Desarrollo del pensamiento computacional en estudiantes de bachillerato a través de entornos de programación textual y basados en bloques

Leidy Lized García Aponte

Universidad Pedagógica Nacional

Facultad de Ciencia y Tecnología

Maestría en Tecnologías de la Información Aplicadas a la Educación

Bogotá, D.C.

2024

Desarrollo del pensamiento computacional en estudiantes de bachillerato a través de entornos de programación textual y basados en bloques

Leidy Lized García Aponte

Directora

Dra. Linda Alejandra Leal Urueña

Trabajo de grado para optar el título de Magister en Tecnologías de la Información Aplicadas a la Educación

Universidad Pedagógica Nacional

Facultad de Ciencia y Tecnología

Maestría en Tecnologías de la Información Aplicadas a la Educación

Bogotá, D.C.

2024

Dedicatoria

A mi esposo, mi hija, mis padres y mis hermanos, quienes han sido mi fuente de fortaleza, apoyo y amor incondicional a lo largo de este camino.

Agradecimientos

Quiero agradecer a Dios, por ser mi fortaleza y guía en cada paso de este camino profesional permitiéndome adquirir nuevos conocimientos en esta maestría.

A mi familia, por su amor incondicional y apoyo constante. A mi esposo, por su comprensión, paciencia y apoyo incondicional, y a mi hija, cuya alegría e inocencia iluminaron mis días y me llenaron de energía para continuar. A mis padres, cuyo ejemplo de esfuerzo y dedicación ha sido mi mayor inspiración; a mis hermanos, por su motivación y apoyo brindado.

Finalmente, a mi asesora de tesis la profesora Linda Alejandra Leal y demás profesores, por compartir su sabiduría, exigencia y confianza, contribuyendo de manera significativa a mi formación personal y profesional.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Introducción

El desarrollo del pensamiento computacional y de las habilidades de programación resultan esenciales en un mundo cada vez más digitalizado, lo que demanda estrategias educativas innovadoras, debido a que permiten a las personas abordar problemas complejos de manera lógica y estructurada, diseñando soluciones creativas y eficaces (Wing J. M., 2006). Es así que, organismos como la UNESCO (2020), promueven y buscan preparar a los jóvenes para enfrentarse a los desafíos del siglo XXI, impulsando a que sean creadores de tecnología y no solo consumidores de esta. Dichas habilidades no solo fortalecen la capacidad de resolver problemas, sino que empoderan a las personas para innovar y adaptarse a un entorno en constante transformación.

Los entornos de programación gráficos y textuales juegan un papel importante en este propósito, ya que facilitan el aprendizaje de la programación en diferentes niveles de complejidad, permitiendo a los estudiantes trabajar en la construcción de sistemas computacionales. El uso de entornos de programación basados en bloques y en texto ofrece diferentes ventajas y limitaciones (Weintropa & Wilensky, 2018). Investigaciones como las de Weintropa y Wilensky (2019), han resaltado que ambos enfoques tienen beneficios en diferentes etapas del aprendizaje, destacando la necesidad de que los educadores y los diseñadores de herramientas ayuden a facilitar la transición entre estos entornos.

En Colombia, los resultados de las pruebas PISA del 2022 y de la prueba Bebras 2023 premiten evidenciar desafios persistentes en competencias algoritmicas y de programación. De las 5 areas evaluadas en la competencia Bebras esta competencia fue la que obtuvo la de menor calificacion con un porcentaje de aprobación de solo el 24,56%, reflejando baja

capacidad de razonamiento e implementacion en procesos o pasos para resolver un problema (Fedesoft, 2023). En tanto, en los procesos de razonamiento matemático asociados con las habilidades de pensamiento computacional, los resultados de las pruebas PISA (2022) ubican a Colombia con un promedio de 383, muy inferior al promedio de la OCDE de 472.

La limitada investigación en el entorno colombiano invita a explorar de qué manera los entornos de programación visual y textual pueden contribuir al desarrollo de las habilidades de programación y el pensamiento computacional en los estudiantes de bachillerato (Quiroz Vallejo, Carmona Mesa, Castrillón Yepes, & Villa Ochoa, 2021). Específicamente, esta tesis se enfocó en investigar acerca de las diferencias en el desarrollo del pensamiento computacional al utilizar entornos de programación textuales y por bloques.

Para ello, se llevó a cabo un estudio cuasiexperimental con seis grupos de estudiantes, tres de grado décimo y tres de grado undécimo. Los primeros, emplearon una serie de actividades de resolución de problemas empleando el entorno de programación por bloques Scratch, mientras los segundos desarrollaron las mismas actividades empleando el entorno de programación textual en Visual Studio Code Python. Ambos grupos fueron evaluados antes y después de la intervención empleando el test de pensamiento computacional diseñado para medir las habilidades de pensamiento computacional con un enfoque principal en pensamiento algorítmico como habilidad cognitiva, el cual busca evaluar la capacidad de los participantes para identificar rutinas algorítmicas, descomposición de problemas, abstracción de elementos esenciales, reconocimiento de patrones y, evaluación y depuración de soluciones (Marc et al., 2022). Los resultados de esta investigación se presentan en los siguientes capítulos.

El capítulo uno se plantea el problema de investigación, destacando el desarrollo del pensamiento computacional como una de las habilidades esenciales del siglo XXI y se define el propósito del estudio en el contexto educativo nacional. En este se destacan los desafíos educativos en Colombia que se evidenciaron por medio de los bajos resultados en pruebas internacionales como PISA y Bebras. Teniendo en cuenta este contexto se plantea la necesidad de analizar las diferencias que puedan existir en el entorno de programación basado en bloque (Scratch) y textual (Visual Studio Code Python) en el desarrollo del pensamiento computacional en estudiantes de bachillerato, buscando identificar diferencias significativas en su efectividad.

El capítulo dos se centra en los antecedentes y para ello se realiza una revisión de las investigaciones previas sobre el desarrollo del pensamiento computacional y se analizan estudios comparativos entre los entornos de programación basados en bloques y textuales, adicionalmente se describe el estado actual de las prácticas educativas en Colombia identificando las oportunidades y desafíos.

El capítulo tres corresponde al Marco Teórico, en este se expone los fundamentos conceptuales del pensamiento computacional y los entornos de programación utilizados para el estudio, explicando los principios, características y aplicaciones, así como su impacto en el aprendizaje de las habilidades computacionales. En este se realiza un análisis de las propuestas descritas por autores como Wing (2006) y Brennan & Resnick (2012). Además, se realiza una explicación de los principios, características, aplicaciones e impacto que tiene el aprendizaje de habilidades computacionales en estudiantes de bachillerato.

El capítulo cuatro describe la Metodología utilizada para el estudio, basada en un enfoque cuasiexperimental con 131 estudiantes de grados decimo y undécimo del Colegio El Carmen Teresiano de Bogotá, los cuales se encontraban divididos, uno de ellos trabajo con el entorno de programación por bloques Scratch y el otro trabajo programación textual en Visual Studio Code Python. Para el estudio se desarrollaron tres fases: pretest, intervención didáctica y postest, con la finalidad de medir el impacto de los entornos de programación en el desarrollo del pensamiento computacional y adicionalmente en este capítulo se describen las actividades diseñadas para trabajar en los dos entornos de programación y se describen aspectos generales que se tuvieron en cuenta para el ingreso al entorno de aprendizaje.

El capítulo cinco presenta los análisis e interpretación de los resultados, en el cual se muestra los análisis de las condiciones iniciales, los efectos sobre la variable dependiente (pos-test), se realiza la comparación del rendimiento de los grupos. Se realiza un análisis estadístico utilizando SPSS, aplicando pruebas de normalidad, homogeneidad de varianzas y un análisis de covarianza (ANCOVA) para controlar las diferencias iniciales entre los grupos y se muestran los análisis en cuanto al desarrollo de las actividades trabajadas en cada entorno de programación por bloques y el entorno de programación textual.

El capítulo seis se realiza la discusión de los resultados obtenidos y se relacionan con investigaciones previas mostrando la efectividad que pudieron tener los dos entornos de programación trabajados.

El capítulo siete resume las principales conclusiones del estudio y se destacan los hallazgos para el diseño de estrategias educativas en el desarrollo del pensamiento computacional.

El capítulo ocho se muestran las contribuciones del estudio, limitaciones y posibles líneas de investigación futuras para continuar investigación sobre el tema.

1. Problema de Investigación

1.1. Pensamiento computacional: una habilidad esencial en el siglo XXI

El pensamiento computacional es considerado como una de las habilidades esenciales del siglo XXI, al igual que la lectura, la escritura o las matemáticas, según lo destaca Wing (2006), quien a su vez fue pionera en el término pensamiento computacional y lo definió como la capacidad de descomponer problemas complejos, pensar algorítmicamente, y desarrollar soluciones eficientes. García-Peñalvo & Cruz-Benito (2016), coinciden en que el pensamiento computacional debe integrarse en el curriculo educativo desde los primeros años, debido a que fomenta habilidades para la resolución de problemas y la creatividad, ademas, permite preparar a los estudiantes para los desafios de la sociedad contemporanea por medio del analisis y solución de problemas en un mundo tecnologicamente complejo.

De acuerdo con la UNESCO:

> "Aprender pensamiento computacional y programación, saber cuáles son los principios lógicos que hacen al funcionamiento de las tecnologías y poder utilizarlas de manera creativa para el diseño y desarrollo de sistemas digitales, son también, competencias fundamentales en un programa de Ciudadanía Digital que busca preparar a los estudiantes para responder a las demandas laborales del siglo XXI" (UNESCO, 2020, p.5).

Diversos programas destacan la importancia de la introducción del desarrollo del pensamiento computacional en el currículo latinoamericano. El Plan Ceibal en Uruguay y Aprende a programar en Chile, se han propuesto introducir en las escuelas el pensamiento computacional. A su vez, Profuturo, en colaboración de La Fundación Telefónica Vivo,

promueven la enseñanza de la programación y la robótica en escuelas públicas de Brasil, incentivando en el aprendizaje por medio de la resolución de problemas reales haciendo uso de la tecnología. Así mismo, la fundación Sadosky ubicada en Buenos Aires (Argentina), promueve por medio de la enseñanza de la programación en escuelas secundarias cursos, capacitaciones docentes y desarrollo de contenidos curriculares, utilizando lenguajes y entornos de programación basados en bloques como Scratch, y entornos basados en texto, principalmente con el uso del lenguaje de programación Python.

En Colombia, las iniciativas han sido lideradas por el Ministerio de las Tecnologias de las Información y las Comunicaciones (MinTic) en alianza con el British Council, quienes desde 2019, desarrollan el programa de formación *"Coding For Kids",* con el que buscan fortalecer en los docentes y niñas y niños colombianos sus competencias de pensamiento lógico y computacional, capacidad creativa y de resolución de problemas. Asi mismo, estas dos instituciones, con apoyo del Ministerio de Educación Nacional, han desarrollado la estrategia Código Verde del proyecto *"Colombia Programa"* (2024), que tiene como propósito generar recursos y oportunidades de desarrollo profesional docente para fomentar el pensamiento computacional en instituciones educativas oficiales de Colombia, con un enfoque en pro de la equidad de género (Micrositio - Colombia Programa - Inicio - Colombia Programa, s. f.). Desde su lanzamiento, la App ha sido descargada por 100000 estudiantes, quienes vienen participando de este componente educativo, sumando más de 1.486 días, 20 horas, 50 minutos y 24 segundos de tiempo efectivo, lo que equivale a más de 4 años dedicados a fortalecer estas habilidades fundamentales para enfrentar los retos tecnológicos del futuro (MinTIC, 2024).

Pese a estas iniciativas, los indicadores de resultado en torno a las habilidades de los estudiantes siguen causando preocupación. De acuerdo con cifras reportadas por la Federación

Colombiana de la Industria del Software y Tecnologías de la Información Relacionadas – Fedesoft, el promedio nacional alcanzado en la más reciente pruebas de Pensamiento computacional Bebras Colombia en el año 2023, aplicada por Bebras Colombia, fue 4.6 sobre 15. Esto representa una reducción de más de 2.25 puntos porcentuales frente al promedio de 2022 que fue de 6.92 (Fedesoft, 2023).

En esta última edición de la prueba participaron 11055 estudiantes de los grados cuarto (°4) a undécimo (11°), de 18 departamentos, quienes participaron en la prueba diagnóstica, la cual midió 5 áreas de conocimiento del pensamiento computacional: algoritmos y programación, estructuras de datos y representaciones, procesos informáticos y hardware, comunicación y redes e interacciones y sistemas y sociedad. Algoritmos y programación fue el área de menor calificación con un porcentaje promedio de 24,2% de puntos aprobados, lo que deja al país en un nivel bajo, teniendo en cuenta que la escala de puntaje del diagnóstico tiene un rango de 0 a 15, siendo 0 muy bajo y 15 muy bueno (Las falencias en el pensamiento computacional, 2024).

Estos datos, junto con los últimos resultados de la prueba internacional PISA (2022), en la que los estudiantes en Colombia obtuvieron puntuaciones inferiores al promedio de la Organización para la Cooperación y el Desarrollo Económico (OCDE). Colombia se ubicó en el puesto 64 de 81 países evaluados, con los siguientes puntajes en las áreas evaluadas: matemáticas (383 puntos), lectura (409 puntos) y ciencias (411 puntos). El 29% de los estudiantes de Colombia alcanzaron al menos el nivel de desempeño 2 en matemáticas, mientras en los países de la OCDE lo alcanzaron el 69%.

Específicamente, la institución educativa en la que se llevó a cabo este estudio, el Colegio el Carmen Teresiano, según las pruebas ICFES 2023 se clasifica en el puesto 129 y con un

puntaje global en el área de matemáticas de 64 puntos. Estos resultados, evidencian que el país enfrenta grandes restos para mejorar la calidad de la educación y su pertinencia en torno a las competencias que demanda el siglo XXI.

En este escenario, el desarrollo de habilidades de resolución de problemas mediante el enfoque del pensamiento computacional puede contribuir en la mejora de las habilidades de los estudiantes y en su desempeño en las pruebas académicas estandarizadas. Al tiempo que resulta esencial para afrontar el déficit de 112.000 programadores en Colombia para 2025, según estudios realizados por MinTIC y Fedesof (S.A.S, 2022), cifras que evidencian la necesidad de fortalecer habilidades de pensamiento computacional y programación en los estudiantes colombianos.

1.2. Desarrollo del pensamiento computacional a través de entornos de programación basados en bloque y textuales

El desarrollo de entornos de programación basada en bloques como Scratch y Blockly ha permitido que estudiantes de nivel de primaria y secundaria desarrollen habilidades de programación de manera intuitiva (Resnick , Rusk, & Maloney, 2009), con lo cual reducen la barrera de entrada a la programación, permitiendo a los estudiantes concentrarse en la resolución de problemas. Estos entornos han sido ampliamente estudiados como herramientas pedagógicas en la enseñanza del pensamiento computacional. Resnick y otros (2009), establecen que Scratch es una herramienta fundamental que permite a los estudiantes de diversas edades aprender programación sin los obstaculos de la sintaxis compleja, enfocándose en la lógica y estructura de los algoritmos. Esto ha facilitado el acceso a la programación a principiantes, haciéndolos más receptivos y motivados.

Grover & Pea (2013), indican que los entornos por bloques contribuyen a la adquisición de competencias como la descomposición de problemas, la identificación de patrones y el uso de abstracciones, todos ellos pilares fundamentales del pensamiento computacional, en donde los entornos basados en bloques simplifican el proceso cognitivo, ayudando a los estudiantes en la visualización del flujo de control en los programas y desarrollando habilidades de resolución de problemas sin sentirse frustrados en errores sintácticos. Estos entornos han demostrado que son accesibles e inclusivos, debido a que pueden ser utilizados por estudiantes con diversas capacidades.

Por otra parte, los entornos de programación textual, como Python, Java, y JavaScript, presentan un enfoque más avanzado que los entornos basados en bloques y son comúnmente utilizados en la educación secundaria y universitaria. Weintrop & Wilensky (2017), realizaron una comparación entre entornos por bloques y textuales, y encontraron que, los lenguajes de programación gráficos tienden a ser más accesibles inicialmente, pero los lenguajes de programación textuales permiten una comprensión más profunda de los principios de programación a medida que los estudiantes progresan.

Grover, Cooper & Pea (2014), señalan que los lenguajes textuales contribuyen a la construcción de pensamiento lógico-matemático, esencial para disciplinas STEM (Ciencia, Tecnología, Ingeniería y Matemáticas), pero uno de los retos es que los errores de sintaxis en estos lenguajes a menudo frustran a los estudiantes que están iniciando en la programación, lo que puede llevar a una disminución en la motivación y el interés por la programación.

Los estudios llevados a cabo por Weintrop & Wilensky (2017) y Grover & Pea (2013) coinciden en que es efectivo introducir a los estudiantes en el desarrollo del pensamiento

computacional inicialmente en entornos basados en bloques y a medida que adquieren experiencia introducirlos en entornos textuales que les permite una comprensión más profunda y los prepara mejor para problemas complejos en programación.

Yusuf & Román-González (2024), indican que, existe un acuerdo común de que el pensamiento computacional permite a los estudiantes desarrollar el pensamiento creativo y las habilidades de resolución de problemas, sin embargo, la adquisición de habilidades de pensamiento computacional a partir de la educación en programación se ha visto desafiada por la dificultad percibida desde hace mucho tiempo de la programación informática, teniendo en cuenta que los entornos de programación grafica permiten una retroalimentación instantánea a diferencia de los entornos de programación textual, pero a su vez identificar y corregir errores de sintaxis requiere atención y resolución de problemas que puede conducir a una carga cognitiva que conlleva a una obstaculización en el proceso de aprendizaje, además de escasez de evidencia de investigación, se presta poca atención a cómo los estudiantes difieren en sus habilidades de pensamiento computacional con respecto a la competencia en programación cuando interactúan con estos entornos.

En esta línea, el propósito de esta investigación es analizar y comparar el impacto que tienen los entornos de programación textual y basados en bloques en el desarrollo del pensamiento computacional de los estudiantes de los grados 10 y 11 de bachillerato del Colegio El Carmen Teresiano de la ciudad de Bogotá. La investigación busca determinar si existen diferencias en el desarrollo del pensamiento computacional cuando se emplean entornos de programación gráficos y textuales durante las clases de informática. Para ello, se diseñaron guías de actividades de resolución de problemas para ser resueltas a través de los entornos de programación basada en bloques Scratch y en el entorno de programación textual de Visual

Studio Code Python, con el fin de analizar sus ventajas y limitaciones durante la solución de problemas de programación. Con este estudio se pretende dar respuesta a la siguiente pregunta de investigación: ¿Existen diferencias significativas en el desarrollo del pensamiento computacional en estudiantes de Bachillerato cuando se emplean entornos de programación gráfica y textual durante el desarrollo de actividades de resolución de problemas?

1.3. Objetivos

Objetivo General:

Evaluar la incidencia de los entornos de programación textual y gráfica en el desarrollo del pensamiento computacional en estudiantes de Bachillerato.

Objetivos específicos:

- Diseñar un conjunto de actividades orientadas al aprendizaje de conceptos computacionales para ser desarrolladas en entornos de programación basados en bloques y en entornos de programación textual.

- Evaluar las habilidades de pensamiento computacional en estudiantes que solucionan problemas de programación empleando entornos de programación textuales y basados en bloques.

- Establecer ventajas y limitaciones de los entornos de programación textual y basados en bloques en la apropiación de conceptos computacionales y en el desarrollo del pensamiento computacional.

2. Antecedentes

2.1. Desarrollo del pensamiento computacional en estudiantes de bachillerato en Colombia

Con respecto al desarrollo del pensamiento computacional en Colombia, la revisión documental permitió identificar los siguientes estudios. Basogain et al., (2017), presenta los resultados de la colaboración internacional de RENATA y UPV/EHU en el desarrollo del proyecto 'Introducción del Pensamiento Computacional en las escuelas de Bogotá y Colombia', el cual fue impartido en diez escuelas públicas de Colombia durante el último trimestre del año académico 2016-17. Este curso introdujo los "conceptos" y "procesos" básicos de Pensamiento Computacional ayudado por el entorno de programación visual Scratch. El curso fue diseñado para ser impartido por los profesores con el apoyo de una plataforma de aprendizaje con el propósito de que los estudiantes aprendieran los principios de un lenguaje de programación informático. Y evidencia que este tipo de iniciativas académicas proporciona a las instituciones beneficios de participar en nuevos campos académicos, tales como el pensamiento computacional, que no están disponibles en las ofertas tradicionales de la escuela, debido a la implementación exitosa del curso que logró introducir conceptos y procesos básicos de pensamiento computacional, facilitando la enseñanza de programación por medio de Scratch. A su vez, las evaluaciones basadas en pruebas y proyectos de Scratch, evidenciaron un impacto positivo en las instituciones educativas, docentes y estudiantes, las cuales tenían revisión por pares.

Por medio del proyecto "enREDAdos" se introdujo a los estudiantes en una metodología basada en el aprendizaje por proyectos y centrada en la creación de Recursos Educativos Digitales Abiertos (REDA), en esta estrategia se buscó fortalecer habilidades de pensamiento

computacional mediante actividades que conecta problemas del contexto local con las Tecnologías de la Información y la Comunicación (TIC), promoviendo el pensamiento crítico y lógico, la colaboración y la creatividad, en el cual se pudo evidenciar mejoras en las competencias computacionales y en la percepción de los estudiantes sobre su capacidad para resolver problemas mediante tecnología (Sinisterra, 2018).

Fernández Díaz et al. (2023), indican que el pensamiento computacional versus el pensamiento matemático muestran una fuerte correlación en estudiantes de séptimo y noveno grado de una institución en Barranquilla, en donde los procesos cognitivos como son: secuencias, correspondencia y depuración del pensamiento computacional contribuyen en el fortalecimiento de la comprensión conceptual, el desarrollo procedimental y la argumentación matemática, en este estudio se evidencia que el desarrollo de habilidades de pensamiento computacional tienen un impacto positivo en el aprendizaje matemático, para lo cual se resalta las técnicas algorítmicas como el planteamiento y solución de problemas por medio de la simulación y modelación, que pueden ser útiles para mejorar los bajos resultados obtenidos por los estudiantes en las pruebas Saber 11.

