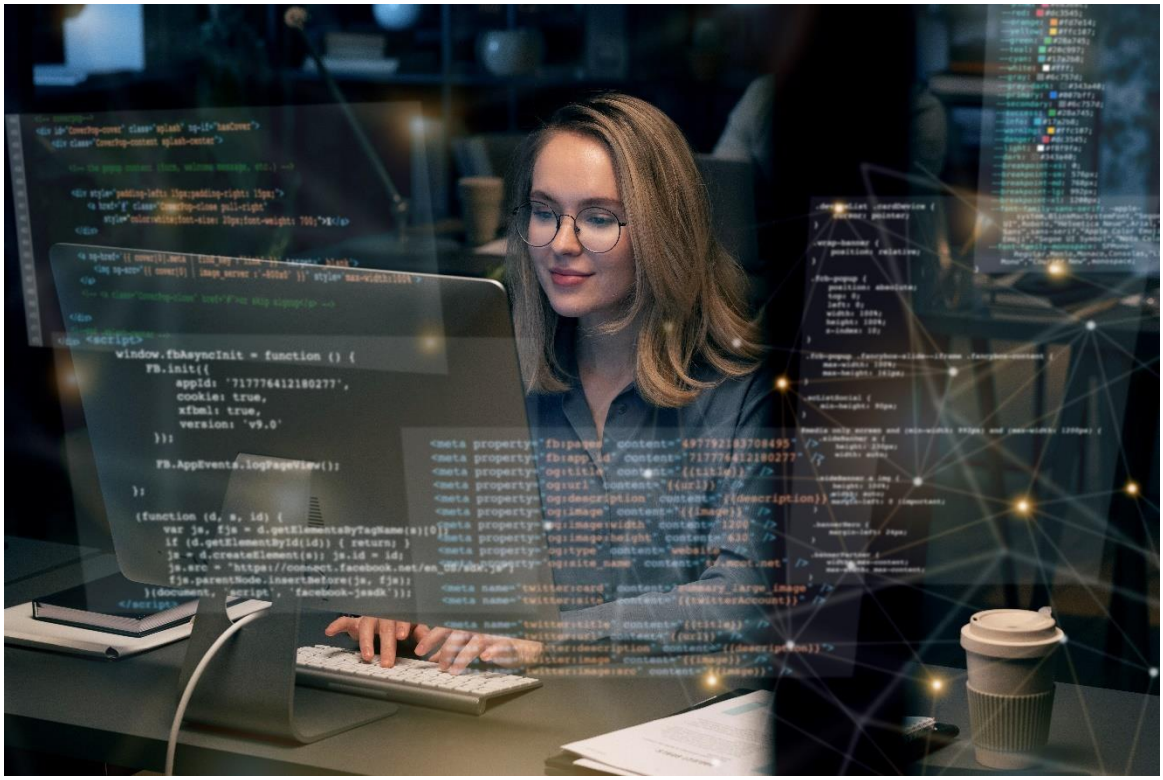
Metodología de la programación



En este epígrafe, se explorarán los conceptos fundamentales de la programación, incluyendo la lógica, la creación de pseudocódigos y ordinogramas, así como la programación orientada a objetos.

1. Lógica de programación.

La lógica de programación es fundamental para el desarrollo de cualquier software. Esta lógica implica el uso de operaciones lógicas y la estructura secuencial de un programa, lo cual permite tomar decisiones, repetir acciones y organizar las instrucciones de manera eficiente. En esta sección, se describirán las operaciones lógicas básicas y cómo se aplican en secuencias de programación para crear programas funcionales y eficientes.

1.1. Descripción y utilización de operaciones lógicas.

Las operaciones lógicas son esenciales para la toma de decisiones dentro de un programa. Estas operaciones permiten evaluar expresiones y ejecutar diferentes bloques de código basados en condiciones específicas. Los principales operadores lógicos son:

- AND (&& en JavaScript, and en Python): Esta operación devuelve verdadero si ambas expresiones evaluadas son verdaderas. Es útil para combinar múltiples condiciones que deben cumplirse simultáneamente.

```javascript
// Ejemplo en JavaScript
let a = true;
let b = true;
console.log(a && b); // true
```

```python
# Ejemplo en Python
a = True
b = True
print(a and b) # True
```

- OR (|| en JavaScript, or en Python): Esta operación devuelve verdadero si al menos una de las expresiones evaluadas es verdadera. Se usa cuando solo una de las condiciones necesita cumplirse.

```javascript
// Ejemplo en JavaScript
let a = true;
let b = false;
console.log(a || b); // true
```

```python
# Ejemplo en Python
a = True
b = False
print(a or b) # True
```

- NOT (! en JavaScript, not en Python): Esta operación invierte el valor de una expresión. Si la expresión es verdadera, NOT devuelve falso, y viceversa. Es útil para negar una condición.

```javascript
// Ejemplo en JavaScript
let a = true;
console.log(!a); // false
```

```python
# Ejemplo en Python
a = True
print(not a) # False
```

A continuación, se presentan ejemplos completos en JavaScript y Python utilizando valores reales para ilustrar cómo se aplican las operaciones lógicas.

