

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“EFECTO DEL USO DE ANDAMIAJES EDUCATIVOS EN UN CURSO INTRODUCTORIO A LA PROGRAMACIÓN”

VICENTE MUÑOZ ROJAS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Federico Meza
Profesor Correferente: Andrea Vásquez

Julio - 2025



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Efecto del uso de andamiajes educativos en un curso introductorio a la programación

Nombre del candidato(a): Vicente Muñoz Rojas

Carrera / Grado: Ingeniero civil en informática.

Campus: Santiago San Joaquín ; **Departamento:** Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Federico Meza, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 23/07/2025

; Firma:

Estudiante o Candidato(a):

Fecha: 23/07/2025

; Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

AGRADECIMIENTOS

Quisiera expresar mi sincero agradecimiento a los y las docentes que me brindaron orientación, compartieron información valiosa, referencias bibliográficas, memorias previas y aportaron con retroalimentación clave para la elaboración de este trabajo. Su disposición y apoyo fueron fundamentales para el desarrollo de esta memoria.

De igual forma, agradezco profundamente a los estudiantes que participaron activamente en este estudio sobre andamiajes educativos. Su compromiso, disposición para participar en las distintas dinámicas, honestidad al responder las encuestas y generosidad al entregar comentarios constructivos fueron esenciales para llevar a cabo esta investigación.

Asimismo, expreso mi gratitud a mis amigos y compañeros de universidad, quienes me acompañaron a lo largo de todo este camino. Su apoyo constante, disposición para ayudar y palabras de ánimo fueron un pilar fundamental tanto durante la carrera como en la escritura de esta memoria. Siempre estuvieron ahí cuando más lo necesitaba.

Finalmente, quiero agradecer a mi pareja, Javiera, por su constante apoyo a lo largo de todo el desarrollo de esta memoria. Su compañía, ánimo incondicional y paciencia fueron fundamentales en los momentos más exigentes, especialmente cuando debía revisar tareas, revisar tickets de salida o analizar encuestas. Sin su presencia y aliento, este proceso habría sido mucho más difícil.

Agradezco también a Nube, quien con su silenciosa pero reconfortante compañía, estuvo a mi lado durante largas jornadas y noches de trabajo, haciendo que todo fuera un poco más llevadero.

RESUMEN

Resumen—Esta memoria estudia el impacto del uso de herramientas de andamiaje pedagógico como lenguajes de bloques y ejercicios tipo Parsons en estudiantes de un curso introductorio de programación en la Universidad Técnica Federico Santa María. Se implementó una metodología experimental con un grupo que utilizó estas herramientas y un grupo de control que siguió un enfoque tradicional. Se aplicaron encuestas para evaluar motivación y autoeficacia, y mini tests para medir el desempeño académico, además de comparar los resultados del primer certamen y la primera tarea. Los resultados indican mejoras en la consistencia del aprendizaje, reducción de errores comunes y un efecto positivo en la motivación y autoeficacia. Aunque algunas diferencias no fueron significativas, se observó una evolución favorable en el grupo experimental, especialmente entre estudiantes con mayores dificultades iniciales. Estos hallazgos respaldan el uso de andamiajes como estrategia efectiva en cursos introductorios de programación.

Palabras Clave—Andamiajes pedagógicos; Introducción a la Programación; Lenguajes de bloques; Ejercicios tipo Parsons; Autoeficacia

ABSTRACT

Abstract—This thesis examines the impact of using pedagogical scaffolding tools, such as block-based languages and Parsons problems, on students in an introductory programming course at the Universidad Técnica Federico Santa María. An experimental methodology was implemented, comparing a group that used these tools with a control group that followed a traditional approach. Surveys were conducted to assess motivation and self-efficacy, and mini-tests were used to measure academic performance, along with a comparison of results from the first exam and the first assignment. The results indicate improvements in learning consistency, a reduction in common errors, and a positive effect on motivation and self-efficacy. Although some differences were not statistically significant, a favorable progression was observed in the experimental group, especially among students with greater initial difficulties. These findings support the use of scaffolding as an effective strategy in introductory programming courses.

Keywords—Pedagogical Scaffolding; Introduction to Programming; Block-Based Languages; Parsons Problems; Self-Efficacy

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
ÍNDICE DE FIGURAS	VII
ÍNDICE DE TABLAS	VII
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Problema Identificado	2
1.2 Errores comunes en estudiantes novatos del curso de programación	2
1.3 Impacto de las Dificultades en el Aprendizaje	4
1.4 Objetivos de la Memoria	5
1.4.1 Objetivo general	5
1.4.2 Objetivos específicos	5
CAPÍTULO 2: MARCO CONCEPTUAL	6
2.1 Carga Cognitiva en el Aprendizaje de Programación	6
2.2 Andamiajes Educativos	6
2.3 Teoría de la Zona de Desarrollo Próximo de Vygotsky	7
2.4 Programación por Bloques y Ejercicios Tipo Parsons	8
2.5 Factores Psicológicos en el Aprendizaje de Programación	9
2.5.1 Motivación	9
2.5.2 Autoeficacia	10
2.6 Relación con Trabajos Previos	10
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	12
3.1 Metodología	12
3.1.1 Descripción de las actividades del grupo experimental	12
3.1.2 Evaluación comparativa entre ambos grupos	13
3.2 EduBlocks	14
3.2.1 Ejemplo	14
3.3 Pagina de ejercicios tipo Parsons	15
3.3.1 Pagina Principal	16
3.3.2 Selección de ejercicios	17
3.3.3 Ejemplo	18
3.3.4 Ejercicios de Condicionales en Python	20
3.3.5 Ejercicios de Ciclos While en Python	20
3.3.6 Ejercicios de Ciclos for en Python	21
3.4 Test para evaluar el Impacto de los Andamiajes Educativos en la Motivación y el Rendimiento en Programación	21

3.4.1	Test Inicial y Final	21
3.4.2	Test Inicial	21
3.4.3	Test Final	22
3.5	Ticket de salida al final de cada Ayudantía	23
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		24
4.1	Descripción del proceso	24
4.1.1	Hardware utilizados en las ayudantías	24
4.2	Verificación de homogeneidad entre grupos	25
4.3	Comparar el rendimiento académico	25
4.3.1	Tarea: Sistema de plantaciones automatizado	25
4.3.2	Ticket de salida:	27
4.3.3	Certamen 1	34
4.4	Motivación y Autoeficacia: Análisis	37
4.4.1	Evaluación Inicial de la Motivación	38
4.4.2	Evaluación Inicial de la Autoeficacia	38
4.4.3	Preguntas adicionales del test final	38
4.4.4	Comparación gráfica: antes y después de los andamiajes	39
4.5	Observación de las experiencias	42
CAPÍTULO 5: CONCLUSIONES		44
5.1	Gestión y desarrollo de las ayudantías	44
5.2	Resultados principales y cumplimiento de objetivos	45
5.3	Alcances y limitaciones	46
5.4	Impacto observado en los estudiantes	47
ANEXOS		48
5.1	A. Tests de salida	48
5.1.1	Ejemplo ticket de salida 1:	48
5.2	B. Ejercicios tipo Parsons	49
5.2.1	Ejercicios sobre estructuras condicionales	49
5.2.2	Ejercicios sobre ciclos While	54
5.2.3	Ejercicios sobre ciclos for	57
5.3	C. Certamen 1	59
REFERENCIAS BIBLIOGRÁFICAS		62

ÍNDICE DE FIGURAS

1	Árbol del problema.	4
2	Plataforma EduBlocks.	7
3	Zona de desarrollo próximo con andamiajes.	7
4	Ejercicios tipo Parsons con python (bloques desordenados).	8
5	Ejercicios tipo Parsons con python (bloques ordenados).	9
6	Plataforma Snap.	11
7	Ejemplo de Lenguaje en Bloques en EduBlocks	14
8	El código equivalente en Python generado por EduBlocks.	15
9	Ejecución del programa.	15
10	Home de la página de ejercicios tipo Parsons	17
11	Selector de ejercicios	18
12	Ejercicio tipo Parsons en estado inicial, con bloques de código desordenados.	18
13	Retroalimentación sobre errores al reorganizar el código.	19
14	Solución correcta al ejercicio tipo Parsons.	19
15	Resumen final del ejercicio tipo Parsons completado.	20
16	Distribución de notas por paralelo	28
17	Distribución de rango de notas por paralelo	29
18	Distribución de rango de notas por paralelo	30
19	Distribución de rango de notas por paralelo	31
20	Distribución de rango de notas por paralelo	32
21	Promedio de notas obtenidas en tickets de salida por grupo.	33
22	Distribución de notas Certamen 1	36
23	Boxplot comparativo de notas entre paralelos en el Certamen 1.	37
24	Gráfico comparativo encuesta motivación y autoeficacia	40
25	Percepción de los andamiajes respecto a la motivación y confianza.	42

ÍNDICE DE TABLAS

1	Comparación de rendimiento de la tarea entre grupos.	26
2	Desempeño por criterios y relación con <i>misconceptions</i>	27
3	Comparación de rendimiento del certamen entre grupos.	35
4	Puntaje asignado por respuesta.	39
5	Comparación de resultados antes y después	40

INTRODUCCIÓN

El aprendizaje de la programación en niveles iniciales representa un desafío significativo tanto para estudiantes como para docentes, especialmente debido a la complejidad de los conceptos abstractos y las barreras cognitivas que enfrentan los aprendices novatos. En este contexto, las herramientas de andamiaje pedagógico, como los lenguajes de bloques y los ejercicios tipo Parsons, han surgido como enfoques prometedores para facilitar la comprensión de los conceptos fundamentales de la programación. Estas herramientas buscan reducir la carga cognitiva inicial, permitiendo que los estudiantes se enfoquen en desarrollar habilidades de resolución de problemas y pensamiento computacional.

El presente trabajo aborda el análisis y evaluación del impacto de estas herramientas de andamiaje pedagógico en el contexto de un curso introductorio de programación. El problema principal radica en determinar si estas estrategias pueden mejorar el rendimiento académico, reducir la frecuencia de errores comunes y fomentar la motivación y autoeficacia de los estudiantes, en comparación con los métodos tradicionales de enseñanza.

La solución propuesta consiste en la implementación y evaluación de un conjunto de actividades pedagógicas desarrolladas específicamente para las ayudantías del curso, basadas en el uso de lenguajes de bloques (como EduBlocks) y ejercicios tipo Parsons. Estas herramientas, consideradas como andamiajes educativos, se utilizan con el propósito de facilitar la comprensión de conceptos fundamentales de programación, reduciendo la carga cognitiva de los estudiantes y promoviendo un aprendizaje progresivo.

Durante las ayudantías, los estudiantes del grupo experimental trabajan con estos recursos educativos, mientras que el grupo de control sigue un enfoque tradicional centrado exclusivamente en la escritura de código en Python. A lo largo de las sesiones, se aplican instrumentos de evaluación como tests de salida (mini tests), encuestas de motivación y autoeficacia, con el fin de monitorear el progreso del estudiantado y capturar información relevante sobre su experiencia de aprendizaje.

Los resultados obtenidos muestran indicios de que el uso de andamiajes pedagógicos no solo mejora la consistencia del rendimiento académico, sino que también influye positivamente en la percepción de autoeficacia de los estudiantes. Además, se observó una reducción en la frecuencia de errores comunes en programación y un efecto nivelador que benefició especialmente a quienes presentaban mayor rezago inicial. Estos hallazgos respaldan el valor de incorporar estrategias de apoyo estructurado en cursos introductorios, como medio para mejorar tanto el desempeño como la experiencia de aprendizaje de los estudiantes.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

El curso Introducción a la Programación es una asignatura fundamental en los planes de estudio de todas las carreras de Ingeniería Civil de la Universidad Técnica Federico Santa María (UTFSM). Este curso tiene como objetivo principal desarrollar en los estudiantes el pensamiento computacional para resolver problemas de ingeniería mediante la programación estructurada, utilizando Python como lenguaje base.

Dado que este curso es obligatorio para todas las carreras de Ingeniería Civil, incluye a estudiantes con perfiles diversos, muchos de los cuales no tienen experiencia previa en programación. La falta de esta base, sumada a la complejidad inherente de aprender a programar, crea un entorno desafiante para la enseñanza y el aprendizaje [Guzdial, 2004].

1.1. Problema Identificado

El curso Introducción a la Programación ha mostrado ser un desafío significativo para un gran número de estudiantes, especialmente aquellos que no provienen de áreas relacionadas con la informática. En el campus San Joaquín, la tasa de aprobación del curso oscila entre el 70 % y el 80 %, y rara vez supera este último valor. Esto significa que entre el 20 % y el 30 % de los estudiantes no logran aprobar el curso cada año.

Además, la Universidad Técnica Federico Santa María cuenta con aproximadamente 3000 estudiantes que cursan esta asignatura entre sus cinco sedes. Esta gran cantidad de alumnos enfrenta una variedad de dificultades que van desde errores básicos de sintaxis hasta problemas más complejos relacionados con la lógica de programación. Los errores comunes incluyen la incorrecta asignación de variables, problemas con la indentación, el uso incorrecto de ciclos y condicionales, y la confusión entre conceptos matemáticos y de programación.

1.2. Errores comunes en estudiantes novatos del curso de programación

Los estudiantes novatos en cursos introductorios de programación enfrentan una serie de dificultades recurrentes que han sido ampliamente documentadas en la literatura. Según el trabajo de revisión titulado “Students Misconceptions and Other Difficulties in Introductory Programming: A Literature Review” [Qian y Lehman, 2018], existen varios errores y conceptos erróneos (*misconceptions*) que suelen aparecer en esta etapa de aprendizaje, afectando negativamente el progreso de los estudiantes.

Entre los errores más comunes se destacan:

- Confusión con la asignación de variables: Los estudiantes a menudo confunden la asignación de valores a variables con la igualdad matemática. Esto les lleva a interpretar incorrectamente las operaciones de asignación, creyendo que expresiones como $x = x + 1$ son inválidas o ilógicas.
- Dificultades con la sintaxis: Los errores de sintaxis son comunes entre los principiantes y pueden incluir problemas como olvidar puntos y comas, corchetes, paréntesis o cometer errores en la indentación. La sintaxis específica del lenguaje puede representar una carga cognitiva adicional que interfiere en la comprensión de los conceptos lógicos subyacentes.
- Problemas con el control de flujo: Conceptos como los bucles while y for, o las estructuras condicionales if-else, suelen causar confusión. Los estudiantes tienden a tener dificultades para comprender cuándo y cómo estas estructuras cambian el flujo de ejecución del programa.
- Manipulación incorrecta de listas o arrays: Los estudiantes novatos suelen tener problemas al trabajar con estructuras de datos como listas, especialmente en lo que respecta a la indexación, la manipulación de elementos y el recorrido de las listas mediante bucles.
- Errores lógicos: Además de los errores de sintaxis, los estudiantes también cometen errores lógicos, como no prever correctamente todas las posibles condiciones de entrada o no anticipar el comportamiento del programa en diferentes escenarios.
- Conceptos erróneos sobre las funciones: Los estudiantes pueden tener problemas para entender el concepto de funciones como bloques reutilizables de código y suelen confundir el paso de parámetros y el retorno de valores con las operaciones de entrada y salida del programa.
- Confusión con conceptos de otras materias: Es común que los estudiantes confundan conceptos de programación con ideas de otras materias, como matemáticas o física. Por ejemplo, pueden interpretar erróneamente operaciones lógicas o el uso de variables debido a su familiaridad con las nociones matemáticas, lo que genera problemas al comprender la lógica de los algoritmos.

Estos errores y *misconceptions* no solo frenan el progreso en el curso, sino que también aumentan la frustración de los estudiantes, lo que puede desmotivarlos. La literatura sugiere que, para mejorar el aprendizaje de los estudiantes, es importante no solo corregir estos errores una vez que ocurren, sino también prevenir su aparición a través de metodologías pedagógicas que faciliten la comprensión de los conceptos de programación.

1.3. Impacto de las Dificultades en el Aprendizaje

Las dificultades mencionadas no solo impactan el rendimiento académico de los estudiantes, sino que también afectan negativamente su confianza y motivación para aprender programación. La falta de habilidades básicas en programación puede convertirse en una barrera significativa para su progreso en el curso y en otros cursos posteriores que requieren conocimientos de programación.

La enseñanza de la programación a estudiantes sin experiencia previa es particularmente desafiante, especialmente en un contexto donde la programación no forma parte del currículum escolar como en Chile. Los estudiantes que nunca han tenido contacto con la codificación enfrentan una curva de aprendizaje muy empinada, lo que aumenta la probabilidad de desmotivación y abandono del curso.

Para entender mejor las dificultades que enfrentan los estudiantes y cómo se relacionan entre sí, se presenta un árbol del problema (Ver figura 1). Este diagrama ayuda a visualizar las causas y efectos asociados con la baja comprensión y el alto índice de reprobación en el curso Introducción a la Programación (En el árbol, el problema se define como “Los estudiantes enfrentan dificultades en el curso de programación”).



Figura 1: Árbol del problema.
Fuente: Elaboración propia.

1.4. Objetivos de la Memoria

El desarrollo de esta memoria tiene como propósito principal analizar y evaluar el impacto de la implementación de herramientas de andamiaje pedagógico, como lenguajes de bloques y ejercicios tipo Parsons, en el aprendizaje de los estudiantes en un curso introductorio de programación.

1.4.1. Objetivo general

Evaluar el impacto del uso de herramientas de andamiaje pedagógico, como lenguajes de bloques y ejercicios tipo Parsons, en el rendimiento académico y la comprensión de conceptos fundamentales en un curso introductorio de programación.

1.4.2. Objetivos específicos

- Comparar el rendimiento académico de los estudiantes que utilizan andamiajes pedagógicos, como lenguajes en bloques y ejercicios tipo Parsons, frente a aquellos que emplean métodos tradicionales de programación en Python.
- Determinar la influencia de los andamiajes pedagógicos, como lenguajes en bloques y ejercicios tipo Parsons, en la motivación y confianza de los estudiantes para aprender programación.
- Analizar la influencia de las herramientas de andamiaje en la frecuencia de errores comunes entre estudiantes novatos.