El desarrollo de habilidades para resolver problemas por medio del pensamiento computacional en educación media en Colombia según Mantilla Guiza y Negro Bennasar (2021), siguiendo el modelo de Brennan y Resnick (2012), se organiza en dimensiones conceptuales, practicas y perspectivas que han demostrado una correlacion significativa en el fortalecimiento de habilidades como son la abstraccion, la descomposicion y el diseño de algoritmos, esto en el contexto de la pandemia (COVID 19), en donde el pensamiento computacional ha permitido la continuidad educativa mediante ambientes virtuales y en donde se subraya la importancia de la integracion de enfoques innovadores que combinan la teoría y práctica, adaptados al marco

educativo colombiano, para maximizar el impacto del pensamiento computacional en la formación de ciudadanos digitales competentes.

Espinal et al., (2023), en el estudio " Capacidades y dificultades de los estudiantes para la transferencia de un lenguaje de programación basado en bloques a otros lenguajes de programación: un estudio de caso en Colombia", analizaron la capacidad de los estudiantes para transferir habilidades desde un lenguaje de programación MakeCode realizando actividades de transferencia en el lenguaje de programación en bloques (Scratch), hacia lenguajes más avanzados, como es el lenguaje de programación textual (Python) durante un protocolo de entrevista. Para ello, su principal enfoque era identificar qué conocimientos y habilidades se transferían con éxito y cuales generaban dificultades. Los resultados mostraron que los estudiantes lograron transferir conceptos como secuencias y bucles, pero presentaron inconvenientes en sintaxis y abstracción. Aunque se evidenciaron dificultades, también se puede destacar que para los estudiantes que tenían conocimientos en programación grafica lograron avanzar al lenguaje de programación Python más rápidamente. Así mismo, se logró identificar que, si bien la mayoría de los estudiantes pueden transferir entre lenguajes de programación basados en bloques, en su mayoría luchan por explicar un programa en un lenguaje de programación basado en texto y a su vez resolver un nuevo desafío de codificación. Para ello consideran que los diseñadores instruccionales deben revisar las diferentes estrategias para facilitar la transferencia de los estudiantes a lenguajes de programación, particularmente cuando es difícil para quienes no hablan inglés.

Por lo tanto, la inclusión del pensamiento computacional en el currículo puede conllevar a la preparación de estudiantes para el mercado laboral, aunque su impacto en el desempeño académico y el desarrollo de habilidades practicas requieren equipar a los estudiantes en

competencias necesarias que reflejen la creciente necesidad del uso de la tecnología en sus prácticas académicas.

2.2. Estudios comparativos de entornos de programación basados en bloques y en texto

Estudios previos reconocen que los recursos proporcionados por una interfaz específica dan forma a la modalidad – programación basada en bloques o textual - y esa modalidad impacta en las prácticas de programación. Asimismo, la modalidad, la interfaz y las características del entorno y la representación son maleables y pueden afectar tanto lo que los usuarios pueden hacer como la forma en la que lo pueden hacer (Weintrop & Wilensky, 2018).

Price & Barnes (2015), compararon los logros de 17 estudiantes novatos de grado sexto que utilizaban entornos de bloques frente a 14 estudiantes de grado séptimo que emplearon un entorno textual basado en la Web, denominado Tiled Grace. El ejercicio se dividió en 9 secciones, con un texto tutorial que presentaba cada sección con sus objetivos diferenciándose únicamente en las interfaces, haciendo referencia a "bloques" o a "código".

Se evaluó la eficacia y el interés de los estudiantes por medio de una encuesta previa y posterior, haciendo uso de las escalas de uno (1) a cinco (5), en donde las puntuaciones de conocimiento oscilaron entre cero (0) y tres (3), en la evaluación de eficacia los resultados en la encuesta previa con el grupo que realizo actividades en el entorno de bloques se obtiene M=3.51 (SD=0.90), en la encuesta posterior M=3.88 (SD=0.73), y el grupo que realizo actividades en el entorno de texto se obtiene en el cuestionario previo M=3.59 (SD=0.70), en el cuestionario posterior M=3.75 (SD=0.93).

Así mismo, en la evaluación de interés se obtiene en el cuestionario previo en el grupo que realizo actividades en el entorno por bloques M=4.24 (SD=0.70) y en el posterior M=4.26 (DS=0.53), y en el grupo que realizo actividades en el entorno en texto en el cuestionario previo obtuvieron M=3.76 (SD=0.71) y en el cuestionario posterior M=3.93 (SD=0.81).

Se logro evidenciar que las puntuaciones de conocimiento para los estudiantes que realizaron actividades en el entorno de programación grafica en el cuestionario previo fue de M=1.53 (SD=0.94) y el posterior de M=1.30 (SD=0.95), y en los estudiantes que realizaron actividades en el entorno de programación textual en el cuestionario previo fue M=1.14 (SD=0.86) y en el cuestionario posterior M=1.13 (SD=0.99). A su vez, se realizó una prueba de Mann-Whitney en la cual se pudo evidenciar que el grupo por bloques obtuvo una respuesta significativa en la calificación de interés de W=163, p=0.040. Por lo tanto, no hubo otras diferencias significativas entre los grupos.

Por otro lado, entornos basados en bloques como Scratch facilitan una disminución significativa de la barrera de entrada al pensamiento computacional, permitiendo que los estudiantes principiantes adquieran competencias fundamentales en las estructuras algorítmicas y desarrollen capacidades de razonamiento lógico, sin verse obstaculizados por las complejidades sintácticas de los lenguajes de programación tradicionales (Malan & Leitner, 2007). Las comparaciones entre novatos, de edades que oscilan entre 11 y 13 años, que usaron interfaces de programación textual y basada en bloques encontraron que el grupo de bloques completó significativamente más objetivos en menos tiempo y pasó significativamente menos tiempo inactivo en comparación con el grupo de texto, por lo anterior se muestra, la inactividad del grupo de bloques fue de M=407.2 (SD=238.9) a diferencia del grupo de texto que fue de M=793.5 (SD=368.3), y el tiempo que permanecieron activos los del grupo de bloques fue de

M=1866.8 (SD=617.4) y el grupo de texto de M=1414.5 (SD=463.1), lo cual evidencia significativamente que el grupo por bloques obtuvo menor inactividad (Price & Barnes, 2015).

Otros estudios han analizado entornos híbridos que combinan características de las interfaces basadas en bloques y en texto (Weintropa & Wilensky, 2018). Los resultados señalan que en estudiantes de grado 12, los entornos basados en bloques pueden ayudar en futuros aprendizajes en escenarios de programación basados en texto, especialmente en lo que se refiere al conocimiento conceptual y la capacidad de crear programas exitosos, mostrando mejores desempeños en futuros cursos de Java una vez completado el plan de estudios, debido a que , en promedio, los estudiantes en la condición de bloques produjeron programas de mayor duración que sus compañeros de texto e híbridos. En 10 de las 13 asignaciones en el plan de estudios de 5 semanas, los estudiantes de bloques produjeron los programas más largos en promedio, y los estudiantes en la condición híbrida produjeron los programas más largos en las otras tres asignaciones. La programación, tanto en entornos basados en bloques como en entornos textuales, ha demostrado ser una herramienta eficaz para fomentar el desarrollo del pensamiento computacional en los estudiantes. Sin embargo, la efectividad de cada entorno en cuanto a la adquisición de habilidades computacionales varía según el nivel educativo, el contexto, y los objetivos pedagógicos.

Grover & Pea (2013), identificaron que los entornos graficos permiten a los estudiantes principiantes comprender conceptos de programacion en educación media y secundaria e indican que estos entornos pueden ser beneficiosos para luego avanzar a entornos textuales. Malan & Leitner (2007), observaron que por medio de los entornos graficos los estudiantes de secundaria y bachillerato interactuaron con conceptos complejos de forma más accesible e intuitiva.

Price & Barnes (2015), indicaron que los estudiantes que trabajaron en entornos graficos muestran un mayor compromiso y una menor frustración, sugiriendo que los entornos graficos permiten ofrecer una introducción amigable al pensamiento computacional. En este estudio se exploró el impacto de los entornos de bloques y textuales en estudiantes novatos, encontrando que el 81% de los estudiantes que usaron entornos de bloques lograron completar las actividades, mientras que solo el 54% de quienes usaron entornos textuales tuvieron el mismo éxito. Además, los estudiantes que trabajaron en entornos basados en bloques mostraron un mayor compromiso y una menor frustración, lo cual sugiere que los entornos basados en bloquespueden ofrecer una introducción más amigable al pensamiento computacional.

Weintropa & Wilensky (2018), encontaron que los entornos de programación por bloques son mas efectivos para la enseñanza a estudiantes de educación secundaria en cuanto a los principios basicos del pensamiento computacional como es la descomposición de problemas y la creación de algoritmos sencillos, según los datos estadisticos demuestran como la modalidad de programación basada en bloques, texto o hibrida, para lo cual, se identifico que los estudiantes en modalidad hibrida ejecutaron los programas en promedio (1073), en comparacion con la modalidad bloques (733) y texto (742.9), lo cual destaca que las interfaces hibridas pueden fomentar practicas de programación productivas. Sin embargo, dicha investigación sugiere que los estudiantes que logran aprender y comprender los lenguajes textuales desarrollan una comprensión más significativa de los conceptos avanzados de programación y pensamiento computacional.

Los resultados de este estudio muestran que los estudiantes de bloques ejecutan un programa un promedio de 733 veces, los estudiantes de hibrido ejecutan sus programas un promedio de 1073 veces y los estudiantes de texto ejecutan sus programas 742,9 veces en

promedio. Un cálculo ANOVA del número promedio de ejecuciones por estudiante en cada condición muestra que hay una diferencia estadísticamente significativa entre la condición $F(2,89) = 8,71$, $p < .001$. Un análisis post hoc de Tukey HSD muestra que los estudiantes en la condición híbrida ejecutaron sus programas significativamente más a menudo que las otras condiciones (en comparación con Bloques $p < .001$, en comparación con Texto $p = .003$), mientras que no hubo diferencia en el número de ejecuciones entre los estudiantes de Bloques y Texto ($p = .86$).

Algunos estudios sugieren que los estudiantes que inician con entornos de programación basada en bloques y realizan la transición a lenguajes textuales obtienen un rendimiento superior en la resolución de problemas algorítmicos complejos, mejora en la capacidad de manejar estructuras de datos y algoritmos avanzados, en comparación con aquellos que solo usaron entornos de bloques (Weintrop & Wilensky, 2019).

La comparación directa entre ambos tipos de entornos, especialmente en el nivel de educación secundaria, sugiere que cada uno tiene ventajas en diferentes etapas del aprendizaje: los entornos basados en bloques son ideales para la introducción de estudiantes a conceptos de programación, para mejorar la motivación y reducir la frustración, especialmente en estudiantes con menos experiencia en programación. Además, los estudiantes que usan entornos basados en bloques muestran mayor disposición a experimentar y a participar en proyectos creativos (Malan y Leitner, 2007). Mientras que, los entornos textuales son más adecuados para estudiantes con una base previa en programación. La complejidad añadida de la sintaxis les permite desarrollar habilidades avanzadas de pensamiento computacional, aunque la curva de aprendizaje es más pronunciada. Como mencionan Grover y Pea (2013), los entornos textuales preparan mejor a los estudiantes para futuros desafíos en la programación

profesional.

El análisis de estos estudios sugiere que la combinación de ambos enfoques puede ser una estrategia pedagógica efectiva. Un modelo progresivo en el que los estudiantes comienzan con entornos basados en bloques y luego avanzan a entornos textuales ha encontró que los estudiantes que primero usan entornos basados en bloques muestran un 35% más de éxito al pasar a entornos textuales, en comparación con aquellos que comienzan directamente con texto (Weintrop & Wilensky, 2019). Además, estos estudiantes tienen un 20% menos de errores sintácticos cuando finalmente trabajan en entornos textuales, lo que sugiere que la base proporcionada por los bloques ayuda a mitigar las dificultades sintácticas.

Finalmente, los resultados de las investigaciones que han comparado el desarrollo del pensamiento computacional cuando se emplean entornos de programación textual y basados en bloques señalan que los estudiantes logran mayores avances en el aprendizaje en entornos basados en bloques en comparación con alternativas basadas en texto, pero que estos avances no resultan en una diferencia en la instrucción basada en texto con respecto al aprendizaje conceptual, cambios de actitud o prácticas de programación exitosas (Weintrop & Wilensky, 2019).

3. Marco Teórico

3.1. Pensamiento Computacional

El término pensamiento computacional fue introducido por Seymour Papert en los años 60 bajo la idea de que la programación de computadoras puede proporcionar a los niños una manera de pensar acerca de su propio pensamiento y aprender sobre su propio aprendizaje (MIT, 2016).

Posteriormente, se define por Wing J. M., (2006) como: "el Pensamiento Computacional implica la resolución de problemas, el diseño de sistemas y la comprensión de la conducta humana, haciendo uso de los conceptos fundamentales de la informática" (p. 33).

De acuerdo con Wing (2008):

> **"**El pensamiento computacional es un tipo de pensamiento analítico. Comparte con el pensamiento matemático las formas generales en las que podemos abordar la solución de un problema. Comparte con el pensamiento de ingeniería las formas generales en las que podemos abordar el diseño y la evaluación de un sistema grande y complejo que opera dentro de las limitaciones del mundo real. Comparte con el pensamiento científico las formas generales en las que podemos abordar la comprensión de la computabilidad, la inteligencia, la mente y el comportamiento humano". (p. 3717).

La *International Society for Tecnology in Education & Computer Science Teachers Association* (2011), propuso una definición del pensamiento computacional como un proceso de resolución de problemas que incluye, pero no se limita, a: formular problemas de una manera que nos permita usar una computadora y otras herramientas para ayudar a resolverlos, organizar y

analizar datos de forma lógica, representación de datos a través de abstracciones como modelos y simulaciones, automatización de soluciones a través del pensamiento algorítmico (una serie de pasos ordenados), identificar, analizar e implementar posibles soluciones con el objetivo de lograr la mayor combinación eficiente y eficaz de pasos y recursos, generalizar y transferir este proceso de resolución de problemas a una amplia variedad de problemas.

Estas habilidades están respaldadas y mejoradas por una serie de disposiciones o actitudes que son dimensiones esenciales del pensamiento computacional y que incluyen: confianza en el manejo de la complejidad, persistencia en trabajar con problemas difíciles, tolerancia para la ambigüedad, la capacidad de lidiar con problemas abiertos y la capacidad de comunicarse y trabajar con otros para lograr una meta o solución común (ISTE, 2011).

Brennan & Resnick (2012), desarrollaron un marco conceptual enfocado en el uso de la plataforma Scratch para enseñar y evaluar el pensamiento computacional. Este enfoque se basa en tres dimensiones clave: conceptos computacionales (secuencias, bucles, condicionales, operadores, datos), practicas computacionales (experimentación y depuración, construcción incremental, reutilización), y perspectivas computacionales.

Más recientemente, Lafuente Martínez, Levéque, Benítez, Hardebolle, & Dehler Zufferey (2022), adaptarón cinco componentes para definir el pensamiento computacional: pensamiento algoritmico, descomposición, abstracción, reconocimiento de patrones y evaluación/depuración. El pensamiento algoritmico tiene como objetivo identificar y representar rutinas para resolver problemas o tareas, como instrucciones ordenadas paso a paso; la descomposicion, desglosar un problema, algoritmo o procesar en partes mas pequeñas de modo que los resultados parciales puedan integrarse más tarde para resolver o comprender mas

facilmente el problema completo; la abstraccion, identificar elementos esenciales de un problema o proceso, esto implica simplificar y ocultar detalles; el reconocimiento de patrones, inferir e identificar patrones, tendencias o regularidades en un determinado problema o proceso; y la evalución/depuración, revisar la adecuacion de soluciones o elementos a un problema.

3.2. Entornos de programación textual

La programación basada en texto o textual hace referencia a la secuencia de instrucciones que componen un algoritmo y que están escritas en un lenguaje de programación determinado, basado en las normas establecidas en el propio lenguaje. Este conjunto de sentencias es analizado y "traducido" por el procesador para realizar las tareas para las que ha sido creado (Programación y Robótica, s. f.).

A diferencia de la programación basada en bloques, los lenguajes de programación textual requieren unos conocimientos más amplios, así como un alto grado de abstracción. Aprender un lenguaje de programación textual requiere aprender la sintaxis y la gramática específica del lenguaje, lo cual dificulta su uso (Programación y Robótica, s. f.). No obstante, desde la década de los 60 se han ido desarrollando distintos entornos de programación y en la actualidad existen multitud de lenguajes y de entornos visuales más amigables para el usuario (INTEF, 2022).

A diferencia de los entornos gráficos, donde las instrucciones se representan mediante bloques visuales, los entornos textuales exigen una mayor precisión y comprensión de las reglas del lenguaje de programación. Estos entornos son considerados una etapa más avanzada en el proceso de aprendizaje de la programación, ya que requieren que los estudiantes dominen tanto el pensamiento lógico como el manejo de errores sintácticos (Grover & Pea, 2013).

Los entornos de programación textuales pueden clasificarse según el nivel educativo, nivel de dificultad del lenguaje de programación y ámbito de aplicación: es así que Python, es uno de los lenguajes de programación textuales más populares en la enseñanza inicial de la programación. Python es conocido por su sintaxis simple y legible, lo que facilita la transición de los estudiantes desde entornos de bloques a la programación textual (Lye & Ling Koh, 2014). Python se utiliza ampliamente en educación secundaria y universitaria, así como en el mundo profesional.

3.2.1. Entorno Visual Studio Code Python

Es uno de los lenguajes más recomendados para enseñar a programar debido a su simplicidad y versatilidad. Es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea, permitiendo a los estudiantes ver inmediatamente el resultado de sus instrucciones y facilitando la depuración de errores.

A diferencia de otros lenguajes textuales como Java o C++, Python es conocido por su sintaxis sencilla que se asemeja al lenguaje natural, lo que facilita el aprendizaje de los estudiantes. Al eliminar la complejidad de la sintaxis, Python permite a los estudiantes centrarse en los conceptos fundamentales de la programación y el pensamiento computacional, como la descomposición de problemas, abstracción, y algoritmos (Román-González et al., 2015).

Python no solo es útil para aprender a programar, sino que también tiene aplicaciones en múltiples disciplinas, incluyendo la inteligencia artificial, ciencia de datos, desarrollo web y automatización. Esto motiva a los estudiantes a aprender Python porque pueden ver su utilidad en áreas del mundo real. Tiene una gran cantidad de bibliotecas y módulos que facilitan la enseñanza y el aprendizaje. Es utilizado en muchos currículos escolares para enseñar a los

estudiantes los conceptos básicos de programación y pensamiento computacional, desde la creación de programas sencillos hasta la implementación de algoritmos más avanzados.

3.2.2. Entornos de programación basada en bloques

La programación por bloques proporciona pistas visuales al usuario sobre cómo y dónde se pueden usar los comandos para restringir la composición del programa (Weintropa & Wilensky, 2018). Los entornos de programación basados en bloques son plataformas educativas diseñadas para enseñar conceptos de programación utilizando una interfaz gráfica. En lugar de escribir código textual, los estudiantes manipulan bloques que representan comandos y estructuras lógicas, los cuales pueden ser ensamblados para crear programas. Esta representación visual reduce las barreras relacionadas con la sintaxis, facilitando el aprendizaje de conceptos fundamentales de la programación y del pensamiento computacional (Resnick, y otros, 2009).

Estos entornos permiten la programación por bloques y el desarrollo de diferentes actividades con finalidades de mejorar el desarrollo del pensamiento computacional, al tiempo que se fomenta el pensamiento creativo y sistemático usando la programación para expresar ideas (Resnick , Rusk, & Maloney, 2009). Los entornos de programación basados en bloques incluyen funciones diseñadas para facilitar el acto de programar como son booleana y número/cadena (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010).

Los bloques están diseñados para ensamblarse entre sí de manera coherente, lo que permite a los estudiantes comprender la estructura y el flujo del programa de manera más intuitiva (Weintrop & Wilensky, 2019). Esto no solo ayuda a los estudiantes a concentrarse en la lógica de la programación, sino que también reduce la frustración asociada con los errores

sintácticos comunes en lenguajes textuales (Grover & Pea, 2013).

Existen diferentes tipos de entornos de programación por bloques, que se clasifican según el nivel educativo, las funcionalidades que ofrecen y los ámbitos de aplicación. Sin embargo, en esta investigación nos enfocaremos en Scratch, el cual está diseñado para estudiantes de educación primaria y secundaria y es uno de los más utilizados, permitiéndoles crear proyectos interactivos como juegos y animaciones mientras aprenden conceptos de programación como secuencias, bucles y condicionales (Resnick, y otros, 2009). Scratch ha sido identificado como una herramienta clave para fomentar la creatividad y el pensamiento crítico en estudiantes novatos (Brennan & Resnick, 2012).

Este entorno fue desarrollado por Massachusetts Institute of Technology (MIT), el cual está diseñado para que los estudiantes aprendan conceptos básicos de programación mediante la creación de proyectos interactivos. Dicho entorno fomenta la creatividad, el pensamiento crítico, y la colaboración entre los estudiantes (Resnick et al., 2009). El entorno facilita el desarrollo de pensamiento computacional mediante la construcción de secuencias, algoritmos, y el uso de estructuras condicionales.

