PREPA TECMILENIO

Guía para el profesor

Tecnologías de la Información I

BSTI1003/BTTI1004

Índice

Información general del curso

Modalidades

- Clave banner: BSTI1003/BTTI1004
- Modalidad: Presencial

Competencia del curso

Obtener las bases teóricas y prácticas del razonamiento lógico necesarias para incursionar en el mundo de las ciencias de la computación, a través del lenguaje de programación Python, resolviendo retos interactivos que desafían de manera gradual el pensamiento lógico y estructurado.

Metodología

Características del curso

- El curso se imparte con la técnica didáctica de **Aula Invertida.**
- Tiene una competencia y un proyecto final.
- Está conformado por 12 temas que integran su contenido.
- Posee un examen intermedio, correspondiente a los primeros 6 temas.
- Se desarrollan actividades dentro del aula (individuales o en equipo).

Estructura del curso: versión semestral

Semana	Temario	Actividad	Ponderación
1	Algoritmos y programas	Reto 1	3
2	Paradigmas de programación	Reto 2	3
3	Descomposición de problemas	Reto 3	3
4	Integración	Reto 4	3
5	Elementos de una PC	Reto 5	3
6	Lenguajes de Programación	Reto 6	3
7	Temas 1 al 6	Valoración de avance	0
8	Tema 1 al 6	Examen intermedio	15
9	Depuración	Reto 7	4
10	Tipos de Información	Reto 8	4
11	Condicionales IF	Reto 9	4
12	Comparativos Básico	Reto 10	4
13	Operaciones Lógicas	Reto 11	4
14	Comparativos compuestos	Reto 12	4
15	Temas 1 al 12	Reto final	13
16	Temas 1 al 12	Valoración de avance	0
17	Temas 1 al 12	Examen Final	30

*La entrega de valoración de avance es requisito para habilitar el examen.

Estructura del curso: versión semestral

Semana	Temario	Actividad	Ponderación
1	Algoritmos y programas	Reto 1	3
2	Paradigmas de programación	Reto 2	3
3	Descomposición de problemas	Reto 3	3
4	Integración	Reto 4	3
5	Elementos de una PC	Reto 5	3
6	Lenguajes de Programación	Reto 6	8
7	Tema 1 al 6	Examen intermedio	15
8	Depuración	Reto 7	3
9	Tipos de Información	Reto 8	3
10	Condicionales IF	Reto 9	3
11	Comparativos Básico	Reto 10	3
12	Operaciones Lógicas	Reto 11	3
13	Comparativos compuestos	Reto 12	3
14	Temas 1 al 12	Reto final	14
15	Temas 1 al 12	Examen Final	30

*La versión tetramestral no cuenta con valoraciones.

Modelo didáctico

Este curso se desarrolla bajo la metodología de **Aula Invertida**, un enfoque pedagógico que promueve el aprendizaje activo y la autonomía del estudiante. A través de esta metodología, se busca optimizar el tiempo en el aula, permitiendo que los estudiantes lleguen preparados para la aplicación práctica del conocimiento, mientras el docente guía y retroalimenta su aprendizaje en un entorno de interacción y construcción conjunta del conocimiento.

Estrategias didácticas

Los estudiantes revisarán previamente los materiales de estudio, los cuales estarán disponibles en la plataforma de aprendizaje virtual. Durante la sesión presencial, el docente verificará la comprensión del material a través de preguntas o actividades diagnósticas, aclarará dudas y explicará instrucciones clave para la actividad a realizar en el aula. Las actividades prácticas, diseñadas para reforzar los conceptos estudiados, se llevarán a cabo en equipo o de manera individual bajo la supervisión y orientación del docente. Al finalizar, los productos de aprendizaje deberán ser entregados en la plataforma de aprendizaje virtual para su evaluación y retroalimentación.

Modalidad de enseñanza

Este curso se imparte en **modalidad presencial**, lo que permite un aprendizaje dinámico, colaborativo y centrado en la participación activa de los estudiantes en el aula.

Recursos de aprendizaje

Los libros de texto y materiales obligatorios se encuentran detallados en la sección **Bibliografía**. Además, al finalizar cada tema, los estudiantes contarán con un listado de **palabras clave** que les permitirá profundizar en los conceptos a través de la consulta de fuentes confiables en internet y el uso de herramientas de inteligencia artificial para reforzar su comprensión y ampliar su conocimiento.

Evaluación del aprendizaje

El desempeño del estudiante será medido a través de una combinación de **actividades, exámenes, proyectos y retos**, diseñados para evaluar la aplicación del conocimiento y el desarrollo de competencias clave. Todas las actividades cuentan con una **rúbrica de evaluación**, disponible en la plataforma Canvas, y recibirán **retroalimentación directa del docente** para favorecer la mejora continua del aprendizaje. Las fechas y detalles específicos de cada evaluación pueden consultarse en la sección **Calendario del curso**.