CAPÍTULO 2

MARCO CONCEPTUAL

El presente capítulo establece los fundamentos teóricos y conceptuales sobre los cuales se basa la solución propuesta en esta memoria. A través de la definición de conceptos clave y su interrelación con la problemática abordada, se busca delimitar el contexto de la investigación, estableciendo un lenguaje común que facilite la comprensión del trabajo y su relevancia en el ámbito educativo.

2.1. Carga Cognitiva en el Aprendizaje de Programación

La teoría de la carga cognitiva, desarrollada por John Sweller, postula que la capacidad cognitiva de los individuos es limitada, y que la sobrecarga de información puede dificultar el aprendizaje efectivo [Sweller, 1988]. En el contexto de la enseñanza de la programación, esta teoría es particularmente relevante, dado que los estudiantes deben procesar simultáneamente múltiples elementos, como la sintaxis del lenguaje, la lógica de los algoritmos, y la solución de problemas. Esta sobrecarga puede llevar a la frustración y a la dificultad para internalizar los conceptos clave, especialmente entre aquellos estudiantes que no tienen experiencia previa en codificación.

Para mitigar esta carga cognitiva, se propone el uso de herramientas pedagógicas que simplifiquen el proceso de aprendizaje. Los andamiajes educativos, por ejemplo, permiten a los estudiantes enfocarse en la lógica y la estructura del código sin ser abrumados por la necesidad de recordar y aplicar correctamente la sintaxis desde el inicio.

2.2. Andamiajes Educativos

El concepto de andamiaje educativo se refiere a las estrategias de apoyo temporal que se brindan a los estudiantes para facilitar el aprendizaje de nuevos conceptos o habilidades. Estos apoyos se retiran gradualmente a medida que los estudiantes desarrollan competencia y autonomía en la tarea. Jerome Bruner fue uno de los pioneros en introducir este concepto en la educación, inspirado en las ideas de la teoría de la zona de desarrollo próximo de Vygotsky [Vygotsky, 1978].

En la enseñanza de la programación, los andamiajes pueden tomar la forma de plataformas de programación por bloques, como EduBlocks ¹ (Ver figura 2), que permiten a los estudiantes ensamblar código de manera intuitiva. Al reducir la carga cognitiva inicial, los estudiantes

¹<https://edublocks.org/>

pueden concentrarse en entender la lógica y el flujo de los programas, antes de enfrentarse a la tarea de escribir código por completo.

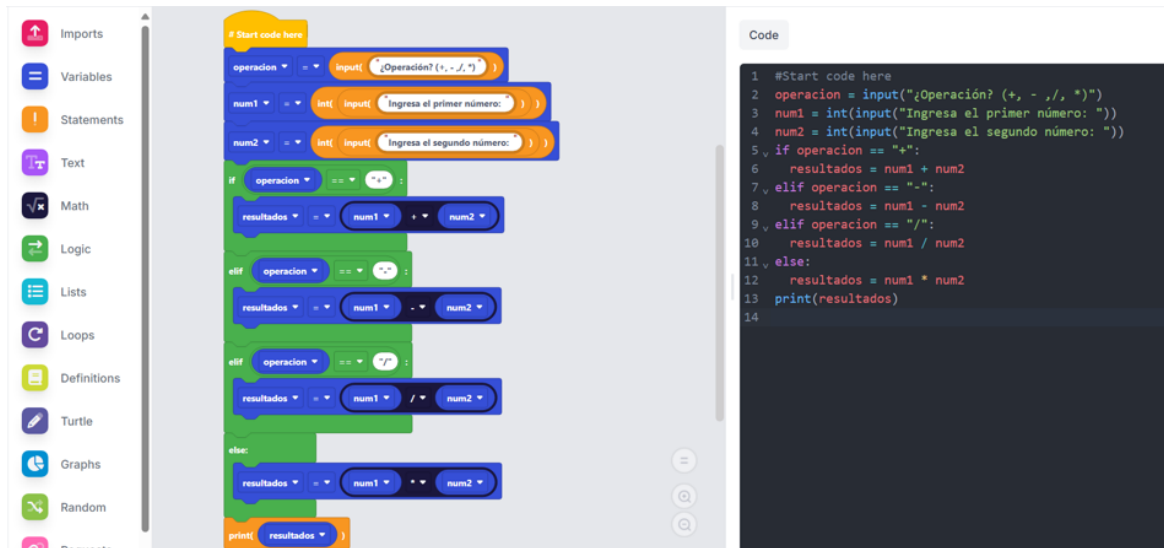


Figura 2: Plataforma EduBlocks.
Fuente: Elaboración propia.

2.3. Teoría de la Zona de Desarrollo Próximo de Vygotsky

Lev Vygotsky, psicólogo y teórico del desarrollo cognitivo, propuso la teoría de la zona de desarrollo próximo (ZDP), la cual describe el espacio entre lo que un estudiante es capaz de hacer por sí solo y lo que puede lograr con la ayuda de un mentor o mediante el uso de andamiajes u otros tipos de asistencia (Ver Figura 3) [Vygotsky, 1978]. Según esta teoría, el aprendizaje es más efectivo cuando se proporciona asistencia dentro de esta zona, permitiendo al estudiante alcanzar un nivel superior de comprensión y habilidad.



Figura 3: Zona de desarrollo próximo con andamiajes.
Fuente: Elaboración propia.

Aplicada al contexto de la enseñanza de la programación, la ZDP sugiere que los estudiantes se benefician de recibir apoyo en las etapas iniciales del aprendizaje, especialmente en tareas que están fuera de su alcance independiente. Los andamiajes educativos, como la programación por bloques y los ejercicios tipo Parsons, operan dentro de esta zona, guiando a los estudiantes mientras desarrollan las competencias necesarias para codificar de manera autónoma.

2.4. Programación por Bloques y Ejercicios Tipo Parsons

La programación por bloques es una metodología de enseñanza que simplifica el proceso de codificación al permitir que los estudiantes ensamblen programas utilizando bloques visuales que representan estructuras de código. Esta metodología no solo facilita la comprensión de la lógica de programación, sino que también reduce errores comunes relacionados con la sintaxis. Herramientas como EduBlocks (Ver figura 2) permiten a los estudiantes iniciarse en la programación de una manera más accesible, haciendo la transición hacia la codificación en texto plano más gradual y menos intimidante.

Por otro lado, los ejercicios tipo Parsons (ver figura 4 y 5) consisten en fragmentos de código desordenados que los estudiantes deben reorganizar para formar un programa funcional. Esta técnica, también conocida como “Parsons problems”, ha demostrado ser efectiva para fortalecer la comprensión de la estructura y lógica del código, al tiempo que elimina la presión de escribirlo desde cero. Además, estos ejercicios se consideran una herramienta valiosa y atractiva para los cursos introductorios de programación [Parsons y Haden, 2006].

Drag from here



Figura 4: Ejercicios tipo Parsons con python (bloques desordenados).

Fuente: Elaboración propia.

Construct your solution here

```
a = 10
```

```
b = 30
```

```
suma = a + b
```

```
print(suma)
```

Figura 5: Ejercicios tipo Parsons con python (bloques ordenados).

Fuente: Elaboración propia.

Ambos enfoques no están diseñados específicamente para trabajar en conjunto, pero su combinación puede ser altamente beneficiosa. Al integrarse, tienen el potencial de reducir la carga cognitiva inicial, permitiendo que los estudiantes se enfoquen en los aspectos conceptuales de la programación antes de enfrentarse a los desafíos de la sintaxis y la codificación detallada.

2.5. Factores Psicológicos en el Aprendizaje de Programación

En el contexto de la enseñanza de la programación, los factores psicológicos como la motivación y la autoeficacia juegan un papel fundamental en el rendimiento académico y la persistencia de los estudiantes. Estos factores influyen en la disposición de los estudiantes para enfrentarse a los desafíos y persistir en la resolución de problemas, lo cual es crucial en disciplinas que presentan una alta curva de aprendizaje, como la programación [Tek *et al.*, 2018].

2.5.1. Motivación

La motivación es uno de los principales determinantes del éxito académico. Ryan y Deci, en su investigación “Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions”, destacan la importancia de la motivación intrínseca y extrínseca en el aprendizaje [Ryan y Deci, 2000]. La motivación intrínseca se refiere al deseo de aprender por el interés y la satisfacción personal, mientras que la motivación extrínseca está impulsada por recompensas externas, como las calificaciones o el reconocimiento. En el aprendizaje de la programación, los estudiantes motivados intrínsecamente son más propensos a involucrarse activamente con los contenidos, lo que mejora su capacidad para resolver problemas complejos y aprender nuevas habilidades.

En el contexto del curso Introducción a la Programación, muchos estudiantes pueden sentirse desmotivados debido a la falta de experiencia previa en programación y a la dificultad de

los conceptos que se presentan. Sin embargo, el uso de herramientas de andamiaje pedagógico, como los lenguajes de bloques y los ejercicios tipo Parsons, puede aumentar la motivación al proporcionar un entorno de aprendizaje más accesible y menos intimidante. Estas herramientas simplifican las tareas iniciales, lo que permite a los estudiantes experimentar logros tempranos y, a su vez, fomentar una mayor motivación para continuar aprendiendo.

2.5.2. Autoeficacia

La autoeficacia es la creencia de una persona en su capacidad para llevar a cabo acciones necesarias para alcanzar un objetivo. Según el psicólogo Albert Bandura, la autoeficacia influye en la forma en que los individuos enfrentan los desafíos, su nivel de esfuerzo y su persistencia ante las dificultades [Bandura, 1997]. En el aprendizaje de la programación, una alta autoeficacia permite que los estudiantes aborden problemas con confianza y persistan en la búsqueda de soluciones, mientras que una baja autoeficacia puede llevar a la evitación y a la frustración.

Los estudiantes en el curso Introducción a la Programación especialmente aquellos sin experiencia previa, pueden tener una baja percepción de su autoeficacia al enfrentarse a tareas que requieren lógica algorítmica y resolución de problemas. Las herramientas de andamiaje pedagógico proporcionan una estructura que guía el aprendizaje y facilita la construcción gradual de habilidades. Al reducir la complejidad inicial, los estudiantes logran pequeños éxitos tempranos, lo que refuerza su creencia en su capacidad para aprender programación y, en última instancia, incrementa su autoeficacia.

2.6. Relación con Trabajos Previos

Un trabajo previo relevante es la memoria titulada “Impacto del uso de un lenguaje basado en bloques en el desarrollo de habilidades de programación” de Francisco Vásquez Pinto [Pinto, 2020]. En dicho estudio, se comparan dos grupos de estudiantes del curso de programación en la UTFSM. A uno de los grupos se le enseña utilizando Snap!, un lenguaje de pseudocódigo en bloques (ver figura 6), mientras que al otro grupo se le instruye utilizando Python, de la manera tradicional.

Los resultados de ese trabajo no muestran diferencias significativas entre los dos grupos en términos de rendimiento al finalizar el curso. Ambos grupos reportaron sentirse motivados con la programación y consideraron que el curso les fue útil. Sin embargo, el objetivo principal del estudio era observar el impacto de un lenguaje basado en bloques en el desarrollo de habilidades de programación. Un aspecto importante que surge de la investigación es que, aunque los estudiantes del grupo Snap! pueden haber adquirido habilidades de pensamiento computacional, no estuvieron directamente expuestos a la codificación en un lenguaje de programación tradicional como Python. Esto plantea la cuestión de si el uso de un lenguaje



Figura 6: Plataforma Snap.
Fuente: Elaboración propia.

en bloques realmente contribuye al desarrollo de habilidades de programación per se, ya que no estuvieron escribiendo código en un entorno de programación tradicional.

Este trabajo es un antecedente clave para la presente investigación, ya que destaca la importancia de evaluar si los lenguajes en bloques permiten a los estudiantes no solo desarrollar el pensamiento computacional, sino también adquirir habilidades de codificación transferibles a otros lenguajes textuales como Python. Por esta razón, se ha optado por utilizar una plataforma que integra un lenguaje en bloques con Python, como EduBlocks. Esto permite que los estudiantes no se enfrenten a un contenido completamente diferente al del curso, a diferencia del estudio anterior que creó una versión especial del curso BJC-USM para el grupo de experimentación.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

La presente propuesta de solución aborda el problema identificado en los cursos introductorios de programación, mediante la implementación y comparación de estrategias pedagógicas basadas en andamiajes educativos. Estas estrategias incluyen el uso de una plataforma de programación por bloques, EduBlocks, y ejercicios tipo Parsons diseñados en un programa desarrollado específicamente para este estudio. El objetivo es evaluar cómo estas herramientas impactan el aprendizaje de los estudiantes en cuanto a comprensión, motivación y reducción de errores comunes.

3.1. Metodología

Se conformarán dos grupos de estudiantes inscritos en el curso Introducción a la Programación: un grupo experimental, que empleará las herramientas de andamiajes educativos exclusivamente durante las ayudantías, y un grupo de control, que tendrá ayudantías siguiendo el enfoque tradicional basado en Python, sin el uso de estas herramientas pedagógicas.

3.1.1. Descripción de las actividades del grupo experimental

1. Primera y segunda semana: Uso de EduBlocks

- Los estudiantes del grupo experimental trabajarán con la plataforma EduBlocks, enfocándose en la comprensión de la sintaxis.
- El uso de bloques visuales reducirá la carga cognitiva asociada a la sintaxis, permitiendo un enfoque en la lógica del código.
- Durante estas semanas, las actividades estarán limitadas a la resolución de programas secuenciales.

2. Tercera, cuarta y quinta semana: Ejercicios tipo Parsons

- El mismo grupo experimental realizará actividades con ejercicios tipo Parsons, organizando fragmentos de código desordenados para formar programas funcionales.
- Los ejercicios se implementarán en una página web diseñada por el autor.

- Durante estas semanas, las actividades estarán enfocadas en la resolución de problemas y en la realización de ejercicios tipo Parsons en Python, abarcando estructuras condicionales, ciclos while, ciclos for, y combinaciones de estas en Python.

3.1.2. Evaluación comparativa entre ambos grupos

Ambos grupos seguirán las mismas clases teóricas tradicionales del curso introductorio a la programación, utilizando Python como lenguaje base. La principal diferencia radicará en las actividades desarrolladas durante las ayudantías:

- El grupo experimental utilizará herramientas de andamiajes educativos, como EduBlocks y ejercicios tipo Parsons.
- El grupo de control realizará ayudantías siguiendo el enfoque tradicional, sin el uso de herramientas pedagógicas adicionales.

Para evaluar el impacto de las metodologías, se aplicarán las siguientes evaluaciones:

- **Test de motivación y confianza:** Diseñados para evaluar la motivación y la autoeficacia de los estudiantes en el aprendizaje de programación. Se aplicarán dos pruebas: la primera al inicio de la primera ayudantía y la segunda antes del primer certamen en alguna ayudantía, una vez finalizados los andamiajes educativos.
- **Test de salida al final de cada ayudantía:** De 7 a 10 minutos de duración, diseñados para evaluar el rendimiento. Estos resultados serán clave para comparar el desempeño entre los estudiantes que utilizaron herramientas de andamiajes educativos y aquellos que no.
- **Comparación de notas en el primer certamen:** Se analizarán las calificaciones obtenidas por los estudiantes en el primer certamen del curso, comparando los resultados entre quienes utilizaron herramientas de andamiajes educativos y quienes no.
- **Comparación de notas en la primera tarea:** Se realizará un análisis de las calificaciones obtenidas en la primera tarea del curso, con el fin de identificar diferencias en el desempeño entre los estudiantes que utilizaron herramientas de andamiaje educativo y aquellos que no. Además, se examinará la frecuencia y el tipo de errores conceptuales (*misconceptions*) cometidos, con el propósito de evaluar si el uso de andamiajes contribuye a reducir este tipo de errores.

3.2. EduBlocks

EduBlocks es una plataforma web de programación basada en bloques que permite a los estudiantes crear programas arrastrando y soltando bloques visuales. Durante esta investigación, se utilizará EduBlocks únicamente para trabajar con programas secuenciales la primera y parte de la segunda semana de ayudantías. La elección de esta herramienta busca reducir la carga cognitiva inicial asociada al aprendizaje de la sintaxis, permitiendo que los estudiantes se enfoquen en los conceptos fundamentales de programación.

Referencias a las actividades en EduBlocks:

- Resolución de ejercicios básicos de entrada y salida de datos.
- Programas que involucren operaciones matemáticas simples.
- Uso de estructuras secuenciales para resolver problemas cotidianos.

3.2.1. Ejemplo

En el ejemplo mostrado, se utilizan tres imágenes que ilustran el flujo de trabajo en EduBlocks, un entorno de programación visual diseñado para aprender los conceptos básicos de la programación de una forma intuitiva y amigable.

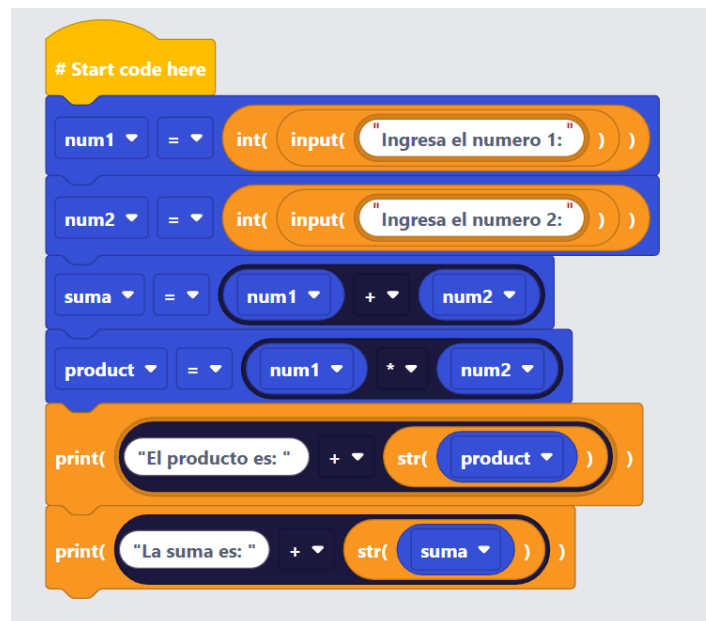


Figura 7: Ejemplo de Lenguaje en Bloques en EduBlocks
Fuente: Elaboración propia.

```
1  #Start code here
2  num1 = int(input("Ingresa el numero 1:"))
3  num2 = int(input("Ingresa el numero 2:"))
4  suma = num1 + num2
5  product = num1 * num2
6  print("El producto es: " + str(product))
7  print("La suma es: " + str(suma))
8
```

Figura 8: El código equivalente en Python generado por EduBlocks.
Fuente: Elaboración propia.