Además, Brennan y Resnick (2012) introdujeron un marco para evaluar el desarrollo del pensamiento computacional utilizando Scratch. Este marco se basa en tres dimensiones: conceptos computacionales (secuencias, bucles, condicionales, operadores, datos), prácticas computacionales (experimentación, depuración, construcción incremental), perspectivas computacionales (percepciones sobre la programación y su utilidad). Es así como, el éxito de Scratch radica en su capacidad para motivar a los estudiantes a aprender programación sin

enfrentar las barreras comunes de los lenguajes textuales, lo que ha sido validado en numerosos estudios sobre su impacto positivo en la educación (Grover & Pea, 2013; Resnick et al., 2009).

4. Metodología

4.1. Tipo de investigación

Se realizó una investigación cuasiexperimental, ya que se trabajó con grupos preexistentes de estudiantes de grados décimo y undécimo del Colegio El Carmen Teresiano en Bogotá. El estudio se desarrolló con seis cursos, tres de grado decimo y tres de grado undécimo. Los participantes fueron organizados en dos grupos experimentales: Un grupo trabajó en actividades empleando el entorno basado en bloques Scratch, mientras que el otro grupo realizó las mismas actividades utilizando el entorno textual Python por medio de Visual Studio Code. Este diseño permitió comparar los efectos de ambos entornos en el desarrollo del pensamiento computacional, controlando las diferencias iniciales mediante la aplicación de un pretest.

Los estudiantes fuerón divididos en dos grupos: el primer grupo realizó actividades de solución de problemas de programación empleando el entorno de programación basado en bloques Scratch y el segundo grupo desarrolló las mismas actividades pero empleando el entorno de programación textual Visual Studio Code para la programción en lenguaje Python. Se aplicó como pre-test y pos-test la prueba de pensamiento computacional validada por Lafuente Martínez, Levéque, Benítez, Hardebolle, & Dehler Zufferey (2022).

4.1.1. Variables

Para alcanzar el objetivo de la investigación, se identificaron las siguientes variables con el fin de responder a la pregunta de investigación y evaluar la validez de las hipótesis propuestas.

La variable dependiente es el resultados del test de pensamiento computacional.

La variable independiente es el entorno de programación que tomará dos valores: basado en bloques y textual.

Tabla 1.

Variables y definiciones

Variable	Definición conceptual	Definición operacional	Dimensiones	Indicadores de la dimensión	Evaluación
Desarrollo del pensamiento computacional	Capacidad de los estudiantes para resolver problemas mediante el uso de habilidades algorítmicas y conceptos computacionales.	Medido a través del Test de Pensamiento Algorítmico, que evalúa habilidades como descomposición, abstracción, pensamiento algorítmico y evaluación de soluciones.	Habilidades de pensamiento computacional.	Puntuación total obtenida en el Test de Pensamiento Algorítmico y puntuaciones por dimensiones. Escala de tres puntos: "no aplica", "aplica parcialmente" y "aplica totalmente".	Aplicación del Test de Pensamiento Algorítmico como pretest y postest antes y después de la intervención.
Tipo de entorno de programación	Entorno de programación utilizado para enseñar conceptos de pensamiento computacional.	Los estudiantes se agruparán en dos categorías según el entorno utilizado: entorno basado en bloques (Scratch) y entorno textual (Python) Visual Studio Code.	Bloques y texto.	Tipo de entorno asignado a los estudiantes (basado en bloques o textual).	Clasificación de los estudiantes en uno de los dos grupos según el entorno utilizado.

4.1.2. Hipótesis

Las hipótesis del estudio son:

H1 – Existen diferencias significativas en el desarrollo de habilidades del pensamiento computacional cuando se realiza entrenamiento previo mediante entornos de programación basado en bloques y textuales.

H0 - No existen diferencias significativas en el desarrollo de habilidades del pensamiento computacional cuando se realiza entrenamiento previo mediante entornos de programación basado en bloques y textuales.

4.1.3. Población y Muestra

El estudio se realizó en el Colegio El Carmen Teresiano, ubicado en la ciudad de Bogotá, Distrito Capital en la localidad de Rafael Uribe Uribe, barrio Country Sur, en el centro oriente de la ciudad. Se conto con una participación de 131 estudiantes de los grados decimos y undécimos, cuyas edades oscilaban entre los 15 y 17 años, de los cuales el 23.7% de los estudiantes indicaron tener 15 años, el 45.8% indicaron tener 16 años, el 29% indicaron tener 17 años y el 1.5% indicaron tener 18 años, y se evidencio una desviación estándar de *SD* = .765.

Tabla 2.

Estadísticos descriptivos

Estadísticos descriptivos

	N	Mínimo	Máximo	Media	Desviación estándar	Asimetría	Curtosis
Edad	131	15	18	16.08	0.765	0.066	0.42

Con respecto al género, se obtuvo una participación de 80 mujeres (femenino) y 51 hombres (masculino), estos estudiantes pertenecen a un estrato socioeconómico 2 (bajo) y 3 (medio-bajo) según la estratificación de Colombia.

Tabla 3.

Genero de los estudiantes

Genero					
		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válido	F	80	61.1	61.1	61.1
	M	51	38.9	38.9	100
	Total	131	100	100	

4.2. Instrumentos de Recolección de Datos

Se empleó la prueba de pensamiento algorítmico para adultos (*Algorithmic Thinking Test for Adults*, diseñada por de Lafuente Martínez et al., (2022. La prueba se compone de 27 ítems de respuesta múltiple que se califican como correcto (1) o incorrecto (0). Esta prueba permite evaluar habilidades relacionadas con el pensamiento computacional, pensamiento algorítmico, descomposición, abstracción y reconocimiento de patrones.

Esta prueba fue validada por 11 expertos (seis mujeres, 5 hombres) en la materia de pensamiento computacional para categorizar los 27 ítems de los cuales 20 de ellos tienen

excelentes índices de confiabilidad que fueron seleccionados para la prueba. Los ítems se probaron en 289 participantes, 137 expertos y 152 novatos en pensamiento computacional. El coeficiente alfa de Cronbach fue de $a = .847$, lo que indica una buena confiabilidad del cuestionario.

Los expertos consideraron que la gran mayoría de los ítems evalúan en primer lugar ya sea pensamiento algorítmico (70% de los ítems), o contenidos algorítmicos (85% de los ítems). Desde una perspectiva de validez de contenido, esto apuntaba a la existencia de un elemento algorítmico en la mayoría de los ítems.

4.3. Procedimiento

Esta investigación se desarrolló en tres fases haciendo uso de un ambiente de aprendizaje en la plataforma Moodle con alojamiento gratuito (Mil aulas) en el cual se organizaron cada uno de los contenidos a trabajar. Por lo tanto, en la fase inicial se realiza una presentación del proyecto, aplicación del pretest de pensamiento computacional, en la segunda fase se realiza el entrenamiento de las actividades propuestas, para lo cual el grado decimo trabajo con el entorno de programación Scratch y el grado undécimo trabajo con el entorno de programación Visual Studio Code Python. Para la tercera fase se realiza la aplicación del postest usando el mismo cuestionario inicial.

4.3.1. Fase Inicial

En la fase inicial se realizó el diseño de las actividades a desarrollar para en el entorno de programación por bloques y en el entorno de programación textual teniendo como referente para

la elaboración de estas el marco conceptual de Brennan & Resnick (2012), y se elige el instrumento para la evaluación de Lafuente Martínez et al., (2022).

En esta investigación se identificó el marco teórico a seguir y el instrumento de evaluación del pensamiento computacional a partir de la indagación de las habilidades necesarias a desarrollar, es así, que se logra identificar diferentes investigaciones en las que se tiene en cuenta como tema de investigación el desarrollo del pensamiento computacional en entornos de programación gráfica, entornos de programación textual, e investigaciones en las que se hace uso de programación grafica-textual para identificar el impacto que pueden llegar a tener en estudiantes de bachillerato, a su vez se identifica el instrumento de evaluación de pensamiento computacional que haya sido validado y que muestre confiabilidad en el mismo.

En este rastreo de investigaciones realizadas, se logra evidenciar que el marco teórico de Brennan & Resnick (2012), muestra conceptos que permiten el desarrollo de la ensenañza del pensamiento computacional en estudiantes de bachillerato en donde se plantea siete conceptos (secuencias, blucles, eventos, paralelismo, condicionales, operadores y datos).

A partir de lo anterior, se desarrollan 11 actividades de las cuales se trabajan los conceptos de datos, condicionales y bucles, es asi que, 3 actividades van orientadas a datos, 4 actividades van orientas a estructuras selectivas (simple, doble) y 4 actividades van orientadas a ciclos (for, while).

Los conceptos de programacion que se utilizaron en cada una de las sesiones se tuvieron en cuenta para el diseño de las actividades para entornos de programación grafica y entornos de programacion textual, en donde se diseñaron unas guias que contenian una breve descripción del concepto a trabajar, un ejercicio práctico como ejemplo en el que se evidenciara el uso y ejecución del programa, y acontinuación, por cada uno de los conceptos se exponian los ejercicios a desarrollar.

Para la intervencion con los estudiantes se informa a las directivas del Colegio El Carmen Teresiano sobre la aplicación de las sesiones para los grados seleccionados (décimo y undecimo), se informa a los estudiantes y padres de familia o tutores, logrando la aprobación de todas las partes mecionadas anteriormente.

El pretest de pensamiento computacional, utilizando el instrumento de Lafuente Martínez et al., (2022), se aplicó durante la cuarta semana del mes de julio del año 2024, por medio de un cuestionario en Moodle, en este se incluyen cada una de las preguntas de la prueba de Pensamiento Algorítmico para adultos. Participaron 131 estudiantes, de los cuales 61 eran de grado décimo y 70 de grado undécimo.

4.3.2. Segunda Fase

La implementación de las actividades en el entorno de programación grafica (Scratch) y programación textual (Visual Studio Code Python), se desarrolló por 11 semanas, con una duración por sesión de 2 horas (120 minutos). En la tabla 4 se muestra las sesiones, el concepto trabajado y el concepto de acuerdo con el marco conceptual de Brennan & Resnick (2012).

Tabla 4.

Sesiones y conceptos trabajados

Sesión	Concepto trabajado	Concepto según Brennan & Resnick (2012)
1	Variables	Datos: implican almacenar, recuperar y actualizar valores.
2		
3		
4	Estructuras selectivas (simple, doble)	Condicionales: capacidad de tomar decisiones en función de ciertas condiciones, lo que permite la expresión de múltiples resultados.
5		
6		
7		
8		

9	Ciclos (for, while)	Bucles: son un mecanismo para ejecutar la misma secuencia varias veces.
10		
11		

Cada una de las sesiones se aplicaron durante el desarrollo de la clase de Tecnologia e Informática, en donde los estudiantes de grado decimo utilizaron el entorno de programación por bloques Scratch y los estudiantes de grado undecimo utilizaron el entorno de programación textual Visual Studio Code Python.

4.3.3. Tercera Fase

La aplicación del postest se realizó durante la tercera semana del mes de octubre del año 2024 y finalmente se procede al analisis y hallazgos de los resultados obtenidos en la aplicación del pretest y postest.

4.4. Descripción de Actividades Basadas en Texto y en Bloques Para el Desarrollo del Pensamiento Computacional

Para el desarrollo de las actividades se realizó por medio del alojamiento gratuito de Moodle (Mil aulas) un ambiente virtual de aprendizaje, en este se permitió el ingreso a cada uno de los estudiantes asignando un usuario y contraseña de manera individual. Se conto con 11 actividades para un total de 33 ejercicios en las cuales se muestra inicialmente un ejemplo de los ejercicios a realizar, teniendo en cuenta el concepto que se quería trabajar y el concepto según Brennan y Resnick (2012).

Se diseñaron dos guías de actividades una para el entorno de programación por bloques con Scratch y otra guía para el entorno de programación textual con Visual Studio Code Python,

las cuales se enfocan en fomentar el pensamiento computacional. En estas guías se fortalecen habilidades como el análisis lógico, la descomposición y la creación de algoritmos por medio de ejercicios prácticos que se enfocan en trabajar conceptos como son: variables, estructuras selectivas (simple, doble) y ciclos o estructuras repetitivas (for, while).

La guía de actividades que se diseñó para grado 10° se desarrolló en el entorno de programación por bloques con Scratch. En esta guía los estudiantes encontraron una explicación de conceptos que permitieran introducirlos en el tema a trabajar, para lo cual, se hace uso de estructuras básicas para el diseño de procesos y toma de decisiones, se presenta elementos de programación en Scratch, como lectura de datos, asignación de variables, resultados y uso de estructuras selectivas y repetitivas.

La guía de actividades que se diseñó para grado 11° se desarrolló en el entorno de programación textual Visual Studio Code Python. En esta guía los estudiantes encontraron una explicación de conceptos que permitieron introducirlos en el tema a trabajar, para lo cual, se hace uso de estructuras básicas para el diseño de procesos y toma de decisiones, se presenta elementos de programación en Visual Studio Code Python, como lectura de datos, asignación de variables, resultados y uso de estructuras selectivas y repetitivas mediante ejercicios cotidianos.

La guía fue dividida en 11 secciones denominadas “Actividades”, cada una de estas contó con uno o más ejercicios diseñados para abordar conceptos específicos de programación, estas actividades incluyen instrucciones claras y casos prácticos como se describe a continuación y como se puede apreciar en el Anexo A. Actividades para el entorno de programación grafica Scratch y Anexo B. Actividades para el entorno de programación textual Visual Studio Code Python.

Inicialmente en la guía se presenta una introducción a los conceptos fundamentales de los diagramas de flujo y algoritmos por medio de un ejemplo, en esta se establecen las bases teóricas para el desarrollo de las actividades, se realiza una explicación de un diagrama de flujo según el concepto y gráficamente como se debe desarrollar según los símbolos estandarizados en los cuales se debe seguir una serie de pasos ordenados para resolver un problema o ejecutar una tarea. Asimismo, se define el concepto de algoritmo y se incluyen elementos clave como la estructura básica de entrada y salida de datos de un programa, mencionando la lectura de datos y el almacenamiento del resultado en una variable y la escritura de este valor en la salida. Adicionalmente se muestra el desarrollo de un ejercicio por medio de un diagrama de flujo y como se desarrolla en Scratch y Visual Studio Code según corresponda.

Las actividades No. 1, No. 2 y No.3 fueron una introducción a estructuras básicas en programación las cuales incluyeron el diseño de programas que incluían cálculos matemáticos simples. Para ello en la Actividad No.1 se abordó el cálculo de impuestos, salarios, conversión de semanas/días, identificación de altura mínima y máxima. En la actividad No. 2 se abordaron ejercicios de conversión de dinero o moneda, cálculo del Índice de Masa Corporal (IMC), velocidad promedio y operaciones geométricas de área y perímetro en un rectángulo. En la actividad No. 3 se calcula los intereses en capital, se calcula promedios y se clasifican datos de acuerdo a una categorización dada.

A continuación se presenta un ejemplo en el que se muestra los conceptos de estructuras algorítmicas selectivas utilizadas en la toma de decisiones lógicas en programación, en el ejemplo se realiza una explicación de las estructuras condicionales “Si entonces” (estructura selectiva simple) y “Si entonces/Sino” (estructura selectiva doble), teniendo en cuenta que estas permiten evaluar condicionales y ejecutar acciones en función de si se cumplen o no ciertos criterios

específicos según la complejidad de cada ejercicio, para lo cual se presentan dos ejercicios prácticos: el primero corresponde a un ejemplo de estructura simple y el segundo corresponde a una estructura selectiva doble.

Las actividades No. 4, No. 5, No. 6, No. 7, No. 8 y No. 9, se trabajó el uso de estructuras condicionales en programación, en estas se abordan ejercicios más complejos aplicando lógica algorítmica en diferentes escenarios. Por lo tanto, la actividad No. 4, se desarrolla ejercicios como clasificación de temperatura en fría, caliente o agradable, determinación de si un año es bisiesto, verificación de la edad e identificación de si un número es primo. La actividad No. 5, contiene ejercicios como conversión de temperatura, cálculo de bonificaciones y clasificación de un número como positivo, negativo o cero. La actividad No. 6 se categorizan datos según la temperatura si es fría, templada o caliente y categoriza la edad de una persona. La actividad No. 7, cálculo de datos según el peso y costo de envió de un paquete, cálculo de números negativos según un número determinado, suma y promedio de números positivos y negativos. La actividad No. 8, identifica números múltiplos de 3, promedio de múltiplos y determina el dinero de una persona para comprar un artículo. Y la actividad No. 9, introduce un sistema de descuentos basados en el tipo de membresía y el monto de compra del usuario.

A continuación, en la guía se presenta un ejemplo práctico en el que se hace uso de estructuras repetitivas como son “repetir” (For) y “mientras” (While), en estos ejemplos se realiza la explicación de los bucles For y While, en donde el bucle For tiene un numero de iteraciones que se define previamente y el bucle While se usa cuando la cantidad de repeticiones depende de una condición que se evalúa cuando no sabemos cuántas veces se debe repetir.

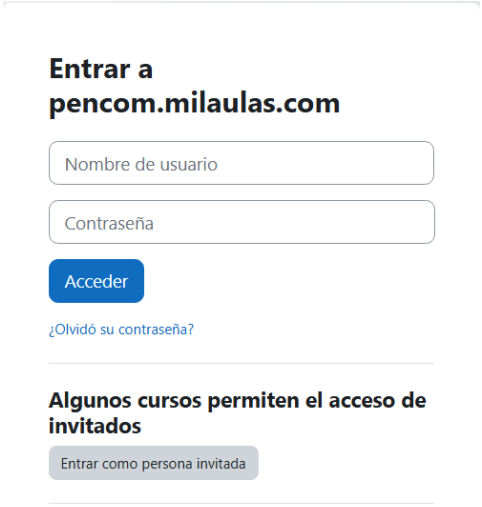
Las actividades No. 10 y No. 11 de la guía se trabajó el uso de bucles como son For y While, para lo cual en la actividad No. 10, los ejercicios calculan la factorial de un número, indican cuantas vocales tiene un texto, muestran la tabla de multiplicar de un numero indicado y cuenta números inversamente. Y en la actividad No. 11, calcula los sueldos de los empleados según el sueldo, las horas extras, sueldo por hora y la nómina según unas indicaciones especificas dadas.

4.4.1. Aspectos generales de ingreso al entorno de aprendizaje

Para el ingreso al aula virtual los estudiantes se dirigieron al siguiente enlace: https://pencom.milaulas.com/login/index.php, a continuación de ello, agregaban las respectivas credenciales como se muestra en la Figura 1.

Figura 1.

Ingreso al ambiente virtual de aprendizaje



A continuación, los estudiantes al ingresar al curso encontraban cuatro (4) secciones denominadas: Test, Variables, Estructuras Selectivas y For/While, cada una de estas con sus respectivas actividades como se muestra en la Figura 2.

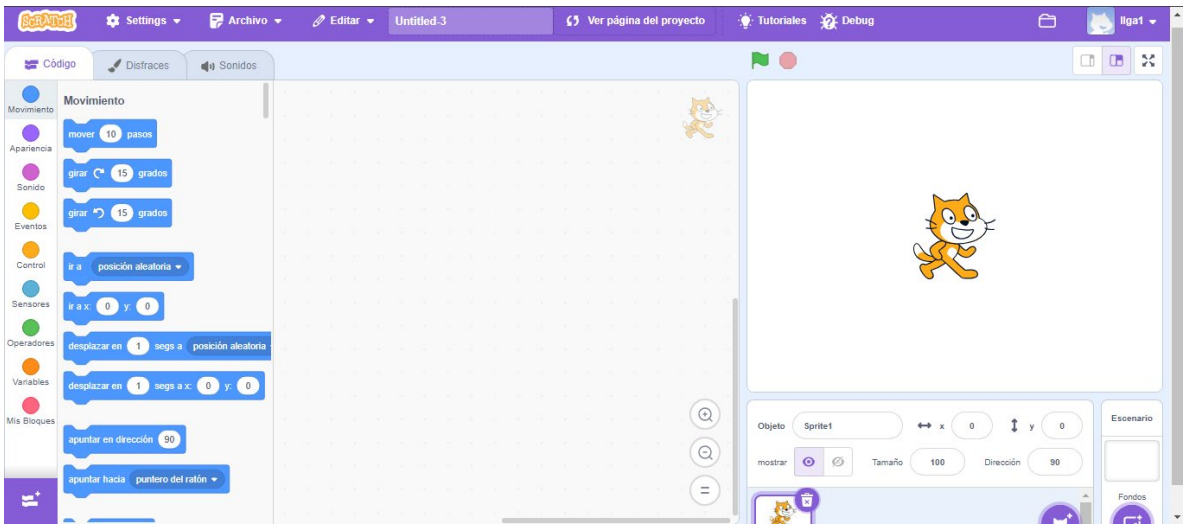
Figura 2.

Secciones del ambiente virtual

Para el desarrollo de cada una de las actividades los estudiantes realizaron un reconocimiento del entorno de programación correspondiente según el grado en el que se encontraban, por lo tanto, grado decimo trabajo en el entorno de programación por bloques (Scratch) y grado undécimo trabajo en el entorno de programación textual (Visual Studio Code Python). En el reconocimiento del entorno de programación por bloques se invitó a los estudiantes a registrarse en el entorno de programación a utilizar (https://scratch.mit.edu/) con su respectivo correo institucional, esto con el fin de que cada uno pudiera acceder y guardar los ejercicios creados sin posibilidad de perdida, así mismo, se realizó un recorrido indicando las partes principales como se muestra en la Figura 3.

Figura 3.