Rol del estudiante y del docente

- **Estudiante:** se espera que participe de manera activa en las sesiones presenciales, realice el autoaprendizaje previo mediante la revisión del material, colabore en actividades de equipo y entregue sus productos en tiempo y forma en la plataforma **Canvas**.
- **Docente:** su papel es el de **facilitador** del aprendizaje, brindando explicaciones, resolviendo dudas, proporcionando retroalimentación oportuna y guiando a los estudiantes en la aplicación de los conceptos en el aula.

Dinámica de trabajo y cronograma

El curso sigue una estructura clara y organizada de actividades, evaluaciones y sesiones presenciales. Toda la información sobre la programación de los temas y actividades se encuentra disponible en la sección **Calendario del curso**, donde los estudiantes podrán consultar fechas clave y planificar su aprendizaje de manera efectiva.

Guía de impartición

Para asegurar la correcta implementación del curso, el docente cuenta con una **Guía de impartición**, un documento que contiene información clave sobre la planificación y desarrollo de las sesiones. Esta guía incluye detalles sobre las estrategias de enseñanza recomendadas, actividades sugeridas, criterios de evaluación y lineamientos para la retroalimentación. Su propósito es proporcionar una estructura clara que facilite la impartición del curso y garantice la alineación con los objetivos de aprendizaje establecidos.

Evaluación

Versión semestral

Elemento	Evaluables	Puntos

1	Reto 1	3
2	Reto 2	3
3	Reto 3	3
4	Reto 4	3
5	Reto 5	3
6	Reto 6	3
7	Valoración de avance	0
8	Examen intermedio	15
9	Reto 7	4
10	Reto 8	4
11	Reto 9	4
12	Reto 10	4
13	Reto 11	4
14	Reto 12	4
15	Reto final	13
16	Valoración de avance	0
17	Examen final	30
	Total	100

Versión tetramestral

Elemento	Evaluables	Puntos
1	Reto 1	3
2	Reto 2	3
3	Reto 3	3
4	Reto 4	3
5	Reto 5	3
6	Reto 6	8

7	Examen intermedio	15
8	Reto 7	3
9	Reto 8	3
10	Reto 9	3
11	Reto 10	3
12	Reto 11	3
13	Reto 12	3
14	Reto final	14
15	Examen final	30
	Total	100