A continuación, se muestra la salida del programa cuando el usuario ejecuta y ingresa dos números cualquiera (10 y 50):

```
Ingresa el numero 1:10
Ingresa el numero 2:50
El producto es: 500
La suma es: 60
```

Figura 9: Ejecución del programa.
Fuente: Elaboración propia.

Este flujo de trabajo permite a los usuarios familiarizarse con conceptos clave de la programación, como la entrada de datos, operaciones aritméticas y la visualización de resultados, todo de manera progresiva y visual.

3.3. Pagina de ejercicios tipo Parsons

La página web fue desarrollada por el autor de esta memoria utilizando el framework React para el frontend y un backend básico con Express, junto con una base de datos MongoDB. Con el fin de mantenerla disponible durante los meses en que se utilizarían los andamiajes educativos, se creó una instancia en Google Cloud Platform, donde se alojó el sitio web.

Esta plataforma está diseñada para implementar ejercicios tipo Parsons de forma interactiva, permitiendo a los estudiantes resolver problemas reorganizando bloques de código desordenados hasta formar programas funcionales. La herramienta ofrece distintos niveles de dificultad, lo que facilita un aprendizaje progresivo desde conceptos fundamentales hasta estructuras más complejas. El código fuente de la página de ejercicios está disponible en el siguiente repositorio de GitHub².

Características principales:

- Ejercicios diseñados para reforzar la lógica de programación y la estructura de código.
- Retroalimentación inmediata sobre el desempeño del estudiante en cada nivel.
- Registro de métricas como tiempo de resolución y errores cometidos, lo que facilita el análisis del progreso de los estudiantes.

Referencias a las actividades:

Se diseñaron y utilizaron un conjunto de ejercicios tipo Parsons para reforzar conceptos clave de programación en Python, organizados de la siguiente manera:

- **Condicionales:** 15 ejercicios distintos enfocados en fortalecer la comprensión y aplicación de estructuras condicionales.
- **Ciclos while:** 10 ejercicios diferentes orientados a mejorar el dominio en el uso de bucles while.
- **Ciclos for:** 10 ejercicios variados destinados a practicar y consolidar el uso de bucles for pero sin el uso de listas.

Los ejercicios están organizados con una dificultad progresiva: los primeros se centran en conceptos básicos y son de muy baja complejidad, mientras que, a medida que se avanza, presentan problemas más desafiantes para estimular el aprendizaje continuo. Además, se incluyen distractores, que son bloques de código que no forman parte de la solución final pero están disponibles como opciones para ser seleccionados.

3.3.1. Pagina Principal

La Página Principal ofrece la interfaz inicial donde los usuarios pueden acceder al selector de ejercicios. En esta página se destacan las principales características del sitio, que permiten a los usuarios aprender y mejorar sus habilidades en programación, resolviendo ejercicios

²<https://github.com/Vichooz/Parsons-Problem-React>

tipo Parsons en Python. Además, se brinda retroalimentación en tiempo real para facilitar el proceso de aprendizaje y optimización de sus conocimientos.

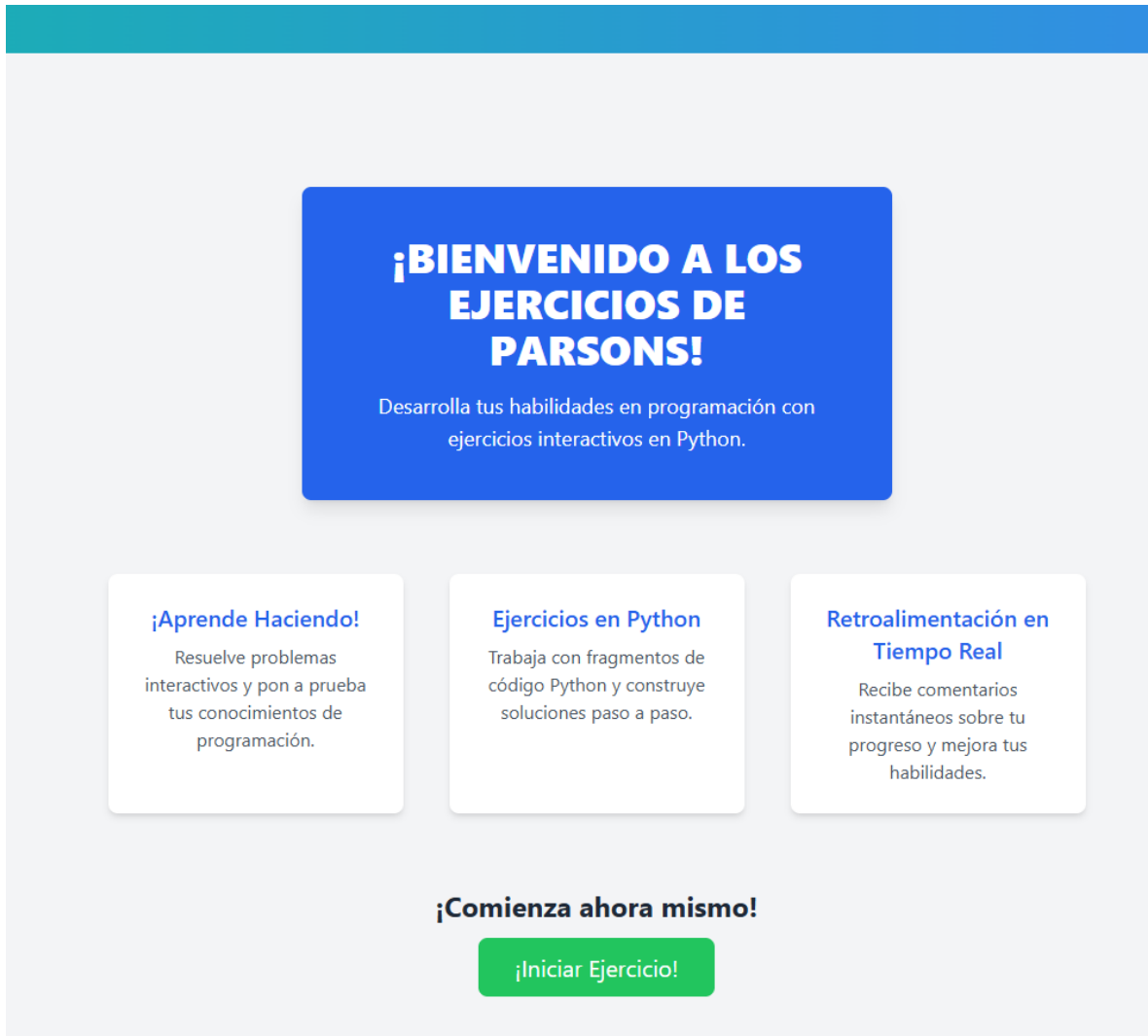


Figura 10: Home de la página de ejercicios tipo Parsons

Fuente: Elaboración propia.

3.3.2. Selección de ejercicios

La sección de Selección de Niveles permite a los estudiantes elegir tanto el nivel de dificultad como el concepto que desean estudiar. El diseño de la interfaz organiza los niveles de forma clara y facilita la navegación entre ellos. Los niveles están distribuidos de la siguiente manera: los primeros 15 niveles corresponden a ejercicios sobre condicionales, del nivel 16 al 25 se enfocan en ejercicios con ciclos while, y del nivel 26 al 35 se centran en ejercicios con ciclos for. Además, se implementó un sistema de colores que permite identificar visualmente el

tipo de ejercicio: los ejercicios sobre condicionales están marcados en verde, los de ciclos while en amarillo, y los de ciclos for en morado.



Figura 11: Selector de ejercicios
Fuente: Elaboración propia.

3.3.3. Ejemplo

A continuación, se muestra un ejemplo de un ejercicio tipo Parsons disponible en la plataforma web. En este caso, el desafío consiste en reorganizar bloques de código para construir un programa funcional que compare dos números, a y b. El objetivo del programa es determinar cuál de los dos números es mayor, si son iguales o simplemente diferentes y mostrarlo por pantalla.

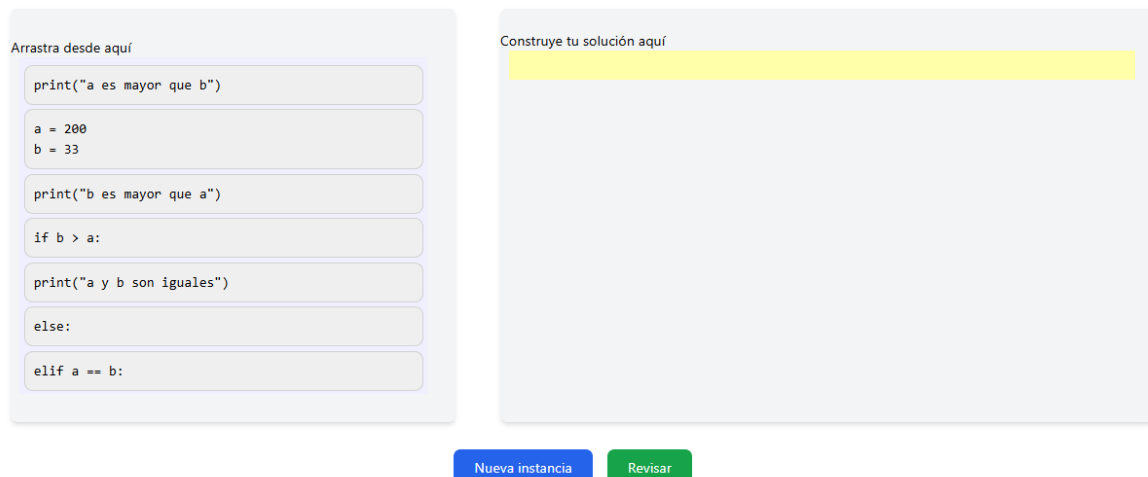


Figura 12: Ejercicio tipo Parsons en estado inicial, con bloques de código desordenados.
Fuente: Elaboración propia.

El estudiante debe reorganizar los bloques de código para lograr que el programa cumpla con la funcionalidad especificada. La interfaz proporciona retroalimentación inmediata sobre los aciertos y errores en el orden del código.

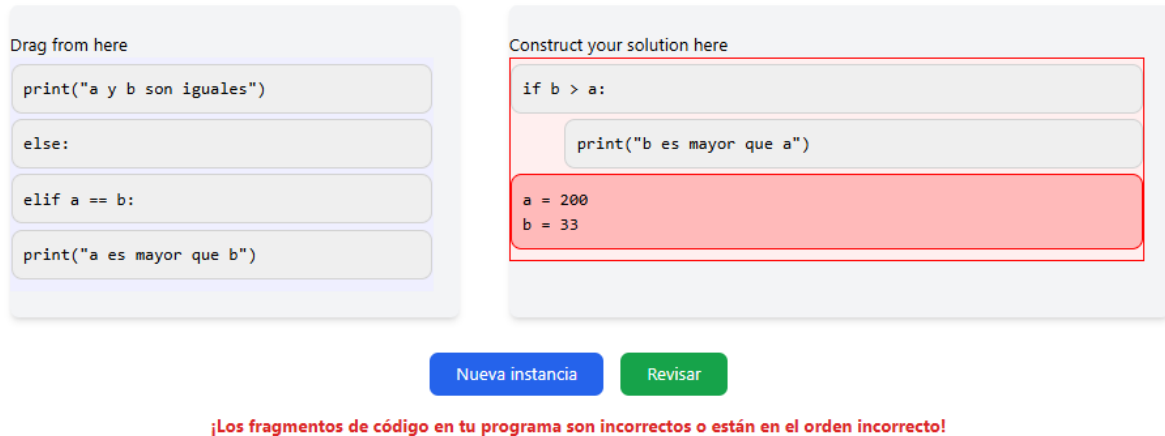


Figura 13: Retroalimentación sobre errores al reorganizar el código.
Fuente: Elaboración propia.

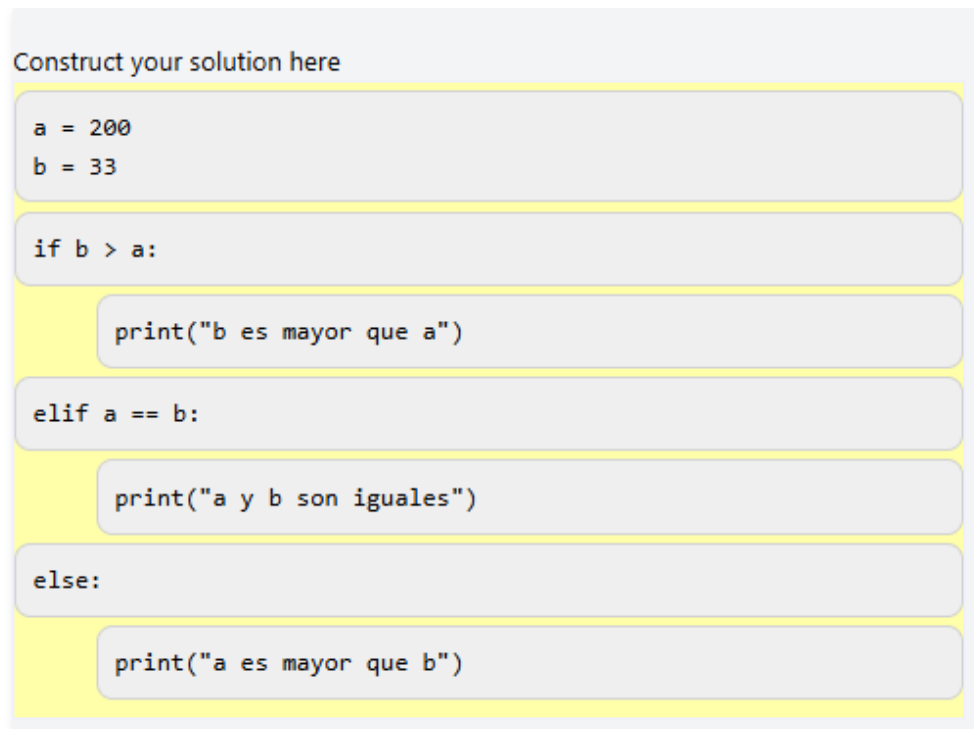


Figura 14: Solución correcta al ejercicio tipo Parsons.
Fuente: Elaboración propia.

Una vez finalizado el ejercicio, el programa genera un resumen que incluye el tiempo empleado y el número de errores cometidos durante la reorganización de los bloques de código. Cada vez que el estudiante considere que ha completado correctamente el ejercicio, podrá seleccionar la opción de verificar su solución. Si la solución resulta incorrecta, se incrementará el contador de errores, ya que el estudiante asumió que su respuesta era correcta cuando no lo era.

¡Ejercicio completado!

¡Felicidades! Has completado el ejercicio
en:

03:06

Errores cometidos: 0

[Siguiente ejercicio](#)

Figura 15: Resumen final del ejercicio tipo Parsons completado.

Fuente: Elaboración propia.

3.3.4. Ejercicios de Condicionales en Python

En la plataforma los estudiantes tendrán acceso a un total de 15 ejercicios diseñados para reforzar el aprendizaje de estructuras condicionales en Python. Durante la ayudantía, se repasarán los conceptos teóricos relacionados con los condicionales. Posteriormente, los estudiantes dispondrán de aproximadamente 20-25 minutos para practicar en la plataforma y hacer ejercicios. Esta actividad se caracteriza por su flexibilidad, ya que no hay un límite de tiempo ni una cantidad mínima de ejercicios que deban completarse. Cada estudiante podrá seleccionar y realizar los ejercicios que prefiera, tantas veces como lo desee (incluso fuera de la ayudantía). Los ejercicios disponibles sin distractores se pueden revisar en el Anexo sección B 5.2.1.

3.3.5. Ejercicios de Ciclos While en Python

En la plataforma los estudiantes tendrán acceso a un total de 10 ejercicios diseñados para reforzar el aprendizaje de bucles `while` en Python. Durante la ayudantía, se repasarán los conceptos teóricos relacionados con los ciclos `while`. Posteriormente, los estudiantes dispondrán de aproximadamente 20-25 minutos para practicar en la plataforma. La cantidad de

distractores es significativamente mayor en comparación con los niveles relacionados con las estructuras condicionales, lo que incrementa el desafío y promueve un análisis más detallado por parte de los estudiantes. Además, se incluirán ejercicios que combinan el uso de ciclos `while` con estructuras condicionales, proporcionando un nivel adicional de complejidad y fomentando la integración de ambos conceptos en la resolución de problemas. Los ejercicios disponibles sin distractores se pueden revisar en el Anexo sección B.5.2.2

3.3.6. Ejercicios de Ciclos `for` en Python

En la plataforma los estudiantes tendrán acceso a un total de 10 ejercicios diseñados para reforzar el aprendizaje de bucles `for` en Python pero sin el uso de listas. Durante la ayudantía, se repasarán los conceptos teóricos relacionados con los ciclos `for` y se usará la plataforma. Al igual que en la sección de ciclos `while` también se combinarán con ejercicios con condicionales. Los ejercicios disponibles sin distractores se pueden revisar en el Anexo sección B.5.2.3

3.4. Test para evaluar el Impacto de los Andamiajes Educativos en la Motivación y el Rendimiento en Programación

3.4.1. Test Inicial y Final

Estos tests tienen como objetivo medir cómo varían la motivación y la autoeficacia de los estudiantes respecto al aprendizaje de programación a lo largo del curso.