En el siguiente ejemplo de JavaScript, el objetivo es determinar si una persona puede obtener un descuento en una tienda en línea. La persona obtiene un descuento si es miembro del programa de fidelidad o si su compra supera los 100 euros. A continuación, se presenta el código en JavaScript que implementa esta lógica:

```javascript
let esMiembro = true; // La persona es miembro del programa de fidelidad
let totalCompra = 120; // La compra total es de 120 euros

// Verificar si la persona obtiene un descuento
if (esMiembro || totalCompra > 100) {
  console.log("La persona obtiene un descuento.");
} else {
  console.log("La persona no obtiene un descuento.");
}
```

- Explicación del código:
  - Se declaran dos variables: esMiembro (un valor booleano que indica si la persona es miembro del programa de fidelidad) y totalCompra (un número que representa el total de la compra).
  - Se utiliza una estructura condicional if para evaluar la condición lógica esMiembro || totalCompra > 100. La operación lógica OR (||) devuelve true si al menos una de las dos condiciones es verdadera.
  - Si la persona es miembro (esMiembro es true) o si el total de la compra es mayor que 100 (totalCompra > 100), se imprime "La persona obtiene un descuento."
  - Si ninguna de las condiciones es verdadera, se imprime "La persona no obtiene un descuento."
- Salida esperada: “La persona obtiene un descuento”.

En el siguiente ejemplo de Python, el problema es determinar si una persona es elegible para votar. Una persona es elegible para votar si tiene al menos 18 años y es ciudadano. A continuación, se presenta el código en Python que implementa esta lógica:

```python
edad = 20  # La persona tiene 20 años
es_ciudadano = True  # La persona es ciudadana

# Verificar si la persona es elegible para votar
if edad >= 18 and es_ciudadano:
    print("La persona es elegible para votar.")
else:
    print("La persona no es elegible para votar.")
```

- Explicación del código:
  - Se declaran dos variables: edad (un número que representa la edad de la persona) y es_ciudadano (un valor booleano que indica si la persona es ciudadana).
  - Se utiliza una estructura condicional if para evaluar la condición lógica edad >= 18 and es_ciudadano. La operación lógica AND (and) devuelve true solo si ambas condiciones son verdaderas.
  - Si la persona tiene al menos 18 años (edad >= 18) y es ciudadana (es_ciudadano es True), se imprime "La persona es elegible para votar."
  - Si alguna de las condiciones no es verdadera, se imprime "La persona no es elegible para votar."
- Salida esperada: “La persona es elegible para votar”.

1.2. Secuencias y partes de un programa.

Un programa se compone de una serie de instrucciones que se ejecutan en una secuencia específica. La estructura de un programa típicamente incluye tres partes principales:

- ► Inicio: La sección donde se inicializan variables y se establecen las condiciones iniciales. Es el punto de partida del programa.
- ► Cuerpo: La sección principal del programa donde se ejecutan las operaciones y se aplican las decisiones lógicas. Aquí se encuentran las funciones, bucles, y demás estructuras de control.
- ► Fin: La sección donde se finaliza el programa, se liberan recursos y se realizan operaciones de limpieza si es necesario.

A continuación, se presentan ejemplos de secuencias simples en pseudocódigo y en JavaScript:

Ejemplo en pseudocódigo:

```
INICIO
  Declarar variable x como entero
  Asignar valor 10 a x
  Si x es mayor que 5 entonces
    Imprimir "x es mayor que 5"
  Fin Si
FIN
```

Ejemplo en JavaScript:

```javascript
// Inicio
let x = 10;

// Cuerpo
if (x > 5) {
  console.log("x es mayor que 5");
}

// Fin
```

Actividad 1

Realiza las siguientes actividades que involucran el uso de operaciones lógicas para tomar decisiones dentro de un programa. Implementa el código siguiendo las indicaciones y prueba diferentes casos para verificar su correcto funcionamiento.

Ejercicio 1: JavaScript

Determina si una persona puede obtener un descuento en una tienda en línea. Para ello:

Declara dos variables: esMiembro y totalCompra.

Implementa una estructura condicional que utilice la operación lógica OR (||) para evaluar si la persona es miembro del programa de fidelidad o si su compra supera los 100 euros.

Imprime "La persona obtiene un descuento." si alguna de las condiciones es verdadera, de lo contrario imprime "La persona no obtiene un descuento."

Ejercicio 2: Python

Determina si una persona es elegible para votar. Para ello:

Declara dos variables: edad y es_ciudadano.

Implementa una estructura condicional que utilice la operación lógica AND (and) para evaluar si la persona tiene al menos 18 años y es ciudadana.

Imprime "La persona es elegible para votar." si ambas condiciones son verdaderas, de lo contrario imprime "La persona no es elegible para votar."

Actividades adicionales:

Modifica los valores de las variables en ambos ejercicios y observa cómo cambia la salida. Explica por qué el programa produce la salida que observas.