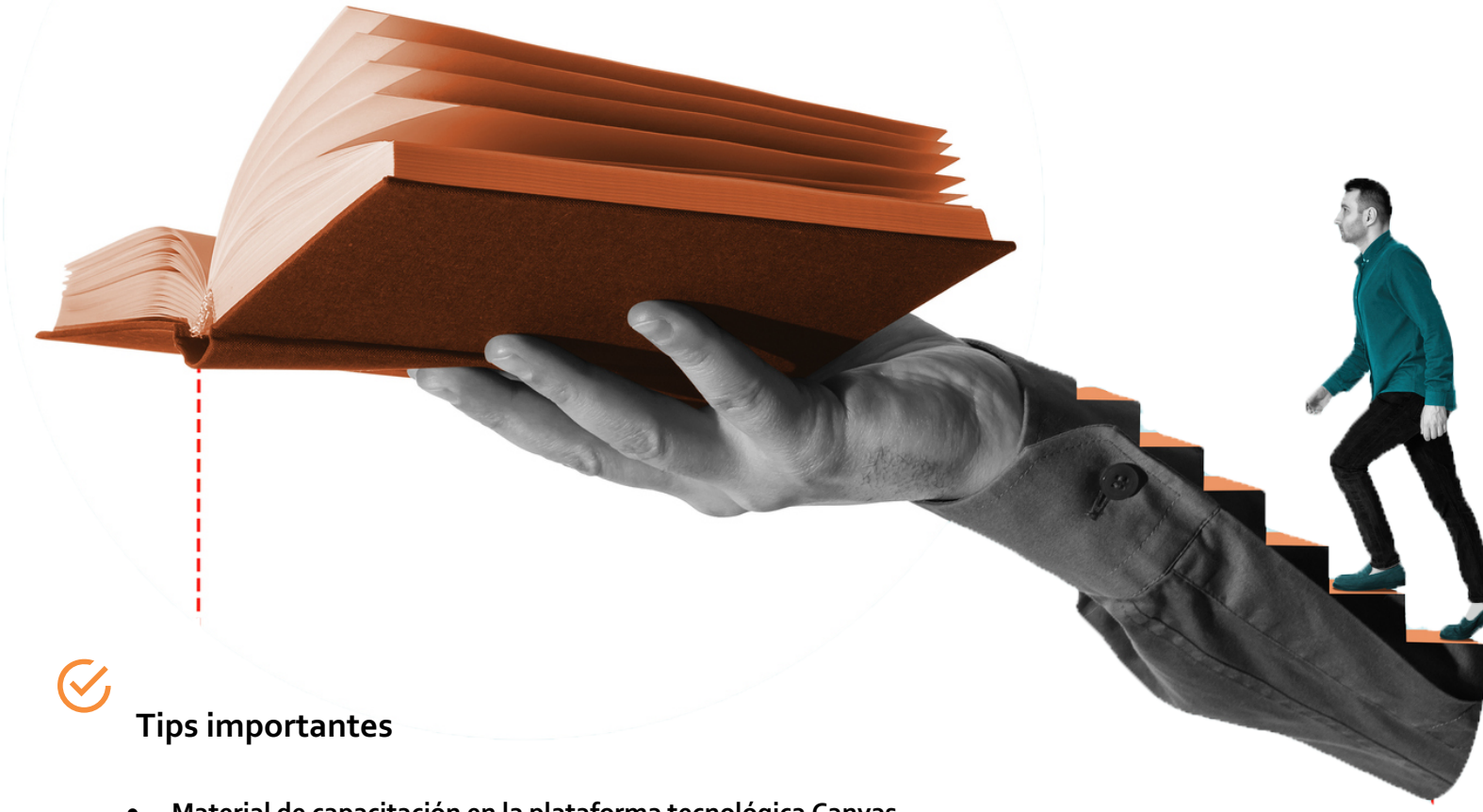
Bibliografía

Libro de apoyo

- Arteaga, M. (2023). *Lógica de programación con Pseint. Enfoque práctico*. Colombia: Fondo Editorial Remington.
- Joyanes, L. (2020). *Fundamentos de Programación: Algoritmos, estructuras de datos y objetos* (5ª ed.). España: McGraw-Hill.

Requisitos especiales

Requisitos especiales	Especificación	Temas en los que se usará
Laboratorio	**Laboratorio (Centro de cómputo)**	Temas 1 a 12



Tips importantes

- **Material de capacitación en la plataforma tecnológica Canvas**
- Tutorial digital para profesores: https://bit.ly/2SbMaNK
- Tutorial digital para alumnos: https://bit.ly/35IBnP6

- **¿En dónde o a quién reporto un error detectado en el contenido del curso?**

Lo puedes reportar a la cuenta atencioncursos@servicios.tecmilenio.mx, pero te pedimos que también reportes sugerencias para el contenido y actividades del curso.

- **¿Quién me informa de la cantidad de sesiones y tiempo de cada sesión en las semanas?**

El coordinador docente te debe proporcionar esta información.

- **¿En qué semanas se aplican los exámenes parciales y el examen final?**

Consulta con tu coordinador docente los calendarios de acuerdo con la modalidad de impartición.

- **¿Tengo que capturar las calificaciones en banner y en la plataforma educativa?**

Sí, es importante que captures calificaciones en la plataforma para que los alumnos estén informados de su avance y reciban retroalimentación de parte tuya de todo lo que realizan en el curso. En banner es el registro oficial de las calificaciones de los alumnos.

Temario

Tema	Título
1	Algorithms and Programs
2	Programming Paradigms
3	Problem Decomposition
4	Integration
5	Computer Components
6	Programming Languages
7	Debugging
8	Data Types
9	IF Statements
10	Basic Comparisons
11	Logical Operations
12	Compound Comparisons

Notas de enseñanza

Tema 1

Comienza explicando de manera clara y concreta qué es un algoritmo, utilizando ejemplos cotidianos como la preparación de un sándwich. Esto facilitará que el estudiante comprenda el concepto desde una perspectiva cercana y práctica.

- Aclara la diferencia entre algoritmo y programa, ilustrando cómo un algoritmo representa una solución lógica y un programa es su implementación mediante un lenguaje que la computadora pueda interpretar.
- Apóyate en recursos audiovisuales para complementar la clase. Puedes proyectarlos en clase o sugerirlos como material de consulta, destacando los aspectos más relevantes antes y después de su visualización.
- Fomenta la participación activa a través de ejercicios prácticos, donde el estudiante represente algoritmos utilizando pseudocódigo y diagramas de flujo. Ofrece retroalimentación puntual, enfocándote en la claridad, lógica y secuencia de las instrucciones.
- Introduce el concepto de codificación como el vínculo entre el algoritmo y su ejecución en un programa, dejando claro que se empleará el lenguaje de programación Python a lo largo del curso.
- Cierra el tema con preguntas de reflexión, por ejemplo:
    - ¿Qué beneficios tiene representar un proceso mediante un algoritmo antes de programarlo?
    - ¿En qué momentos de tu vida cotidiana usas secuencias similares a un algoritmo?
- Verifica que el estudiante logre los siguientes aprendizajes clave:
    - Identificar qué es un algoritmo y distinguirlo de un programa, reconociendo su relevancia en la solución de problemas computacionales.
    - Representar algoritmos mediante pseudocódigo y diagramas de flujo de forma clara, ordenada y lógica.
    - Comprender el concepto de codificación como la traducción de un algoritmo a un lenguaje de programación.

Tema 2

Introduce el tema utilizando una analogía funcional, por ejemplo: construir una aplicación para organizar el tiempo. Esta comparación te permitirá explicar cómo diferentes formas de pensar un problema derivan en distintos enfoques de programación.