3.4.2. Test Inicial

- **Método:** Se realiza en la primera ayudantía, con una duración aproximada de 2-3 minutos.
- **Propósito:** Determinar el estado inicial de motivación y confianza de los estudiantes, para posteriormente evaluar el impacto de los andamiajes pedagógicos (como lenguajes en bloques y ejercicios tipo Parsons).
- **Instrucciones:** A continuación, encontrarás una serie de afirmaciones relacionadas con tu experiencia y percepción sobre el aprendizaje de programación. Marca la opción que mejor refleje tu nivel de acuerdo con cada afirmación. Responde de forma honesta; no hay respuestas correctas ni incorrectas.
- **Escala de respuesta:**

- Totalmente en desacuerdo
- En desacuerdo
- Neutral
- De acuerdo
- Totalmente de acuerdo

Dimensiones Evaluadas:

■ Motivación:

1. Estoy emocionado/a por aprender programación.
2. Disfruto enfrentándome a desafíos relacionados con la programación.
3. Me siento motivado/a para dedicar tiempo adicional a practicar programación fuera de clase.
4. Creo que aprender programación me ayudará en mi futuro académico y profesional.

■ Confianza:

1. Creo que soy bastante bueno/a programando.
2. Creo que puedo resolver problemas básicos de programación por mi cuenta.
3. Si enfrento un desafío nuevo en programación, creo que puedo abordarlo con éxito.

3.4.3. Test Final

- **Método:** El test se aplicará durante la ayudantía previa al primer certamen, una vez que los andamiajes educativos hayan sido implementados y retirados. Su duración será de aproximadamente 10 minutos.
- **Propósito:** Evaluar la evolución de la motivación y confianza en los estudiantes, comparando aquellos que utilizaron andamiajes educativos con quienes no los emplearon.
- **Formato:** El test incluirá preguntas similares al inicial para permitir una comparación de resultados. Además, contendrá un apartado adicional sobre la percepción de los andamiajes educativos utilizados.

3.5. Ticket de salida al final de cada Ayudantía

Diseñados para medir el aprendizaje y los errores comunes de los estudiantes en sesiones específicas.

- **Duración:** 7-10 minutos.
- **Estructura:**
 1. **Evaluación de Conceptos Aprendidos:**
 - Programación secuencial.
 - Condicionales
 - Ciclos While
 - Ciclos For
 2. **Identificación de Errores Comunes:** Actividades orientadas a detectar patrones de error, como:
 - Problemas de indentación.
 - Uso incorrecto de asignación de variables.
 - Control de flujo.
 - Syntaxis.
 - Errores lógicos.
- **Propósito:** Los resultados permitirán comparar el desempeño de los estudiantes que emplearon herramientas de andamiaje educativo (como lenguajes en bloques o ejercicios tipo Parsons) con aquellos que no las utilizaron, analizando así su efectividad.

Los resultados de estos mini tests servirán para comparar el desempeño de los estudiantes que utilizaron herramientas de andamiaje educativo con aquellos que no las emplearon. Esto permitirá analizar la efectividad de dichas herramientas en el proceso de aprendizaje, contribuyendo directamente al cumplimiento de los objetivos planteados en esta memoria.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

4.1. Descripción del proceso

Con el objetivo de evaluar el impacto del uso de andamiajes educativos como lenguajes en bloques y ejercicios tipo Parsons en estudiantes novatos de un curso introductorio a la programación, se trabajó con dos paralelos del curso *Introducción a la Programación* de la Universidad Técnica Federico Santa María (UTFSM). El paralelo 200 fue el grupo experimental, en el cual se implementaron andamiajes educativos durante las ayudantías, mientras que el paralelo 202 actuó como grupo de control, sin el uso de estas herramientas.

En ambos paralelos, la mayoría de los estudiantes cursaban su primer año y provenían de carreras no relacionadas con la informática. Las ayudantías se llevaron a cabo de forma presencial, en el mismo horario para ambos grupos. Se utilizaron las mismas presentaciones en formato PowerPoint y se propusieron los mismos ejercicios, siendo la única diferencia la incorporación de andamiajes educativos en el paralelo 200.

Cada ayudantía tuvo una duración de 1 hora y 10 minutos, y normalmente se estructuró de la siguiente manera:

- 10–15 minutos: repaso teórico del contenido de la semana.
- 20–25 minutos: resolución de ejercicios utilizando el andamiaje correspondiente.
- Tiempo restante: ejercicios de programación donde los estudiantes debían escribir código.
- Últimos 10 minutos: aplicación de un ticket de salida para evaluar el aprendizaje.

4.1.1. Hardware utilizados en las ayudantías

Todas las ayudantías se llevaron a cabo en las instalaciones de la UTFSM, Sede San Joaquín. El paralelo 200 se realizó en la sala LAB-033, mientras que el paralelo 202 tuvo lugar en la sala LAB-037. Ambas salas cuentan con estaciones de trabajo equipadas con el sistema operativo Linux y con las mismas especificaciones técnicas, garantizando condiciones equivalentes para todos los estudiantes.

4.2. Verificación de homogeneidad entre grupos

Antes de realizar comparaciones sobre el rendimiento académico entre los paralelos 200 (experimental) y 202 (control), se evaluó si ambos grupos presentaban condiciones iniciales comparables, es decir, si eran estadísticamente homogéneos. Para ello, se aplicó un test de pensamiento computacional al inicio del semestre a ambos grupos. El test aplicado fue propuesto en el artículo “Validation of a Bebras-Based Test to Assess Computational Thinking Abilities in First-Year College Students”[Meza *et al.*, 2024], el cual fue diseñado para evaluar habilidades de pensamiento computacional en estudiantes de primer año universitario.

Los resultados obtenidos fueron analizados mediante una prueba t de Student para muestras independientes, que permitió comparar los puntajes entre ambos grupos. El paralelo 202 obtuvo un promedio de 72,10 con una desviación estándar de 16,59, mientras que el paralelo 200 alcanzó un promedio de 75,93 con una desviación estándar de 15,56. El valor p asociado a la prueba fue de 0,408, lo que indica que no existen diferencias estadísticamente significativas entre los grupos ($p = 0,408 > 0,05$).

Estos resultados permiten afirmar que los grupos eran comparables al inicio del estudio, por lo que cualquier diferencia posterior podría atribuirse con mayor confianza a la implementación de los andamiajes educativos.

4.3. Comparar el rendimiento académico

4.3.1. Tarea: Sistema de plantaciones automatizado

La tarea fue realizada durante la tercera semana desde el inicio de las ayudantías.

Esta consistió en desarrollar un programa para calcular los costos de plantaciones automatizadas, poniendo en práctica conceptos de programación secuencial y condicional. A continuación, se presenta un análisis comparativo del desempeño entre el grupo experimental (Paralelo 200, que contó con andamiajes educativos) y el grupo de control (Paralelo 202, que siguió el método tradicional), basado en las calificaciones obtenidas y las rúbricas utilizadas para la evaluación. Ambos grupos entregaron correctamente 30 tareas. Para asegurar la validez del análisis, se excluyeron del estudio aquellos estudiantes que no realizaron la entrega o presentaron casos de copia.

Métrica	Grupo Experimental (200)	Grupo Control (202)
Nota promedio	93.8	89.93
Estudiantes con nota 100	14 (47 %)	14 (47 %)
Estudiantes con nota ≥ 90	25 (83 %)	18 (60 %)
Estudiantes con nota ≤ 70	0 (0 %)	3 (10 %)
Desviación estándar	± 6.93	± 11.44
N	30	30

Tabla 1: Comparación de rendimiento de la tarea entre grupos.

Fuente: Elaboración propia.

Los resultados revelan que, si bien ambos grupos presentaron igual proporción de notas máximas (47 % en cada grupo), el uso de andamiajes educativos demostró un impacto diferenciado en el rendimiento general. El grupo experimental mostró una clara ventaja en la consistencia de los aprendizajes, con un 83 % de estudiantes alcanzando notas iguales o superiores a 90 puntos (frente al 60 % del grupo control) y ninguna calificación por debajo de 70 puntos (versus un 10 % en el grupo tradicional). Esta mejora se refleja también en la reducción de la desviación estándar (6.93 frente a 11.44), indicando que los andamiajes no solo elevan el rendimiento promedio (93.8 vs 89.93), sino que homogenizan los resultados, reduciendo la brecha entre estudiantes.

Los resultados también muestran que los andamiajes educativos ayudan principalmente a evitar notas bajas y a mejorar el rendimiento de los estudiantes promedio. Si bien no aumentan el número de notas máximas, sí logran que más alumnos alcancen un “buen nivel” (sobre 90 puntos) y que ninguno quede rezagado (notas bajo 70). Esto sugiere que funcionan como un apoyo que nivela hacia arriba el aprendizaje del estudiante.

Se evaluaron 7 categorías clave. A continuación, el porcentaje de estudiantes que alcanzaron la máxima puntuación en cada criterio:

Criterio	Exp. (%)	Control (%)	Misconception abordado
Gestión de variables	97	100	Confusión asignación o igualdad (= vs ==)
Detección de errores	70	70	Sintaxis (indentación, paréntesis)
Cálculo de área	87	67	Prioridad de operadores matemáticos
Condicionales	90	97	Estructuración lógica de if-else
Costo final	87	77	Casos límite en cálculos acumulativos
Claridad	97	90	Mensajes de salida ambiguos y orden en el código
Variables descriptivas	100	90	Nombres genéricos (x, y)

Tabla 2: Desempeño por criterios y relación con *misconceptions*

Fuente: Elaboración propia.

Los resultados obtenidos refuerzan la importancia de los andamiajes educativos como una estrategia efectiva para mejorar la comprensión y aplicación de conceptos en programación. Aunque ambos grupos demostraron habilidades similares en varios aspectos, el grupo experimental destacó en la reducción de errores relacionados con la prioridad de operadores matemáticos y el manejo de casos límite. Esto sugiere que los andamiajes educativos facilitaron una mejor estructuración del pensamiento lógico de los estudiantes.

Sin embargo, en ciertos criterios específicos, como la gestión de variables y la estructuración de condicionales, el grupo de control mostró un desempeño ligeramente superior. Este hallazgo podría indicar que algunos aspectos del aprendizaje no se vieron directamente influenciados por los andamiajes o que los estudiantes del grupo de control utilizaron estrategias alternativas eficaces para abordar estos desafíos.

A pesar de estos hallazgos, se realizó una prueba t de Student sobre las calificaciones obtenidas en la Tarea 1 para evaluar la significancia estadística de las diferencias entre los paralelos. Los resultados indicaron que la diferencia no es significativa ($p = 0.1199 \geq 0.05$). Esto sugiere que, aunque los andamiajes educativos parecen tener un impacto positivo en la distribución de las notas, el análisis estadístico no proporciona evidencia suficiente para afirmar que este efecto sea significativamente diferente de lo que podría esperarse por azar.

4.3.2. Ticket de salida:

Durante cada sesión de ayudantía, se aplicaron tickets de salida al finalizar la clase con el objetivo de evaluar la comprensión inmediata de los contenidos abordados. Estas breves

evaluaciones consistieron en preguntas o ejercicios de programación relacionados con el tema trabajado en la sesión. La mayoría de las preguntas de estos tickets se centraron en identificar y corregir errores comunes (*misconceptions*). Un ejemplo de cómo son los tickets de salida se presenta en el Anexo sección A 5.1.

Ticket de salida 1: Este ticket evaluó conceptos relacionados con errores comunes (*misconceptions*) y la programación básica secuencial en Python. Estaba compuesto por preguntas de selección múltiple, cada una con un valor de 20 puntos. No se otorgaban puntajes intermedios: las respuestas correctas recibían la puntuación completa, mientras que las incorrectas no sumaban puntos. Un total de 27 estudiantes del grupo de control y 29 del grupo experimental respondieron el ticket de salida.

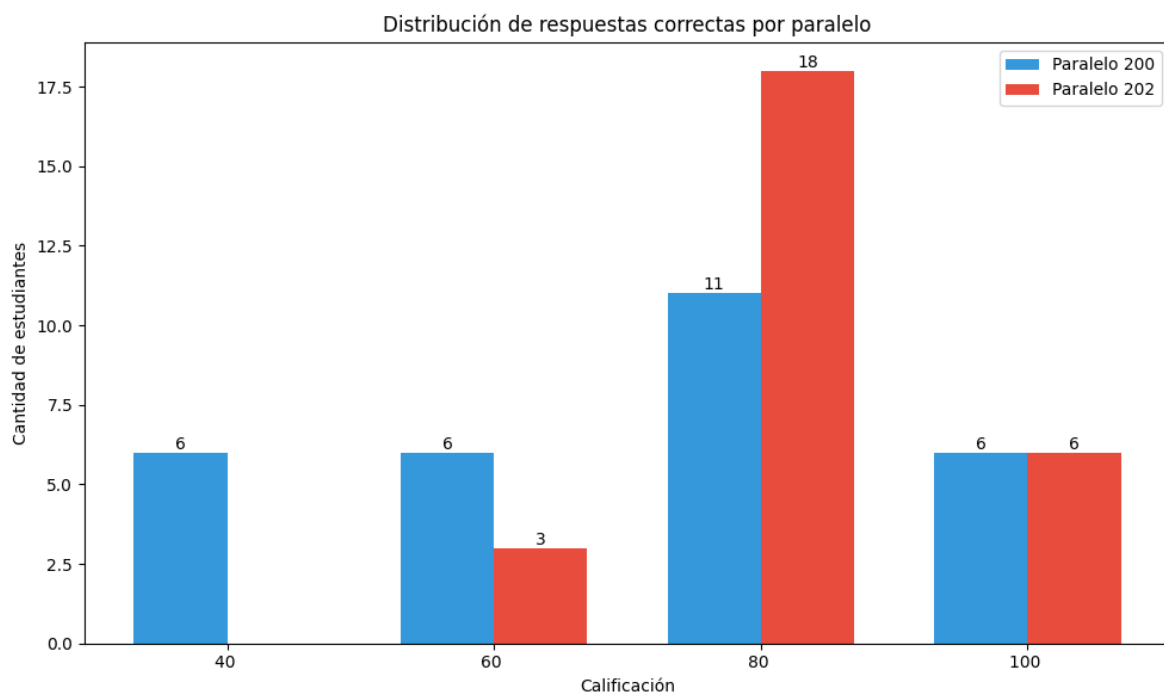


Figura 16: Distribución de notas por paralelo
Fuente: Elaboración propia.

Los resultados muestran que el grupo experimental, que utilizó el andamiaje basado en lenguaje en bloques, obtuvo un promedio de 71,72 puntos con una desviación estándar de 21,05. Por otro lado, el grupo de control alcanzó un promedio de 82,22 puntos, con una desviación estándar de 11,54. Según el análisis estadístico realizado mediante la t-test, la diferencia entre ambos grupos es estadísticamente significativa ($p = 0.02 \leq 0.05$), lo que sugiere que el desempeño de los grupos fue distinto bajo las condiciones evaluadas.

Ticket de salida 2: Este ticket evaluó conceptos relacionados con la programación secuencial. Estaba compuesto por 2 preguntas de selección múltiple y 2 preguntas de desarrollo, cada una con un valor de 25 puntos. Se otorgaban puntos intermedios dependiendo de las respuestas en las preguntas de desarrollo. Un total de 32 estudiantes del grupo de control y 31 del grupo experimental respondieron el ticket de salida.

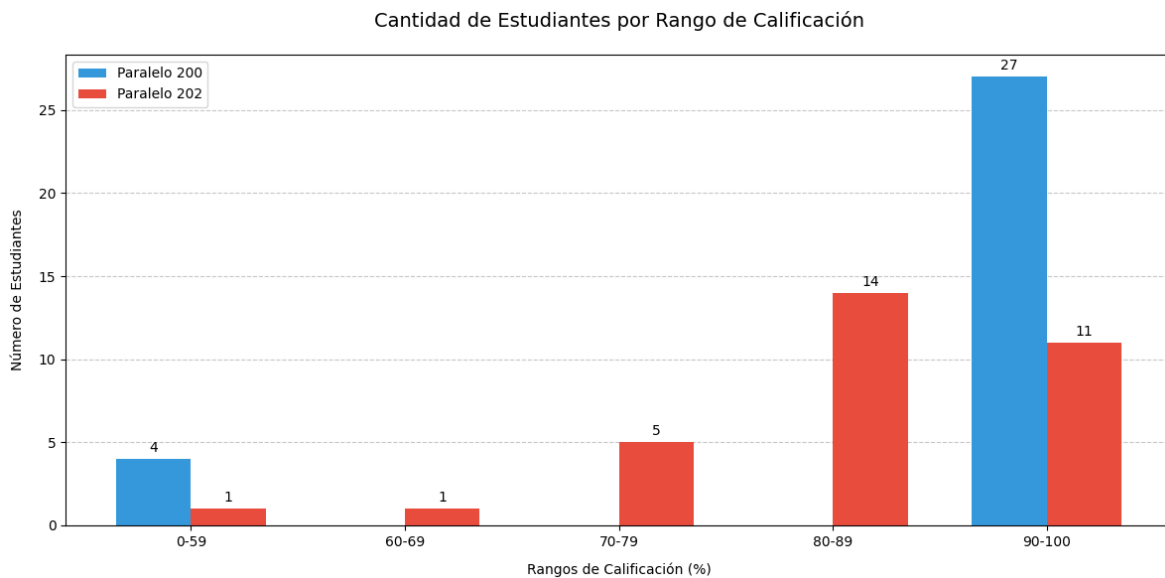


Figura 17: Distribución de rango de notas por paralelo
Fuente: Elaboración propia.

Los resultados del Ticket de Salida 2 revelan diferencias notables entre el grupo experimental y el grupo de control. El grupo experimental alcanzó un promedio superior (93.55 puntos) en comparación con el grupo de control (86.59 puntos), lo que sugiere que la metodología aplicada pudo tener un efecto positivo en el rendimiento general.