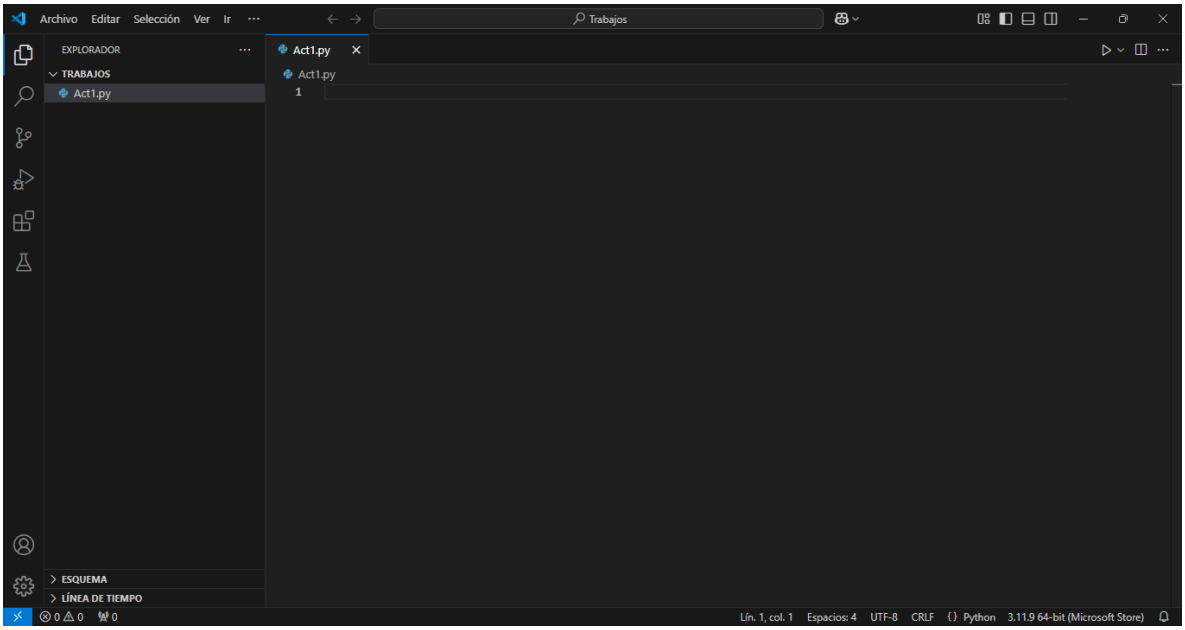
Entorno de programación Scratch



Así mismo, se realiza el reconocimiento del entorno de programación textual en Visual Studio Code Python indicando a los estudiantes como descargar el programa, instalación, extensiones a instalar y creación de carpetas y archivos para la realización de los ejercicios como se muestra en la Figura 4.

Figura 4.

Entorno de programación Visual Studio Code Python



Teniendo en cuenta la Figura 2, los estudiantes iniciaron desarrollando el pretest de Pensamiento Computacional, y secuencialmente desarrollaron cada una de las secciones, finalizando con el postest.

5. Análisis e Interpretación de Resultados

El análisis de datos para esta investigación se llevó a cabo utilizando el Statistical Package for the Social Science- IBM versión 23, software al cual en adelante nos referiremos a SPSS. Cada grupo de respuestas de los 131 estudiantes de los dos grupos que presentaron el pretest y postest se consolidó en una matriz única para realizar el tratamiento estadístico de los datos, con el fin de determinar el efecto de las actividades en entorno de programación gráfica y entorno de programación textual sobre el desarrollo del pensamiento computacional.

En el analisis se tuvieron en cuenta los siguientes parametros: validación de la prueba de pensamiento computacional por medio de la pruena alfa de Cronbach, el nivel de significancia del 5%, verificacion de los supuestos de normalidad, homogeneidad y correlaciones entre las variables. En los siguientes apratados se presentan los análisis en detalle.

5.1.Análisis de las condiciones iniciales

A continuación, se presenta los resultados del pretest de pensamiento computacional con una escala de evaluación de 0 a 5 como se muestra en la tabla 5.

Tabla 5.

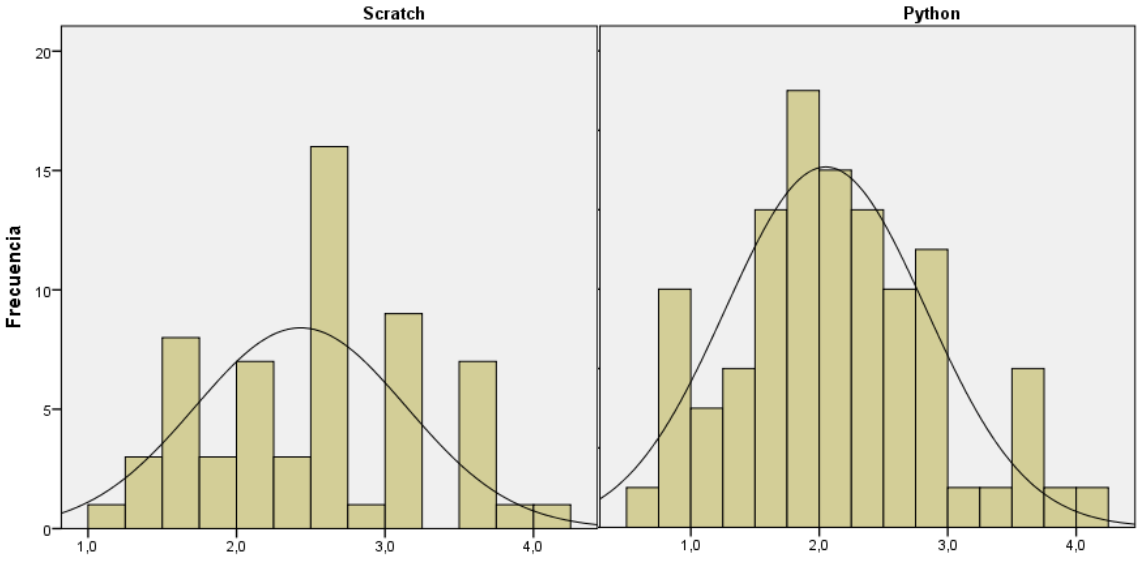
Estadísticos descriptivos del pretest de pensamiento computacional

Actividades	N	Media	Desviación estándar	Mínimo	Máximo	Rango
Scratch	60	2,432	,7116	1,2	4,0	2,8
Visual Studio Code Python	71	2,051	,7800	,6	4,0	3,4

Se puede observar que el grupo que realizo actividades en el entorno de programación por bloques (Scratch), obtuvo un mejor desempeño en la prueba pretest de pensamiento computacional (M=2,43, SD=0,71) mientras que el grupo que realizo actividades en el entorno de programación textual (Visual Studio Code Python), obtuvo un desempeño más bajo (M=2,05), SD= 0,78).

Figura 5.

Resultados del pretest: actividades de programación gráfica y textual



5.1.1. Efectos sobre la variable dependiente (pos-test)

A continuación, se presenta los resultados del postest de pensamiento computacional aplicado con una escala de evaluación de 0 a 5 como se muestra en la tabla 6.

Tabla 6.

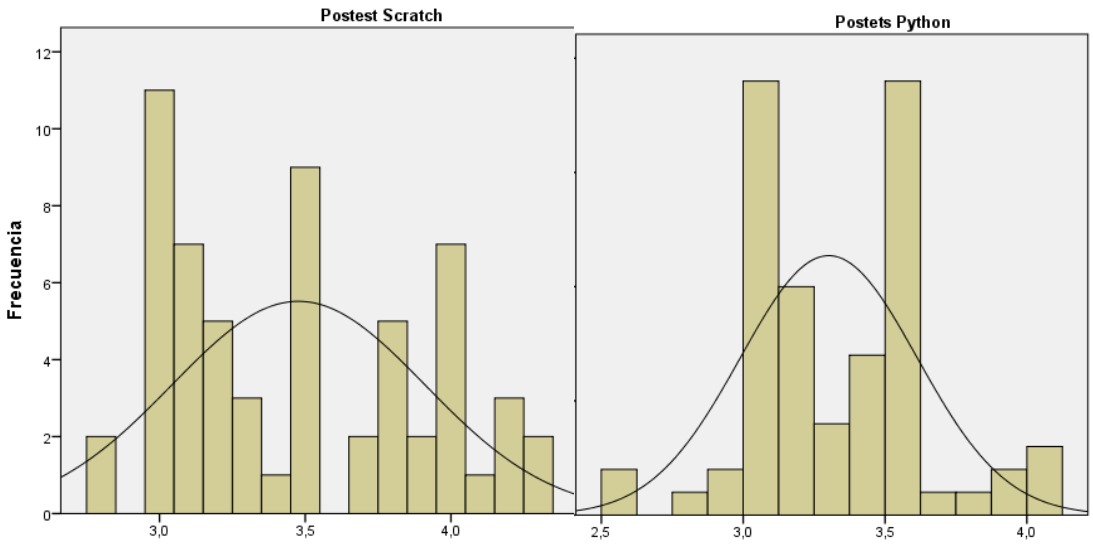
Estadísticos descriptivos del postest de pensamiento computacional

Actividades	N	Media	Desviación estándar	Mínimo	Máximo	Rango
Scratch	60	3,477	,4339	2,8	4,3	1,5
Visual Studio Code Python	71	3,302	,3117	2,5	4,0	1,5

Los resultados del postest muestran que el grupo que trabajó actividades en el entorno de programación por bloques (Scratch) obtuvo una media mayor (M = 3.477, SD = 0.4339), con puntuaciones que oscilaron entre 2.8 y 4.3. Por su parte, el grupo que trabajó con el entorno de programación textual (Visual Studio Code Python) alcanzó una media ligeramente menor (M = 3.302, SD = 0.3117), con puntuaciones entre 2.5 y 4.0. La dispersión de los datos fue menor en el grupo de Visual Studio Code Python en comparación con el grupo de Scratch.

Figura 6.

Comparativo de los resultados del postets



5.1.2. Comparación del rendimiento en los grupos

Tabla 7.

Estadísticos descriptivos del pretest y postest según el entorno de programación

Estadísticos descriptivos					
	N	Mínimo	Máximo	Media	Desviación estándar
PRETEST (Scratch)	131	1.2	4.0	2.432	.7116
PRETEST (Visual Studio Code Python)	131	.6	4.0	2.051	.7800
POSTEST (Scratch)	131	2.8	4.3	3.477	.4339
POSTEST (Visual Studio Code Python)	131	2.5	4.0	3.302	.3117

La comparación entre los datos del pretest y el postest muestra una mejora en ambos grupos. El grupo de Scratch incrementó su media en 1.045 puntos (de 2.432 a 3.477), mientras que el grupo de Visual Studio Code Python mostró un avance mayor de 1.251 puntos (de 2.051 a 3.302). Además, las desviaciones estándar se redujeron en ambos grupos, lo que sugiere que los conocimientos y habilidades del grupo se volvieron más homogéneos tras el entrenamiento con las actividades para el desarrollo de conceptos de pensamiento computacional. Así, el grupo que trabajó con el entorno de programación textual en Visual Studio Code Python mostró un mayor margen de mejora a pesar de partir de puntuaciones iniciales más bajas. Por su parte, el grupo de Scratch mantuvo una ventaja consistente en las puntuaciones medias tanto al inicio como al final de la intervención, aunque con una mejora relativa menor.

El rango de puntuaciones en el postest fue idéntico para ambos grupos (1.5 puntos), aunque situado en diferentes intervalos: entre 2.8 y 4.3 para Scratch, y entre 2.5 y 4.0 para Visual Studio Code Python. Esto sugiere que, si bien el grupo de Scratch mantuvo puntuaciones ligeramente más altas, ambos entornos de programación contribuyeron de manera uniforme a las mejoras en el test de pensamiento computacional.

Estos datos sugieren que ambos entornos de programación son efectivos para el desarrollo de habilidades de pensamiento computacional. La homogeneización del grupo, reflejada en la reducción de las desviaciones estándar, sugiere que ambos entornos contribuyen a nivelar los conocimientos de los estudiantes, independientemente de sus puntos de partida.

5.2. Análisis de covarianza

Para determinar si las diferencias observadas son estadísticamente significativas, se llevó a cabo un análisis de covarianza para evaluar la efectividad de los dos entornos de programación (Scratch y Visual Studio Code Python) controlando las diferencias iniciales entre los grupos utilizando el pretest como covariable.

Para aplicar el análisis de covarianza de manera válida, los datos deben cumplir varios supuestos estadísticos. En primer lugar, los residuos del modelo deben seguir una distribución normal, lo cual se puede verificar mediante pruebas como Shapiro-Wilk o gráficos Q-Q. Las varianzas de los grupos deben ser homogéneas (homocedasticidad), verificable mediante la prueba de Levene. También es esencial la independencia de las observaciones, lo que significa que los datos de un participante no deben estar relacionados con los de otros, y debe existir una relación lineal entre la covariable (pretest) y la variable dependiente (postest) para cada grupo, verificable mediante gráficos de dispersión. Además, las pendientes de regresión deben ser

homogéneas entre los grupos, lo que implica que la relación entre la covariable y la variable dependiente debe ser similar en todos ellos, y la covariable debe medirse con la menor cantidad de error posible y ser fiable.

5.2.1. Verificación de supuestos

En cuanto a las estadísticas descriptivas de los residuos, la media es prácticamente cero (0.0000), lo que indica que el modelo está bien ajustado y que los residuos están equilibrados alrededor de la media. El intervalo de confianza del 95% para la media oscila entre -0.0620 y 0.0620, lo que sugiere una baja variabilidad promedio. Además, la mediana de los residuos (-0.0782) está cerca de la media, lo que es consistente con una distribución simétrica. Los indicadores de forma, como la asimetría (0.194) y la curtosis (-0.482), muestran que la distribución de los residuos es ligeramente asimétrica hacia valores positivos y algo más plana que la normal, pero dentro de márgenes aceptables. La dispersión de los residuos es moderada, con una varianza de 0.129 y un rango de 1.70.

Tabla 8.

Descriptivos

Descriptivos				
			Estadístico	**Desv. Error**
Residuo para POSTEST	**Media**		0	0.03135
	95% de intervalo de confianza para la media	Límite inferior	-0.062	
		Límite superior	0.062	
	Media recortada al 5%		-0.0041	
	Mediana		-0.0782	
	Varianza		0.129	
	Desv. Desviación		0.35883	

	Mínimo	-0.85	
	Máximo	0.84	
	Rango	1.7	
	Rango intercuartil	0.5	
	Asimetría	0.194	0.212
	Curtosis	-0.482	0.42

Respecto a las pruebas de normalidad, los resultados son mixtos. El test de Kolmogorov-Smirnov (KS) muestra un resultado significativo (p = .008), indicando una ligera desviación de la normalidad. Sin embargo, el test de Shapiro-Wilk (SW), considerado más adecuado para muestras pequeñas o moderadas, no encuentra evidencia significativa de desviación de la normalidad (p = .206). En conjunto, esto sugiere que la distribución de los residuos puede ser asumida como aproximadamente normal, lo cual es un requisito clave para el análisis univariante de varianza.

En la verificación de la homogeneidad de varianzas u homocedasticidad, se utilizó la prueba de Levene, es así que, los valores de significancia menores a 0,05 se consideran críticos y se rechaza la hipótesis indicando que no existen diferencias significativas en el desarrollo de habilidades del pensamiento computacional cuando se realiza entrenamiento previo mediante entornos de programación basado en bloques y textuales, por lo cual consideramos que cumplen con el supuesto de homogeneidad de varianzas.

Tabla 9.

Prueba de Homogeneidad de varianzas

	Estadístico de Levene	**gl1**	**gl2**	**Sig.**
Postest	12.379	1	129	,001

Tabla 10.

Pruebas de normalidad

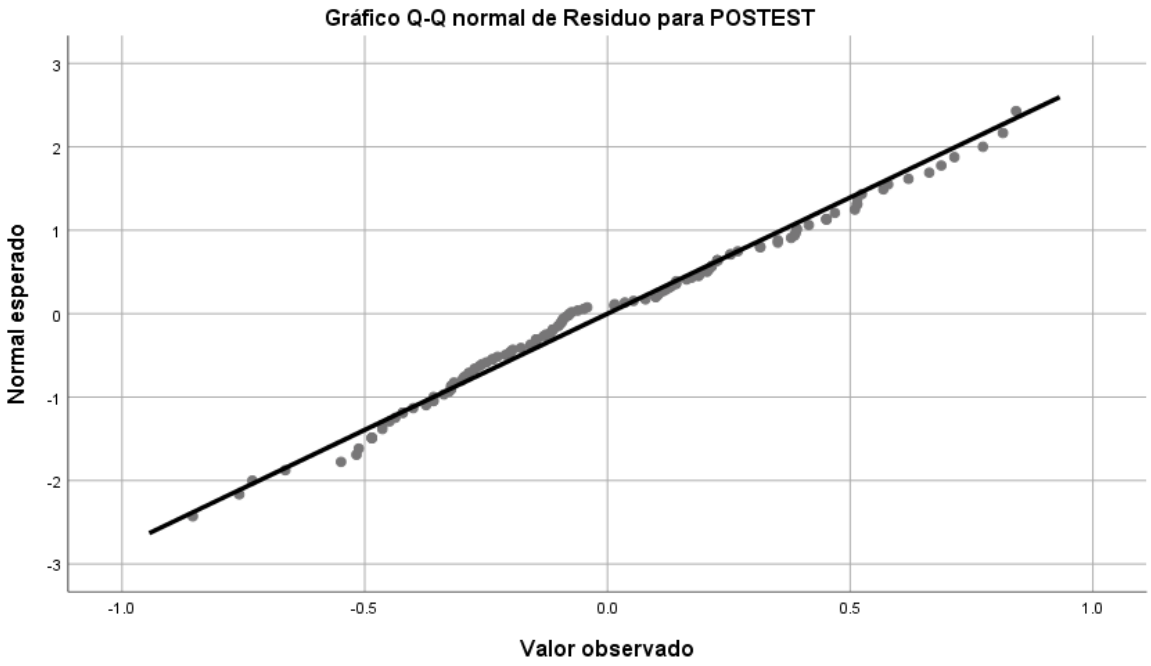
Pruebas de normalidad						
	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Estadístico	Gl	Sig.	Estadístico	gl	Sig.
Residuo para POSTEST	0.093	131	0.008	0.986	131	0.206
a. Corrección de significación de Lilliefors						

Los residuos del modelo están equilibrados y cumplen razonablemente con los supuestos de normalidad. Esto refuerza la validez de los resultados previos y permite confiar en las inferencias obtenidas del análisis.

El primer gráfico Q-Q muestra que la mayoría de los puntos se alinean bien con la línea diagonal, lo que sugiere que los residuos siguen una distribución aproximadamente normal. Sin embargo, hay ligeras desviaciones en los extremos (cola izquierda y derecha), indicando posibles valores atípicos o una distribución menos ajustada en los extremos.

Figura 7.

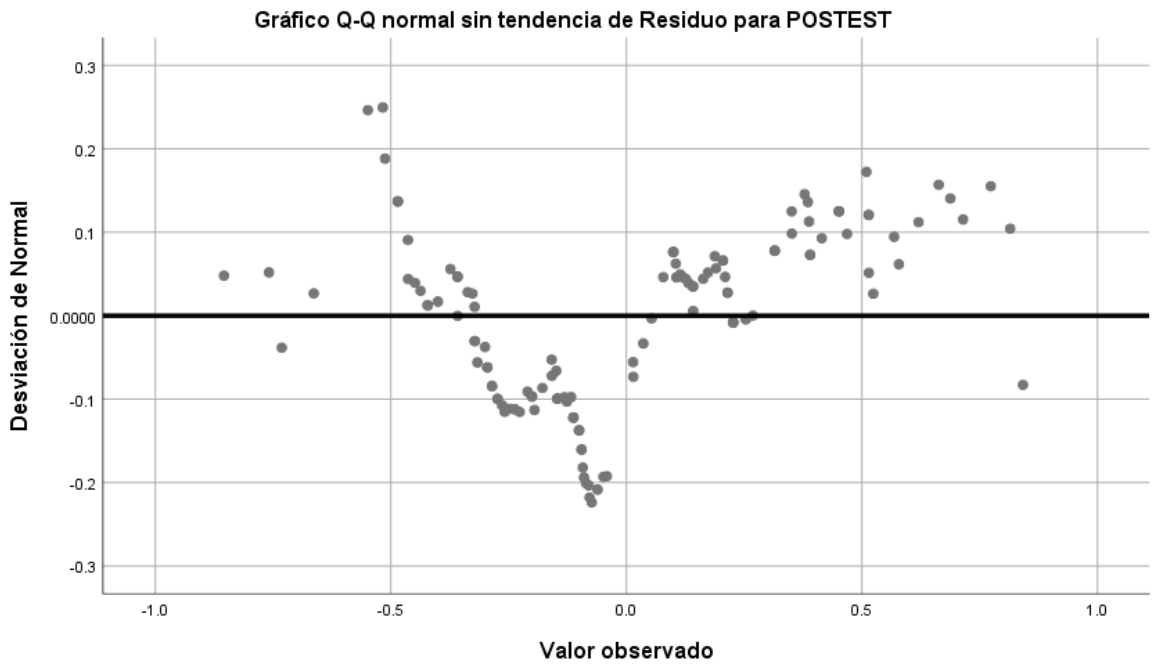
Gráfico Q-Q normal de Residuo para POSTEST



El segundo gráfico Q-Q sin tendencia refuerza esta observación: aunque la mayoría de los puntos se agrupan cerca de cero, hay algunas desviaciones leves hacia los extremos. Estas desviaciones no parecen ser severas y son consistentes con los resultados de las pruebas de normalidad previamente discutidas.

Figura 8.