- Presenta los cuatro paradigmas de programación de manera clara y comparativa:

  - Programación estructurada: destaca su lógica secuencial y uso de condicionales y bucles.
  - Programación reactiva: explica su uso en entornos dinámicos, como videojuegos o aplicaciones que responden a eventos en tiempo real.
  - Programación orientada a objetos: ilustra con ejemplos cotidianos cómo se modela el mundo real mediante clases y objetos.
  - Programación funcional: subraya la importancia de las funciones puras y la ausencia de efectos secundarios.
- Utiliza el video de TICnoticos como apoyo visual, y promueve el análisis de sus contenidos mediante preguntas guía, por ejemplo: ¿Cuál de los paradigmas presentados te resulta más intuitivo?, ¿qué ventajas y desventajas tiene cada uno?
- Propicia actividades donde se apliquen los principios de cada paradigma en situaciones específicas. Puedes usar ejemplos simples, como diseñar una calculadora, una agenda o un juego, y pedir que se resuelva el mismo problema desde distintos enfoques.
- Fomenta la discusión entre estudiantes sobre cuál paradigma sería más adecuado según el contexto, lo que ayudará a desarrollar pensamiento crítico y toma de decisiones informadas.
- Al concluir el tema, formula preguntas de cierre, como:
  - ¿Qué paradigma se adapta mejor a un entorno cambiante con muchos eventos simultáneos?
  - ¿Por qué resulta útil conocer distintos paradigmas a la hora de programar?
- Asegúrate de que el estudiante logre los siguientes aprendizajes clave:
  - Identificar las características principales de los paradigmas de programación: estructurado, reactivo, orientado a objetos y funcional.
  - Distinguir en qué contextos es más adecuado aplicar cada uno.
  - Aplicar los principios básicos mediante ejemplos y ejercicios

Tema 3

Inicia el tema con un ejemplo práctico y visual, como la construcción de una casa. Este tipo de analogías permite explicar cómo un problema complejo se puede dividir en etapas lógicas y manejables.

- Expón la importancia del enfoque de descomposición, destacando que dividir el problema en partes más pequeñas favorece una solución más clara, ordenada y eficiente. Enfatiza que esta estrategia permite pensar de forma estructurada y planificada antes de comenzar a codificar.
- Guía al estudiante en el análisis y jerarquización de objetivos, comenzando por identificar el propósito general del problema y luego dividiéndolo en metas más específicas. A partir de ello, conduce el diseño de un algoritmo principal y algoritmos secundarios que atiendan tareas concretas.

- Incluye actividades prácticas donde el estudiante aplique la descomposición lógica de tareas, diseñando soluciones algorítmicas paso a paso. Ofrece retroalimentación constructiva, enfocándote en la claridad del pensamiento algorítmico, la estructura y la secuencia lógica.
- Motiva la reflexión al cierre del tema con preguntas como:
    - ¿Cómo te ayuda dividir un problema en partes más pequeñas a encontrar una solución más efectiva?
    - ¿Qué dificultades encontraste al jerarquizar objetivos? ¿Cómo las resolviste?
- Asegúrate de que el estudiante logre los siguientes aprendizajes clave:
    - Comprender la importancia de dividir un problema complejo en partes más pequeñas y organizadas para facilitar su solución.
    - Identificar la jerarquía de objetivos dentro de un problema para establecer un algoritmo principal y sus algoritmos secundarios.
    - Diseñar soluciones algorítmicas estructuradas utilizando la descomposición lógica de tareas.

Tema 4

Comienza la clase con una analogía que facilite la comprensión de la integración de componentes computacionales: por ejemplo, compara una computadora con una orquesta, donde cada instrumento (hardware, software, lógica, programación) debe actuar en armonía para lograr una sinfonía coherente.

Introduce el concepto de integración de algoritmos destacando cómo múltiples instrucciones, que por sí solas resuelven tareas simples, pueden conectarse para abordar problemas complejos. Puedes usar ejemplos como el proceso de comprar en línea (selección de producto, carrito, pago, confirmación), resaltando cómo cada paso representa un algoritmo que se integra en un sistema funcional.

Apóyate en recursos visuales y cronológicos para presentar la evolución de la PC. Invita al grupo a reflexionar sobre cómo han cambiado las computadoras a lo largo del tiempo y cuál ha sido su impacto en la sociedad. Utiliza el video sugerido (“Evolución de las Computadoras”) como disparador de conversación crítica.

Explica de forma clara la arquitectura de Von Neumann utilizando diagramas simples. Relaciónala con los dispositivos actuales, haciendo énfasis en cómo este modelo estructura la forma en que las computadoras almacenan y procesan instrucciones y datos.

Durante la clase, fomenta la participación mediante preguntas abiertas como:

- ¿Qué impacto social crees que ha tenido la evolución de la PC?
- ¿Cómo imaginas la arquitectura de una computadora futura?
- ¿Qué ventajas tiene integrar varios algoritmos frente a programarlos de forma aislada?