Sin embargo, se observa una mayor dispersión en las calificaciones del grupo experimental (desviación estándar = 16.76), indicando que los resultados fueron más variables: mientras un grupo significativo de estudiantes logró la máxima puntuación (100 puntos), otros obtuvieron calificaciones considerablemente más bajas (50 puntos). Por otro lado, el grupo de control mostró una distribución más homogénea (desviación estándar = 11.34), con la mayoría de los estudiantes concentrados en un rango intermedio-alto (70-100 puntos). Los resultados del t-test sugieren que la diferencia entre los resultados de ambos grupos es estadísticamente significativa ($p = 0.03 \leq 0.05$).

Ticket de salida 3: Este ticket evaluó conceptos relacionados específicamente con la programación condicional. Estaba compuesto por 1 pregunta de selección múltiple de 20 puntos y 2 preguntas de desarrollo, una de 30 puntos y otra de 50 en la cual se otorgaban puntos

intermedios dependiendo de las respuestas. Un total de 32 estudiantes del grupo de control y 29 del grupo experimental respondieron el ticket de salida.

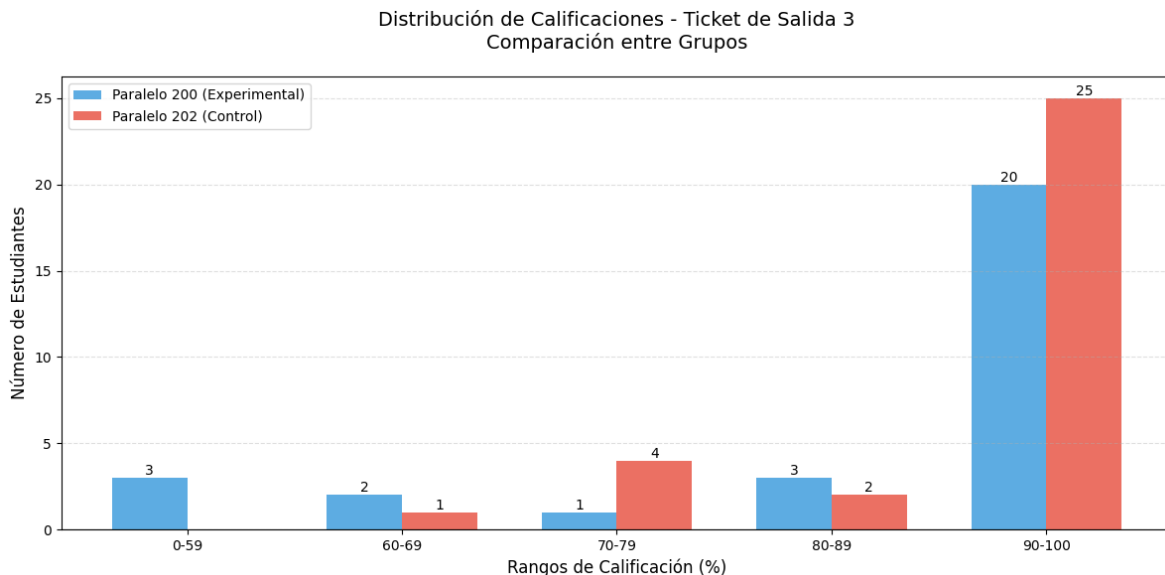


Figura 18: Distribución de rango de notas por paralelo
Fuente: Elaboración propia.

Los resultados obtenidos en el Ticket de Salida 3 evidencian una diferencia en el desempeño entre ambos grupos. El grupo de control obtuvo un promedio de 93,62 puntos con una desviación estándar de 11,25, mientras que el grupo experimental alcanzó un promedio de 86,38 puntos, con una desviación estándar de 18,84. Aunque ambos grupos demostraron un desempeño sólido en la temática evaluada, el grupo de control logró una mayor consistencia en sus resultados, reflejada tanto en su promedio más alto como en su menor dispersión. Aunque en este caso no hay evidencia de diferencia significativa ($p = 0.08 \geq 0.05$).

Ticket de salida 4: Este ticket evaluó conceptos relacionados específicamente con la programación cíclica While. Estaba compuesto por 3 preguntas de desarrollo, dos de 35 puntos y otra de 30 en la cual se otorgaban puntos intermedios dependiendo de la calidad de las respuestas. Un total de 33 estudiantes del grupo de control y 30 del grupo experimental respondieron el ticket de salida.

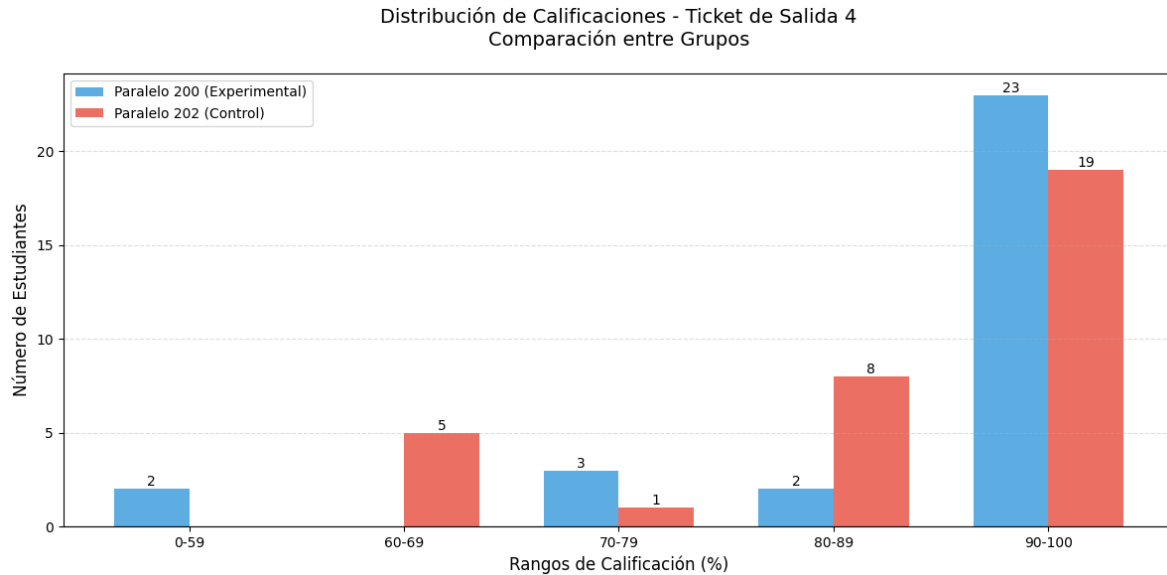


Figura 19: Distribución de rango de notas por paralelo
Fuente: Elaboración propia.

En cuanto al desempeño obtenido, el grupo experimental alcanzó un promedio de 91,33 puntos con una desviación estándar de 16,33, mientras que el grupo de control obtuvo un promedio ligeramente inferior de 88,48 puntos, con una desviación estándar de 12,46. No obstante, a pesar de la diferencia observada entre ambos grupos, el análisis estadístico muestra que esta diferencia no es estadísticamente significativa ($p = 0.45 \geq 0.05$).

Ticket de salida 5: Este ticket evaluó conceptos relacionados específicamente con la programación cíclica for. Estaba compuesto por 3 preguntas de desarrollo, dos de 35 puntos y otra de 30 en la cual se otorgaban puntos intermedios dependiendo de la calidad de las respuestas. Un total de 33 estudiantes del grupo de control y 32 del grupo experimental respondieron el ticket de salida.

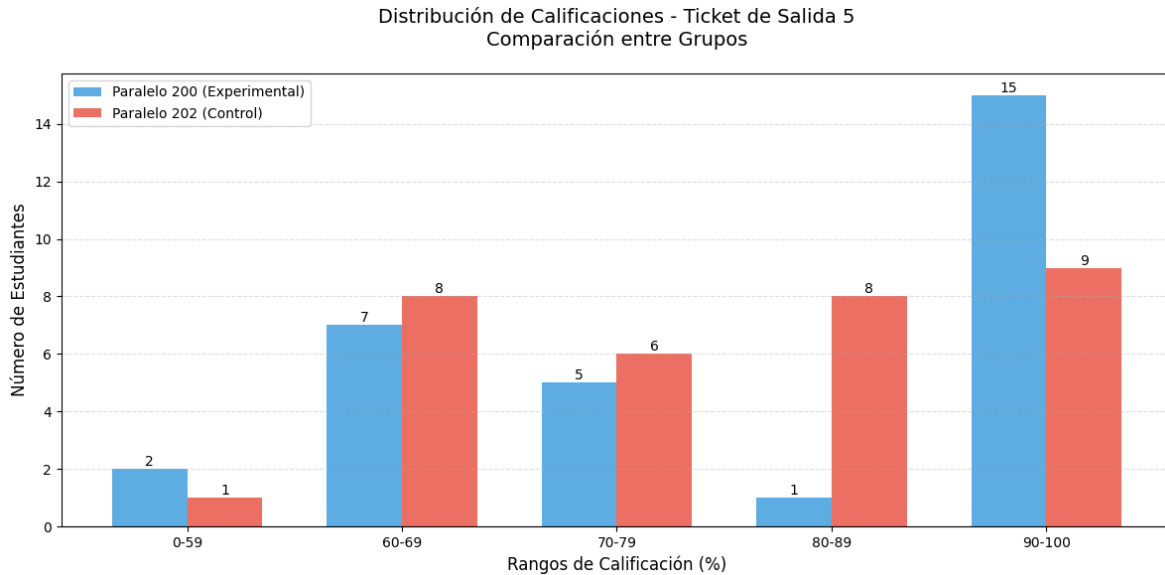


Figura 20: Distribución de rango de notas por paralelo
Fuente: Elaboración propia.

En cuanto al desempeño obtenido, el grupo experimental alcanzó un promedio de 80.33 puntos con una desviación estándar de 18,57, mientras que el grupo de control obtuvo un promedio ligeramente inferior de 77.18 puntos, con una desviación estándar de 12,74. El análisis estadístico indica que esta diferencia no es estadísticamente significativa ($p = 0.45 \geq 0.05$).

Desempeño general A continuación, se presentan los resultados de los tickets de salida aplicados durante semanas en las que se utilizó los andamiajes educativos. El grupo experimental (Paralelo 200) trabajó con ejercicios en lenguaje en bloques y ejercicios tipo Parsons, mientras que el grupo de control (Paralelo 202) utilizó el método tradicional.

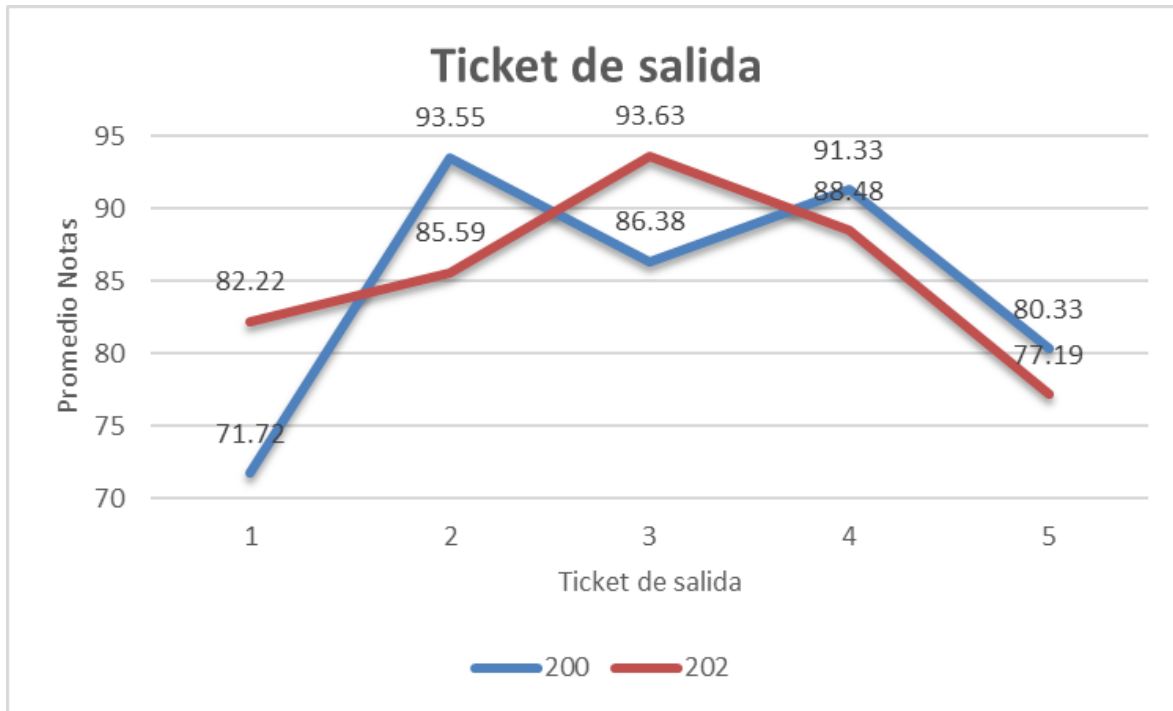


Figura 21: Promedio de notas obtenidas en tickets de salida por grupo.
Fuente: Elaboración propia.

Como se aprecia en la Figura 21, ambos grupos muestran una tendencia general al alza en su rendimiento, pero con diferencias notables en ciertos puntos del proceso. En el primer ticket, el grupo experimental parte con un promedio significativamente menor (71.72), lo que podría indicar una menor familiaridad con los contenidos. Sin embargo, tras la implementación de los andamiajes, se observa una mejora abrupta en el segundo ticket, alcanzando un promedio de 93.55, superando ampliamente al grupo de control (85.59).

En los tickets 3 y 4, los promedios del grupo experimental se mantienen altos y estables (86.38 y 91.33), acercándose o superando nuevamente al grupo de control, que se mantiene más constante a lo largo del tiempo. Esta estabilización sugiere que los andamiajes permitieron nivelar el rendimiento del grupo, favoreciendo especialmente a los estudiantes con mayor rezago inicial.

Los resultados obtenidos a través de las t-test revelan una evolución clara en la efectividad de los andamiajes educativos a lo largo del curso. Durante la fase inicial, correspondiente a los Tickets 1 y 2, se observaron diferencias estadísticamente significativas entre ambos grupos. En el primer ticket, el grupo de control mostró un mejor desempeño que el experimental, lo que podría indicar que los estudiantes necesitaron un período de adaptación a la nueva metodología. Sin embargo, esta situación se revirtió en el segundo ticket, donde fue el grupo experimental el que superó significativamente al de control.

En contraste, durante la fase avanzada del curso, representada por los Tickets 3, 4 y 5, las diferencias entre grupos dejaron de ser estadísticamente significativas ($p > 0.05$ en todos los casos). Esta convergencia progresiva en los resultados sugiere que, mientras los andamiajes fueron particularmente valiosos para acelerar el aprendizaje durante la fase intermedia, ambos métodos terminaron siendo igualmente efectivos cuando los estudiantes alcanzaron niveles más avanzados de competencia.

Estos resultados refuerzan la hipótesis de que los andamiajes educativos pueden ser particularmente efectivos para acelerar el aprendizaje de estudiantes con dificultades al inicio del curso, promoviendo una mejora sostenida en el tiempo. Además, permiten reducir la brecha de rendimiento dentro del grupo, elevando el promedio general sin aumentar significativamente la dispersión de los resultados, salvo en instancias finales donde la heterogeneidad individual tiende a manifestarse con mayor fuerza.

4.3.3. Certamen 1

El Certamen 1 se aplicó al finalizar la sexta semana de ayudantías, y tuvo como objetivo evaluar los contenidos abordados hasta ese momento: programación secuencial, estructuras condicionales, ciclos `while` y `for`, manipulación de cadenas de texto (`string`) y funciones. La evaluación fue individual, escrita y sin acceso a materiales de apoyo, lo que permitió medir el nivel de aprendizaje consolidado de los estudiantes en un contexto de alta exigencia académica. Además, incluyó una pregunta enfocada en el análisis del flujo de ejecución de un fragmento de código (ruteo), que buscaba evaluar la comprensión profunda de la lógica de programación. El certamen aplicado se encuentra disponible en el Anexo, en la sección C 5.3.

A continuación, se presenta una comparación del rendimiento entre el grupo experimental (Paralelo 200) y el grupo de control (Paralelo 202). Para asegurar que la comparación refleje de manera más precisa el impacto de los andamiajes, se excluyeron del análisis los estudiantes que faltaron a dos o más ayudantías en las que se aplicaron herramientas de andamiaje educativo (condición que corresponde a haber asistido a menos del 60 % de las sesiones con andamiajes, de un total de 5). Esta medida buscó garantizar que el grupo experimental estuviera compuesto por estudiantes que realmente participaron activamente en la implementación de estas herramientas pedagógicas. De forma análoga, se aplicó el mismo criterio en el grupo de control, excluyendo a los estudiantes que faltaron a dos o más ayudantías, con el objetivo de obtener un grupo de control comprometido y comparable. Esta depuración permite obtener resultados más sólidos y confiables en la evaluación del efecto de los andamiajes sobre el rendimiento académico.

Adicionalmente, se excluyeron del análisis los estudiantes que estaban cursando la asignatura por segunda vez, a fin de controlar el efecto de la experiencia previa en los resultados. Estas depuraciones permitieron obtener resultados más sólidos y confiables en la evaluación del efecto de los andamiajes sobre el rendimiento académico..

Métrica	Grupo Experimental (200)	Grupo Control (202)
Nota promedio	51.33	50.28
Estudiantes con nota 100	0	2
Estudiantes con nota ≥ 90	2	6
Estudiantes con nota ≥ 55	14	14
Desviación estándar	± 24.21	± 33.22
Mediana:	53.00	46.50
Máximo:	95	100
Mínimo:	7	4
Número de estudiantes	30	32

Tabla 3: Comparación de rendimiento del certamen entre grupos.

Fuente: Elaboración propia.

Como se observa en la Tabla 4.3.3, el promedio de notas entre ambos grupos fue relativamente similar (51.33 en el grupo experimental y 50.28 en el grupo de control), sin una diferencia sustancial. Ambos grupos tenían un número comparable de estudiantes (30 en el grupo experimental y 32 en el grupo de control), lo cual permite una comparación balanceada entre sus rendimientos.

