcázar<br>Trabajo del estudiante                                                                                                                                                                          | <1 % |
| 23 | Cristina Jiménez Hernández. "Diseño, implementación y análisis de diferentes metodologías activas en el proceso de enseñanza-aprendizaje de matemáticas", Universitat Politècnica de Valencia, 2024<br>Publicación                              | <1 % |
| 24 | Espinosa Otoy, Víctor Hugo. "Comparativo de aplicación del método sole en estudiantes de secundaria de las instituciones educativas del distrito del mercado de Lima, 2019", Universidad Católica los Ángeles de Chimbote (Peru)<br>Publicación | <1 % |
| 25 | la-respuesta.com<br>Fuente de Internet                                                                                                                                                                                                          | <1 % |
| 26 | repositorio.utelesup.edu.pe<br>Fuente de Internet                                                                                                                                                                                               | <1 % |
| 27 | Submitted to Universidad Andina Nestor Cáceres Velásquez<br>Trabajo del estudiante                                                                                                                                                              | <1 % |
| 28 | repositorio.uap.edu.pe<br>Fuente de Internet                                                                                                                                                                                                    | <1 % |
| 29 | revista.domhelder.edu.br<br>Fuente de Internet                                                                                                                                                                                                  | <1 % |
| 30 | Submitted to Universidad César Vallejo<br>Trabajo del estudiante                                                                                                                                                                                | <1 % |

|    |                                                                                                                                                                                                                                                          |      |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| 31 | Submitted to Instituto Superior Tecnológico Sudamericano<br>Trabajo del estudiante                                                                                                                                                                       | <1 % |
| 32 | Vilca Paye, Celestino. "Aprendizaje basado en problemas y su incidencia en el desarrollo de las competencias matemáticas en los estudiantes de CEBA Politécnico Los Andes.", Universidad Nacional del Altiplano de Puno (Peru)<br>Publicación            | <1 % |
| 33 | revistalatam.redilat.org<br>Fuente de Internet                                                                                                                                                                                                           | <1 % |
| 34 | transparencia.unitru.edu.pe<br>Fuente de Internet                                                                                                                                                                                                        | <1 % |
| 35 | Submitted to Universidad Estatal Amazonica-<br>Trabajo del estudiante                                                                                                                                                                                    | <1 % |
| 36 | www.uticvirtual.edu.py<br>Fuente de Internet                                                                                                                                                                                                             | <1 % |
| 37 | dspace-uh-tmp.igniteonline.la<br>Fuente de Internet                                                                                                                                                                                                      | <1 % |
| 38 | redined.educacion.gob.es<br>Fuente de Internet                                                                                                                                                                                                           | <1 % |
| 39 | Submitted to Universidad Tecnologica del Peru<br>Trabajo del estudiante                                                                                                                                                                                  | <1 % |
| 40 | repositorio.unaat.edu.pe<br>Fuente de Internet                                                                                                                                                                                                           | <1 % |
| 41 | Submitted to Universidad Nacional de Educacion Enrique Guzman y Valle<br>Trabajo del estudiante                                                                                                                                                          | <1 % |
| 42 | María Jesús Marlene Julca-Asto, Kony Luby Duran-Llano, Gladys Martha Alvarez-Medina, Marleny Donato-Palacios. "El aprendizaje basado en problemas para la enseñanza de las ciencias", Revista Arbitrada Interdisciplinaria Koinonía, 2023<br>Publicación | <1 % |
| 43 | Submitted to Universidad Nacional Santiago Antunez de Mayolo<br>Trabajo del estudiante                                                                                                                                                                   | <1 % |
| 44 | repositorio.uancv.edu.pe<br>Fuente de Internet                                                                                                                                                                                                           | <1 % |
| 45 | www.uv.es<br>Fuente de Internet                                                                                                                                                                                                                          | <1 % |
| 46 | Submitted to Instituto Politecnico Nacional<br>Trabajo del estudiante                                                                                                                                                                                    | <1 % |