Agrega comentarios en tu código para describir lo que hace cada parte de este.

2. Ordinogramas.

Los ordinogramas son herramientas visuales que ayudan a diseñar y entender la lógica de un programa antes de escribir el código. En esta sección, se describirá cómo crear y utilizar ordinogramas para representar operaciones y secuencias en un programa.

2.1. Descripción de un ordinograma.

Un ordinograma, también conocido como diagrama de flujo, es una representación gráfica de un algoritmo o proceso. Utiliza diversos símbolos estandarizados para ilustrar las diferentes etapas y decisiones en un programa. Los ordinogramas ayudan a visualizar la lógica de un programa, identificar posibles errores y optimizar el flujo de trabajo.

2.2. Elementos de un ordinograma.

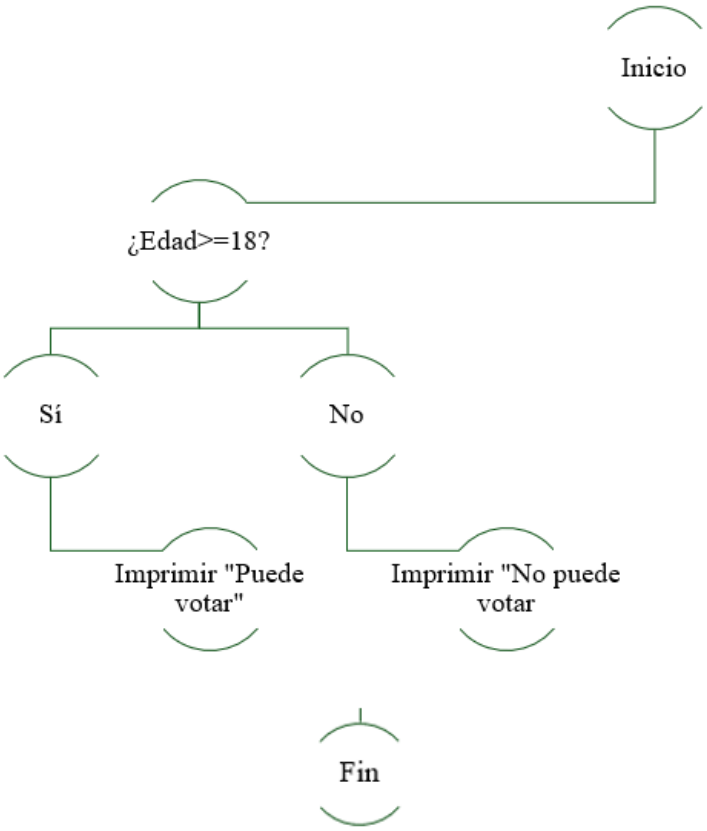
Un ordinograma se compone de varios elementos básicos que representan diferentes partes del proceso. Estos elementos son esenciales para construir un diagrama claro y funcional que describa la lógica de un programa.

Los elementos básicos de un ordinograma son los siguientes:

- → Inicio y fin: Señalan el punto de partida y el punto de conclusión del proceso.
- → Proceso: Indica una operación que modifica el estado del sistema.
- → Decisión: Plantea una pregunta con dos posibles respuestas (sí o no), determinando la bifurcación del flujo.
- → Entrada/Salida: Denota la entrada de datos en el sistema o la salida de datos del mismo.
- → Conector: Establece la unión entre diferentes partes del diagrama, facilitando su continuidad en casos complejos.
- → Líneas de flujo: Muestran la dirección en la que avanza el proceso, permitiendo seguir la secuencia de operaciones de manera clara.

Además, un ordinograma puede incluir anotaciones adicionales para proporcionar contexto o detalles específicos de cada paso, así como subdivisiones que ayudan a desglosar procesos más complejos en partes manejables.

A continuación, se presenta un ejemplo gráfico de un ordinograma que muestra el proceso de verificación de edad para determinar si una persona puede votar:



2.3. Operaciones en un programa.

Las operaciones en un programa pueden ser de tipo lógico o aritmético. Los ordinogramas representan visualmente estas operaciones, facilitando la comprensión y el diseño de la lógica del programa. Ambas pueden ser representadas en un ordinograma para mejorar la comprensión del flujo de un programa. A continuación, se describe cada tipo:

Operaciones lógicas: Estas operaciones se emplean para realizar comparaciones y tomar decisiones dentro de un programa. Los resultados de estas operaciones son verdadero o falso. Las operaciones lógicas básicas incluyen:

- AND: Es verdadero si ambas condiciones son verdaderas.
- OR: Es verdadero si al menos una de las condiciones es verdadera.
- NOT: Invierte el valor de verdad de la condición.

En un ordinograma, estas operaciones suelen representarse en un bloque de decisión.