Al analizar otras métricas, se aprecian diferencias relevantes. Por ejemplo, el grupo de control presentó un mayor número de estudiantes con notas sobresalientes (6 estudiantes con nota ≥ 90 , incluyendo 2 con nota 100), mientras que el grupo experimental no registró ningún caso de nota máxima y sólo 2 estudiantes superaron los 90 puntos. Sin embargo, cuando se considera el umbral de aprobación (nota ≥ 55), ambos grupos tienen exactamente 14 estudiantes aprobados, lo que indica que, en términos de logro mínimo, los andamiajes educativos no representaron una ventaja significativa en la cantidad de aprobados.

Es importante destacar que el grupo experimental presentó una desviación estándar considerablemente menor (24.21 vs 33.22), lo que sugiere una distribución de notas más homogénea y menos dispersa. Aunque el grupo de control alcanzó el puntaje máximo absoluto (100), también mostró una mayor dispersión, con notas tan bajas como 4 puntos, mientras que en el grupo experimental la nota más baja fue 7. Esta diferencia sugiere que el uso de andamiajes podría haber ayudado a reducir los casos extremos de bajo rendimiento, estabilizando el desempeño general del grupo. En este sentido, aunque los andamiajes no impulsaron a los estudiantes a obtener notas significativamente más altas en promedio, sí habrían cumplido un rol nivelador y de contención pedagógica, contribuyendo a una mayor equidad en el proceso de aprendizaje. Este patrón se visualiza claramente en la Figura 22, donde la distribución del grupo experimental muestra una mayor concentración en rangos medios, mientras que el grupo de control presenta una curva bimodal con mayor presencia tanto en notas bajas como en rendimientos sobresalientes.

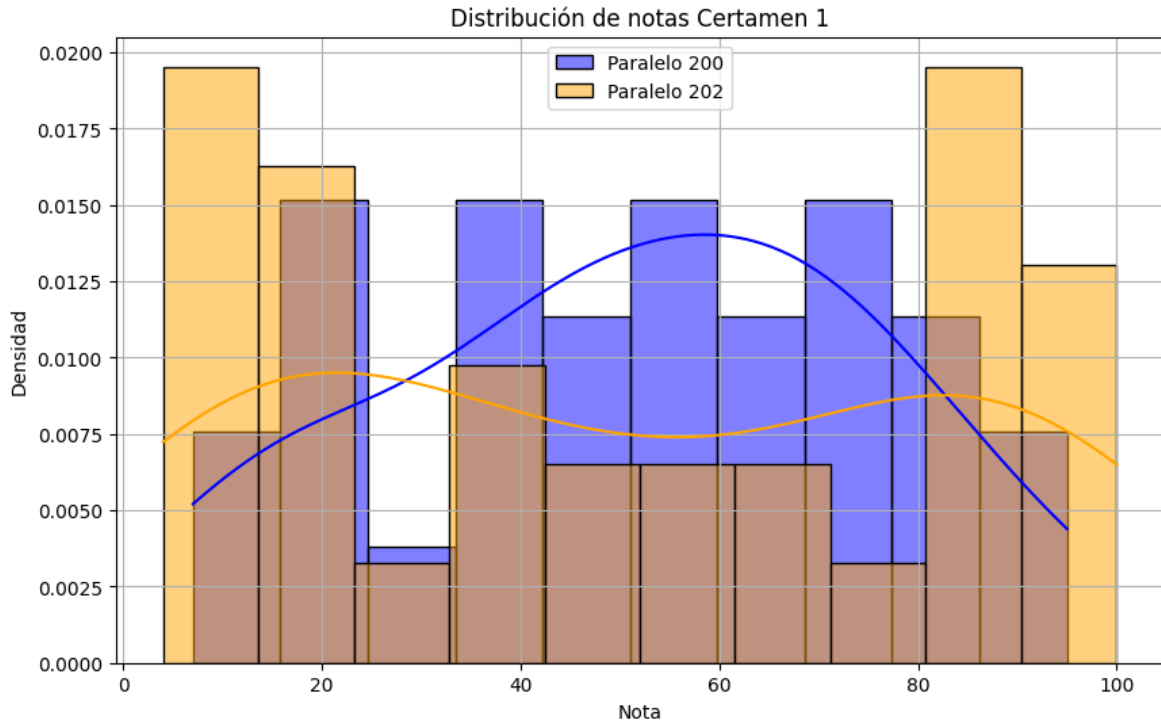


Figura 22: Distribución de notas Certamen 1
Fuente: Elaboración propia.

Como se observa en la figura 22, el grupo experimental presenta una distribución de notas que se aproxima mas a una curva normal, centrada en los rangos medios de rendimiento. En cambio, el grupo de control muestra una distribución más dispersa y bimodal, con una mayor concentración tanto en las notas más bajas como en las más altas.

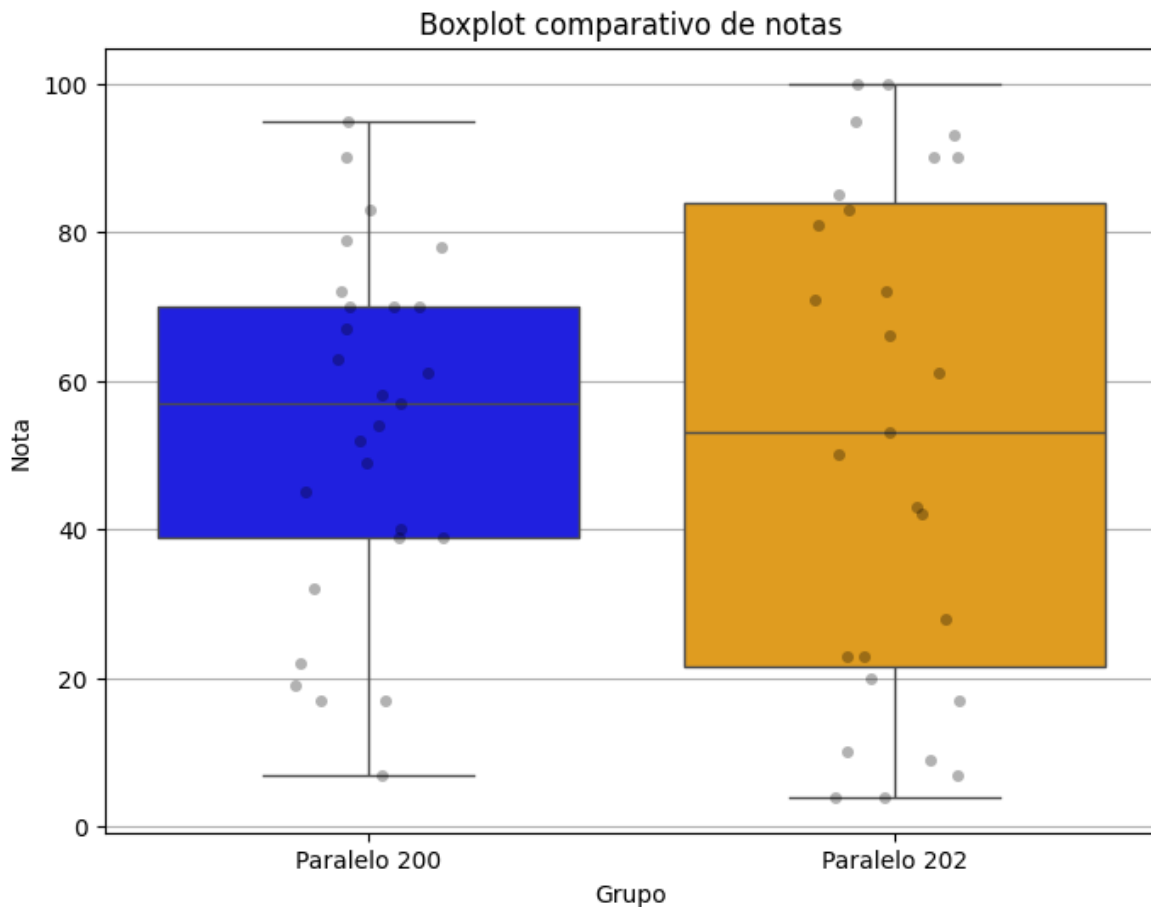


Figura 23: Boxplot comparativo de notas entre paralelos en el Certamen 1.

Fuente: Elaboración propia.

Como se muestra en la figura 23, el grupo experimental exhibe una distribución más concentrada en torno a la mediana, con menor presencia de valores extremos. En contraste, el grupo de control presenta una mayor dispersión de notas y varios valores atípicos, especialmente hacia los extremos inferiores y superiores. Este comportamiento refuerza la idea de que los andamiajes pedagógicos no necesariamente elevan los puntajes máximos, pero sí tienden a reducir la variabilidad y estabilizar el desempeño académico general.

4.4. Motivación y Autoeficacia: Análisis

En esta sección se analiza la evolución de la motivación y la autoeficacia de los estudiantes del paralelo 200 (grupo experimental) a lo largo del curso. Para ello, se aplicó un **test inicial** durante la primera semana, antes de la implementación de herramientas de andamiaje educativo, y un **test final** una semana antes del primer certamen, tras varias sesiones con actividades diseñadas bajo este enfoque pedagógico.

El test fue respondido por los mismos estudiantes en ambos momentos, lo que permite establecer comparaciones directas sobre su estado emocional y su percepción de competencia en programación. Además, el test final incluyó preguntas adicionales enfocadas en evaluar el cambio percibido por los estudiantes, así como el impacto subjetivo de los andamiajes en su motivación y confianza.

4.4.1. Evaluación Inicial de la Motivación

La motivación fue evaluada a través de las siguientes cuatro afirmaciones:

- 1. Estoy emocionado/a por aprender programación.**
- 2. Disfruto enfrentándome a desafíos relacionados con la programación.**
- 3. Me siento motivado/a para dedicar tiempo adicional a practicar programación fuera de clase.**
- 4. Creo que aprender programación me ayudará en mi futuro académico y profesional.**

4.4.2. Evaluación Inicial de la Autoeficacia

La autoeficacia se midió mediante las siguientes afirmaciones:

- 5. Creo que soy bastante bueno/a programando.**
- 6. Creo que puedo resolver problemas básicos de programación por mi cuenta.**
- 7. Si enfrento un desafío nuevo en programación, creo que puedo abordarlo con éxito.**

4.4.3. Preguntas adicionales del test final

Para complementar el análisis, el test final incluyó ítems diseñados para capturar el cambio percibido por los estudiantes y su apreciación respecto al impacto de los andamiajes educativos. Estas preguntas adicionales son las siguientes:

- Los andamiajes educativos (ejercicios interactivos, material adicional, etc.) me ayudaron a sentirme más motivado/a.
- Los andamiajes educativos aumentaron mi confianza al enfrentar ejercicios de programación.

4.4.4. Comparación gráfica: antes y después de los andamiajes

A continuación se presenta un gráfico de barras que compara las respuestas obtenidas en el test inicial y el test final de motivación y autoeficacia, aplicados al paralelo 200. Este gráfico permite visualizar cómo varió la percepción de los estudiantes a lo largo del curso, tras la implementación de herramientas de andamiaje educativo.

Para este análisis, las respuestas cualitativas fueron transformadas a una escala numérica con el objetivo de facilitar la comparación cuantitativa. La conversión fue realizada según la siguiente escala:

Respuesta	Puntaje asignado
Totalmente en desacuerdo	1
En desacuerdo	2
Neutral	3
De acuerdo	4
Totalmente de acuerdo	5

Tabla 4: Puntaje asignado por respuesta.

Fuente: Elaboración propia.

Cabe destacar que tanto en el test inicial como en el test final participaron 29 estudiantes, lo que asegura la comparabilidad directa entre ambas mediciones.

Esta escala permitió calcular promedios por pregunta, facilitando así la construcción del gráfico que se presenta a continuación.

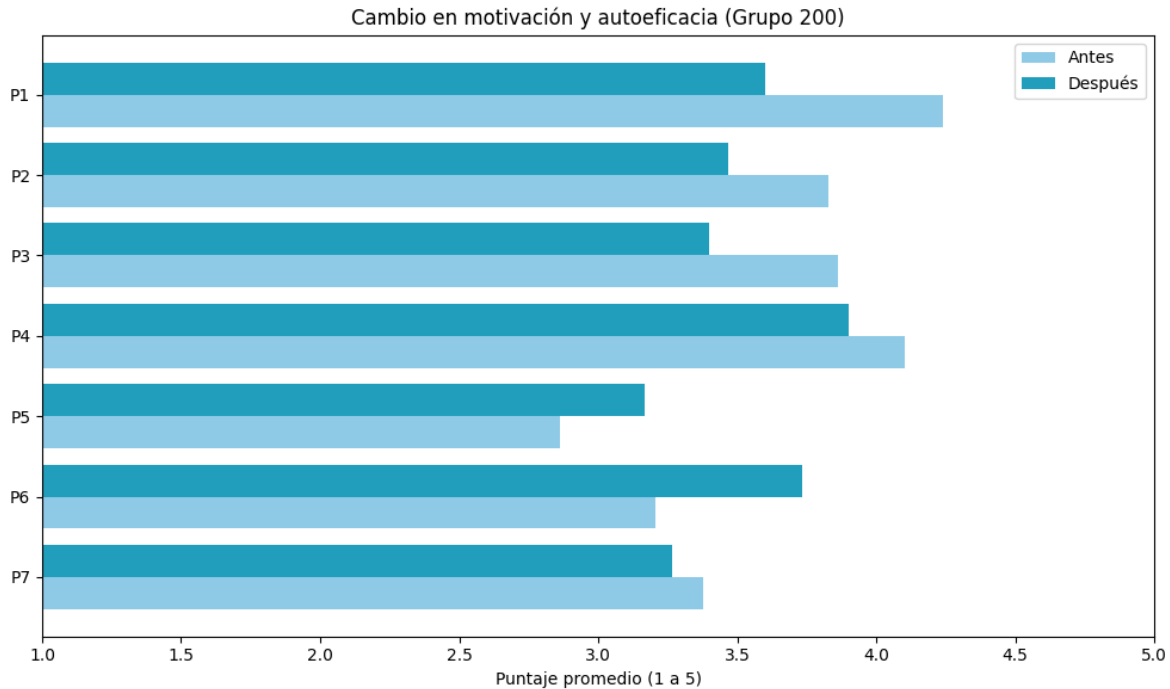


Figura 24: Gráfico comparativo encuesta motivación y autoeficacia
Fuente: Elaboración propia.

A continuación, se presenta una tabla que muestra los promedios obtenidos para cada afirmación evaluada antes y después de la implementación de los andamiajes educativos. Las preguntas 1 a 4 corresponden a la dimensión de **motivación**, mientras que las preguntas 5 a 7 evalúan la **autoeficacia** percibida por los estudiantes en relación a la programación. La tabla incluye también la diferencia absoluta entre ambas mediciones y el porcentaje de cambio relativo.

Pregunta	Antes	Después	Diferencia (+/-)	% Cambio
P1	4.24	3.60	-0.64	-15.1
P2	3.83	3.47	-0.36	-9.4
P3	3.86	3.40	-0.46	-12.0
P4	4.10	3.90	-0.20	-5.0
P5	2.86	3.17	0.30	10.6
P6	3.21	3.73	0.53	16.4
P7	3.38	3.27	-0.11	-3.3

Tabla 5: Comparación de resultados antes y después
Fuente: Elaboración propia.

Respecto a los resultados, se observa que las respuestas relacionadas con la **motivación** (P1 a P4) presentan una ligera disminución en sus promedios tras el curso, siendo P1 (*Estoy emo-*

cionado/a por aprender programación) la que muestra la mayor caída porcentual (-15.1 %). Esta tendencia podría reflejar un ajuste en las expectativas iniciales de los estudiantes, a medida que enfrentaron la dificultad real asociada al aprendizaje de programación.

Adicionalmente, esta disminución en la motivación también podría explicarse por factores externos al curso, como el avance del semestre académico. A medida que transcurren los meses, es común que la carga académica aumente y que los estudiantes deban afrontar evaluaciones exigentes como certámenes y controles, lo que puede impactar negativamente en su motivación general.

En contraste, en las afirmaciones asociadas a la **autoeficacia** (P5 a P7) se evidencian cambios mixtos: P5 y P6 aumentan de manera significativa (+10.6 % y +16.4 % respectivamente), lo que sugiere un fortalecimiento de la percepción de competencia en programación básica y resolución de problemas. Esto indica que, aunque la motivación general pudo disminuir levemente, los estudiantes se sintieron más capaces de enfrentar desafíos de programación al finalizar el curso.