|    |                                                                                                                                                                                                                                                                                                      |      |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|
| 47 | Salas Guzmán, Erick Anthony Wilber. "Aprendizaje basado en problemas en el logro de la competencia matemática en los estudiantes del segundo grado del nivel secundario de la Institución Educativa "José Olaya" – Satipo, 2019", Universidad Católica los Ángeles de Chimbote (Peru)<br>Publicación | <1 % |
| 48 | Submitted to Universidad Femenina del Sagrado Corazón<br>Trabajo del estudiante                                                                                                                                                                                                                      | <1 % |
| 49 | Submitted to Universidad Internacional de la Rioja<br>Trabajo del estudiante                                                                                                                                                                                                                         | <1 % |
| 50 | repositorio.umsa.bo<br>Fuente de Internet                                                                                                                                                                                                                                                            | <1 % |
| 51 | repositorio.uti.edu.ec<br>Fuente de Internet                                                                                                                                                                                                                                                         | <1 % |
| 52 | www.scribd.com<br>Fuente de Internet                                                                                                                                                                                                                                                                 | <1 % |

Excluir citas  
Excluir bibliografía

Activo  
Activo

Excluir coincidencias

< 30 words





ESCUELA DE

**POSGRADO**UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL  
DE HUAMANGA

**ACTA DE SUSTENTACIÓN DE TESIS PARA OPTAR**  
**EL GRADO ACADÉMICO DE MAESTRO(A) EN DOCENCIA UNIVERSITARIA**  
**RESOLUCIÓN DIRECTORAL N°00687-2026-UNSCH-EPG/D.**

Siendo las 08:00 a.m. del 20 de julio de 2026 se reunieron en el auditorium de la Escuela de Posgrado de la Universidad Nacional de San Cristóbal de Huamanga, el Jurado Examinador y Calificador de Tesis, presidido por el **Dr. MARCELINO POMASONCCO ILLANES** Director (e) de la Escuela de Posgrado, el **Dr. ROLANDO ALFREDO QUISPE MORALES** Director de la Unidad de Posgrado de la Facultad de Ciencias de la Educación, e integrado por los siguientes miembros: **Dr. LUIS LUCIO ROJAS TELLO** y el **Dr. ALBERTO ALFREDO PALOMINO RIVERA**; para la sustentación oral y pública de la tesis titulada: **APRENDIZAJE BASADO EN PROBLEMAS PARA MEJORAR LAS COMPETENCIAS DE PROGRAMACIÓN EN LOS ESTUDIANTES DE INGENIERÍA DE SISTEMAS, UNSCH 2024**, presentado por el **Bach. PELAYO QUISPE BAUTISTA**. Teniendo como asesora a la **Mtra. ELINAR CARRILLO RIVERO**.

Acto seguido se procedió a la exposición de la tesis, con el fin de optar el Grado Académico de **MAESTRO(A) EN DOCENCIA UNIVERSITARIA**. Formuladas las preguntas, éstas fueron absueltas por el graduando.

A continuación, el Jurado Examinador y Calificador de Tesis procedió a la votación, la que dio como resultado el siguiente calificativo: DIECISIETE ( 17 ).

| CALIFICACION (x)               |   |
|--------------------------------|---|
| Aprobado(a) por Unanimidad.    | X |
| Aprobado(a) por Mayoría.       |   |
| Desaprobado(a) por Unanimidad. |   |
| Desaprobado(a) por Mayoría.    |   |

Marcar con un aspa (x).

Luego, el presidente del Jurado recomienda que la Escuela de Posgrado proponga que se le otorgue al **Bach. PELAYO QUISPE BAUTISTA**, el Grado Académico de **MAESTRO(A) EN DOCENCIA UNIVERSITARIA**. Siendo las 9.10 AM .....hrs. se levanta la sesión.

Se extiende el acta en la ciudad de Ayacucho, a las 9.10 AM .....hrs. del 20 de julio de 2026.

.....  
**Dr. MARCELINO POMASONCCO ILLANES**  
Director (e) de la Escuela de Posgrado.

.....  
**Dr. ROLANDO ALFREDO QUISPE MORALES**  
Director de la UPG-FCE.

.....  
**Dr. LUIS LUCIO ROJAS TELLO**  
Miembro.

.....  
**Dr. ALBERTO ALFREDO PALOMINO RIVERA**  
Miembro.

.....  
**Dr. ENRIQUE JAVIER AGUILAR FELICES**  
Secretario Docente.

**Observaciones:**