Propicia que los estudiantes trabajen en equipo para representar gráficamente la integración de un conjunto de algoritmos en un caso práctico, y luego expliquen su funcionamiento al resto del grupo.

Finaliza con un repaso guiado que refuerce los tres puntos clave:

1. Comprender la forma en que los algoritmos se integran en sistemas más amplios.
2. Identificar los hitos históricos en la evolución de la computadora personal.
3. Explicar la arquitectura de Von Neumann y su relevancia.

Tema 5

Inicia la clase con una pregunta detonante como:
¿Qué crees que sucede dentro de tu computadora cuando presionas el botón de encendido?
Esto activará la curiosidad del estudiante y lo conectará con su experiencia cotidiana.

Divide el tema en dos grandes bloques: hardware y software. Comienza por el hardware explicando los componentes físicos clave (CPU, RAM, disco duro, periféricos), utilizando una computadora abierta o esquemas visuales para ilustrar las partes reales. Acompaña con una analogía práctica: por ejemplo, compara el CPU con el "cerebro" y la RAM con la "mesa de trabajo" de la computadora.

Al abordar el software, establece claramente la diferencia entre el sistema operativo y el software de aplicación. Puedes pedir a los estudiantes que compartan ejemplos de programas que usan a diario (como Word, Excel, Zoom) y que identifiquen si pertenecen a uno u otro tipo.

Refuerza el vínculo entre hardware y software destacando que uno no puede funcionar sin el otro. Usa un ejemplo interactivo: simula un problema común (como que no se detecte un teclado) y discútelo como un grupo, analizando si es un problema de hardware o de software.

Durante la clase, sugiere el video recomendado sobre los componentes internos de una PC, animando al estudiante a anotar las funciones clave de cada parte para discutirlas después.

Finaliza la sesión con preguntas de reflexión como:

- ¿Qué aspectos considerarías al elegir una computadora para tus estudios?
- ¿Cómo podrías diagnosticar un problema básico en tu equipo?
- ¿Qué relación existe entre el sistema operativo y los programas que usas a diario?

Guía a los estudiantes para que logren los siguientes aprendizajes clave:

1. Identificar las principales partes del hardware de una computadora y describir su función dentro del sistema.
2. Distinguir entre hardware y software, comprendiendo cómo interactúan.

Reconocer la importancia del software de sistema y de aplicación en el uso cotidiano de una PC.

Tema 6

Inicia la clase con una pregunta provocadora como:
¿Cómo crees que una computadora 'entiende' que debe abrir una ventana, reproducir música o mostrarte una notificación?
Este tipo de preguntas activa la curiosidad y conecta el contenido con la experiencia diaria del estudiante.

Explica que los lenguajes de programación son herramientas que nos permiten traducir ideas humanas en instrucciones comprensibles para las máquinas. Muestra ejemplos simples de código en Python o Scratch para ilustrar cómo se escriben y ejecutan esas instrucciones.

Divide la sesión en tres bloques didácticos:

1. Definición y propósito de los lenguajes de programación.
2. Sintaxis y su importancia en la precisión del código.
3. Objetos, métodos y argumentos como elementos estructurales.

Utiliza ejemplos visuales como recetas de cocina para explicar la sintaxis: una instrucción mal escrita (como olvidar un ingrediente) impide obtener el resultado esperado. Para los objetos y métodos, usa analogías con objetos reales (por ejemplo, una lámpara como objeto que puede encenderse o apagarse mediante un método).

Propón ejercicios en clase donde el estudiante practique la escritura de instrucciones básicas con estructuras sencillas. Refuerza el aprendizaje visual con fragmentos del video sugerido ("¿Para qué se usa cada lenguaje de programación?") y propón una discusión posterior sobre qué lenguaje consideran más adecuado para ciertas aplicaciones.

Cierra la clase con una reflexión grupal o individual:

- ¿Por qué es importante escribir código con precisión?
- ¿Qué similitudes encuentras entre diferentes lenguajes de programación?
- ¿Cómo aplicarías estos conocimientos para resolver un problema cotidiano?

Asegúrate de que el estudiante logre:

1. Comprender la definición y relevancia de los lenguajes de programación en el desarrollo de software.
2. Identificar la sintaxis básica y su impacto en el correcto funcionamiento del código.
3. Aplicar los conceptos de objetos, métodos y argumentos en estructuras lógicas funcionales.

Tema 7

Inicia la clase con una actividad vivencial: muestra un algoritmo con un error simple (por ejemplo, una suma que da un resultado incorrecto) y plantea la pregunta:
¿Dónde crees que está el problema?

Este ejercicio detonará el pensamiento crítico desde el inicio y conectará con la experiencia real de programar. Define el concepto de depuración como el proceso de encontrar y corregir errores en un programa, pero subraya que va más allá de "arreglar lo que no funciona"; también implica mejorar la eficiencia y estabilidad del código.