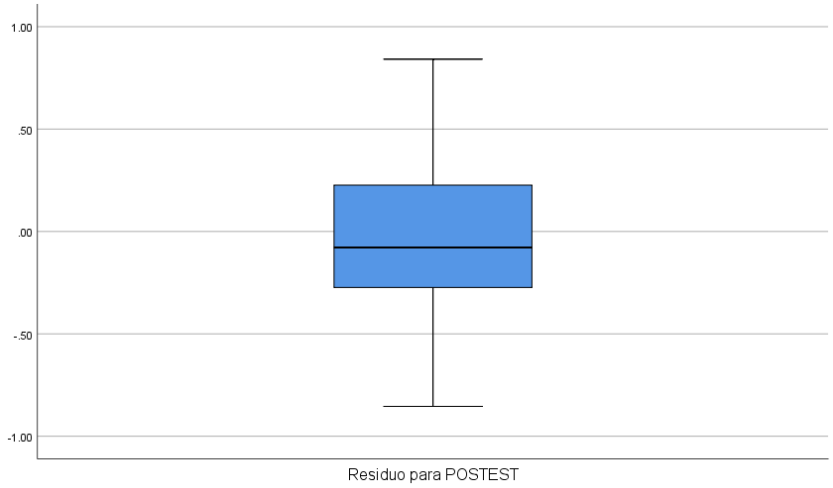
Gráfico Q-Q normal sin tendencia de Residuo para POSTEST



El diagrama de caja muestra que los residuos están distribuidos mayormente dentro de los límites del rango intercuartil. No se observan valores atípicos extremos, lo que sugiere que los residuos son razonablemente homogéneos y están dentro de un rango aceptable. La mediana está cerca de cero, lo que confirma la simetría observada en los descriptivos y la normalidad relativa.

Figura 9.

Diagrama de Caja y Bigotes



El análisis gráfico y descriptivo de los residuos indica que, en general, cumplen con el supuesto de normalidad, con una ligera desviación en los extremos que no parece afectar de manera crítica los resultados del modelo. La distribución es razonablemente homogénea y simétrica, y no se identificaron valores atípicos significativos. Estos resultados validan la interpretación previa del modelo ANOVA univariado.

5.2.2. Análisis univariado de varianza

El análisis univariado de varianza muestra que el modelo general es significativo ($F = 8.285$, $p = .000$), explicando un 10.1% de la varianza en los resultados del postest, $F(2, 128) = 8.285$, $p < .001$, R^2 ajustado = .101. Esto indica que las variables pretest y entorno de programación contribuyen de manera conjunta a explicar los resultados posteriores, aunque la varianza explicada es moderada.

Tabla 11.

Pruebas de efectos inter-sujetos

Pruebas de efectos inter-sujetos					
Variable dependiente: POSTEST					
Origen	**Tipo III de suma de cuadrados**	**gl**	**Media cuadrática**	**F**	**Sig.**
Modelo corregido	2.167[a]	2	1.084	8.285	.000
Intersección	125.144	1	125.144	956.957	.000
Pretest	1.170	1	1.170	8.946	.003
Entorno de programación	.490	1	.490	3.745	.055
Error	16.739	128	.131		
Total	1517.061	131			
Total corregido	18.906	130			

a. R al cuadrado = .115 (R al cuadrado ajustada = .101)

Las puntuaciones del pretest mostraron una influencia significativa en los resultados finales, (F= 8.946, p = .003), lo que sugiere que el rendimiento inicial influye de manera importante en el rendimiento final (postest) y confirmando la importancia de controlar las diferencias iniciales.

Por otro lado, el factor entorno de aprendizaje muestra un efecto no significativo (F= 3.745, p = .055), lo que implica que no hay diferencias claras entre los entornos de aprendizaje en los resultados del postest después de controlar por el pretest.

Estos resultados sugieren que ambos entornos de programación (Scratch y Visual Studio Code Python) son comparablemente efectivos para desarrollar el pensamiento computacional. Así, el rendimiento previo es un predictor clave del rendimiento final, mientras que las diferencias entre grados no son suficientemente fuertes para ser consideradas significativas en este análisis.

El bajo porcentaje de varianza explicada (10.1%) indica la probable existencia de otros factores importantes no incluidos en este análisis que influyen en el desarrollo del pensamiento computacional. Sobre este punto es importante señalar que en este estudio el entorno de programación está vinculado al grado escolar (Scratch en décimo y Visual Studio Code Python en undécimo), lo que podría introducir una variable de confusión en los resultados. Futuros estudios podrían beneficiarse de un diseño que separe estos factores para evaluar independientemente el efecto del entorno de programación y el nivel académico.

Con base en el análisis realizado, no se puede afirmar que existan diferencias significativas atribuibles únicamente al entorno de programación en el rendimiento del postest. En el análisis de efectos inter-sujetos, el efecto principal del entorno de programación no fue significativo ($p = .055$), está muy cerca pero no alcanza el umbral convencional de $p < .05$, lo que sugiere que, al considerar únicamente esta variable, las diferencias en el postest entre los estudiantes que utilizaron distintos entornos de programación no son suficientemente grandes como para ser estadísticamente significativas. Sin embargo, al incluir la interacción entre el entorno de programación y el pretest, esta interacción resultó ser significativa ($p < .001$). Esto implica que la relación entre el pretest y el postest varía en función del entorno de programación, es decir, este modera la influencia del rendimiento inicial sobre el rendimiento posterior.

Así, aunque el entorno de programación por sí solo no parece tener un efecto significativo, sí existen diferencias significativas en cómo el rendimiento inicial (pretest) influye en el rendimiento final (postest), dependiendo del entorno utilizado. Esto sugiere que las estrategias pedagógicas o las intervenciones deberían considerar tanto el entorno de programación como el nivel de rendimiento inicial de los estudiantes en la prueba de pensamiento computacional, ya que su interacción tiene un impacto importante en los resultados.

5.3. Análisis complementarios

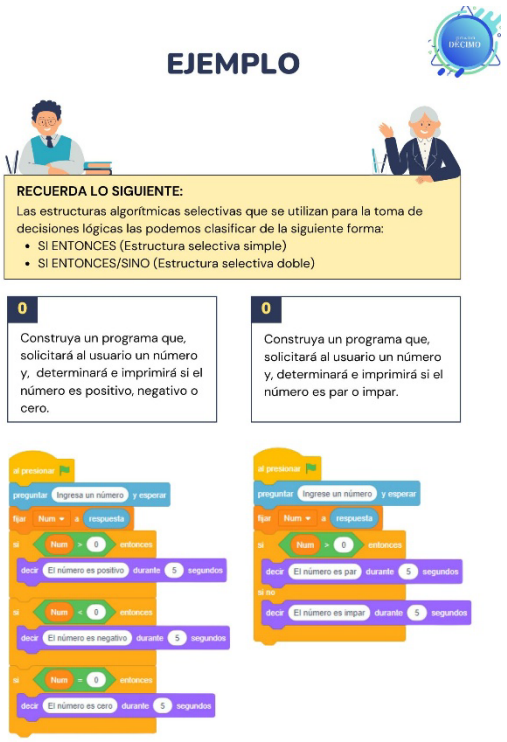
El análisis de las actividades desarrolladas en el entorno de programación por bloques Scratch y en el entorno de programación textual Visual Studio Code Python permitieron interpretar los resultados estadísticos obtenidos, identificar ventajas y limitaciones en el desarrollo del pensamiento computacional. En este apartado muestra la naturaleza de las actividades y las características de cada uno de los entornos que pudieron influir en el rendimiento de los estudiantes, considerando algunos factores como la facilidad de uso, comprensión de conceptos y resolución de problemas. Así mismo, se analiza los desafíos a los que se pudieron enfrentar los estudiantes como son sintaxis en los entornos de programación trabajados e interactividad, esto con el fin de comprender el impacto en la adquisición de habilidades de programación y pensamiento computacional.

5.3.1. Análisis de las actividades de solución de problemas en el entorno de programación por bloques (Scratch)

Durante el desarrollo de cada una de las sesiones los estudiantes encontraban un ejemplo introductorio con el objetivo de orientar el desarrollo de cada una de las actividades propuestas y motivar en su aprendizaje, en dicho ejemplo se mostraba información para recordar, un ejercicio práctico con su respectiva solución (Figura 5). A su vez, se indicó a los estudiantes que la entrega de las actividades se realizaría por medio del ambiente virtual (Moodle) a partir de la publicación del archivo (para el caso de Scratch), imágenes, carpetas y archivos que pudieran evidenciar el desarrollo de estas, teniendo en cuenta que cualquiera de las anteriores opciones seria valida siempre y cuando evidenciara todo el trabajo realizado.

Figura 10.

Ejemplo de actividad en Scratch



Durante el desarrollo de las clases se realizaba la explicación del tema con su respectivo ejemplo, y se logró evidenciar que algunos estudiantes eran más agiles que otros en el desarrollo de las mismas en cuanto a la sintaxis. Otros indicaron de manera verbal que los temas eran complejos teniendo en cuenta que su experiencia en programación era nula o poca.

En la Figura 6 se puede apreciar la solución de uno de los ejercicios sobre variables en el entorno de programación por bloques (Scratch) por dos estudiantes, uno de ellos logro desarrollar con éxito y el otro obtuvo dificultades que no permitieron la solución del mismo.

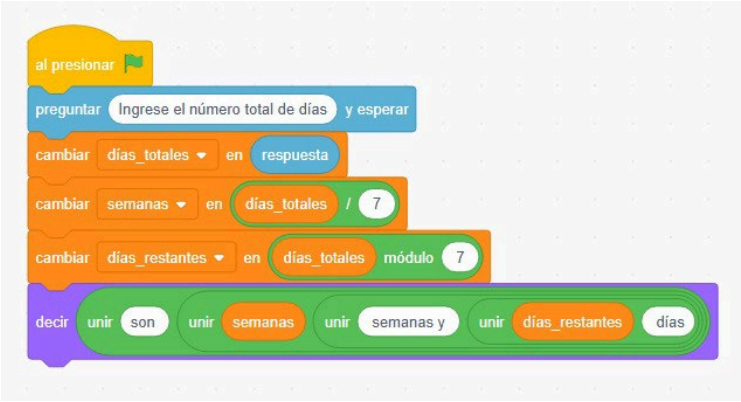
Figura 11.

Ejercicio No. 2 - Variables

Uno de los errores comunes que se evidenciaron por parte de los estudiantes en el desarrollo de los ejercicios a medida que iban avanzando fue el reconocimiento de los bloques que se generaban en la creación de variables debido a que como se muestra en la Figura 12, confundían el bloque de fijar con el bloque de cambiar y esto causo alteraciones en los resultados del ejercicio propuesto debido a que dicho bloque cambia la variable especificada en una cantidad determinada. Esto debido a que dentro de las configuraciones de Scratch se encuentra la opción de cambio de idioma, para lo cual, el idioma español tiene dos opciones: español (España) y español latinoamericano, los cuales tienen traducciones diferentes que causo confusión en los estudiantes.

Figura 12.

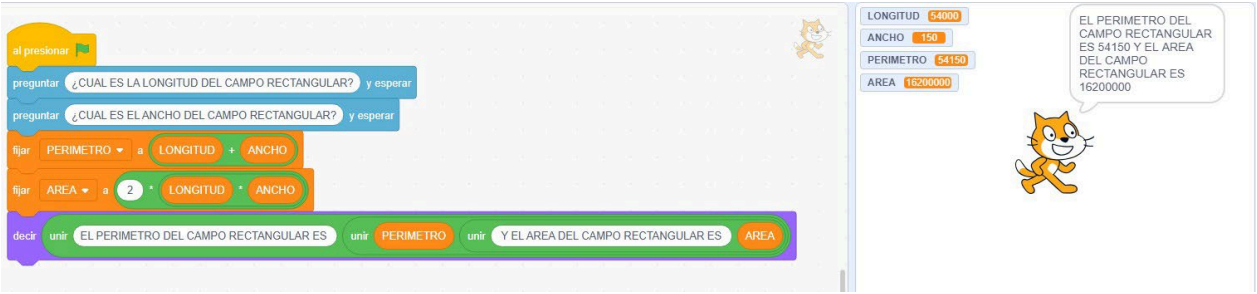
Ejercicio No. 3 - Variables



En el desarrollo de las actividades se pudo evidenciar la confusión en el desarrollo de fórmulas matemáticas que causaron errores en los resultados de los ejercicios propuestos como se evidencia en la Figura 13, donde se desarrollaron de manera incorrecta las fórmulas que no permitieron el óptimo desarrollo del ejercicio y mostraron resultados incorrectos según los datos ingresados para la ejecución del ejercicio.

Figura 13.

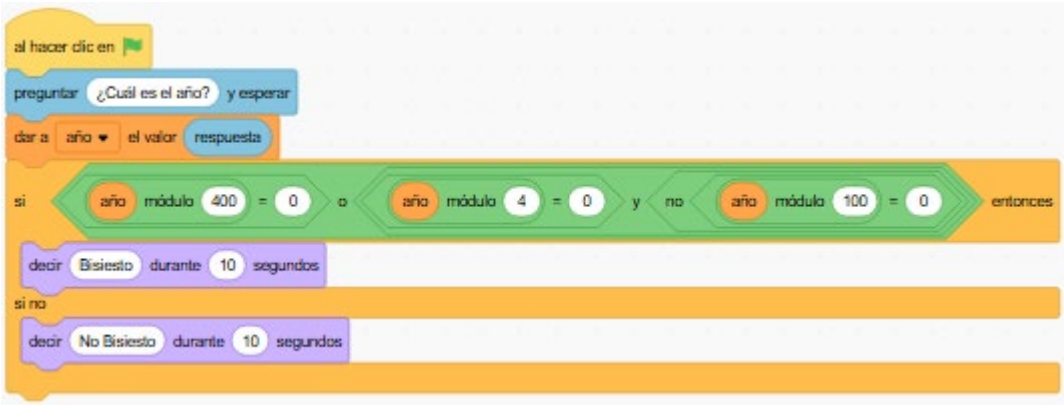
Ejercicio No. 8 - Variables



Los estudiantes a medida que iban desarrollando las actividades reconocieron nuevas funciones que permitieron la solución de los mismos como se muestra en la Figura 14, así mismo, se fueron enfrentando a nuevos desafíos en los que era importante la concentración y los pilares del pensamiento computacional para lograr identificar que se pedía en cada ejercicio logrando mejores resultados en el desarrollo de los mismos.

Figura 14.

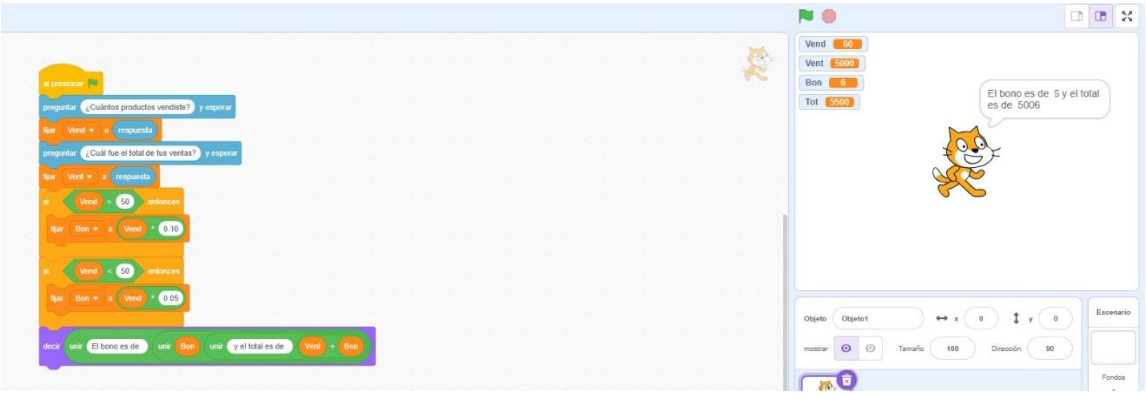
Ejercicio No. 13 - Estructuras selectivas



Sin embargo, se presentaron errores en cuanto al uso de las variables por la similitud en la denominación del nombre de estas, provocando el uso erróneo de las mismas debido a que algunos estudiantes agregaban nombres cortos pero que podían no ser tan claros para ellos mismos y provoco que el uso de estas se viera afectado en el desarrollo de los ejercicios como se muestra en la Figura 15.

Figura 15.

Ejercicio No. 17 - Estructuras selectivas

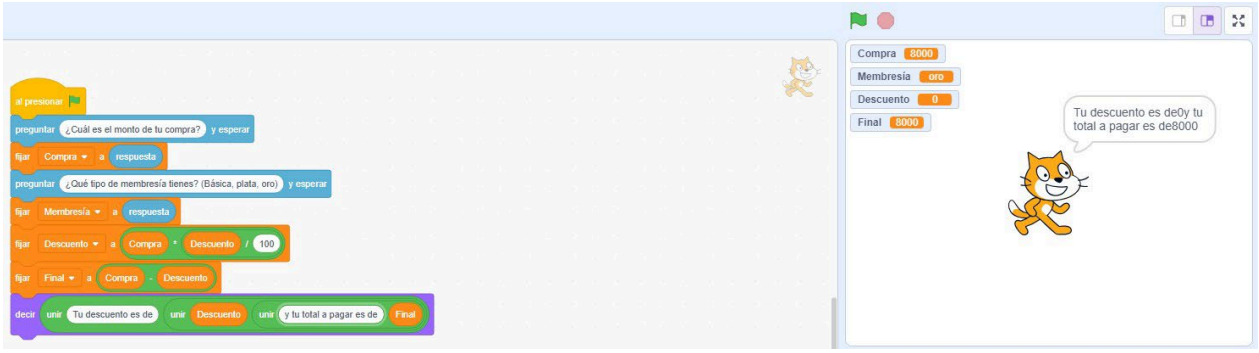


Debido a la extensión de algunos ejercicios se evidencio que algunos estudiantes no desarrollaron los ejercicios en su totalidad debido a la complejidad de estos y según lo que indicaron se mostraron inseguros en el desarrollo de los mismos debido a que no comprendieron

si el uso de condicionales tenía un límite o si por el contrario se podía hacer uso de los bloques condicionales según lo requiriera el ejercicio como se muestra en la Figura 16. Es así, como a partir de ello, surgen dudas en los estudiantes que no comprendieron el ejercicio y estas se van resolviendo para el óptimo desarrollo de ejercicios futuros.

Figura 16.

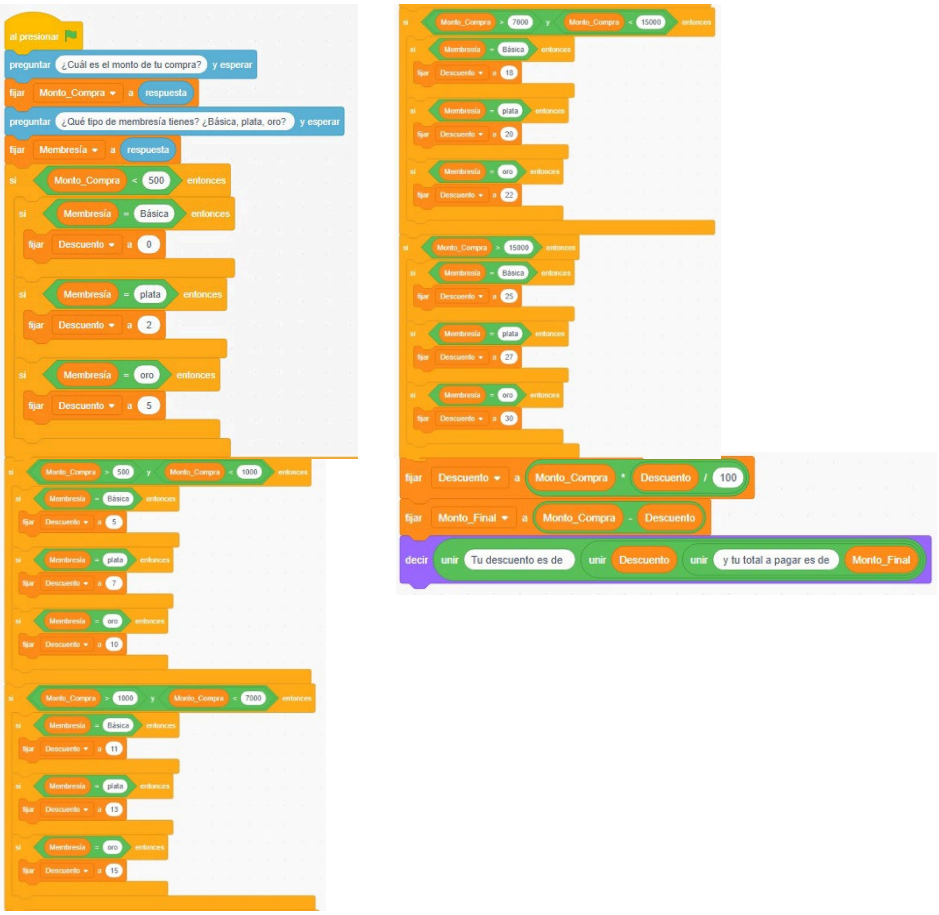
Ejercicio No. 27 - Estructuras selectivas - incompleto



Sin embargo, en su mayoría se evidencio que los estudiantes a medida que avanzaban en el desarrollo de los ejercicios y con la guía como material de apoyo lograron descomponer el mismo ejercicio planteado anteriormente dividiendo el problema en partes pequeñas e identificando patrones que permitieron enfocarse en la información que era relevante y llegar al desarrollo del mismo como se muestra en la Figura 17.

Figura 17.

Ejercicio No. 27 - Estructuras selectivas - terminado

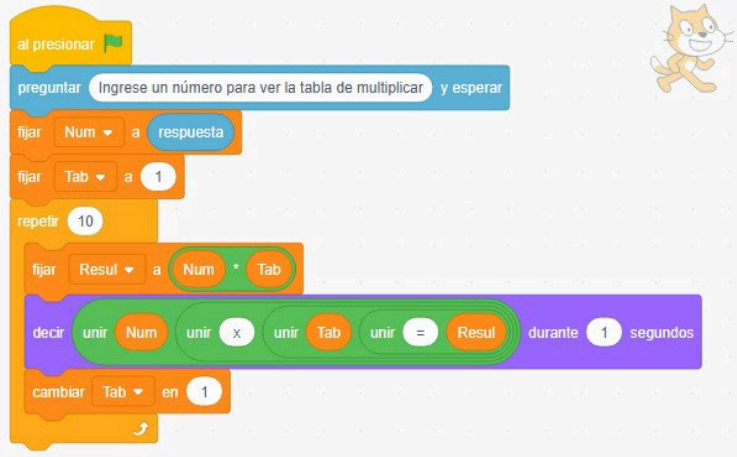


Los ejercicios planteados en las guías motivaron a los estudiantes debido a que ellos indicaban que muchos de estos parten de lo cotidiano que se puede vivir en el día a día de una persona y que inconscientemente nuestras vidas giran en torno a que podemos programar cada una de las actividades que realizamos, así mismo, indicaron que algo que puede ser tedioso para los niños durante los primeros años es el aprendizaje de las tablas de multiplicar desde sus experiencias y por medio de Scratch se vuelve más interesante aprender estas debido a la interactividad que se puede tener con el programa, sin embargo, indicaron que a futuro ellos pueden realizar el mismo programa pero cambiando los escenarios para que estos se han más

atractivos para quienes lo utilicen. Por lo anterior, se muestra en la Figura 18 el desarrollo del ejercicio y se deja el a los estudiantes la posibilidad de ampliar el ejercicio volviéndolo más interactivo y menos aburrido para quien lo pueda utilizar a futuro.

