Comparaciones de igualdad

Ideas clave:

- Podemos comparar valores utilizando **operadores de comparación**.
- El operador de igualdad == compara dos valores y nos dice si son iguales o no.

¿Qué es un operador de comparación?

Los operadores de comparación nos permiten comparar valores.

El primer operador de comparación que estudiaremos es el operador de igualdad ==. Este operador compara dos valores:

- Si son iguales, el resultado es **True**.
- Si no son iguales, el resultado es **False**.

```
print(2 == 2) # True (verdadero)
print(2 == 3) # False (falso)
```

Podemos utilizar los operadores de comparación con variables

```
manzanas = 7
naranjas = 5
print(manzanas == naranjas) # False
```

Ejercicio

Utiliza print y el operador de igualdad para comparar si tenemos la misma cantidad de calcetines izquierdos que derechos.

```
calcetinesIzquierdos = 13
calcetinesDerechos = 17
print() # Modifica esta línea
```

Comparaciones de mayor y menor

Ideas clave:

- Podemos comparar valores utilizando operadores de comparación.
- El operador de igualdad == compara dos valores y nos dice si son iguales o no.
- Existen operadores de comparación como mayor que > y menor que <.

Operador de mayor que

El operador **mayor que** > compara dos valores:

- Si el primer valor es mayor que el segundo, el resultado es verdadero o en inglés **True**.
- Si el primer valor no es mayor que el segundo, el resultado es falso en inglés **False**.

```
print(2 > 3) # False
print(3 > 2) # True
```

Operador de menor que

También existe el operador de **menor que** <, el cual compara dos valores, resultando en true si el primer valor es menor que el segundo, y false si no lo es.

```
print(2 < 3) # True
print(3 < 2) # False
```

Ejercicio
Un programador ha escrito el siguiente código para comparar dos números. Sin embargo, el programa no está funcionando correctamente. ¿Puedes corregirlo únicamente cambiando el signo de comparación?

```
a = 4
b = 3
c = a < b  # Modifica esta línea
print("a es mayor que b: " + str(c))
```

Otros tipos de comparadores

Ideas clave

- Podemos comparar valores utilizando operadores de comparación.
- El operador de igualdad == compara dos valores y nos dice si son iguales o no.
- Existen operadores de comparación como mayor que > y menor que <.
- El operador de mayor o igual que >= compara dos valores y nos dice si el primero es mayor o igual al segundo, y el operador de menor o igual que <= compara dos valores y nos dice si el primero es menor o igual al segundo.

- El operador de desigualdad != compara dos valores y nos dice si son diferentes.

Operadores de comparación

Existen varios operadores de comparación que podemos utilizar en Python. A continuación, se muestra una tabla con los operadores de comparación más comunes:

OPERADOR	TIPO DE COMPARACIÓN	EJEMPLO	RESULTADO
==	Igualdad	2 == 2	true
>	Mayor que	3 > 2	true
<	Menor que	2 < 3	true
>=	Mayor o igual que	3 >= 3	true
<=	Menor o igual que	3 <= 3	true
!=	Distinto	1 != 2	true

Ejercicio
Un programador ha escrito esta sección de código para evaluar si tiene suficiente dinero para comprar un producto. Sin embargo, el programa tiene un caso no contemplado. ¿Puedes corregirlo?

Tip: El caso no contemplado puede ser resuelto utilizando uno de los operadores de comparación mencionado en la tabla anterior.

```
dinero = 500
costo = 500
me_alcanza = dinero > costo # Modifica esta línea
print(me_alcanza)
```

Introducción a flujo
Ideas clave:

- Los programas se ejecutan de manera lineal, de arriba a abajo.
- Podemos cambiar el flujo de un programa utilizando condiciones.

¿Qué es el flujo?

El flujo de un programa es el orden en el que se ejecutan las instrucciones. Hasta el momento, los programas que hemos construido han seguido un flujo lineal, lo que significa que las instrucciones se ejecutan en el mismo orden, de la primera línea a la última. Sin embargo, en la mayoría de los programas necesitamos tomar decisiones y ejecutar instrucciones diferentes dependiendo de ciertas condiciones. Podemos manipular el flujo de un programa utilizando la instrucción `if`.

La sintaxis de un `if` es la siguiente:

```
if condición:
  # Código a ejecutar si la condición es verdadera
```

Veamos un ejemplo:

```
edad = 18
if edad >= 18:
  print("Eres mayor de edad")
```

En el código mostrado, la variable `edad` tiene un valor de 18. Luego, se evalúa si la variable `edad` es mayor o igual a 18 y sólo si es así, se ejecuta el código dentro del bloque 'if'.

La indentación (Espaciado / Sangría) previo al print es *obligatoria*, python utiliza esa información para determinar que ese código solo debe ejecutarse si se cumple la condición.

Es importante destacar que, aunque en este ejemplo el valor está asignado directamente a una variable, en la práctica este valor podría provenir de un formulario, un archivo, ser ingresado por el usuario o generado por un número aleatorio.

Con condiciones podemos crear programas que tomen decisiones dependiendo de la información disponible.