Es importante mencionar que los resultados podrían estar influenciados por factores externos, como el progreso natural de los estudiantes a lo largo del curso o el efecto de otras actividades complementarias. Fue por esto mismo que el test final incluyó preguntas específicas sobre la percepción de los estudiantes respecto a cómo los andamiajes educativos afectaron su motivación y confianza en la programación 4.4.3. Los resultados de esta evaluación se presentan a continuación:

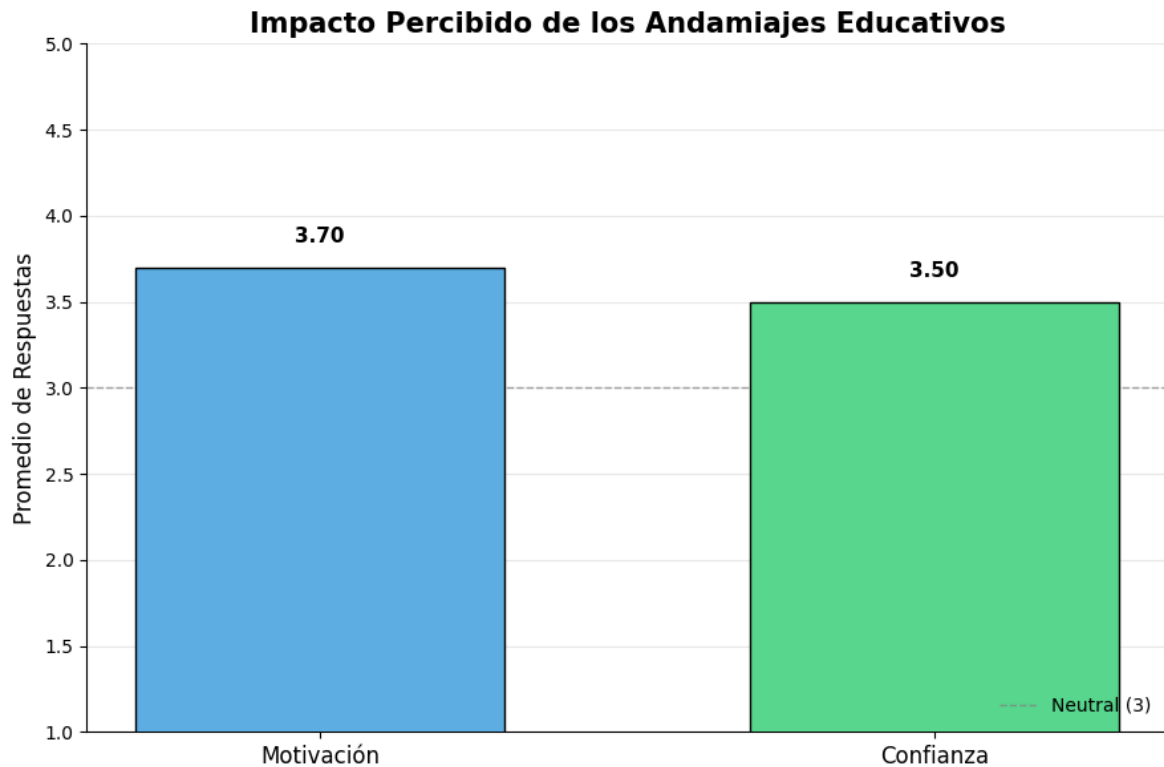


Figura 25: Percepción de los andamiajes respecto a la motivación y confianza.

Fuente: Elaboración propia.

Los resultados, presentados en la Figura 25, muestran que los estudiantes evaluaron positivamente el impacto de los andamiajes. En promedio, la motivación alcanzó una puntuación de 3.70, mientras que la confianza obtuvo un 3.50, ambos valores situándose por encima del nivel neutral (3 en la escala utilizada). Esto sugiere que, en general, los estudiantes percibieron que las herramientas de andamiaje contribuyeron favorablemente tanto a su entusiasmo por aprender programación como a su confianza para enfrentar nuevos desafíos en esta área.

4.5. Observación de las experiencias

Durante las sesiones de ayudantías del grupo experimental, se observaron distintos comportamientos, actitudes y comentarios por parte de los estudiantes que permiten identificar percepciones y efectos cualitativos del uso de andamiajes educativos. A continuación, se resumen las principales observaciones:

- Varios estudiantes se ofrecieron voluntariamente para resolver ejercicios tipo Parsons en el computador que se estaban proyectando las ayudantías, y en todos los casos

estos ejercicios fueron resueltos de forma correcta, lo que sugiere una comprensión activa y colaborativa del contenido.

- Algunos estudiantes manifestaron interés por realizar ejercicios tipo Parsons incluso sobre contenidos que aún no habían sido abordados formalmente en clase, lo que podría interpretarse como una señal de curiosidad y disposición para aprender de forma autónoma.
- En relación con la herramienta EduBlocks (usada para introducir la programación mediante bloques la primera semana de ayudantía), algunos estudiantes comentaron que les resultaba compleja la página y que preferían escribir código directamente.
- Fuera del día oficial de las ayudantías, se registró que estudiantes completaron al menos 54 niveles de ejercicios tipo Parsons en línea (al ingresar su nombre o ROL, los datos quedaban registrados), lo cual evidencia un interés de los estudiantes al uso del andamiaje.
- Si bien algunos estudiantes demostraron un interés genuino por la programación durante las primeras semanas, con el paso del tiempo también se observaron signos de frustración asociados a la carga del ramo, a las dificultades propias de la vida universitaria y al cansancio acumulado por evaluaciones de otras asignaturas, las cuales se realizaban en el bloque inmediatamente anterior a las ayudantías. Estos factores pudieron influir negativamente en su motivación y desempeño.

CAPÍTULO 5

CONCLUSIONES

El presente trabajo tuvo como propósito evaluar el efecto del uso de herramientas de andamiaje educativos específicamente, lenguajes de bloques y ejercicios tipo Parsons en el aprendizaje de estudiantes novatos en un curso de “Introducción a la Programación” impartido por la Universidad Técnica Federico Santa María. Para ello, se diseñó una metodología experimental aplicada a un grupo de alumnos (Paralelo 200) que utilizó estas herramientas en las ayudantías, y se comparó con otro grupo control (Paralelo 202) que siguió una metodología tradicional sin andamiajes.

Durante el desarrollo del estudio se aplicaron actividades diferenciadas en cinco sesiones de ayudantías, además de múltiples instrumentos de evaluación que permitían comparar a los grupos.

5.1. Gestión y desarrollo de las ayudantías

Uno de los aspectos más relevantes durante la implementación de las ayudantías fue el desarrollo y puesta en marcha de la plataforma web de ejercicios tipo Parsons. Desde un inicio, se concibió esta herramienta no solo como un recurso complementario para las sesiones semanales, sino como un apoyo constante que los estudiantes pudieran utilizar en cualquier momento. Por ello, se optó por alojar la página web en una instancia de Google Cloud Platform, asegurando así su disponibilidad continua sin interrupciones.

Durante el desarrollo de la plataforma (que tomó aproximadamente dos meses) se realizó una búsqueda de recursos y herramientas existentes, se implementó el sistema utilizando React en el frontend, Express en el backend, y MongoDB como base de datos. Además, se recibió retroalimentación de distintos profesores del ramo, lo cual permitió realizar mejoras tanto en los ejercicios como en el diseño de la interfaz.

En las ayudantías se abordaban los contenidos correspondientes a cada semana, y posteriormente se destinaba un tiempo para que los estudiantes trabajaran con el andamiaje educativo. Además, se les motivaba constantemente a utilizar la plataforma fuera del horario de clases como una herramienta de práctica y refuerzo. A quienes manifestaban dudas a través de correo o Discord, especialmente sobre condicionales o estructuras cíclicas (while, for), se les recomendaba primero resolver ejercicios en la plataforma antes de comenzar a programar directamente, ya que esto les permitía adquirir mayor claridad y confianza en los conceptos.

Un dato destacable fue que el uso de la plataforma aumentó significativamente el día previo al certamen, registrándose el mayor número de ejercicios resueltos fuera del horario de ayu-

dantía. Este comportamiento sugiere un alto nivel de aceptación de la herramienta por parte de los estudiantes, así como una percepción positiva respecto a su utilidad para el estudio del certamen.

5.2. Resultados principales y cumplimiento de objetivos

Los hallazgos cuantitativos obtenidos a lo largo de este estudio proporcionan una evidencia del efecto positivo de los andamiajes educativos en el aprendizaje de la programación. Estos resultados nos ayudan al cumplimiento de los objetivos específicos planteados en esta memoria.

En primer lugar, respecto al rendimiento académico, los tickets de salida revelaron una evolución significativa. Si bien en el Ticket de Salida 1 el grupo de control mostró un desempeño superior (promedio de 82.22 vs. 71.72 del experimental, con una diferencia estadísticamente significativa $p = 0.02$), esta situación se revirtió drásticamente en el Ticket de Salida 2. Aquí, el grupo experimental no solo superó al de control (93.55 vs. 86.59), sino que la diferencia también fue estadísticamente significativa ($p = 0.03$). Esta dinámica sugiere un período inicial de adaptación a la nueva metodología en el grupo experimental, seguido de una aceleración en el aprendizaje que les permitió alcanzar y superar al grupo de control. Los tickets 3, 4 y 5, aunque no mostraron diferencias estadísticamente significativas, evidenciaron una convergencia en el rendimiento de ambos grupos, con el grupo experimental manteniendo promedios altos y estables (86.38, 91.33 y 80.33 respectivamente). Esto indica que los andamiajes fueron particularmente valiosos en las etapas intermedias del curso, nivelando el rendimiento y promoviendo una mejora sostenida, especialmente para aquellos estudiantes que podían haber iniciado con un rezago mayor.

En cuanto al Certamen 1, los promedios de ambos grupos fueron notablemente similares (51.33 para el experimental y 50.28 para el control), lo que podría sugerir una ausencia de impacto. Sin embargo, un análisis más profundo de la distribución de las notas revela una diferencia cualitativa crucial. El grupo experimental mostró una desviación estándar considerablemente menor (24.21 vs. 33.22), lo que se traduce en una distribución de notas más homogénea y menos dispersa. Esto implica que, aunque los andamiajes no impulsaron a los estudiantes a obtener notas sobresalientes en mayor medida, sí actuaron como un factor nivelador y de contención pedagógica. Al reducir la dispersión y minimizar los casos de rendimiento bajo, los andamiajes contribuyeron a una mayor equidad en el proceso de aprendizaje, asegurando que un mayor número de estudiantes alcanzara un nivel de comprensión aceptable. Esto es fundamental en cursos introductorios donde el objetivo principal es sentar una base sólida para todos los estudiantes.

La tarea evaluativa sobre programación secuencial y condicional reforzó estos hallazgos. El grupo experimental obtuvo un promedio superior (93.8 vs. 89.93 del grupo control), una mayor proporción de estudiantes con notas ≥ 90 (83 % vs. 60 %), y una menor incidencia de calificaciones por debajo de 70 (0 % vs. 10 %). Además, el análisis por criterios mostró que

el grupo experimental destacó en la reducción de errores relacionados con la prioridad de operadores matemáticos y el manejo de casos límite, sugiriendo que los andamiajes facilitaron una mejor estructuración del pensamiento lógico. Aunque la prueba t de Student indicó que la diferencia no fue estadísticamente significativa, la tendencia observada es cualitativamente positiva, apuntando a que los andamiajes ayudan a homogeneizar el rendimiento y a prevenir el rezago.

Finalmente, las encuestas de motivación y autoeficacia aplicadas al grupo experimental evidenciaron un impacto leve pero positivo. Si bien las preguntas relacionadas con la motivación general mostraron una ligera disminución en los promedios tras el curso, lo cual puede atribuirse a factores externos como la carga académica general del semestre, las afirmaciones asociadas a la autoeficacia aumentaron significativamente (+10.6 % y +16.4 % respectivamente). Este es un hallazgo crucial: a pesar de las fluctuaciones en la motivación general, los estudiantes se sintieron más capaces y competentes para enfrentar desafíos de programación y resolver problemas básicos. Además, las preguntas adicionales del test final revelaron que los estudiantes percibieron positivamente el impacto de los andamiajes en su motivación (promedio de 3.70) y confianza (promedio de 3.50), ambos por encima del nivel neutral. Esto confirma que las herramientas de andamiaje, ayudaron a fortalecer la autoeficacia, un factor psicológico fundamental para la persistencia en el aprendizaje de la programación.

En síntesis, los resultados permiten afirmar que se cumplieron los objetivos específicos propuestos: se logró comparar el rendimiento académico de los grupos, medir la influencia de los andamiajes en la motivación y confianza de los estudiantes, y analizar la frecuencia de errores comunes, demostrando que estas herramientas son una adición valiosa a la enseñanza introductoria de la programación.

5.3. Alcances y limitaciones

Si bien los resultados obtenidos en esta investigación son prometedores y ofrecen valiosas perspectivas sobre el impacto de los andamiajes educativos, es fundamental reconocer las limitaciones inherentes al diseño y la ejecución del estudio. En primer lugar, el número de sesiones con andamiajes fue reducido (cinco ayudantías para cuatro temas distintos), lo que acota la posibilidad de generar un estudio a mayor profundidad. Asimismo, la comparación entre grupos pudo haber estado influenciada por variables externas no controladas, como diferencias de dedicación personal, estilos de aprendizaje o experiencia previa.

Además, una limitación importante es que el estudio se realizó con un único paralelo (Paralelo 200) como grupo experimental, lo que restringe la generalizabilidad de los resultados a una población estudiantil más amplia. Aunque se tomaron medidas para asegurar la comparabilidad entre el grupo experimental y el grupo de control, la replicación del estudio en múltiples paralelos o instituciones sería necesaria para confirmar la validez externa de los hallazgos. Esto permitiría determinar si los efectos observados son específicos de este grupo o si pueden extrapolarse a otros contextos y perfiles de estudiantes.

Finalmente, aunque se realizó un esfuerzo por cuantificar el impacto en la motivación y la autoeficacia, la naturaleza subjetiva de estos aspectos psicológicos representa un desafío constante en su medición. Las encuestas aplicadas reflejan la percepción de los estudiantes en un momento determinado, por lo que sus respuestas podrían no expresar de forma completamente precisa su experiencia o evolución a lo largo del tiempo.

5.4. Impacto observado en los estudiantes

Desde el punto de vista cualitativo, se observaron señales de mayor involucramiento y disposición hacia el aprendizaje en el grupo experimental. Esto se reflejó en comportamientos como:

- Participación activa en la resolución de ejercicios tipo Parsons.
- Interés por abordar ejercicios incluso antes de haber visto los contenidos formalmente.
- Uso voluntario de la plataforma fuera del horario de clase.
- Comentarios positivos sobre la utilidad de los ejercicios y observaciones críticas, lo que evidencia una reflexión activa sobre las herramientas pedagógicas empleadas.

Estas observaciones complementan los hallazgos cuantitativos y refuerzan la idea de que los andamiajes educativos no solo impactan en el rendimiento, sino también en la experiencia de aprendizaje y percepción del estudiante.

ANEXOS

En esta sección se incluye material complementario utilizado durante el desarrollo del curso, específicamente:

- Pruebas y evaluaciones realizadas en las sesiones de ayudantía
- Ejercicios de programación tipo Parsons disponibles en la plataforma virtual
- Certamen 1

5.1. A. Tests de salida

Para cada ticket de salida se diseñaron cinco modelos distintos, con preguntas similares pero no idénticas entre sí. Por motivos institucionales, solo se presenta un ejemplo representativo, correspondiente al Ticket de Salida 1, el cual abordaba contenidos de Programación Secuencial y nociones básicas de programación en Python.

5.1.1. Ejemplo ticket de salida 1:

Pregunta 1: ¿Cuál es el resultado de `7 // 2` en Python?

- a) 3.5
- b) 3
- c) 4.5
- d) 2

Pregunta 2: ¿Qué imprime el siguiente código?

```
x = 7
y = 2
x = int(5 + x / y)
print("El resultado es:", x)
```

Pregunta 3: Explica con tus propias palabras la diferencia entre el operador /, // y % en Python.

Pregunta 4: Si $x=3.99$, ¿qué resultado devuelve `int(x)`?

- a) 3.99
- b) 4
- c) 3
- d) Error

5.2. B. Ejercicios tipo Parsons

Esta subsección contiene problemas de reorganización de código (Parsons problems) en su versión básica, los cuales consisten en ordenar fragmentos de código sin distractores.

5.2.1. Ejercicios sobre estructuras condicionales

1. Escribe un programa que solicite un número al usuario e imprima si el número es **par** o **impar**.

```
numero = int(input("Ingresa un número: "))
if numero % 2 == 0:
    print("El número es par.")
else:
    print("El número es impar.")
```

2. Escribe un programa que pida la temperatura en grados Celsius y clasifique el clima en **Frío** (menos de 10°C), **Templado** (de 10°C a 25°C) o **Calor** (más de 25°C).

```
temperatura = int(input("Ingresa la temperatura"))
if temperatura < 10:
    print("Frío")
elif 10 <= temperatura <= 25:
    print("Templado")
else:
    print("Calor")
```

3. Crea un programa que pida la edad de una persona e imprima si es **mayor de edad** o **menor de edad**.

```
edad = int(input("Ingresa tu edad: "))
if edad >= 18:
    print("Eres mayor de edad.")
else:
    print("Eres menor de edad.")
```

4. Diseña un programa que solicite dos números y muestre cuál de ellos es **mayor**.

```
num1 = int(input("Ingresa el primer número: "))
num2 = int(input("Ingresa el segundo número: "))
if num1 > num2:
    print("El primer número es mayor.")
elif num1 < num2:
    print("El segundo número es mayor.")
else:
    print("Ambos números son iguales.")
```

5. Crea un programa que solicite un número y verifique si es **positivo**, **negativo** o **cero**.

```
numero = int(input("Ingresa un número: "))
if numero > 0:
    print("El número es positivo.")
elif numero < 0:
    print("El número es negativo.")
else:
    print("El número es cero.")
```

6. Escribe un programa que determine cuál de tres números ingresados es el **mayor**.

```
num1 = int(input("Ingresa el primer número: "))
num2 = int(input("Ingresa el segundo número: "))
num3 = int(input("Ingresa el tercer número: "))
if num1 >= num2 and num1 >= num3:
    print("El número mayor es: " + num1)
elif num2 >= num1 and num2 >= num3:
    print("El número mayor es: " + num2)
else:
    print("El número mayor es: " + num3)
```

7. Diseña un programa que pida al usuario un número del 1 al 7 y determine si corresponde a un día de la semana laboral (**lunes a viernes**) o fin de semana (**sábado o domingo**). Si el número no está en el rango, imprime **"Número fuera de rango"**.

```
numero = int(input("Ingresa un número del 1 al 7: "))
if numero >= 1 and numero <= 7:
    if numero <= 5:
        print("Es un día laboral.")
    else:
        print("Es un fin de semana.")
else:
    print("Número fuera de rango.")
```