Figura 18.

Ejercicio No. 30 – Estructuras de repetición



Las explicaciones de los diferentes temas durante cada sesión de clase permitieron que los estudiantes pudieran expresar las dudas que se iban generando durante el desarrollo de las actividades y las guías realizadas permitieron que fuera un material de estudio para consultar fuera de las clases. Sin embargo, cabe resaltar que algunos ejercicios no fueron desarrollados en su totalidad por parte de los estudiantes debido a la complejidad que estos pudieron presentar y que como indicaron la poca experiencia en lenguajes de programación pudo ser uno de los factores que influyera en su desarrollo.

Para los estudiantes el entorno de programación Scratch les pareció interactivo y visualmente indicaron que debido a que los bloques ya se encuentran creados les facilito poder construir programas aplicando conceptos de programación teniendo en cuenta que estos deben

encajar uno en el otro para poder hacer uso de ellos. Adicionalmente, la guía que se realizó permitió ser un material de estudio en donde podían revisar y aplicar los conceptos que se iban trabajando durante las sesiones.

Por medio de la programación en Scratch identificaron que esta contribuye en el desarrollo de habilidades transversales debido a que pudieron aplicar la programación en contextos de matemáticas y esto ayuda a enriquecer sus aprendizajes y potenciar su creatividad en la resolución de problemas de la vida cotidiana de manera lógica, en donde la toma de decisiones es recurrente y experimentar la prueba y error para abstraer la información es esencial ya que permite dar solución a problemas complejos.

5.3.2. Análisis de las actividades de solución de problemas en entorno de programación textual (Visual Studio Code Python)

En el desarrollo de las secciones del ambiente de aprendizaje los estudiantes encontraban un ejemplo introductorio con el objetivo de orientar el desarrollo de cada una de las actividades propuestas y motivar su aprendizaje. En dicho ejemplo se mostraba información para recordar, un ejercicio práctico con su respectivo desarrollo como se muestra en la Figura 19. A su vez, se indicó a los estudiantes que la entrega de las actividades se realizaría por medio del ambiente virtual (Moodle) y el formato de entrega seria imágenes, carpetas y archivos que pudieran evidenciar el desarrollo de estas, teniendo en cuenta que cualquiera de las anteriores opciones seria valida siempre y cuando evidenciara todo el trabajo realizado.

Figura 19.

Ejemplo de actividad en Visual Studio Code Python

Durante el desarrollo de las sesiones de clase los estudiantes tuvieron acceso a la guía creada en la cual encontraban un ejemplo del tema que se trabajaba y se realizaba explicación para aclarar dudas que se pudieran presentar. Sin embargo, se presentaron dificultades en la creación de los programas (Figura 20), los errores más frecuentes estaban relacionados con la creación de las variables a utilizar, pero no se determina el tipo de datos a almacenar, generando un error en el almacenamiento de los datos lo cual no permite que el programa se ejecute en su totalidad.

Figura 20.

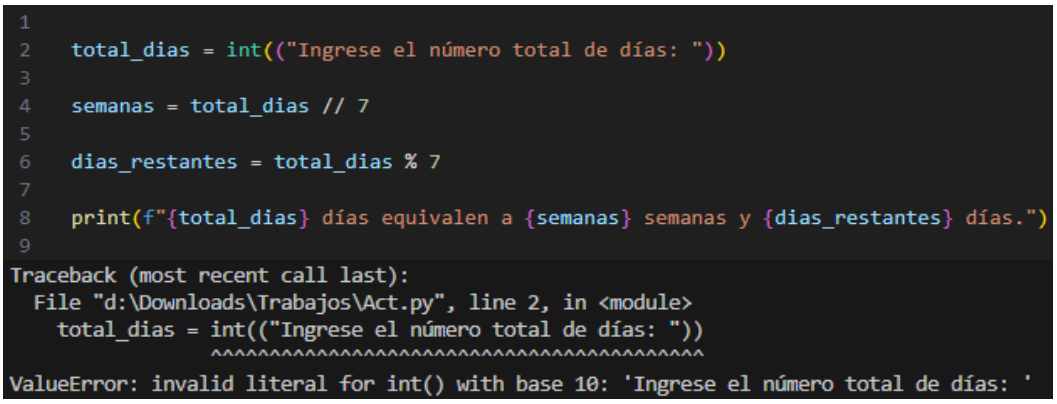
Ejercicio No. 1 - Variables

```
precio_base = (input("Ingrese el precio del producto antes de impuestos: "))
tasa_impuesto = (input("Ingrese la tasa de impuestos (%): "))
impuesto = precio_base * (tasa_impuesto / 100)
precio_final = precio_base + impuesto
print("Precio base: $", precio_base, ", Tasa de impuesto: ",
      tasa_impuesto, "%" ,", Precio final con impuestos: $", precio_final)
```

Se observo que algunos estudiantes presentaron dificultades en la comprensión de los errores de sintaxis que se generaban debido a la falta de familiaridad con los mensajes de error que se generaban y la falta de estrategias para interpretar y corregir dichos errores, para lo cual, esto genero disminución en el ritmo de trabajo de los estudiantes para el desarrollo de las actividades y genero cierta frustración en algunos casos, afectando la motivación para continuar con el desarrollo de los ejercicios. Alguno de los errores que presentaron se muestra en la Figura 21 en el cual se evidencia el desarrollo del ejercicio y en el cual el estudiante no agrega en la línea dos la función “input” la cual permite que se pida la información al usuario según lo que se requiere que el programa realice.

Figura 21.

Ejercicio No. 3 - Variables



A continuación, se evidencia la solución a uno de los ejercicios como se muestra en el Figura 22, en donde el estudiante opta por desarrollar el ejercicio haciendo uso solo de la condición “if” para pedir la temperatura en grados Celsius y no hace uso de una segunda condición como es “elif” y “else”, y a su vez, para la ejecución del programa olvida hacer uso de la función “print” para mostrar el resultado del ejercicio según la ejecución del mismo.

Figura 22.

Ejercicio No. 12 - Estructuras Selectivas - incompleto

```
Tempe = float(input("Ingrese la temperatura en grados Celsius: "))
if Tempe < 15:
    Res = "Hace frío"
if Tempe >= 15 and Tempe <= 25:
    Res = "La temperatura es agradable"
if Tempe > 25:
    Res = "Hace calor"
```

Sin embargo, a medida que se fueron desarrollando las clases, los estudiantes fueron obteniendo una mejor comprensión de los temas y por medio de las explicaciones al iniciar las clases y los ejemplos prácticos generados por medio de las guías se fue facilitando la asimilación de conceptos. Para ello, se evidencio un avance en el desarrollo de las actividades propuestas y permitió abordar los ejercicios con mayor éxito en su solución como se muestra en la Figura 23.

Figura 23.

Ejercicio No. 12 - Estructuras selectivas - completo

```

temperatura = float(input("Ingrese la temperatura en grados Celsius: "))

if temperatura < 15:
    print("Hace frío.")
elif temperatura > 25:
    print("Hace calor.")
else:
    print("La temperatura es agradable.")

```

```
Ingrese la temperatura en grados Celsius: 15
La temperatura es agradable.
```

Los estudiantes indicaban que algunos de los ejercicios eran complejos y debido a que Visual Studio Code Python no tenía ayudas para la creación del código era difícil desarrollar los ejercicios propuestos. En la Figura 24, se muestra errores que se cometían en el desarrollo de los ejercicios como son errores de sintaxis y código incompleto.

Figura 24.

Ejercicio No. 15 - Estructuras selectivas

```
numero = int(input("Ingrese un número para comprobar si es primo: ")
primo = True
if primo = True
    prit(f"El número {numero} es primo")
    else
        prit(f"El número {numero} no es primo")
        for n in range (1, 100)
        es_primo = True
        es_primo = False
        prit(n)
```

A continuación, en la Figura 25 se muestra el desarrollo de un ejercicio en el que algunos estudiantes logran identificar la estructura correcta, pero a su vez en la Figura 26 se muestra el error que cometieron algunos estudiantes en donde la falta de concentración pudo ser una desventaja debido a que en el ejercicio se indica hacer uso de la estructura “repetir” y hacen uso de la estructura “for”.

Figura 25.

Ejercicio No. 31 - Estructuras de repetición – correcto

```
1   Nume = int(input("Ingrese un número para contar hacia atrás: "))
2   while Nume >= 1:
3       print(Nume)
4       Nume = Nume - 1
```

Figura 26.

Ejercicio No. 31 - Estructuras de repetición – incorrecto

```
1   N = int(input("Ingrese un número "))
2   for N >= 1:
3       print(N)
4       N = N - 1
```

Durante el proceso se pudo evidenciar un esfuerzo constante por parte de los estudiantes para desarrollar los ejercicios propuestos teniendo en cuenta los retos que puede llegar a implicar programar para muchos por primera vez. Sin embargo, a pesar de las dificultades iniciales, muchos lograron comprender conceptos básicos y aplicar soluciones a los ejercicios propuestos, evidenciando una disposición positiva por aprender y la capacidad para adaptarse a nuevos retos. No obstante, se evidencio que los estudiantes cometen errores de sintaxis en donde un paréntesis, una coma, una letra faltante o mal escrita les genera frustración debido a que la ejecución del programa se ve afectada y aunque el programa muestra los errores que se generan, se les dificulta encontrar y solucionar estos.

Aunque la sintaxis tiende a facilitar la lectura y escritura los estudiantes indicaron que es un lenguaje nuevo para ellos y por lo tanto no es tan comprensible debido a que se dificulta saber que palabras o caracteres se debe utilizar y el orden adecuado para que no afecte el funcionamiento de los ejercicios, así mismo, aunque muestra los errores en la ejecución, no tiene capsulas informativas o ayudas que permitan el orden adecuado del código.

6. Discusión de Resultados

El entorno de programación por bloques (Scratch) y el entorno de programación textual (Visual Studio Code Python) mostraron ser efectivos para desarrollar habilidades de pensamiento computacional, si bien el análisis de covarianza no permitió identificar diferencias estadísticamente significativas. El análisis de las sesiones de solución de problemas permitió observar que los estudiantes que trabajaron con Scratch lograron mejores resultados en cuanto a conceptos fundamentales debido a que el entorno se muestra de manera intuitiva, mientras que los estudiantes que trabajaron en el entorno de Visual Studio Code mostraron dificultades relacionadas con errores de sintaxis. Estos resultados son consistentes con lo reportado por Price y Barnes (2015), quienes encontraron que los entornos de bloques mejoran la motivación inicial y reducen la frustración, permitiendo que los estudiantes se enfoquen en la lógica y los conceptos básicos sin preocuparse por la sintaxis.

Durante las sesiones de clase se evidencio que los entornos de texto tienden a ser más desafiantes y que los estudiantes que logran adaptarse a ellos desarrollan una percepción más sólida de su capacidad para programar logrando solucionar los errores que se puedan generar durante la ejecución de cada programa, logrando una mejor abstracción de la información y resolviendo problemas cada vez más complejos. Este hallazgo concuerda con Weintrop y Wilensky (2018), quienes argumentan que la complejidad sintáctica de los lenguajes textuales fomenta una comprensión más profunda de los conceptos avanzados de programación.

Los estudiantes que trabajaron en el entorno de programación por bloques Scratch mostraron mayor interés debido a que el programa les permite tener mayor interactividad en la conexión de los bloques, ayudando a la solución del ejercicio. Esto coincide con lo reportado por

Grover y Pea (2013) y Malan y Leitner (2007), quienes destacan que las interfaces gráficas, como Scratch, son más accesibles y fomentan un mayor compromiso inicial. No obstante, el reconocimiento de los tipos de bloques fue una de las dificultades que enfrentaron aquellos sin experiencia previa en este entorno. Por otro lado, los estudiantes en entornos textuales encontraron más dificultades para adaptarse a la sintaxis, a la escritura completa de las instrucciones y a la interpretación de los errores de ejecución del programa.

A partir de los resultados de este estudio se puede confirmar que los dos entornos de programación, tanto basados en bloques Scratch como basados en texto Visual Studio Code Python, contribuyeron en el desarrollo del pensamiento computacional en los estudiantes, teniendo en cuenta que los retos a los que se enfrenta cada entorno son diferentes. Así, las observaciones llevadas a cabo permiten identificar que el entorno de bloques favorece el pensamiento lógico y la concentración en la solución de los problemas, mientras que en el entorno textual se privilegia la identificación cada uno de los conceptos y elementos que componen al desarrollo de los ejercicios. Esto sugiere que los entornos de programación por bloques pueden llegar a ser más efectivos en las etapas iniciales del aprendizaje y con estudiantes sin experiencia previa en programación debido a su accesibilidad y facilidad de uso. Esto coincide con los hallazgos de Resnick et al., (2009) y Brennan y Resnick (2012).

A pesar de las dificultades iniciales relacionadas con la sintaxis del lenguaje de programación, la creacion y uso de variables y la resolucion de problemas aplicando conceptos computacionales más complejos, el grupo que utilizo Visual Studio Code Python mostró una mejora en terminos relativos frente al grupo de Scratch, de esta manera, el grupo que utilizó el entorno de programación textual pudo verse más desafiado lo que permitio la ganacia de habilidades de pensamiento computacional, lo cual indica que, aunque los entornos de

programación textual tienden a ser mas demandantes podrian llegar a ser mas efectivos para los estudiantes en niveles educativos superiores. Estos resultados confirman hallazgos previos que sugieren que el proceso educativo pueda manejar una progresión de los aprendizajes, iniciando con entornos de programación por bloques y se realizando la transición a entornos de programación textual (Weintrop & Wilensky, 2017; Grover & Pea, 2013).

7. Conclusiones

La creación de actividades diferenciadas para los entornos de programación por bloques y textual permitió abordar conceptos computacionales adaptados a las particularidades de los entornos de programación Scratch y Visual Studio Code Python. Las guías diseñadas mostraron ser efectivas para estructurar el aprendizaje en ambos entornos de programación, facilitando la comprensión progresiva de los conceptos computacionales abordados a través de ejercicios y problemas conectados con el entorno cotidiano de los estudiantes. No obstante, las guías pueden mejorarse teniendo en cuenta que para la mayoría de los estudiantes que participaron en este estudio esta era su primera experiencia con un entorno de programación y algunos ejercicios mostraron cierto nivel de dificultad. Este puede ser uno de los factores que pudo afectar los resultados debido a que los estudiantes tuvieron dificultades en completar los ejercicios propuestos a medida que iba aumentando la complejidad de los conceptos computacionales utilizados.

Este resultado muestra la importancia de proporcionar materiales didácticos adaptados de manera más precisa al nivel de conocimientos previos de los estudiantes a los que van dirigidas, así como la necesidad de ampliar el tiempo de trabajo en el área de tecnología con el fin de poder avanzar en el abordaje de los conceptos computacionales más complejos.

A pesar de que los resultados no revelaron diferencias estadísticamente significativas entre ambos entornos, los estudiantes que iniciaron con entornos basados en bloques presentaron un rendimiento inicial mayor, mientras que los estudiantes en entornos textuales demostraron un mayor progreso relativo.

Los resultados permitieron identificar algunas ventajas y limitaciones relacionadas con los entornos de programación. El entorno de programación por bloques Scratch redujo las barreras relacionadas con la sintaxis, no obstante, se presentaron problemas en la identificación correcta de los bloques, el uso de variables y la comprensión de conceptos computacionales más complejos, tal como los condicionales, entre los estudiantes sin experiencia previa.

Mientras que el entorno de programación textual basado en Python las principales dificultades estuvieron asociadas con el uso correcto de los tipos de datos a almacenar en las variables; dificultades en la comprensión de los errores de sintaxis, la falta de experiencia y estrategias para solucionar dichos errores; errores en las secuencias de instrucciones y en su escritura completa. La ausencia de ayuda en la interfaz de programación amplió las dificultades y generó frustración de algunos estudiantes.

Los resultados de este estudio sugieren que los entornos basados en bloques facilitan una introducción más amigable, especialmente para estudiantes sin experiencia previa en programación, mejorando la motivación por el aprendizaje y disminuyendo la frustración a la hora de completar las soluciones, interpretar y corregir los errores de sintaxis. No obstante, la transición a lenguajes textuales puede ayudar en la profundización de conceptos y habilidades de pensamiento computacional más complejas, una vez se ha avanzado en la familiarización con el uso de conceptos de programación. Lo que sugiere que ambos entornos de programación pueden ser efectivos y complementarios, demostrando su relevancia para la formación integral en competencias de pensamiento computacional en el contexto colombiano.

8. Contribuciones, Limitaciones y Proyecciones

8.1. Contribuciones

Este estudio compara el impacto de los entornos de programación por bloques (Scratch) y entornos de programación textual (Visual Studio Code Python) en el desarrollo del pensamiento computacional en estudiantes de bachillerato proporcionando datos importantes para enriquecimiento educativo. Por medio, de los hallazgos se evidencia la efectividad de un modelo progresivo que inicie en entornos de programación por bloques y transite hacia entornos de programación textuales.

El diseño de las actividades implementadas puede servir como referencia para futuros trabajos de investigación o programación de clases que deseen integrar los entornos de programación por bloques y textuales, brindando conceptos computacionales, en los que se busque promover la incorporación de enfoques progresivos en los currículos de educación secundaria y alineados a las necesidades de la sociedad actual y las demandas del mercado laboral en competencias digitales.

8.2. Limitaciones del estudio

El tiempo limitado en el que se llevó a cabo la intervención, así como la ausencia de experiencia previa de los estudiantes en entornos de programación, sumado a las ausencias por parte de los estudiantes durante la realización de las actividades pudieron afectar los resultados obtenidos.

8.3. Proyecciones

El estudio realizado aporta datos importantes sobre el impacto de los entornos de programación en el desarrollo del pensamiento computacional en el contexto colombiano, específicamente evidencia como Scratch y Visual Studio Code Python tiene cierto potencial a nivel educativo, así mismo, se recomienda la implementación de estudios a largo plazo que combinen entornos de programación por bloques y textuales que evidencien el impacto a largo plazo en competencias avanzadas. A su vez, se recomienda la ampliación de la muestra en la que se puedan incluir instituciones de diferentes características socioeconómicas y culturales, y la capacitación de docentes que refuercen la aplicación del uso pedagógico en estos entornos de programación.

Referencias

Attaway, D. (2016). *Matlab: A Practical Introduction to Programming and Problem Solving 4th Edición.* Butterworth-Heinemann.

Basogain, X., Olabe, M., Olabe, J. C., Rico, M. J., Rodríguez, L., & Amórtegui, M. (2017). Pensamiento computacional en las escuelas de Colombia: colaboración internacional de innovación en la educación. *Edmetic*. doi:https://recursos.educoas.org/sites/default/files/5188.pdf

Brackmann, C., Román González, M., Robles, G., Moreno León, J., Casali, A., & Barone, D. (2017). Development of Computational Thinking Skills through Unplugged Activities in Primary School. *Association for Computing Machinery*, 65-72.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *MIT Media Lab*.

Campbell, D., & Stanley, J. (1995). *Diseños experimentales y cuasiexperimentales en la investigación social.* Buenos Aires, Argentina: Talleres Gáficos Color.

Espinal, A., Vieira, C., & Guerrero-Bequis, V. (2023). Student ability and difficulties with transfer from a block-based programming language into other programming languages: a case study in Colombia. Computer Science Education, 33(4), 567-599. https://doi.org/10.1080/08993408.2022.2079867

Fedesoft. (2023). *Resultados Bebras*. Obtenido de Resultados Bebras: https://fedesoft.org/resultados-bebras-2023/

Fernández Díaz, O. R., Delgado Lechuga, G., Esquiaqui González, M., & Castellar Rodríguez , A. (2023). Pensamiento Computacional versus Pensamiento Matemático correlación en aprendizaje de estudiantes de educación media en Colombia. *Dialnet, 29*(3), 98-111. Obtenido de https://dialnet.unirioja.es/servlet/articulo?codigo=9102145

Flanagan, D. (2020). *JavaScript: The Definitive Guide, 7th Edition.* O'Reilly Media.

García-Peñalvo , F., & Cruz-Benito , J. (2016). Computational thinking in pre-university education. *Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality – TEEM'16* , https://doi.org/10.1145/3012430.3012490.

Grover , S., & Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher*, 38-43. doi:doi.org/10.3102/0013189X12463051

Grover , S., Cooper , S., & Pea, R. (2014). "Systems of Assessments" for Deeper Learning of Computational Thinking in K-12. *Conference: Annual Meeting of the American Educational Research Association*. Obtenido de chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://www.life-slc.org/docs/LSLC_rp_A204_Grover-Cooper-Pea_ITiCSE-2014.pdf

Grover, S., & Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher*, 38-43. doi:doi.org/10.3102/0013189X12463051

Harvey, B., & Mönig, J. (2010). Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists? *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 183-187. Obtenido de https://www.researchgate.net/publication/229043489_Bringing_No_Ceiling_to_Scratch_Can_One_Language_Serve_Kids_and_Computer_Scientists

Información, M. d., & Programa, C. (2024). *Ruta Código Verde*. Obtenido de Ruta Código Verde: https://mintic.gov.co/colombiaprograma/847/w3-channel.html

INTEF. (2022). *Programación y Robótica.* Obtenido de Programación y Robótica: https://formacion.intef.es/catalogo/mod/book/tool/print/index.php?id=69

International Society for Tecnology in Education, (., & Computer Science Teachers Association, (. (Agosto de 19 de 2011). *Computational Thinking: Teacher Resources*. Obtenido de

Computational Thinking: Teacher Resources: https://www.iste.org/store/attachmentdownload.aspx?id=2159

ISTE, C. &. (2011). *Operational Definition of Computational Thinking for K–12 Education.* Obtenido de Operational Definition of Computational Thinking for K–12 Education: https://cdn.iste.org/www-root/Computational_Thinking_Operational_Definition_ISTE.pdf

KABACOFF, R. (2015). *R in Action Data analysis and graphics with R.* Manning Publications.