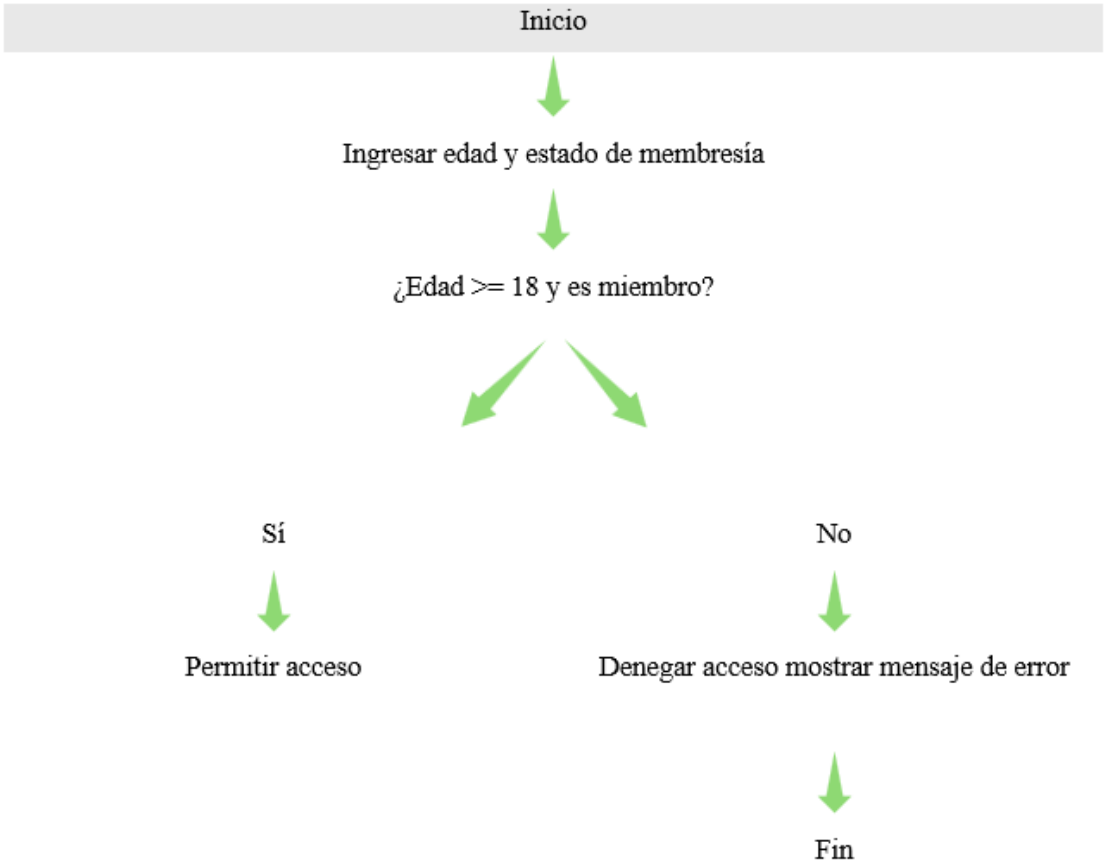
Operaciones aritméticas: Estas operaciones se utilizan para realizar cálculos matemáticos. Las operaciones aritméticas básicas incluyen:

- Suma (+): Adición de dos números.
- Resta (-): Sustracción de un número a otro.
- Multiplicación (*): Producto de dos números.
- División (/): Cociente de un número dividido por otro.
- Módulo (%): Residuo de una división.

En un ordinograma, estas operaciones suelen representarse en un bloque de proceso.

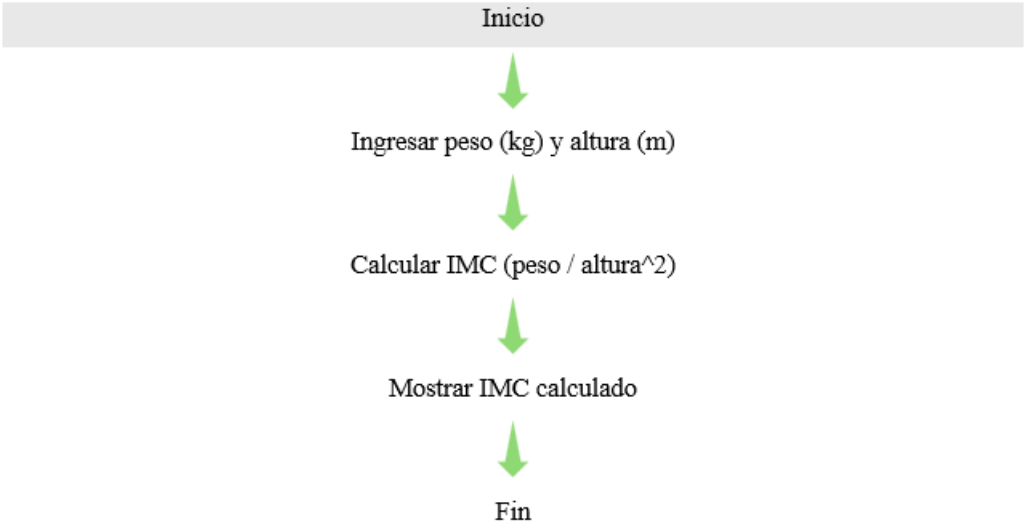
A continuación, se presentan ejemplos de operaciones lógicas y aritméticas representadas en ordinogramas.