Ejercicio

El siguiente programa está mal configurado, alguien cambió la contraseña y ahora no funciona correctamente. Debería mostrar que el código es correcto si la variable `codigo` es igual a "1234". Sin embargo, el programa no muestra nada en absoluto. ¿Puedes corregirlo?

```
codigo = "9371" # Modifica esta línea
if codigo == "1234":
  print("Código correcto")
```

Introducción a diagramas de flujo

Ideas clave:

- Los programas por defecto se ejecutan de manera lineal, de arriba a abajo.
- Podemos cambiar el flujo de un programa utilizando sentencias de control de flujo.
- Los diagramas de flujo son una herramienta visual que nos permite representar el flujo de un programa.

Diagramas de flujo

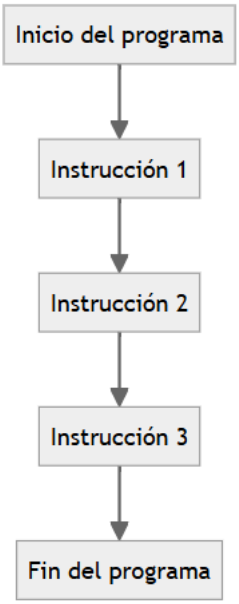
El flujo de un programa es el orden en el que se ejecutan las instrucciones y, si bien podemos seguir el flujo de un programa leyendo el código, a veces es más fácil visualizarlo. Para esto se utilizan los diagramas de flujo, que son una herramienta visual que nos permite representar el flujo de un programa.

Los diagramas de flujo se componen de bloques que representan instrucciones y flechas que indican el flujo de ejecución. Cada bloque tiene una forma específica que indica el tipo de instrucción que representa. Por ejemplo, un bloque rectangular representa una instrucción, un rombo representa una condición y un óvalo representa el inicio o fin del programa.

Código y diagramas de flujo

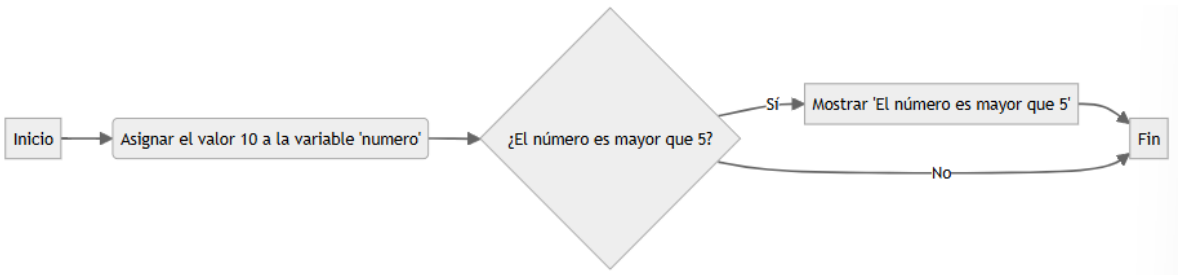
Veamos un ejemplo lineal primero.

```
print("Inicio del programa")
print("Instrucción 1")
print("Instrucción 2")
print("Instrucción 3")
```

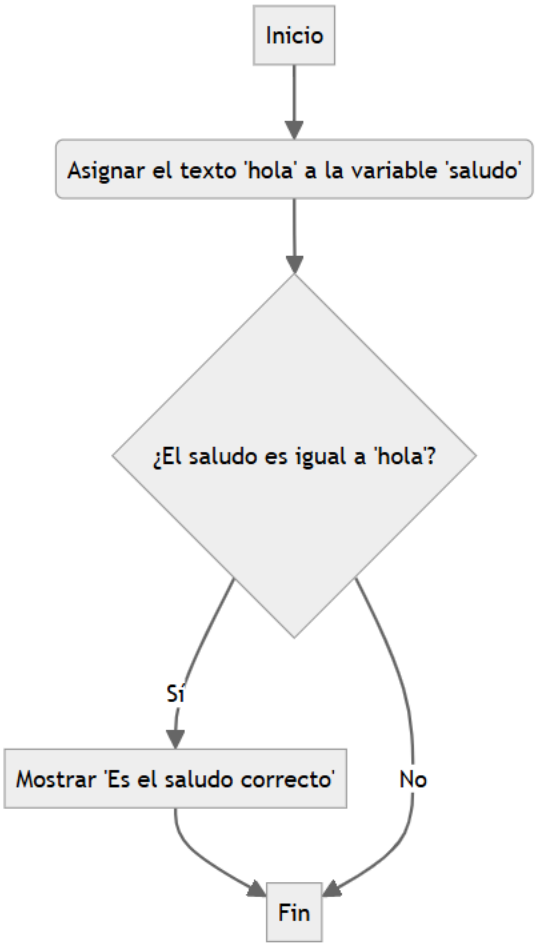


Veamos un ejemplo con una condición.

```
numero = 10
if numero > 5:
    print("El número es mayor que 5")
```



Ejercicio
Implementa el siguiente código a partir de un diagrama de flujo.




```
# Escribe tu código aquí
saludo = 'hola'
if saludo == 'hola':
    print('Es el saludo correcto')
# Fin
```

Indentando con espacios

Ideas clave:

- Los programas se ejecutan de manera lineal, de arriba a abajo.
- Podemos cambiar el flujo de un programa utilizando condiciones.
- La indentación es la cantidad de espacios en blanco que hay al principio de una línea de código.
- La indentación en Python indica que una línea de código está dentro de un if. Formalizaremos esta idea más adelante.