Las falencias en el pensamiento computacional. (2024, febrero 25). www.vanguardia.com. https://www.vanguardia.com/economia/local/2024/02/25/las-falencias-en-el-pensamiento-computacional/

Lye, S., & Ling Koh, J. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *ELSEVIER*, 51-61. doi:doi.org/10.1016/j.chb.2014.09.012

Malan, D., & Leitner, H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, 223-227. doi:https://doi.org/10.1145/1227504.1227388

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, 1-15.

Mantilla Guiza, R. R., & Negro Bennasar, F. (2021). Pensamiento computacional, una estrategia educativa en tiempos de pandemia. *INNOEDUCA*, 89-106. doi:https://doi.org/10.24310/innoeduca.2021.v7i1.10593

Marc, L. M., Lévêque, O., Benitez Baena, I., Hardebolle, C., & Dehler Zufferey, J. (2022). Assessing Computational Thinking: Development and Validation of the Algorithmic Thinking Test for Adults. *Journal of Educational Computing Research*, *60*,

073563312110578. https://doi.org/10.1177/07356331211057819

McFarland, D. (2014). *JavaScript & jQuery: The Missing Manual, 3rd Edition.* O'Reilly Media.

Meyers, S. (2014). *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14.* O'Reilly Media.

Micrositio - Colombia Programa - Inicio - Colombia Programa. (s. f.). Micrositio - Colombia Programa. Recuperado 27 de febrero de 2025, de http://www.mintic.gov.co/colombiaprograma/847/w3-channel.html

Microsoft. (2020). *MakeCode Documentation*. Obtenido de MakeCode Documentation: https://www.microsoft.com/en-us/makecode?rtc=1

MinTIC. (02 de Octubre de 2024). *Código Verde*. Obtenido de Código Verde: https://www.mintic.gov.co/portal/inicio/Sala-de-prensa/Noticias/396380:Desde-su-lanzamiento-100-000-estudiantes-han-descargado-la-App-Codigo-Verde-para-aprender-sobre-pensamiento-computacional

MIT, N. (01 de Agosto de 2016). *El profesor emérito Seymour Papert, pionero del aprendizaje construccionista, muere a los 88 años*. Obtenido de El profesor emérito Seymour Papert, pionero del aprendizaje construccionista, muere a los 88 años: https://news.mit.edu/2016/seymour-papert-pioneer-of-constructionist-learning-dies-0801

Phillips Villaveces, J. (2020). La Introducción al Pensamiento Computacional. *Escuela de Ciencias Exactas e Ingeniería - ECEI*.

Programación y Robótica: Programación textual | Aula En Abierto. (s. f.). Recuperado 27 de febrero de 2025, de https://formacion.intef.es/aulaenabierto/mod/book/view.php?id=2747&chapterid=3063

Price, T. W., & Barnes, T. (2015). Comparing Textual and Block Interfaces in a Novice

Programming Environment. *ACM Association for Computing Machinery*, 91-99.

ProFuturo. (2021). *Cinco anos promovendo a inovação no ensino*. Obtenido de Cinco anos promovendo a inovação no ensino: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://profuturo.education/wp-content/uploads/2022/07/memoria-profuturo-2021-por.pdf

Pruebas PISA 2022: Colombia, un sistema educativo resiliente que requiere cambios estructurales para mejorar su calidad - Pruebas PISA 2022: Colombia, un sistema educativo resiliente que requiere cambios estructurales para mejorar su calidad. (s. f.). Portal MEN - Presentación. Recuperado 26 de diciembre de 2024, de https://www.mineducacion.gov.co/1780/w3-article-417751.html

Quiroz Vallejo, D. A., Carmona Mesa, J. A., Castrillón Yepes, A., & Villa Ochoa, J. A. (2021). RED. Revista de Educación a Distancia. Núm. 68, Vol. 21. Artíc. 7, 30-Nov-2021 DOI: http://dx.doi.org/10.6018/red.485321 Integración del Pensamiento Computacional en la educación primaria y secundaria en Latinoamérica: una revisión sistemática de literat. *Revista de Educación a Distancia (RED), 21*, 1-33. doi:http://dx.doi.org/10.6018/red.485321

Resnick , M., Rusk, N., & Maloney, J. (2009). Scratch: programming for all. *Communications of the ACM*, 60-67.

Resnick, M., Maloney, J., Monroy Hernández , A., Rusk, N., Eastmond, E., & Brennan, K. (2009). Scratch: Programing for all. Communications of the ACM. 60-67.

Román González, M., Pérez González, J., & Jiménez Fernández, C. (2015). Computational Thinking Test: design & general psychometry. *III Congreso Internacional sobre Aprendizaje, Innovación y Competitividad*, 1-23. doi:10.13140/RG.2.1.3056.5521

S.A.S, E. L. R. (2022, septiembre 6). De no tomar acciones, Colombia tendría déficit de 112.000 desarrolladores en 2025. Diario La República. https://www.larepublica.co/alta-gerencia/de-no-tomar-acciones-colombia-tendria-deficit-de-112-000-desarrolladores-en-2025-3440141

Sadosky, F. (2019). *Program.AR: Programación en la Escuela*. Obtenido de Program.AR: Programación en la Escuela: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://program.ar/wp-content/uploads/2023/07/ProgramAR_CAF_10.pdf

Sinisterra, B. E. (2018). *Creación de materiales para Recursos Educativos Digitales Abiertos (REDA): Una Estrategia de aprendizaje por proyectos que aporta al desarrollo de pensamiento computacional en el ciclo de educación media en la escuela Normal Superior de Leticia.* Obtenido de Creación de materiales para Recursos Educativos Digitales Abiertos (REDA): Una Estrategia de aprendizaje por proyectos que aporta al desarrollo de pensamiento computacional en el ciclo de educación media en la escuela Normal Superior de Leticia: https://intellectum.unisabana.edu.co/handle/10818/33818

Stroustrup, B. (2013). *The C++ Programming Language.* Addison-Wesley.

UNESCO. (2020). Education for sustainable development: a roadmap. *UNESCO*. doi:https://doi.org/10.54675/YFRE1448

UNESCO. (2020). La ciudadanía digital como política pública en educación en América Latina. *Repositorio UNESCO*, 5. Obtenido de La ciudadanía digital como política pública en educación en América Latina: https://unesdoc.unesco.org/ark:/48223/pf0000376935?posInSet=6&queryId=0568ca9c-b3a1-42bf-8e38-624b384bf925

Weintrop, D., & Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in

High School Computer Science Classrooms. *ACM Transactions on Computing Education (TOCE), Volume 18, Issue 1*, 1-25. doi:https://doi.org/10.1145/3089799

Weintrop, D., & Wilensky, U. (2019). Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Journal of Computer Science Education*, 67-88. doi:doi.org/10.1016/j.compedu.2019.103646

Weintropa, D., & Wilensky, U. (2018). How block-based, text-based, and hybrid block/text modalities shape. *International Journal of Child-Computer Interaction*, 83-92. Obtenido de https://www.sciencedirect.com/science/article/abs/pii/S2212868917300314

Wing, J. (2008). Computational thinking and thinking about computing. *hilosophical Transactions of The Royal Society A Mathematical Physical and Engineering Sciences* , 3717-3725.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*, 33-35. doi:https://doi.org/10.1145/1118178.1118215

Wolber, D., Abelson, H., Spertus , E., & Looney, L. (2014). *App Inventor 2: Create your own Android apps.* O'Reilly Media.

Yusuf, A., & Román-González, M. (2024). Interaction Patterns During Block-based Programming Activities Predict Computational Thinking: Analysis of the Differences in Gender, . *Cognitive Load, Spatial Ability, and Programming Proficiency. IntechOpen.*, 1-39. doi:https://doi.org/10.5772/acrt.36

Anexos

Anexo A. Actividades para el entorno de programación grafica Scratch

EJEMPLO

- Un diagrama de flujo tiene como objetivo identificar y representar rutinas para resolver problemas o tareas, como instrucciones ordenadas paso a paso.
- Un algoritmo es una secuencia detallada de pasos o instrucciones que conducen a la realización de una tarea o a la solución de un problema.

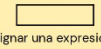
0

Construya un diagrama de flujo y un programa en Scratch tal que dado los datos enteros A y B, escriba el resultado de la siguiente expresión:

$$\frac{(A+B)^2}{3}$$

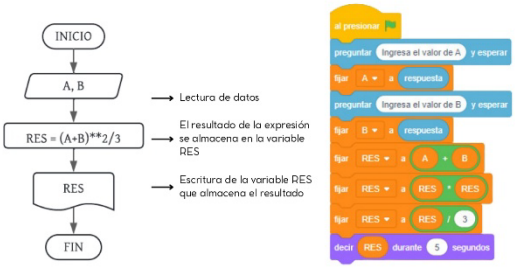
RECUERDA LO SIGUIENTE:

- Para realizar un proceso se utiliza el símbolo:



- Para asignar una expresión o valor a una variable, se utiliza un bloque de asignación:

Variable= expresión o valor



ACTIVIDAD N.1

1

Construya un diagrama de flujo y un programa en Scratch que, dado el precio de un producto antes de impuestos y la tasa de impuestos (en porcentaje), calcula el precio final del producto después de aplicar los impuestos.

2

Construya un diagrama de flujo y un programa en Scratch que, dado el salario mensual de un empleado, calcula su salario anual considerando que recibe dos pagas extras al año. Luego, imprime el salario anual en la consola.

3

Construya un diagrama de flujo y un programa en Scratch que, dado un número de días, calcule cuántas semanas y días completos hay en ese periodo. El programa debe:

- Solicitar al usuario el número total de días.
- Calcular cuántas semanas completas hay en ese número de días.
- Calcular cuántos días sobran después de contar las semanas completas.
- Imprimir el número de semanas y el número de días restantes.

4

Construya un diagrama de flujo y un programa en Scratch que, dado un número N de personas y su altura en centímetros, determine la persona más alta y la más baja. El programa debe:

- Solicitar al usuario la cantidad de personas (N).
- Solicitar al usuario la altura de cada persona.
- Determinar la altura máxima y mínima entre las personas.
- Imprimir la altura de la persona más alta y la más baja.

ACTIVIDAD N.2

5

Construya un programa en Scratch que, dado un monto de dinero en una moneda, lo convierta a otra moneda usando una tasa de cambio fija. Solicite al usuario que ingrese el monto de dinero. Solicite al usuario que ingrese la tasa de cambio (por ejemplo, 1 dólar = 0.85 euros). Calcule el monto convertido (monto convertido = monto * tasa de cambio). Imprima el monto convertido.

6

Construya un programa en Scratch que, dado el peso en kilogramos y la altura en metros de una persona, calcule e imprima su Índice de Masa Corporal (IMC).

- Solicite al usuario que ingrese su peso en kilogramos.
- Solicite al usuario que ingrese su altura en metros.
- Calcule el IMC (IMC = peso / (altura * altura)).
- Imprima e indique la categoría de nivel de peso según el IMC.

7

Construya un programa en Scratch que, dada la cantidad de millas recorridas y el tiempo en horas, calcule y muestre la velocidad promedio en kilómetros por hora (km/h). El programa debe solicitar al usuario ingresar la cantidad de millas recorridas y el tiempo en horas, convertir las millas a kilómetros (1 milla = 1.60934 kilómetros), y luego calcular la velocidad promedio. Finalmente, debe mostrar este valor en la consola.

8

Construya un programa en Scratch que, dado como entrada la longitud y el ancho de un campo rectangular, calcule y muestre su perímetro y área. Luego, muestra estos valores en la consola.

ACTIVIDAD N.3

9

Construya un programa en Scratch que, dado un capital inicial, una tasa de interés anual y un número de años, calcule e imprima el valor futuro del capital usando interés compuesto.

- Solicite al usuario que ingrese el capital inicial.
- Solicite al usuario que ingrese la tasa de interés anual (en porcentaje).
- Solicite al usuario que ingrese el número de años.
- Calcule el valor futuro del capital (valor futuro = capital inicial * (1 + tasa de interés/100)^número de años).
- Imprima el valor futuro.

10

Construya un programa en Scratch que, dado un grupo de calificaciones de estudiantes, calcule e imprima el promedio de las mismas.

- Solicite al usuario que ingrese el número de calificaciones.
- Solicite al usuario que ingrese cada calificación.
- Calcule el promedio de las calificaciones (promedio = suma de calificaciones / número de calificaciones).
- Imprima el promedio.

11

Construya un programa en Scratch Construya un programa que, dada la fecha de nacimiento de una persona, calcule su edad actual y determine su categoría de acuerdo a la siguiente clasificación:

Menor de 12 años	Niño
Entre 13 y 17 años	Adolescente
Entre 18 y 64 años	Adulto
Mayor de 65 años	Anciano

- Solicite al usuario que ingrese su fecha de nacimiento (día, mes y año).
- Calcule la edad actual de la persona.
- Determine la categoría de la persona basada en su edad.
- Imprima la edad y la categoría de la persona.

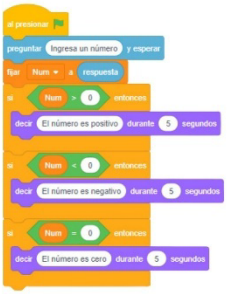
EJEMPLO

RECUERDA LO SIGUIENTE:
Las estructuras algorítmicas selectivas que se utilizan para la toma de decisiones lógicas las podemos clasificar de la siguiente forma:

- SI ENTONCES (Estructura selectiva simple)
- SI ENTONCES/SINO (Estructura selectiva doble)

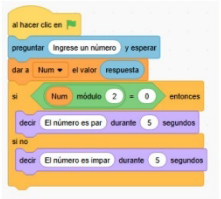
0

Construya un programa que, solicitará al usuario un número y, determinará e imprimirá si el número es positivo, negativo o cero.



0

Construya un programa que, solicitará al usuario un número y, determinará e imprimirá si el número es par o impar.



ACTIVIDAD N.4

12

Construya un programa en Scratch que, lea la temperatura en grados Celsius e imprima si hace frío, calor o una temperatura agradable. Define:

- Frío si la temperatura es menor a 15°C.
- Calor si la temperatura es mayor a 25°C.
- Agradable si la temperatura está entre 15°C y 25°C, inclusivo.

13

Construya un programa en Scratch que, dado un año, determine e imprima si el mismo es bisiesto.

- Solicite al usuario que ingrese un año.
- Utilice el condicional "si entonces" para verificar si el año es bisiesto (un año es bisiesto si es divisible por 4, pero no por 100, a menos que sea divisible por 400).
- Imprima "Bisiesto" si el año es bisiesto.
- Imprima "No Bisiesto" si el año no es bisiesto.

14

Construya un programa en Scratch que, dada la edad de una persona, determine e imprima si la misma es mayor de edad.

- Solicite al usuario que ingrese su edad.
- Utilice el condicional "si entonces" para verificar si la persona es mayor de edad (18 años o más).
- Imprima "Mayor de Edad" si la persona tiene 18 años o más.
- Imprima "Menor de Edad" si la persona tiene menos de 18 años.

15

Construya un programa en Scratch que, se encargue de comprobar si un número es o no primo. A continuación, imprime los números primos entre 1 y 100

ACTIVIDAD N.5

16

Construya un programa en Scratch que, dada la temperatura en grados Celsius, verifique si está por debajo de 0 grados. Si es así, convierta la temperatura a grados Fahrenheit y muestre el resultado en la consola. La fórmula para convertir de Celsius a Fahrenheit es:

$$°F = \frac{9}{5} * °C + 32$$

17

Construya un programa en Scratch que, dada la cantidad de productos vendidos por un vendedor en un mes, aplique un bono del 10% sobre el total de ventas si vendió más de 50 productos. Si vendió 50 productos o menos, aplique un bono del 5%. El programa debe solicitar al usuario ingresar la cantidad de productos vendidos y el precio total de los productos, calcular el bono y mostrar el total de las ventas con el bono incluido en la consola.

18

Construya un programa en Scratch que, dado un número entero, determine si el número es positivo, negativo o cero. El programa debe solicitar al usuario ingresar un número entero, realizar la verificación correspondiente y mostrar el resultado en la consola.

ACTIVIDAD N.6

19

Construya un programa en Scratch que, dada una temperatura en grados Celsius, determine e imprima si es fría (menos de 15°C), templada (entre 15°C y 30°C) o caliente (más de 30°C).

- Solicite al usuario que ingrese una temperatura en grados Celsius.
- Utilice el condicional "si entonces / sino" para calificar la temperatura.
- Imprima "Fría" si la temperatura es menos de 15°C.
- Imprima "Templada" si la temperatura está entre 15°C y 30°C.
- Imprima "Caliente" si la temperatura es más de 30°C.

20

Construya un programa en Scratch que, dada la edad de una persona, determine e imprima la categoría de edad en la que se encuentra la misma.

CATEGORÍAS	
Niño	Menos de 13 años
Adolescente	Entre 13 y 19 años
Joven adulto	Entre 20 y 35 años
Adulto	Entre 36 y 55 años
Adulto mayor	Entre 56 y 70 años
Anciano	Más de 70 años

- Solicite al usuario que ingrese su edad.
- Utilice el condicional "si entonces / sino" para determinar la categoría de edad de la persona.
- Imprima la categoría de edad correspondiente.

ACTIVIDAD N.7

21

Construya un programa en Scratch que, dado el peso de un paquete en kilogramos, calcule el costo de envío según las siguientes reglas:

- Los primeros 2 kg son gratuitos.
- Los siguientes 3 kg tienen un costo de $5 por kg.
- El resto tiene un costo de $8 por kg, pero si el peso del paquete es mayor a 20 kg, entonces el costo es de $10 por kg.

El programa debe solicitar al usuario ingresar el peso del paquete, calcular el costo de envío y mostrar el precio total del envío (peso total = peso + costo de envío) en la consola.

22

Crea un programa en Scratch que, dado como datos N números enteros, determine cuántos números negativos hay entre estos números. El programa debe solicitar al usuario ingresar la cantidad de números (N), luego solicitar cada uno de los números, contar cuántos de ellos son negativos y mostrar el resultado en la consola.

23

Construya un programa en Scratch que, dado como datos 100 números enteros, obtenga la suma de los números positivos y el promedio de los números negativos. El programa debe solicitar al usuario ingresar 100 números, calcular la suma de los positivos y el promedio de los negativos, y mostrar ambos resultados en la consola.

ACTIVIDAD N.8

24

Construya un programa en Scratch que, dado N números enteros, determine cuántos de ellos son múltiplos de 3, calcule el promedio de los números múltiplos de 3 y el promedio de todos los números. El programa debe solicitar al usuario ingresar la cantidad de números (N), luego solicitar cada uno de los números, realizar los cálculos correspondientes y mostrar los resultados en la consola.

25

Construya un programa en Scratch que, lea la cantidad de dinero que tiene una persona y determine si puede comprar un artículo cuyo precio es fijo. Si puede comprarlo, debe imprimirse "Compra realizada". Si no puede, debe imprimirse "Fondos insuficientes".

26

Construya un programa en Scratch que, pregunte al usuario su renta anual y muestre por pantalla el tipo impositivo que le corresponde. Los tramos impositivos para la declaración de la renta en un determinado país son los siguientes:

RENTA	TIPO IMPOSITIVO
Menos de 10000€	5%
Entre 10000€ y 20000€	15%
Entre 20000€ y 35000€	20%
Entre 35000€ y 60000€	30%
Más de 60000€	45%

ACTIVIDAD N.9

16

Construya un programa en Scratch que, dado el monto de la compra de un cliente y su tipo de membresía (Básica, Plata, Oro), determine el descuento aplicable y lo que el cliente debe pagar. El descuento se efectúa con base en el siguiente criterio:

Descuentos según el monto de la compra:

- **Si el monto es menor que $500:**
  - Membresía Básica: No hay descuento
  - Membresía Plata: 2% de descuento
  - Membresía Oro: 5% de descuento
- **Si el monto está comprendido entre $500 y $1000:**
  - Membresía Básica: 5% de descuento
  - Membresía Plata: 7% de descuento
  - Membresía Oro: 10% de descuento
- **Si el monto está comprendido entre $1000 y $7000:**
  - Membresía Básica: 11% de descuento
  - Membresía Plata: 13% de descuento
  - Membresía Oro: 15% de descuento
- **Si el monto está comprendido entre $7000 y $15000:**
  - Membresía Básica: 18% de descuento
  - Membresía Plata: 20% de descuento
  - Membresía Oro: 22% de descuento
- **Si el monto es mayor a $15000:**
  - Membresía Básica: 25% de descuento
  - Membresía Plata: 27% de descuento
  - Membresía Oro: 30% de descuento

El programa debe incluir las siguientes características:

1. Solicitar al usuario el monto de la compra.
2. Solicitar al usuario su tipo de membresía (Básica, Plata, Oro).
3. Determinar el descuento aplicable.
4. Calcular el monto final a pagar.
5. Imprimir el monto del descuento y el monto final a pagar.

EJEMPLO

RECUERDA LO SIGUIENTE:

- La estructura *repetir*, conocida como *FOR*, es una estructura adecuada para utilizar en un ciclo que se ejecutará un número definido de veces.
- La estructura *mientras*, conocida como *while*, es la estructura para utilizar cuando no sabemos el número de veces que éste se ha de repetir.