Diferencia claramente entre los tipos de errores:

- Errores de sintaxis: errores de escritura que impiden que el programa se ejecute.
- Errores lógicos: el código corre, pero el resultado es incorrecto.
- Errores de rendimiento: el programa funciona, pero utiliza demasiados recursos.

Utiliza ejemplos visuales y fragmentos de código reales. Propón ejercicios donde los estudiantes identifiquen y corrijan errores en pequeños algoritmos, y fomenta que trabajen en pareja para explicarse mutuamente los fallos y cómo los resolvieron.

Cierra con una reflexión guiada que incluya preguntas como:

- ¿Qué aprendiste sobre tu forma de pensar al depurar un programa?
- ¿Cómo podrías aplicar la depuración en otras áreas, como la resolución de problemas personales o académicos?

Asegúrate de que el estudiante logre:

1. Identificar los diferentes tipos de errores que pueden surgir en un algoritmo.
2. Analizar la eficiencia del uso de recursos en un algoritmo.
3. Aplicar técnicas básicas de depuración para mejorar el funcionamiento de un programa.

Tema 8

Comienza la clase con una pregunta que invite a la reflexión y conecte con la vida cotidiana:
¿Sabías que para tu computadora no existe diferencia entre una letra, un número o una palabra, si no le indicas claramente qué es?
Esto servirá para introducir la importancia de representar correctamente la información en programación.

Explica que toda acción informática requiere manejar datos de forma clara y estructurada. Introduce el concepto de variable como un contenedor que almacena información para ser usada por el programa.

Desglosa el contenido en tres partes clave:

1. Representación interna de la información (cómo la computadora interpreta datos).
2. Definición y función de las variables.
3. Tipos de variables más comunes: numéricas (int, float), cadenas de texto (string), lógicas (booleanos), entre otras.

Utiliza ejemplos simples para ilustrar cada tipo:

- Números enteros para contar objetos.
- Cadenas de texto para guardar nombres.
- Booleanos para representar estados: verdadero o falso.

Haz hincapié en que elegir el tipo de variable adecuado mejora la claridad, rendimiento y escalabilidad del programa. Refuerza esto con un ejemplo: ¿qué pasa si guardas un número como texto por error?

Finaliza la clase con preguntas de análisis:

- ¿Por qué es importante declarar correctamente una variable?
- ¿Qué tipo de dato usarías para representar la temperatura? ¿Y para un nombre de usuario?
- ¿Cómo afecta una mala elección de tipo de dato al funcionamiento del algoritmo?

Verifica que el estudiante logre:

1. Comprender cómo se representa internamente la información en una computadora.
2. Identificar qué es una variable y su función en un algoritmo.
3. Distinguir entre los diferentes tipos de variables y saber cuándo utilizar cada una.

Tema 9

Inicia la clase con una situación cotidiana que requiera una decisión:
Si está lloviendo, ¿qué haces? ¿Sales con paraguas o te quedas en casa?

Este tipo de analogía conecta con la estructura básica de una instrucción IF y facilita su comprensión inicial. Explica que las estructuras condicionales permiten que un programa tome decisiones con base en ciertas condiciones, haciendo que el flujo de ejecución se adapte según la lógica del contexto.

Desglosa las principales formas de condicionales:

1. IF THEN – ejecuta una acción si se cumple la condición.
2. IF ELSE – ejecuta una acción si se cumple, y otra si no.
3. IF ELSE IF – evalúa múltiples condiciones de forma secuencial.

Utiliza representaciones visuales (diagramas de flujo o tablas de decisión) para que el estudiante visualice el camino que sigue el programa según las condiciones que se presenten.

Propón ejercicios en los que el estudiante escriba algoritmos sencillos que incluyan decisiones múltiples.
Puedes usar situaciones como:

- Si el estudiante aprueba el examen, muestra "Felicidades"; de lo contrario, muestra "Sigue estudiando".
- Si la temperatura es mayor a 30°C, enciende el ventilador; si es menor, apágalo.

Cierra la clase con preguntas que impulsen la reflexión:

- ¿Cómo cambiaría tu algoritmo si tuviera que tomar más de dos decisiones?
- ¿Por qué es importante que las condiciones estén correctamente escritas?
- ¿Qué errores comunes podrías encontrar al usar condicionales?

Verifica que el estudiante logre los siguientes aprendizajes clave:

1. Comprender la lógica condicional básica (IF, IF-ELSE, IF-ELSE IF).
2. Identificar la sintaxis y estructura correcta de cada tipo de condicional.

Aplicar estructuras condicionales para resolver problemas mediante evaluación de alternativas.

Tema 10

Comienza la clase con una situación práctica que requiera comparar valores:
Si tienes dos ofertas de productos, ¿cómo decides cuál comprar? ¿Comparas precio, calidad, disponibilidad?

Este ejemplo permite introducir el concepto de comparación lógica, que también ocurre en programación para tomar decisiones automatizadas.