Un ejemplo común de operación lógica es la verificación de múltiples condiciones para determinar el flujo del programa. En este caso va a ser la verificación de acceso basado en edad y membresía:



- Descripción del ejemplo:
  - Inicio: El proceso comienza.
  - Proceso: Se ingresan la edad y el estado de membresía.
  - Decisión: Se verifica si la edad es mayor o igual a 18 años y si la persona es miembro.
  - Resultado de la decisión:
    - Sí: Si ambas condiciones son verdaderas, se permite el acceso.
    - No: Si alguna condición es falsa, se deniega el acceso y se muestra un mensaje de error.
  - Fin: El proceso termina.

Un ejemplo común de operación aritmética es el cálculo de un valor basado en la entrada del usuario. En este caso se va a efectuar el cálculo del IMC (Índice de Masa Corporal):



- Descripción del ejemplo:
    1. Inicio: El proceso comienza.
    2. Proceso: Se ingresan el peso y la altura del usuario.
    3. Proceso: Se calcula el IMC utilizando la fórmula peso dividido por altura al cuadrado.
    4. Proceso: Se muestra el IMC calculado.
    5. Fin: El proceso termina.

2.4. Implementación de elementos y operaciones en un ordinograma.

La implementación de elementos y operaciones en un ordinograma permite convertir el pseudocódigo en una representación visual que facilita la comprensión y el diseño del programa. A continuación, se describen los pasos detallados para lograr esta conversión.

Cada ordinograma debe comenzar y terminar con un símbolo de inicio y fin, representados por figuras etiquetadas como "Inicio" y "Fin". Las declaraciones de variables y las asignaciones de valores se representan cada uno conteniendo una sola operación o instrucción. Las decisiones que contienen una pregunta o condición se representan con dos flechas para indicar los posibles resultados: verdadero (Sí) y falso (No).

Dependiendo del resultado de la condición, cada ruta de salida lleva a una acción diferente basada en la evaluación de la condición. Las flechas conectan todos los símbolos, indicando el flujo de control desde el inicio hasta el fin del proceso, asegurando que el flujo lógico del programa se siga correctamente.

A continuación, se exponen estos pasos utilizando un ejemplo.

Pseudocódigo:

```
INICIO
  Declarar variable x como entero
  Asignar valor 10 a x
  Si x es mayor que 5 entonces
    Imprimir "x es mayor que 5"
  Sino
    Imprimir "x no es mayor que 5"
  Fin Si
FIN
```

Conversión a un ordinograma:

- Inicio: El ordinograma comienza con un óvalo etiquetado como "Inicio".
- Declaración: Un rectángulo se usa para la declaración de la variable x y otro para asignar el valor 10 a x.
- Decisión: Un rombo contiene la condición x > 5.
- Resultado de la decisión:
- Si la condición es verdadera (Sí), un rectángulo indica la operación "Imprimir 'x es mayor que 5'".
- Si la condición es falsa (No), otro rectángulo indica la operación "Imprimir 'x no es mayor que 5'".
- Fin: El ordinograma concluye con un óvalo etiquetado como "Fin".

Actividad 2

Relaciona los siguientes términos con sus definiciones:

1. AND
2. OR
3. Suma (+)
4. Resta (-)
5. Multiplicación (*)
6. División (/)
7. Módulo (%)
8. Inicio
9. Fin
10. Decisión

A. Cociente de un número dividido por otro.
B. Es verdadero si ambas condiciones son verdaderas.
C. El ordinograma concluye con este símbolo.
D. Residuo de una división.
E. Sustracción de un número a otro.
F. Un rombo contiene esta operación en el ordinograma.
G. Producto de dos números.
H. Es verdadero si al menos una de las condiciones es verdadera.
I. El ordinograma comienza con este símbolo.
J. Adición de dos números.

3. Pseudocódigos.

Los pseudocódigos permiten representar la lógica de programación de manera estructurada y comprensible antes de escribir el código real. En esta sección, se enseñará cómo describir y crear pseudocódigos, acompañados de ejemplos prácticos.

3.1. Descripción de pseudocódigo.

El pseudocódigo es una forma de describir un algoritmo utilizando una notación informal que combina elementos del lenguaje natural y de la programación. No sigue la sintaxis estricta de un lenguaje de programación específico, lo que lo hace fácil de leer y entender tanto para programadores como para personas sin experiencia en programación.

Al no depender de la sintaxis de un lenguaje de programación específico, el pseudocódigo es más fácil de leer y entender. Además, puede adaptarse fácilmente a cualquier lenguaje de programación, facilitando la traducción del pseudocódigo a código real.

También mejora la comunicación entre los miembros del equipo de desarrollo, ya que todos pueden entender la lógica del programa sin importar su experiencia con un lenguaje de programación particular. Y, ayuda a planificar y estructurar la lógica de un programa antes de comenzar a codificar, lo que puede reducir errores y aumentar la eficiencia.