Qué es la indentación

La indentación es la cantidad de espacios en blanco que hay al principio de una línea de código. En Python, la indentación es muy importante porque indica que una línea de código está dentro de un if, pero para que funcione correctamente, debemos ser consistentes con el uso de espacios. A continuación, se muestra un ejemplo de código con una indentación inconsistente:

```
codigo = "1234"
if codigo == "1234":
  print("Código correcto")
    print("Segundo mensaje pero agregamos mas espacios")
```

Al ejectuar el código anterior, obtendremos el siguiente error: `IndentationError: unexpected indent`. Este error se produce porque estamos ocupando una indentación inconsistente. El primer print tiene dos espacios y el segundo tiene cuatro. Para corregirlo, ambos deben tener la misma cantidad de espacios sin importar si son dos, cuatro u otra cantidad.

Ejercicio
Modifica el siguiente código para que funcione.

```
codigo = "1234"
if codigo == "1234":
  print("Código correcto")
    print("Segundo mensaje") # Modifica esta línea
```

Ideas clave:

- Los programas se ejecutan de manera lineal, de arriba a abajo.
- Podemos cambiar el flujo de un programa utilizando condiciones.
- El operador de igualdad == compara dos valores y nos dice si son iguales o no.

A continuación aplicaremos los conceptos aprendidos en el ejercicio anterior.

Ejercicio
Se ha diseñado un programa en Python con la intención de imprimir un mensaje de bienvenida a los usuarios cuyo nombre coincide con "Alex".

Sin embargo, tras realizar algunas modificaciones en el código, el programa ha dejado de funcionar correctamente y ya no muestra el mensaje de bienvenida adecuado. Corrige el error en el siguiente código para que funcione correctamente.

```
nombreUsuario = "Juan" # Modifica esta línea
if nombreUsuario == "Alex":
  print("Bienvenido, Alex");
```

Bloques de código

Ideas clave

- Podemos cambiar el flujo de un programa utilizando condiciones.
- Un bloque es un conjunto de instrucciones que se ejecutan juntas.

¿Qué es un bloque?

Un bloque de código o simplemente bloque es un conjunto de instrucciones que se ejecutan juntas.

En Python, los bloques se delimitan por la indentación, como se muestra en el siguiente ejemplo:

```
if condición:
  # Conjunto de instrucciones si la condición se cumple
else:
  # Conjunto de instrucciones si la condición no se cumple
```

Los bloques resultan útiles para ejecutar un conjunto de instrucciones si una condición se cumple, o si no se cumple.

Veamos un ejemplo práctico:

```
color = "azul"
if color === "azul":
  print("La variable color es azul")
  print("El azul es mi color favorito")
```

En este caso, ambos mensajes se mostrarán ya que la condición se cumple. Sin embargo, si cambiamos el valor de color a "rojo", entonces ninguno de los mensajes se mostrará.

Ahora, probemos el mismo código dejando solo el primer mensaje dentro del bloque:

```
color = "rojo"
```

```
if (color == "azul")
  print("La variable color es azul")
print("El azul es mi color favorito")
```

En este segundo ejemplo, si la variable color es "azul", se mostrarán ambos mensajes. Pero si fuera cualquier otro color, se mostrará únicamente el segundo mensaje, independientemente de si la condición se cumple o no.

Desde el momento en que se escribe el bloque if, la indentación es obligatoria. Python utiliza esa información para determinar que ese código solo debe ejecutarse si se cumple la condición.

Ejercicio
El programador que escribió este código no dejó los prints dentro del bloque if por lo que el código no funciona como debería. ¿Puedes corregirlo?

```
# Escribe tu código aquí
formulario = "incompleto";
if formulario == "completo":
print("Formulario completo");
print("Enviando email");
print("Este mensaje debe aparecer siempre");
# Fin
```

Ejercicio de bloque

Ideas clave

- Podemos cambiar el flujo de un programa utilizando condiciones.
- Un bloque es un conjunto de instrucciones que se ejecutan juntas.
- En Python los bloques se delimitan por la indentación.

A continuación aplicaremos lo aprendimos sobre bloques del ejercicio anterior.

Ejercicio
Un programador está desarrollando un videojuego y ha escrito el siguiente código para que, cuando la puerta esté abierta, se muestre un mensaje. Sin embargo, el programa no está funcionando correctamente. ¿Puedes corregirlo dejando el mensaje respectivo dentro del bloque donde sea necesario?

```
# Escribe tu código aquí
puerta = "cerrada"
if puerta == "abierta":
  print("Detrás de la puerta hay un tesoro")
print("Dentro del tesoro hay 50 monedas de oro")
print("Este mensaje debe aparecer siempre independientemente de si la puerta está abierta o cerrada")
# Fin
```