Explica que los operadores comparativos básicos (>, <, =) permiten establecer condiciones que determinan el curso de acción de un algoritmo. En otras palabras, ayudan al programa a decidir "qué hacer" según los valores que recibe o procesa.

Presenta cada operador con ejemplos concretos:

- Mayor que (>): "Si el saldo es mayor que $1000, autoriza la transferencia."
- Menor que (<): "Si la temperatura es menor que 10°, enciende la calefacción."
- Igual a (=): "Si la contraseña ingresada es igual a la almacenada, permite el acceso."

Puedes utilizar diagramas de flujo para ilustrar cómo se comporta un programa al evaluar condiciones con operadores comparativos. Anima a los estudiantes a construir su propio ejemplo con cada tipo de comparación.

Reproduce el video recomendado ("Operadores aritméticos y de comparación") para reforzar visualmente el contenido y propicia una conversación posterior:
¿Qué tipo de operador usarías para verificar si un usuario tiene la edad suficiente para acceder a una plataforma?

Proporciona ejercicios en los que los estudiantes escriban pequeñas condiciones usando cada uno de los tres operadores. Por ejemplo:

- Comparar edades de dos personas.
- Verificar si un número ingresado por el usuario es mayor a un límite.
- Validar si una respuesta del usuario es igual a la esperada.

Finaliza la clase con preguntas que favorezcan la reflexión lógica:

- ¿Qué operador usarías si quieres validar que un número no es menor que otro?
- ¿Por qué es importante elegir correctamente el operador en una condición?
- ¿Cómo influye una mala comparación en la ejecución del programa?

Verifica que el estudiante logre:

1. Comprender el significado y uso de los operadores >, < y = en expresiones condicionales.
2. Aplicar operadores comparativos en estructuras if para tomar decisiones lógicas.
3. Distinguir entre los diferentes tipos de comparaciones y cómo afectan el flujo del programa.

Tema 11

Inicia la clase con ejemplos cotidianos que requieran decisiones basadas en múltiples condiciones:
"Saldré si NO llueve o si traigo paraguas."
"Si es lunes Y tengo clase, entonces llevo mi computadora."
Este tipo de situaciones cotidianas permite introducir con claridad los operadores lógicos y su uso en programación.
Explica que las operaciones lógicas permiten combinar condiciones para tomar decisiones más precisas dentro de un programa. Aborda uno a uno los operadores:

- AND (Y): todas las condiciones deben cumplirse.
- OR (O): basta con que una condición se cumpla.
- NOT (NO): invierte el valor lógico.
- XOR (O exclusivo): solo una de las condiciones puede cumplirse, pero no ambas.

Apóyate en tablas de verdad y representaciones visuales que muestren los posibles resultados de combinar condiciones con cada operador. Puedes usar semáforos, horarios de clases o decisiones familiares para que los estudiantes practiquen con ejemplos significativos.
Proyecta el video recomendado ("Operadores Lógicos o Booleanos") y propicia una actividad posterior donde el estudiante construya expresiones lógicas en pseudocódigo. Ejemplo:

- "Si el usuario está registrado Y ha iniciado sesión, permite el acceso."
- "Si el archivo no está vacío, guarda el contenido."

Fomenta ejercicios colaborativos donde los estudiantes propongan escenarios, identifiquen las condiciones, y seleccionen el operador lógico más adecuado para construir la lógica de decisión.
Cierra la clase con preguntas para reforzar el pensamiento lógico:

- ¿En qué casos elegirías usar AND en lugar de OR?
- ¿Cómo influye el operador NOT en una condición?
- ¿Qué pasaría si malinterpretas la lógica de combinación?

Verifica que el estudiante logre:

1. Comprender el funcionamiento de los operadores lógicos básicos: AND, OR, NOT y XOR.
2. Identificar cómo combinar múltiples condiciones lógicas en un algoritmo.
3. Aplicar operadores lógicos en expresiones condicionales para tomar decisiones en distintos escenarios.

Tema 12

Inicia la clase con una situación de la vida real que requiera evaluar rangos o condiciones múltiples. Por ejemplo: Para acceder a una beca necesitas tener entre 18 y 25 años y un promedio mayor o igual a 8.5. ¿Cómo programarías esta condición?
Este ejemplo activa el pensamiento lógico y contextualiza el uso de comparativos compuestos en escenarios reales.

Explica que los **operadores comparativos compuestos** permiten expresar condiciones más detalladas, evaluando valores que se ubican dentro de un intervalo, o que se diferencian de uno específico. Aborda los tres principales:

- Mayor o igual que (>=)
- Menor o igual que (<=)
- Diferente de (≠)

Utiliza esquemas visuales y líneas numéricas para ilustrar cómo se representan los rangos. Invita a los estudiantes a generar ejemplos similares, como:

- "Edad válida para votar: mayor o igual a 18."
- "Calificación aceptable: menor o igual a 10."
- "Entrada inválida: diferente de 'sí' o 'no'."