3.2. Creación del pseudocódigo.

El pseudocódigo es flexible y su formato puede variar dependiendo de las convenciones del equipo o del individuo que lo escribe. Su objetivo principal es describir la lógica del programa de manera clara y comprensible. No está destinado a ser ejecutado en una computadora, sino a ser leído por un humano.

En este apartado se exponen algunos ejemplos de pseudocódigo para resolver problemas comunes.

Ejemplo 1: Algoritmo para encontrar el mayor de dos números

Problema: Se necesita determinar cuál de los dos números ingresados es el mayor.

Pseudocódigo:

```
INICIO
  Declarar variables num1 y num2
  Leer num1 y num2
  Si num1 es mayor que num2 entonces
    Imprimir "num1 es mayor que num2"
  Sino
    Imprimir "num2 es mayor que num1"
  Fin Si
FIN
```

- Descripción del pseudocódigo:
  - Inicio: El algoritmo comienza.
  - Declaración y lectura: Se declaran dos variables, num1 y num2, y se leen sus valores.
  - Condición: Se evalúa si num1 es mayor que num2.
  - Resultado: Se imprime cuál número es mayor basado en la condición.
  - Fin: El algoritmo termina.

Ejemplo 2: Algoritmo para calcular la suma de los primeros N números naturales

Problema: Calcular la suma de los primeros N números naturales dados por el usuario.

Pseudocódigo:

```
INICIO
  Declarar variable N
  Leer N
  Declarar variable suma y asignar 0
  Para i desde 1 hasta N hacer
    suma = suma + i
  Fin Para
  Imprimir "La suma de los primeros N números naturales es: " + suma
FIN
```

- Descripción del pseudocódigo:
  - Inicio: El algoritmo comienza.
  - Declaración y lectura: Se declara la variable N y se lee su valor.
  - Inicialización: Se declara la variable suma y se inicializa a 0.
  - Bucle Para: Un bucle se ejecuta desde 1 hasta N, sumando cada valor a suma.
  - Resultado: Se imprime el resultado de la suma.
  - Fin: El algoritmo termina.

4. Objetos.

Los objetos son componentes fundamentales en la programación orientada a objetos, que permiten modularizar y reutilizar el código. A continuación, se describirán los conceptos, funciones, comportamientos, y atributos de los objetos, así como la creación de los mismos.

4.1. Descripción de objetos.

Un objeto es una instancia de una clase que encapsula datos y comportamientos relacionados. Los datos se representan mediante atributos (también conocidos como propiedades), y los comportamientos se implementan a través de métodos (también conocidos como funciones). Los objetos permiten organizar el código en unidades coherentes y reutilizables.

Las principales características de los objetos son las siguientes:

- Encapsulamiento: Agrupa datos y métodos que operan sobre esos datos dentro de una misma unidad, protegiendo los datos del acceso directo desde el exterior.
- Herencia: Permite que los objetos hereden propiedades y métodos de otras clases, facilitando la reutilización y extensión del código.
- Polimorfismo: Habilidad de los objetos para ser tratados como instancias de su clase base, permitiendo que el mismo método pueda tener diferentes comportamientos en distintas clases derivadas.
- Abstracción: Oculta los detalles complejos del sistema, mostrando solo las características esenciales y relevantes para el contexto de uso.

Un ejemplo en JavaScript podría ser el siguiente:

```javascript
// Definición de una clase Persona
class Persona {
  constructor(nombre, edad) {
    this.nombre = nombre; // Atributo nombre
    this.edad = edad; // Atributo edad
  }

  // Método para mostrar información de la persona
  mostrarInformacion() {
    console.log(`Nombre: ${this.nombre}, Edad: ${this.edad}`);
  }
}

// Creación de un objeto persona
const persona1 = new Persona("Beatriz", 30);
persona1.mostrarInformacion(); // Salida: Nombre: Beatriz, Edad: 30
```

El ejemplo en Python sería el siguiente:

```python
# Definición de una clase Persona
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre # Atributo nombre
        self.edad = edad # Atributo edad

    # Método para mostrar información de la persona
    def mostrar_informacion(self):
        print(f"Nombre: {self.nombre}, Edad: {self.edad}")

# Creación de un objeto persona
persona1 = Persona("Beatriz", 30)
persona1.mostrar_informacion() # Salida: Nombre: Beatriz, Edad: 30
```

4.2. Funciones de los objetos.

Los métodos son funciones definidas dentro de una clase que describen los comportamientos que los objetos pueden realizar. Los métodos permiten manipular los atributos del objeto y proporcionar funcionalidades específicas.

Los tipos de métodos son los siguientes:

- Métodos de instancia:
  Operan sobre instancias individuales de una clase. Utilizan los datos contenidos en el objeto y pueden modificar sus atributos. Estos métodos son invocados sobre objetos específicos y tienen acceso directo a los atributos de la instancia a través de la palabra clave this en JavaScript o self en Python.