0

Construya un programa en Scratch que, solicitará al usuario un número "n" y calculará la suma de los primeros "n" números enteros utilizando un bucle "repetir" o "For".

EJEMPLO

RECUERDA LO SIGUIENTE:

- La estructura *repetir*, conocida como *FOR*, es una estructura adecuada para utilizar en un ciclo que se ejecutará un número definido de veces.
- La estructura *mientras*, conocida como *while*, es la estructura para utilizar cuando no sabemos el número de veces que éste se ha de repetir.

0

Construya un programa en Scratch que, solicitará al usuario que ingrese un número y seguirá solicitando hasta que el usuario ingrese un número negativo.

ACTIVIDAD N.10

28

Construya un programa en Scratch que, calcule el factorial de un número entero positivo proporcionado por el usuario.

29

Construya un programa en Scratch que, lea una cadena de texto y cuente cuántas vocales (a, e, i, o, u) contiene.

30

Construya un programa en Scratch que, imprima la tabla de multiplicar de un número N.

- Solicite al usuario que ingrese el valor de N.
- Utilice la estructura "repetir" para imprimir la tabla de multiplicar de N del 1 al 10.

31

Construya un programa en Scratch que, cuente hacia atrás desde un número N hasta 1.

- Solicite al usuario que ingrese el valor de N.
- Utilice la estructura "repetir" para contar hacia atrás desde N hasta 1

32

Construya un programa en Scratch que, calcule la media de N números ingresados por el usuario.

- Solicite al usuario que ingrese el valor de N.
- Utilice la estructura "repetir" para pedir al usuario que ingrese N números.
- Calcule la media de los N números e imprima el resultado.

ACTIVIDAD N.11

33

Construya un programa en Scratch que, dado el sueldo de cada uno de los trabajadores en una empresa y el número de horas extra trabajadas por cada trabajador, calcule el total de la nómina y el pago total por las horas extra. El pago por hora extra es del 50% adicional sobre el sueldo por hora.

Detalles del Ejercicio:

1. Solicite el sueldo mensual de cada trabajador (suponiendo 10 trabajadores).
2. Solicite el número de horas extra trabajadas por cada trabajador.
3. Calcule el sueldo por hora para cada trabajador.
4. Calcule el pago por horas extra (50% adicional sobre el sueldo por hora).
5. Calcule el total de la nómina incluyendo el pago por horas extra.
6. Imprima el total de la nómina y el pago total por horas extra.

	SUELDO	HORAS EXTRAS	SUELDO POR HORA	PAGO HORAS EXTRAS	NOMINA
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Anexo B. Actividades para el entorno de programación textual Visual Studio Code

Python

EJEMPLO

- Un diagrama de flujo tiene como objetivo identificar y representar rutinas para resolver problemas o tareas, como instrucciones ordenadas paso a paso.
- Un algoritmo es una secuencia detallada de pasos o instrucciones que conducen a la realización de una tarea o a la solución de un problema.

0

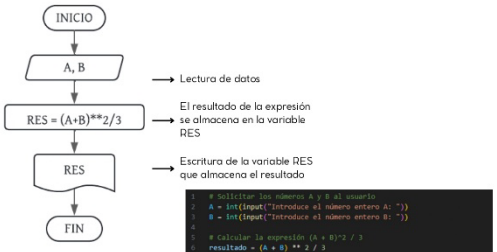
Construya un diagrama de flujo y un programa en Python tal que dado los datos enteros A y B, escriba el resultado de la siguiente expresión:

$$\frac{(A+B)^2}{3}$$

RECUERDA LO SIGUIENTE:

- Para realizar un proceso se utiliza el símbolo: ▭
- Para asignar una expresión o valor a una variable, se utiliza un bloque de asignación:

Variable= expresión o valor




```
# Solicitar los números A y B al usuario
A = int(input("Introduce el número entero A: "))
B = int(input("Introduce el número entero B: "))

# Calcular la expresión (A + B)^2 / 3
resultado = (A + B) ** 2 / 3

# Mostrar el resultado
print(f"El resultado de la expresión (A + B)^2 / 3 es: {resultado}")
```

ACTIVIDAD N.1

1

Construya un diagrama de flujo y un programa en Python que, dado el precio de un producto antes de impuestos y la tasa de impuestos (en porcentaje), calcula el precio final del producto después de aplicar los impuestos. Luego, muestra estos valores en la consola.

2

Construya un diagrama de flujo y un programa en Python que, dado el salario mensual de un empleado, calcula su salario anual considerando que recibe dos pagas extras al año. Luego, imprime el salario anual en la consola.

3

Construya un diagrama de flujo y un programa en Python que, dado un número de días, calcule cuántas semanas y días completos hay en ese periodo. El programa debe:

- Solicitar al usuario el número total de días.
- Calcular cuántas semanas completas hay en ese número de días.
- Calcular cuántos días sobran después de contar las semanas completas.
- Imprimir el número de semanas y el número de días restantes.

4

Construya un diagrama de flujo y un programa en Python que, dado un número N de personas y su altura en centímetros, determine la persona más alta y la más baja. El programa debe:

- Solicitar al usuario la cantidad de personas (N).
- Solicitar al usuario la altura de cada persona.
- Determinar la altura máxima y mínima entre las personas.
- Imprimir la altura de la persona más alta y la más baja.

ACTIVIDAD N.2

5

Construya un programa en Python que, dado un monto de dinero en una moneda, lo convierta a otra moneda usando una tasa de cambio fija. Solicite al usuario que ingrese el monto de dinero. Solicite al usuario que ingrese la tasa de cambio (por ejemplo, 1 dólar = 0.85 euros). Calcule el monto convertido (monto convertido = monto * tasa de cambio). Imprima el monto convertido.

6

Construya un programa en Python que, dado el peso en kilogramos y la altura en metros de una persona, calcule e imprima su Índice de Masa Corporal (IMC).

- Solicite al usuario que ingrese su peso en kilogramos.
- Solicite al usuario que ingrese su altura en metros.
- Calcule el IMC (IMC = peso / (altura * altura)).
- Imprima e indique la categoría de nivel de peso según el IMC.

7

Construya un programa en Python que, dada la cantidad de millas recorridas y el tiempo en horas, calcule y muestre la velocidad promedio en kilómetros por hora (km/h). El programa debe solicitar al usuario ingresar la cantidad de millas recorridas y el tiempo en horas, convertir las millas a kilómetros (1 milla = 1.60934 kilómetros), y luego calcular la velocidad promedio. Finalmente, debe mostrar este valor en la consola.

8

Construya un programa en Python que, dado como entrada la longitud y el ancho de un campo rectangular, calcule y muestre su perímetro y área. Luego, muestra estos valores en la consola.

ACTIVIDAD N.3

9

Construya un programa en Python que, dado un capital inicial, una tasa de interés anual y un número de años, calcule e imprima el valor futuro del capital usando interés compuesto.

- Solicite al usuario que ingrese el capital inicial.
- Solicite al usuario que ingrese la tasa de interés anual (en porcentaje).
- Solicite al usuario que ingrese el número de años.
- Calcule el valor futuro del capital (valor futuro = capital inicial * (1 + tasa de interés/100)^número de años).
- Imprima el valor futuro.

10

Construya un programa en Python que, dado un grupo de calificaciones de estudiantes, calcule e imprima el promedio de las mismas.

- Solicite al usuario que ingrese el número de calificaciones.
- Solicite al usuario que ingrese cada calificación.
- Calcule el promedio de las calificaciones (promedio = suma de calificaciones / número de calificaciones).
- Imprima el promedio.

11

Construya un programa en Python Construya un programa que, dada la fecha de nacimiento de una persona, calcule su edad actual y determine su categoría de acuerdo a la siguiente clasificación:

Menor de 12 años	Niño
Entre 13 y 17 años	Adolescente
Entre 18 y 64 años	Adulto
Mayor de 65 años	Anciano

- Solicite al usuario que ingrese su fecha de nacimiento (día, mes y año).
- Calcule la edad actual de la persona.
- Determine la categoría de la persona basada en su edad.
- Imprima la edad y la categoría de la persona.

EJEMPLO


RECUERDA LO SIGUIENTE:
Las estructuras algorítmicas selectivas que se utilizan para la toma de decisiones lógicas las podemos clasificar de la siguiente forma:

- SI ENTONCES (Estructura selectiva simple)
- SI ENTONCES/SINO (Estructura selectiva doble)

0

Construya un programa en Python que, dado como entrada la cantidad de días que un libro ha sido devuelto tarde a la biblioteca, calcule y muestre la multa correspondiente. La multa diaria es de $2, pero si el libro ha sido devuelto con más de 10 días de retraso, se aplica una multa adicional de $10. El programa debe solicitar al usuario ingresar la cantidad de días de retraso, calcular la multa y mostrar este valor en la consola.

```python
dias_retraso = int(input("Introduce la cantidad de días de retraso: "))
multa = dias_retraso * 2
if dias_retraso > 10:
    multa = multa + 10
print("La multa total es:", multa)
```

```
Introduce la cantidad de días de retraso: 3
La multa total es: 6
```

```
Introduce la cantidad de días de retraso: 15
La multa total es: 40
```

EJEMPLO


RECUERDA LO SIGUIENTE:
Las estructuras algorítmicas selectivas que se utilizan para la toma de decisiones lógicas las podemos clasificar de la siguiente forma:

- SI ENTONCES (Estructura selectiva simple)
- SI ENTONCES/SINO (Estructura selectiva doble)

0

Construya un programa en Python que, dado como entrada la cantidad de horas trabajadas en una semana y la tasa de pago por hora, calcule el salario semanal. Si el número de horas trabajadas es mayor a 40, las horas adicionales se consideran horas extras y se pagan a 1.5 veces la tasa de pago normal. El programa debe solicitar al usuario ingresar la cantidad de horas trabajadas y la tasa de pago por hora, luego calcular el salario semanal teniendo en cuenta las horas extras si es necesario, y finalmente mostrar el salario semanal en la consola.

```python
horas_trabajadas = float(input("Introduce la cantidad de horas trabajadas en la semana: "))
tasa_pago = float(input("Introduce la tasa de pago por hora: "))
salario_semanal = 0.0

if horas_trabajadas > 40:
    horas_extras = horas_trabajadas - 40
    salario_horas_extras = horas_extras * tasa_pago * 1.5
    salario_regular = 40 * tasa_pago
    salario_semanal = salario_regular + salario_horas_extras
else:
    salario_semanal = horas_trabajadas * tasa_pago

print("El salario semanal es:",salario_semanal)
```

```
Introduce la cantidad de horas trabajadas en la semana: 50
Introduce la tasa de pago por hora: 15600
El salario semanal es: 858000.0
```

ACTIVIDAD N.4


12

Construye un programa en Python que, lea la temperatura en grados Celsius e imprima si hace frío, calor o una temperatura agradable. Define:

- Frío si la temperatura es menor a 15°C.
- Calor si la temperatura es mayor a 25°C.
- Agradable si la temperatura está entre 15°C y 25°C, inclusivo.

13

Crea un programa en Python que, dado un año, determine e imprima si el mismo es bisiesto.

- Solicite al usuario que ingrese un año.
- Utilice el condicional "si entonces" para verificar si el año es bisiesto (un año es bisiesto si es divisible por 4, pero no por 100, a menos que sea divisible por 400).
- Imprima "Bisiesto" si el año es bisiesto.
- Imprima "No Bisiesto" si el año no es bisiesto.

14

Construya un programa en Python que, dada la edad de una persona, determine e imprima si la misma es mayor de edad.

- Solicite al usuario que ingrese su edad.
- Utilice el condicional "si entonces" para verificar si la persona es mayor de edad (18 años o más).
- Imprima "Mayor de Edad" si la persona tiene 18 años o más.
- Imprima "Menor de Edad" si la persona tiene menos de 18 años.

15

Construya un programa en Python que, se encargue de comprobar si un número es o no primo. A continuación, imprime los números primos entre 1 y 100

ACTIVIDAD N.5


16

Construya un programa en Python que, dada la temperatura en grados Celsius, verifique si está por debajo de 0 grados. Si es así, convierta la temperatura a grados Fahrenheit y muestre el resultado en la consola. La fórmula para convertir de Celsius a Fahrenheit es:

$$°F = \frac{9}{5} * °C + 32$$

17

Construya un programa en Python que, dada la cantidad de productos vendidos por un vendedor en un mes, aplique un bono del 10% sobre el total de ventas si vendió más de 50 productos. Si vendió 50 productos o menos, aplique un bono del 5%. El programa debe solicitar al usuario ingresar la cantidad de productos vendidos y el precio total de los productos, calcular el bono y mostrar el total de las ventas con el bono incluido en la consola.

18

Construya un programa en Python que, dado un número entero, determine si el número es positivo, negativo o cero. El programa debe solicitar al usuario ingresar un número entero, realizar la verificación correspondiente y mostrar el resultado en la consola.

ACTIVIDAD N.6

19

Construya un programa en Python que, dada una temperatura en grados Celsius, determine e imprima si es fría (menos de 15°C), templada (entre 15°C y 30°C) o caliente (más de 30°C).

- Solicite al usuario que ingrese una temperatura en grados Celsius.
- Utilice el condicional "si entonces / sino" para calificar la temperatura.
- Imprima "Fría" si la temperatura es menos de 15°C.
- Imprima "Templada" si la temperatura está entre 15°C y 30°C.
- Imprima "Caliente" si la temperatura es más de 30°C.

20

Construya un programa en Python que, dada la edad de una persona, determine e imprima la categoría de edad en la que se encuentra la misma.

CATEGORÍAS	
Niño	Menos de 13 años
Adolescente	Entre 13 y 19 años
Joven adulto	Entre 20 y 35 años
Adulto	Entre 36 y 55 años
Adulto mayor	Entre 56 y 70 años
Anciano	Más de 70 años

- Solicite al usuario que ingrese su edad.
- Utilice el condicional "si entonces / sino" para determinar la categoría de edad de la persona.
- Imprima la categoría de edad correspondiente.

ACTIVIDAD N.7

21

Construya un programa en Python que, dado el peso de un paquete en kilogramos, calcule el costo de envío según las siguientes reglas:

- Los primeros 2 kg son gratuitos.
- Los siguientes 3 kg tienen un costo de $5 por kg.
- El resto tiene un costo de $8 por kg, pero si el peso del paquete es mayor a 20 kg, entonces el costo es de $10 por kg.

El programa debe solicitar al usuario ingresar el peso del paquete, calcular el costo de envío y mostrar el precio total del envío (peso total = peso + costo de envío) en la consola.

22

Crea un programa en Python que, dado como datos N números enteros, determine cuántos números negativos hay entre estos números. El programa debe solicitar al usuario ingresar la cantidad de números (N), luego solicitar cada uno de los números, contar cuántos de ellos son negativos y mostrar el resultado en la consola.

23

Construya un programa en Python que, dado como datos 100 números enteros, obtenga la suma de los números positivos y el promedio de los números negativos. El programa debe solicitar al usuario ingresar 100 números, calcular la suma de los positivos y el promedio de los negativos, y mostrar ambos resultados en la consola.

ACTIVIDAD N.8

24

Construya un programa en Python que, dado N números enteros, determine cuántos de ellos son múltiplos de 3, calcule el promedio de los números múltiplos de 3 y el promedio de todos los números. El programa debe solicitar al usuario ingresar la cantidad de números (N), luego solicitar cada uno de los números, realizar los cálculos correspondientes y mostrar los resultados en la consola.

25

Construya un programa en Python que, lea la cantidad de dinero que tiene una persona y determine si puede comprar un artículo cuyo precio es fijo. Si puede comprarlo, debe imprimirse "Compra realizada". Si no puede, debe imprimirse "Fondos insuficientes".

26

Construya un programa en Python que, pregunte al usuario su renta anual y muestre por pantalla el tipo impositivo que le corresponde. Los tramos impositivos para la declaración de la renta en un determinado país son los siguientes:

RENTA	TIPO IMPOSITIVO
Menos de 10000€	5%
Entre 10000€ y 20000€	15%
Entre 20000€ y 35000€	20%
Entre 35000€ y 60000€	30%
Más de 60000€	45%

ACTIVIDAD N.9

27

Construya un programa en Python que, dado el monto de la compra de un cliente y su tipo de membresía (Básica, Plata, Oro), determine el descuento aplicable y lo que el cliente debe pagar. El descuento se efectúa con base en el siguiente criterio:

Descuentos según el monto de la compra:

- **Si el monto es menor que $500:**
  - Membresía Básica: No hay descuento
  - Membresía Plata: 2% de descuento
  - Membresía Oro: 5% de descuento
- **Si el monto está comprendido entre $500 y $1000:**
  - Membresía Básica: 5% de descuento
  - Membresía Plata: 7% de descuento
  - Membresía Oro: 10% de descuento
- **Si el monto está comprendido entre $1000 y $7000:**
  - Membresía Básica: 11% de descuento
  - Membresía Plata: 13% de descuento
  - Membresía Oro: 15% de descuento
- **Si el monto está comprendido entre $7000 y $15000:**
  - Membresía Básica: 18% de descuento
  - Membresía Plata: 20% de descuento
  - Membresía Oro: 22% de descuento
- **Si el monto es mayor a $15000:**
  - Membresía Básica: 25% de descuento
  - Membresía Plata: 27% de descuento
  - Membresía Oro: 30% de descuento

El programa debe incluir las siguientes características:

1. Solicitar al usuario el monto de la compra.
2. Solicitar al usuario su tipo de membresía (Básica, Plata, Oro).
3. Determinar el descuento aplicable.
4. Calcular el monto final a pagar.
5. Imprimir el monto del descuento y el monto final a pagar.

EJEMPLO

RECUERDA LO SIGUIENTE:

- La estructura *repetir*, conocida como *FOR*, es una estructura adecuada para utilizar en un ciclo que se ejecutará un número definido de veces.
- La estructura *mientras*, conocida como *while*, es la estructura para utilizar cuando no sabemos el número de veces que éste se ha de repetir.

0

Construya un programa en Python que, lea 10 números enteros proporcionados por el usuario y calcule la suma y el promedio de esos números.

```
# Inicializar la suma de los números
suma = 0

# Solicitar 10 números al usuario y calcular la suma
for i in range(10):
    numero = int(input(f"Introduce el número {i+1}: "))
    suma += numero

# Calcular el promedio
promedio = suma / 10

# Mostrar la suma y el promedio
print(f"La suma de los 10 números es: {suma}")
print(f"El promedio de los 10 números es: {promedio}")
```

```
Introduce el número 1: 2
Introduce el número 2: 10
Introduce el número 3: 15
Introduce el número 4: 30
Introduce el número 5: 52
Introduce el número 6: 69
Introduce el número 7: 56
Introduce el número 8: 5
Introduce el número 9: 2
Introduce el número 10: 3
La suma de los 10 números es: 244
El promedio de los 10 números es: 24.4
```

EJEMPLO

RECUERDA LO SIGUIENTE:

- La estructura *repetir*, conocida como *FOR*, es una estructura adecuada para utilizar en un ciclo que se ejecutará un número definido de veces.
- La estructura *mientras*, conocida como *while*, es la estructura para utilizar cuando no sabemos el número de veces que éste se ha de repetir.

0

Construya un programa en Python que, solicite al usuario ingresar números enteros positivos hasta que el usuario ingrese un número negativo. Al final, el programa debe imprimir la suma de todos los números positivos ingresados.

```
# Inicializar la suma de los números
suma = 0

# Solicitar números al usuario y sumar los positivos hasta que se ingrese un número negativo
numero = int(input("Introduce un número entero positivo (o un número negativo para terminar): "))

while numero >= 0:
    suma += numero
    numero = int(input("Introduce otro número entero positivo (o un número negativo para terminar): "))

# Mostrar la suma de los números positivos
print(f"La suma de los números positivos ingresados es: {suma}")
```

```
Introduce un número entero positivo (o un número negativo para terminar): 2
Introduce otro número entero positivo (o un número negativo para terminar): 6
Introduce otro número entero positivo (o un número negativo para terminar): 8
Introduce otro número entero positivo (o un número negativo para terminar): -6
La suma de los números positivos ingresados es: 16
```

ACTIVIDAD N.10

28

Construya un programa en Python que, calcule el factorial de un número entero positivo proporcionado por el usuario.

29

Construya un programa en Python que, lea una cadena de texto y cuente cuántas vocales (a, e, i, o, u) contiene.

30

Construya un programa en Python que, imprima la tabla de multiplicar de un número N.

- Solicite al usuario que ingrese el valor de N.
- Utilice la estructura "repetir" para imprimir la tabla de multiplicar de N del 1 al 10.

31

Construya un programa en Python que, cuente hacia atrás desde un número N hasta 1.

- Solicite al usuario que ingrese el valor de N.
- Utilice la estructura "repetir" para contar hacia atrás desde N hasta 1

32

Construya un programa en Python que, calcule la media de N números ingresados por el usuario.

- Solicite al usuario que ingrese el valor de N.
- Utilice la estructura "repetir" para pedir al usuario que ingrese N números.
- Calcule la media de los N números e imprima el resultado.

ACTIVIDAD N.11

33

Construya un programa en Python que, dado el sueldo de cada uno de los trabajadores en una empresa y el número de horas extra trabajadas por cada trabajador, calcule el total de la nómina y el pago total por las horas extra. El pago por hora extra es del 50% adicional sobre el sueldo por hora.

Detalles del Ejercicio:

1. Solicite el sueldo mensual de cada trabajador (suponiendo 10 trabajadores).
2. Solicite el número de horas extra trabajadas por cada trabajador.
3. Calcule el sueldo por hora para cada trabajador.
4. Calcule el pago por horas extra (50% adicional sobre el sueldo por hora).
5. Calcule el total de la nómina incluyendo el pago por horas extra.
6. Imprima el total de la nómina y el pago total por horas extra.

	SUELDO	HORAS EXTRAS	SUELDO POR HORA	PAGO HORAS EXTRAS	NOMINA
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					