8. Escribe un programa que pida al usuario las calificaciones de un estudiante en tres exámenes y calcule el promedio. Luego, el programa debe determinar si el estudiante tiene una **Nota excelente** (promedio de 9 o más), **Aprobado** (promedio entre 6 y 8 inclusive) o **Reprobado** (promedio menor a 6).

```
nota1 = int(input("Ingresa la nota del primer examen: "))
nota2 = int(input("Ingresa la nota del segundo examen: "))
nota3 = int(input("Ingresa la nota del tercer examen: "))
promedio = (nota1 + nota2 + nota3) / 3
if promedio >= 9:
    print("Nota excelente")
elif 6 <= promedio < 9:
    print("Aprobado")
else:
    print("Reprobado")
```

9. Escribe un programa que determine si un triángulo es **equilátero**, **isósceles** o **escaleno** según las longitudes de sus lados.

```
a = float(input("Ingresa la longitud del primer lado: "))
b = float(input("Ingresa la longitud del segundo lado: "))
c = float(input("Ingresa la longitud del tercer lado: "))
if a+b > c and a+c > b and b+c > 1:
    if a == b == c:
        print("El triángulo es equilátero.")
    elif a == b or a == c or b == c:
        print("El triángulo es isósceles.")
    else:
        print("El triángulo es escaleno.")
```

```
else:  
    print("Las longitudes ingresadas no forman un triángulo válido.")
```

10. Escribe un programa que actúe como una calculadora, permitiendo al usuario elegir entre **suma**, **resta**, **multiplicación** o **división**. Asegúrate de manejar el error de división por cero correctamente. Los condicionales deben ir en el orden de +, -, * y /.

```
operacion = input("Elige una operación (+, -, *, /): ")  
numero1 = float(input("Ingresa el primer número: "))  
numero2 = float(input("Ingresa el segundo número: "))  
if operacion == "+":  
    print("Resultado:", numero1 + numero2)  
elif operacion == "-":  
    print("Resultado:", numero1 - numero2)  
elif operacion == "*":  
    print("Resultado:", numero1 * numero2)  
elif operacion == "/":  
    if numero2 == 0:  
        print("Error: No se puede dividir por cero.")  
    else:  
        print("Resultado:", numero1 / numero2)  
else:  
    print("Operación no válida.")
```

11. Escribe un programa que solicite el año actual y un año de referencia. Luego, compara ambos años para determinar si han **pasado** o **faltan años** para llegar al año ingresado. Si el año ingresado es el mismo que el actual, informa que ambos son **iguales**.

```
anio_actual = int(input("Ingresa año actual: "))  
anio = int(input("Ingresa un año: "))  
if anio < anio_actual:  
    dif = anio_actual - anio  
    print("Han pasado " + dif + " años desde el año " + anio)  
elif anio > anio_actual:  
    dif = anio - anio_actual  
    print("Faltan " + dif + " años para el año " + anio3)  
else:  
    print("El año ingresado es el año actual: " + anio_actual)
```

12. Escribe un programa que pida al usuario cuánto dinero tiene y cuánto cuesta un producto. Luego, el programa debe determinar si el dinero es **suficiente** para comprar el producto, si **falta** dinero o si tiene dinero de **sobra**.

```
dinero = float(input("¿Cuánto dinero tienes? "))
precio_produc = float(input("¿Cuánto cuesta el producto? "))
if dinero > precio_produc:
    sobra = dinero - precio_produc
    print("Tienes dinero y te sobra:" + sobra)
elif dinero == precio_produc:
    print("Tienes justo el dinero necesario.")
else:
    falta = precio_produc - dinero
    print("No tienes suficiente dinero. Te falta: "+falta)
```

13. Escribe un programa que pida un número al usuario y lo clasifique según su valor y su proximidad al cero. Si el número es negativo y tiene una distancia mayor a 100 unidades del cero, se clasificará como **Lejos negativo**. Si es negativo y está a 100 unidades o menos del cero, será **Cerca negativo**. Si es positivo y tiene más de 50 unidades del cero, será **Lejos positivo**, y si está a 50 unidades o menos, será **Cerca positivo**. Si el número es exactamente cero, se clasificará como Cero

```
numero = float(input("Ingresa un número: "))
if numero == 0:
    print("Cero")
elif numero < 0:
    if abs(numero) > 100:
        print("Lejos negativo")
    else:
        print("Cerca negativo")
else:
    if numero > 50:
        print("Lejos positivo")
    else:
        print("Cerca positivo")
```

14. Escribe un programa que pida el salario de una persona y determine en qué rango se encuentra. Los rangos son los siguientes: si el salario es mayor a 0 y menor o igual a 5000, la persona está en el **Rango 1**; si el salario es mayor a 5000 y menor o igual a 10000, la persona está en el **Rango 2**; si el salario es mayor a 10000 y menor o igual a 20000, la persona está en el **Rango 3**; si el salario es mayor a 20000, la persona está en el **Rango 4**; y si el salario es menor o igual a 0, se debe indicar que el salario no es válido.

```
salario = float(input("Ingresa tu salario: "))
if salario <= 0:
```

```
        print("El salario no es válido")
    elif salario <= 5000:
        print("Rango 1")
    elif salario <= 10000:
        print("Rango 2")
    elif salario <= 20000:
        print("Rango 3")
    else:
        print("Rango 4")
```

15. Escribe un programa que determine si una figura geométrica es un **cuadrado**, un **rectángulo** o **ninguna** de estas, basándose en la longitud de sus lados. Si se ingresan cuatro valores para los lados de la figura.

```
lado1 = float(input("Ingresa la longitud del primer lado: "))
lado2 = float(input("Ingresa la longitud del segundo lado: "))
lado3 = float(input("Ingresa la longitud del tercer lado: "))
lado4 = float(input("Ingresa la longitud del cuarto lado: "))
if lado1 == lado2 == lado3 == lado4:
    print("Es un cuadrado.")
elif lado1 == lado3 and lado2 == lado4:
    print("Es un rectángulo.")
else:
    print("No es una figura válida.")
```

5.2.2. Ejercicios sobre ciclos While

1. Escribe un programa que imprima los números del **1 al 10** utilizando un bucle while.

```
i = 1
while i <= 10:
    print(i)
    i += 1
```

2. Crea un programa que solicite un número al usuario y calcule la suma de todos los números **desde 1 hasta el número ingresado** utilizando un bucle while y imprima la suma final.

```
n = int(input("Ingresa un número: "))
suma = 0
```

```
i = 1
while i <= n:
    suma += i
    i += 1
print("La suma es:", suma)
```

3. Diseña un programa que le pida al usuario que ingrese un número. El programa debe continuar solicitando números al usuario hasta que este ingrese un **número negativo o cero**. Después de cada número positivo ingresado, el programa debe mostrar el mensaje **Número positivo**. Si el número ingresado es negativo o cero, el programa debe mostrar el mensaje **Número negativo o cero** y finalizar.

```
while numero >= 0:
    numero = int(input("Ingresa un número: "))
    if numero > 0:
        print("Número positivo")
    else:
        print("Número negativo o cero")
```

4. Escribe un programa que calcule el factorial de un número ingresado por el usuario utilizando un bucle `while`. El programa debe solicitar al usuario un número entero, luego calcular su **factorial** multiplicando sucesivamente los números desde 1 hasta el número ingresado. Al final, debe mostrar el resultado del factorial.

```
n = int(input("Ingresa un número: "))
factorial = 1
i = 1
if i < 0:
    print("No esta definido")
else:
    while i <= n:
        factorial *= i
        i += 1
    print("El factorial es:", factorial)
```

5. Implementa un programa que simule un **sistema de contraseña**. El programa debe solicitar la contraseña hasta que el usuario ingrese la correcta.

```
contraseña_correcta = "segura123"
contraseña = ""
while contraseña != contraseña_correcta:
    contraseña = input("Ingresa la contraseña: ")
print("Acceso concedido.")
```

6. Crea un programa que imprima los números **pares entre 1 y 20** utilizando un bucle `while`.

```
i = 2
while i <= 20:
    print(i)
    i += 2
```

7. Diseña un programa que simule un menú interactivo donde el usuario pueda elegir entre **varias opciones**. El programa debe continuar mostrando el menú hasta que el usuario seleccione la opción **Salir**.

```
opcion = ""
while opcion != "Salir":
    print("Menú:")
    print("1. Opción 1")
    print("2. Opción 2")
    print("3. Salir")
    opcion = input("Selecciona una opción: ")
```

8. Escribe un programa que solicite números al usuario y calcule el **promedio de los números ingresados**. El programa debe finalizar cuando se ingrese un número igual a cero.

```
suma = 0
contador = 0
numero = None
while numero != 0:
    numero = int(input("Ingresa un número (0 para salir): "))
    if numero != 0:
        suma += numero
        contador += 1
promedio = suma / contador if contador > 0 else 0
print("El promedio es:", promedio)
```

9. Implementa un programa que cuente cuántos **dígitos** tiene un número ingresado por el usuario utilizando un bucle `while`.

```
numero = int(input("Ingresa un número: "))
contador = 0
while numero > 0:
    numero //= 10
    contador += 1
print("El número tiene", contador, "dígitos.")
```

10. Crea un programa que permita al usuario adivinar un número aleatorio entre 1 y 100. El programa debe continuar hasta que el usuario adivine correctamente.

```
import random
numero_secreto = random.randint(1, 100)
adivinado = False
while not adivinado:
    intento = int(input("Adivina el número: "))
    if intento == numero_secreto:
        print("¡Correcto!")
        adivinado = True
    elif intento < numero_secreto:
        print("Demasiado bajo.")
    else:
        print("Demasiado alto.")
```

5.2.3. Ejercicios sobre ciclos for

1. Diseña un programa que invierta una cadena de texto ingresada por el usuario.

```
palabra = input("Ingresa una palabra: ")
invertida = ""
for letra in palabra:
    invertida = letra + invertida
print("La palabra invertida es:", invertida)
```

2. Crea un programa que solicite un número al usuario y calcule la suma de todos los números desde 1 hasta el número ingresado.

```
n = int(input("Ingresa un número: "))
suma = 0
for i in range(1, n + 1):
    suma += i
print("La suma es:", suma)`;
```

3. Escribe un programa que imprima la tabla de multiplicar del 5 utilizando un bucle for.

```
for i in range(1, 11):
    print("5 x " + str(i) + " = " + str(5 * i))
```

4. Crea un programa que solicite una palabra al usuario y la imprima letra por letra utilizando un bucle for.

```
palabra = input("Ingresa una palabra: ")
for letra in palabra:
    print(letra)
```

5. Escribe un programa que muestre solo los números pares del 1 al 10 utilizando un bucle for.

```
for numero in range(1, 11):
    if numero % 2 == 0:
        print(numero)
```

6. Escribe un programa que calcule el factorial de un número ingresado por el usuario.

```
numero = int(input("Ingresa un número: "))
factorial = 1
for i in range(1, numero + 1):
    factorial *= i
print("El factorial de", numero, "es:", factorial)`;
```

7. Escribe un programa que imprima los números del 1 al 10 utilizando un bucle for.

```
for i in range(1, 11):
    print(i)
```

8. Escribe un programa que imprima los números del 1 al 5 utilizando un bucle for.

```
for i in range(1, 6):
    print(i)
```

9. Escribe un programa que imprima la tabla del 3 (del 3 al 30) utilizando un bucle for.

```
for i in range(3, 31, 3):
    print(i)
```

10. Crea un programa que verifique si un número es primo utilizando un bucle for.

```
numero = int(input("Ingresa un número: "))
es_primo = True
for i in range(2, numero):
    if numero % i == 0:
        es_primo = False
if numero > 1 and es_primo:
    print("El número es primo.")
else:
    print("El número no es primo.")
```

5.3. C. Certamen 1

Contexto: Invernaderos

Los agricultores de Pythonia han implementado modernos invernaderos automáticos, divididos en filas, cada una compuesta por varias secciones. Las filas pueden tener distinta cantidad de secciones, y hay un sensor de humedad en cada sección.

El estado de humedad de un invernadero se representa mediante un `string`, en el que las distintas filas están separadas entre sí por el símbolo “|”, y las secciones dentro de las filas por un guión “-“. Los valores de humedad de cada sección son valores enteros que van de 0 a 9.

Por ejemplo, el string “5-7-3-5|8-2|6-2-5|9-1-1-1-9” representa un invernadero compuesto por 4 filas y 14 secciones en total:

Fila	Secciones	Humedad por sección
1	4	5, 7, 3, 5
2	2	8, 2
3	3	6, 2, 5
4	5	9, 1, 1, 1, 9

Pregunta 1 [20 %] - Ruteo y salida del programa

En la hoja de respuesta correspondiente, realice el ruteo del programa que se incluye a continuación. Además, indique en el recuadro de la pantalla lo que imprime como salida. Cada vez que el valor de una variable cambie, escríbalo en una nueva fila de la tabla. Si una variable es de tipo `string`, su valor debe ir entre comillas. La tabla tiene suficientes filas.

Programa para el ruteo

```
def misterio(s):  
    i = 2  
    m = int(s[0])  
    while i < len(s):  
        if int(s[i]) > m:  
            m = int(s[i])  
        i += 2  
    return m
```

```
x = "2-5-8|1-9-1|9-1-1"
y = ""
i = 0
while i < 15:
    a = misterio(x[i:i+5])
    print(x[i:i+5], a)
    y = y + str(a) + "-"
    i += 6

print(y[:-1])
```

Pregunta 2 [40 %] - Función filas_secas

Escriba la función `filas_secas(invernadero, umbral)`, que recibe como parámetros un `string` correspondiente al estado de un invernadero, y un entero que indica el umbral de tolerancia. Este umbral determinara si una sección se considera seca o no. La función debe retornar la cantidad de filas completamente secas, es decir, aquellas en que todas sus secciones tienen humedad menor que el umbral.

Puede suponer que los parámetros cumplen con el formato indicado en el contexto. No es necesario hacer validaciones.

Nota: Hay muchas formas de implementar la función solicitada. No es obligatorio usar la función `misterio` del ruteo, pero si decide usarla, no es necesario que la copie nuevamente.

Ejemplos:

```
>>> print(filas_secas('5-7-3-5|8-2|6-2-5|9-1-1-1-9', 8))
2
>>> print(filas_secas('5-7-3-5|8-2|6-2-5|9-1-1-1-9', 7))
1
```

Explicación: En el primer caso, la primera y la tercera fila están completamente secas, pues los valores de humedad de todas sus secciones están por debajo del valor umbral 8. Por lo tanto, la función retorna 2, pues hay dos filas en esa condición. En el segundo caso, sólo la tercera fila está completamente seca, pues es la única para la que los indicadores de humedad de todas sus secciones están por debajo del valor umbral 7. Por esto la función retorna 1.

Pregunta 3 [40 %] - Programa de interacción

Escriba un programa que solicite al usuario ingresar un valor mínimo aceptable de humedad, para luego solicitar de manera repetitiva el estado actual de distintos invernaderos, siguiendo el formato descrito para el `string`.

Para cada invernadero, el programa debe indicar la humedad promedio de todas sus secciones, redondeada al entero más cercano. El proceso continúa hasta que se ingresa un invernadero que no tenga filas secas.

Antes de finalizar, el programa debe indicar la humedad promedio mínima y máxima para todos los invernaderos ingresados, sin considerar el último.

Ejemplo de ejecución:

```
Ingrese valor umbral: 8
Ingrese estado de un invernadero: 5-7-3-5|8-2|6-2-5|9-1-1-1-9
Secas: 2
Humedad promedio: 5
Ingrese estado de un invernadero: 6-7-7|7-6|6-5-6|7
Secas: 4
Humedad promedio: 6
Ingrese estado de un invernadero: 4-2-9|1-2-3-4-5-6-7-8|9
Secas: 0
El invernadero no tiene secciones secas
Invernaderos considerados: 2
Humedad promedio máxima: 6
Humedad promedio mínima: 5
```

Nota: Puede suponer que la función `filas_secas(invernadero, umbral)` de la P2 se encuentra implementada y funciona correctamente. No es necesario volver a copiarla y puede utilizarla aunque no haya respondido correctamente esa pregunta. Además, puede suponer que los `strings` ingresados cumplirán siempre con el formato descrito en el contexto y que siempre se ingresará al menos un invernadero que contiene filas secas.

Importante: Puede utilizar únicamente materia estudiada en la asignatura, hasta la Unidad 5.

REFERENCIAS BIBLIOGRÁFICAS

- [Bandura, 1997] Bandura, A. (1997). *Self-efficacy: The Exercise of Control*. W H Freeman/Times Books/ Henry Holt & Co, New York, NY, US. ISBN: 0-7167-2850-8.
- [Guzdial, 2004] Guzdial, M. (2004). Programming Environments for Novices. En Fincher, S. y Petre, M., editores, *Computer Science Education Research*, pp. 127–154. Taylor & Francis, Lisse, The Netherlands, 1 edición.
- [Meza et al., 2024] Meza, F., Vásquez, A., y San Martín, D. (2024). Validation of a bebras-based test to assess computational thinking abilities in first-year college students. En *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2, SIGCSE 2024*, p. 1750–1751, New York, NY, USA. Association for Computing Machinery.
- [Parsons y Haden, 2006] Parsons, D. y Haden, P. (2006). Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. En *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52, ACE '06*, p. 157–163, AUS. Australian Computer Society, Inc.
- [Pinto, 2020] Pinto, F. J. V. (2020). Impacto del Uso de un Lenguaje Basado en Bloques en el Desarrollo de Habilidades de Programación. Memoria para optar al título de Ingeniero Civil Informático, Departamento de Informática, Universidad Técnica Federico Santa María, Chile. Federico Meza, Profesor Guía.
- [Qian y Lehman, 2018] Qian, Y. y Lehman, J. (2018). Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.*, 18(1).
- [Ryan y Deci, 2000] Ryan, R. M. y Deci, E. L. (2000). Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology*, 25(1):54–67.
- [Sweller, 1988] Sweller, J. (1988). Cognitive Load during Problem Solving: Effects on Learning. *Cognitive Science*, 12(2):257–285.
- [Tek et al., 2018] Tek, F. B., Benli, K. S., y Deveci, E. (2018). Implicit Theories and Self-Efficacy in an Introductory Programming Course. *IEEE Transactions on Education*, 61(3):218–225.
- [Vygotsky, 1978] Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Oxford, England.