Explícale al estudiante que en ocasiones es necesario evaluar una condición dentro de otra estructura repetitiva, como un ciclo. El bucle if se refiere al uso de condiciones dentro de bucles para controlar la ejecución de ciertas instrucciones.
Ejemplo práctico:

- En una lista de números, imprimir únicamente los que sean mayores o iguales a 80.

Puedes integrarlo en ejercicios que incluyan un ciclo for y una condición if dentro del mismo. Esto fortalecerá su comprensión de cómo evaluar condiciones múltiples dentro de estructuras iterativas.

Introduce los if anidados como decisiones jerárquicas, es decir, condiciones dentro de otras condiciones. Utiliza ejemplos claros como:

- "Si el usuario es mayor de edad, entonces…
  Si también tiene membresía activa, entonces permitir acceso premium."

Explica que los if anidados permiten evaluar una lógica más profunda cuando una decisión depende del cumplimiento de otra. Anímalos a crear sus propios ejemplos con al menos dos niveles de anidación.

Fomenta ejercicios prácticos en los que se simulen decisiones como validar acceso a una app, clasificar niveles de riesgo, o detectar respuestas incorrectas. Propón desafíos que impliquen detectar errores comunes, como invertir el orden de comparación o usar el operador equivocado.

Cierra la clase con preguntas de análisis:

- ¿Qué operador usarías para verificar si una temperatura está dentro de un rango ideal?

- ¿Cómo puedes representar una condición de exclusión con un comparador "≠"?
- ¿Por qué es importante usar comparativos compuestos y no solo simples?

Verifica que el estudiante logre:

1. Identificar el significado y uso de los operadores >=, <= y ≠.
2. Aplicar condiciones compuestas en estructuras de decisión para evaluar rangos y exclusiones.
3. Analizar escenarios donde estas comparaciones mejoran la precisión de las decisiones en un algoritmo.

Proyecto final

Descripción

Poner en práctica todos los conocimientos adquiridos durante el semestre creando un juego donde el flujo de la conversación cambie dependiendo de las respuestas del usuario.

Requerimientos:

1. Haber completado todas las actividades anteriores en la plataforma Ozaria.
2. Diálogo o libreto de interacción entre al menos dos personajes.
3. Diagrama de flujo para una conversación entre al menos dos personajes:

a. Con al menos 10 posibles rutas en la historia, dependiendo de las respuestas del usuario.
b. Con al menos 10 diálogos de profundidad en cada ruta de la historia.
c. Redactar en una cuartilla, el aprendizaje que le ha dejado la materia.

Instrucciones:

1. Accede a la plataforma Ozaria y comienza el proyecto final: Creador de historias.
2. Codifica la historia creada usando el diagrama de flujo del punto 3 de requerimientos y siguiendo las instrucciones de la plataforma.
3. Prueba tu juego, identifica y corrige los errores de sintaxis de tu código.
4. Revisa que el flujo de tu historia corresponda con el de tu historia original.

Anexo 1

	Niveles de desempeño			
Criterio	**Altamente competente (100% - 86%)**	**Competente (85% - 70%)**	**Aún sin desarrollar la competencia (69% - 0%)**	
1. Desempeño del juego en la plataforma.	**30 a 27**	**26 a 23**	**22 a 0**	30%
	El juego funciona sin errores, tiene tiempo de carga menor a 5 segundos en al menos dos navegadores y se ejecuta sin interrupciones o bloqueos al menos tres veces consecutivas.	El juego funciona con menos de tres errores de sintaxis o ejecución, y es accesible, aunque con algunas limitaciones de velocidad o compatibilidad.	El juego presenta más de tres fallos funcionales o no puede reproducirse en la plataforma.	
2. Diagrama de flujo.	**30 a 27**	**26 a 23**	**22 a 0**	30 %
	El diagrama representa al menos 10 decisiones explícitas, con conectores lógicos, condiciones y salidas bien definidas.	El diagrama tiene al menos 6 decisiones claras y muestra una secuencia lógica en el 80 % de sus rutas.	El diagrama omite más del 30 % de las decisiones o carece de relación lógica en sus secuencias.	
3. Complejidad del juego.	**20 a 18**	**17 a 15**	**14 a 0**	20 %
	Incluye múltiples niveles con condiciones anidadas o al	Incluye una estructura clara con dos rutas diferenciadas y	Contiene una sola ruta o mecánica repetitiva, con	

	menos tres rutas de interacción diferenciadas, con estructura organizada y nombres comprensibles.	lógica básica parcialmente desarrollada.	estructura simple o sin finalizar.	
4. Complejidad de la historia	**20 a 18**	**17 a 15**	**14 a 0**	20 %
	La narrativa incluye introducción, al menos dos desarrollos y un desenlace, con descripciones detalladas en al menos tres escenas y múltiples desenlaces.	La historia sigue una línea lógica con al menos dos desenlaces alternos, aunque con menor profundidad descriptiva.	Incluye una única ruta, sin coherencia entre decisiones o con finales poco diferenciados.	