  Por ejemplo, en una clase "Coche", un método de instancia como acelerar puede incrementar el atributo velocidad de un objeto específico de la clase "Coche":

```javascript
class Coche {
    constructor(marca, modelo, velocidad) {
      this.marca = marca;
      this.modelo = modelo;
      this.velocidad = velocidad;
    }

    // Método de instancia para acelerar el coche
    acelerar(incremento) {
      this.velocidad += incremento;
    }
  }

  const miCoche = new Coche("Toyota", "Corolla", 0);
  miCoche.acelerar(50); // Incrementa la velocidad del objeto miCoche
```

- Métodos de clase:
  Están asociados a la clase en sí misma y no a instancias individuales. Estos métodos se utilizan para operaciones que son relevantes para la clase como un todo y se declaran con la palabra clave static en algunos lenguajes de programación. Los métodos de clase no pueden acceder a los atributos de instancia, pero pueden acceder a los atributos estáticos de la clase.

Por ejemplo, en una clase "Coche", un método de clase como crearCochePorDefecto puede crear y devolver una nueva instancia de la clase con valores predefinidos:

```
class Coche {
    constructor(marca, modelo, velocidad) {
      this.marca = marca;
      this.modelo = modelo;
      this.velocidad = velocidad;
    }

    // Método de clase para crear un coche con valores predefinidos
    static crearCochePorDefecto() {
      return new Coche("Genérico", "Modelo", 0);
    }
  }

  const cochePorDefecto = Coche.crearCochePorDefecto();
```

- Métodos estáticos:
  Definidos con la palabra clave static en algunos lenguajes de programación. Estos métodos no pueden acceder a los atributos de instancia ni a los métodos de clase, pero son útiles para realizar tareas comunes relacionadas con la clase. Los métodos estáticos operan independientemente de las instancias de la clase y se llaman directamente desde la clase.

  Por ejemplo, en una clase "Coche", un método estático como convertirKmAHr puede convertir una velocidad de kilómetros por hora a millas por hora:

```
class Coche {
    constructor(marca, modelo, velocidad) {
      this.marca = marca;
      this.modelo = modelo;
      this.velocidad = velocidad;
    }

    // Método estático para convertir km/h a mph
    static convertirKmAHr(velocidadKm) {
      return velocidadKm * 0.621371;
    }
  }

  const velocidadEnMillas = Coche.convertirKmAHr(100); // Conversión de 100 km/h a mph
```

4.3. Comportamientos de los objetos.

El comportamiento de un objeto se refiere a las acciones que puede realizar, las cuales se definen a través de sus métodos. Los métodos son funciones asociadas a un objeto y operan sobre sus atributos, permitiendo modificar su estado o realizar tareas específicas.

Los métodos se declaran dentro de la clase del objeto y se utilizan para implementar la lógica necesaria para las operaciones del objeto. Por ejemplo, en un objeto "Coche", los métodos podrían incluir acciones como acelerar, frenar, y girar. Estos métodos modifican los atributos del objeto, como la velocidad o la dirección, reflejando los cambios en su estado.

En un objeto "Coche", el método acelerar podría incrementar el atributo velocidad, mientras que el método frenar podría reducirlo. Esto permite controlar el estado del coche de manera intuitiva y organizada.

Actividad 3

¿Qué es el encapsulamiento y cómo se relaciona con los objetos en la programación orientada a objetos?

¿Qué es el polimorfismo y cómo se utiliza en la programación orientada a objetos?

¿Cuál es la diferencia entre métodos de instancia, métodos de clase y métodos estáticos en la programación orientada a objetos?

4.4. Atributos de los objetos.

Los atributos de un objeto son variables que almacenan su estado. Cada atributo tiene un nombre y un valor, y se accede a ellos utilizando la notación de punto (.) en la mayoría de los lenguajes de programación orientada a objetos.

Los atributos se definen dentro de una clase y se inicializan cuando se crea un objeto de esa clase. Estos atributos pueden representar diversas propiedades del objeto, como su nombre, su tamaño o su estado. Por ejemplo, un objeto "Persona" puede tener atributos como nombre y edad.

En un objeto "Persona", nombre y edad serían atributos que almacenan el nombre y la edad de la persona. Estos valores pueden ser modificados o utilizados por los métodos de la clase.

4.5. Creación de objetos.

La creación de un objeto se realiza mediante la instanciación de una clase. Esto implica llamar al constructor de la clase, que es un método especial utilizado para inicializar los atributos del objeto y prepararlo para su uso. El funcionamiento es el siguiente:

- Definición de la clase: Primero, se define la clase que describe el objeto. Esta clase incluye los atributos y métodos que caracterizan al objeto.
- Constructor: La clase tiene un constructor, que es una función especial utilizada para inicializar los atributos del objeto cuando se crea una nueva instancia de la clase.
- Instanciación: Para crear un objeto, se llama al constructor de la clase, pasando los valores iniciales necesarios para los atributos.

En una clase "Coche", el constructor podría inicializar atributos como marca, modelo y velocidad. Al crear un objeto "Coche" usando el constructor, se asignan valores específicos a estos atributos, configurando el estado inicial del objeto.

```javascript
// Definición de la clase Coche
class Coche {
    constructor(marca, modelo, velocidad) {
      this.marca = marca;
      this.modelo = modelo;
      this.velocidad = velocidad;
    }

    // Método para acelerar el coche
    acelerar(incremento) {
      this.velocidad += incremento;
    }

    // Método para frenar el coche
    frenar(decremento) {
      this.velocidad -= decremento;
      if (this.velocidad < 0) this.velocidad = 0;
    }
  }

  // Creación de un objeto de la clase Coche
  const miCoche = new Coche("Toyota", "Corolla", 0);
```

En este ejemplo se muestra cómo se define una clase, se utilizan atributos para almacenar el estado del objeto, y se crean métodos para definir su comportamiento. La instanciación de la clase "Coche" crea un nuevo objeto con los valores iniciales especificados, listo para interactuar mediante sus métodos.

5. Ejemplos de códigos en diferentes lenguajes.

Se presentarán ejemplos prácticos de códigos en distintos paradigmas de programación, incluyendo lenguajes estructurales, de scripts, y orientados a objetos. Esto ayudará a comprender cómo se aplican los conceptos aprendidos en diversos contextos.

5.1. Códigos en lenguajes estructurales.

Los lenguajes estructurales se caracterizan por la organización del código en bloques lógicos, utilizando estructuras de control como bucles y condicionales. Un ejemplo típico es el lenguaje C.

Un ejemplo en C es el siguiente:

```c
#include <stdio.h>

int main() {
    int i, n, sum = 0;

    printf("Ingrese un número: ");
    scanf("%d", &n);

    for(i = 1; i <= n; ++i) {
        sum += i;
    }

    printf("La suma de los primeros %d números naturales es: %d\n", n, sum);
    return 0;
}
```

- Descripción:
  - Se solicita al usuario que ingrese un número.
  - Se utiliza un bucle for para sumar los primeros n números naturales.
  - Se muestra el resultado en pantalla.

5.2. Códigos en lenguajes scripts.

Los lenguajes de scripts se utilizan para automatizar tareas dentro de aplicaciones más grandes. Un ejemplo común es JavaScript, que se usa ampliamente en desarrollo web.

Un ejemplo en JavaScript es el siguiente:

```javascript
function calcularFactorial(n) {
    if (n === 0 || n === 1) {
        return 1;
    }
    return n * calcularFactorial(n - 1);
}

let numero = 5;
console.log(`El factorial de ${numero} es ${calcularFactorial(numero)}`);
```

- Descripción:
  - Se define una función recursiva calcularFactorial para calcular el factorial de un número.
  - Se invoca la función con un valor específico y se muestra el resultado en la consola.

5.3. Códigos en lenguajes orientados a objetos.

Los lenguajes orientados a objetos se basan en la creación y manipulación de objetos. Un ejemplo típico es Python.

Un ejemplo en Python es el siguiente:

```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        return f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años."

persona1 = Persona("Ana", 30)
print(persona1.saludar())
```

- Descripción:
  - Se define una clase Persona con un constructor que inicializa los atributos nombre y edad.
  - Se incluye un método saludar que devuelve una cadena con el saludo.
  - Se crea una instancia de la clase y se llama al método saludar.

Actividad 4

Define una clase "Libro" que tenga los atributos "título", "autor" y "año_publicación". El constructor de la clase debe inicializar estos atributos. Además, define un método "descripcion" que devuelva una cadena con el formato "Título: [título], Autor: [autor], Año: [año_publicación]". Luego, crea una instancia de la clase con valores específicos para los atributos y llama al método "descripcion" para mostrar la información del libro.

---

6. Prueba de autoevaluación.

¿Cuál es el principal objetivo de la lógica de programación?

a) Crear interfaces de usuario atractivas.

b) Organizar las instrucciones de manera eficiente para tomar decisiones y repetir acciones.

c) Generar gráficos en programas de diseño.

¿Qué operador lógico se utiliza en Python para invertir el valor de una expresión?

a) AND

b) OR

c) NOT

¿Cuál es el símbolo que representa el punto de partida en un ordinograma?

a) Proceso

b) Inicio

c) Conector

¿Qué elemento de un ordinograma indica la entrada o salida de datos?

a) Proceso

b) Decisión

c) Entrada/Salida

En la programación orientada a objetos, ¿qué término describe la agrupación de datos y métodos dentro de una misma unidad?

a) Herencia

b) Polimorfismo

c) Encapsulamiento

Las operaciones lógicas básicas incluyen AND, OR y ____.

Un ____ es una representación gráfica de un algoritmo o proceso.

En la programación orientada a objetos, un ____ es una instancia de una clase que encapsula datos y comportamientos relacionados.

El pseudocódigo es una forma de describir un algoritmo utilizando una notación informal que combina elementos del lenguaje natural y de la ____.

Los métodos de instancia en una clase se utilizan para operar sobre ____ individuales de esa clase.

Lenguaje de guion