Operadores y estructuras de control

Operadores y expresiones en Python

Objetivos

El objetivo de esta sección es aprender sobre los diferentes tipos de operadores en Python, como los aritméticos, de comparación y lógicos. Además, se pretende aprender cómo utilizar estos operadores en expresiones y cómo crear ejemplos de código que los utilicen.

Introducción

Los operadores y expresiones son esenciales para la programación, ya que permiten realizar cálculos y tomar decisiones en los programas. En Python, hay tres tipos de operadores: aritméticos, de comparación y lógicos.

Operadores aritméticos

Los operadores aritméticos en Python incluyen la suma (+), la resta (-), la multiplicación (*), la división (/), el módulo (%), la exponenciación (**), y la división entera (//). Estos operadores se utilizan para realizar cálculos numéricos.

Suma (+)

El operador de suma se utiliza para sumar dos valores y obtener un resultado numérico. Por ejemplo:

```
a = 7
b = 3
suma = a + b
print("Suma:", suma)  # Imprime: Suma: 10
```

Resta (-)

El operador de resta se utiliza para restar un valor de otro y obtener un resultado numérico. Por ejemplo:

```
a = 7
b = 3
resta = a - b
print("Resta:", resta)  # Imprime: Resta: 4
```

Multiplicación (*)

El operador de multiplicación se utiliza para multiplicar dos valores y obtener un resultado numérico. Por ejemplo:

```
a = 7
b = 3
multiplicacion = a * b
print("Multiplicación:", multiplicacion)  # Imprime: Multiplicación: 21
```

División (/)

El operador de división se utiliza para dividir un valor por otro y obtener un resultado numérico. Por ejemplo:

```
a = 7
b = 3
division = a / b
print("División:", division)  # Imprime: División: 2.3333333333333335
```

Módulo (%)

El operador de módulo se utiliza para obtener el resto de una división y obtener un resultado numérico. Por ejemplo:

```
a = 7
b = 3
modulo = a % b
print("Módulo:", modulo)  # Imprime: Módulo: 1
```

Exponenciación (**)

El operador de exponenciación se utiliza para elevar un valor a una potencia y obtener un resultado numérico. Por ejemplo:

```
a = 7
b = 3
exponenciacion = a ** b
print("Exponenciación:", exponenciacion)  # Imprime: Exponenciación: 343
```

División entera (//)

El operador de división entera se utiliza para dividir un valor por otro y obtener el resultado entero. Por ejemplo:

```
a = 7
b = 3
division_entera = a // b
print("División entera:", division_entera)  # Imprime: División entera: 2
```

Operadores de comparación

Los operadores de comparación en Python incluyen igual (==), diferente (!=), mayor que (>), menor que (<), mayor o igual que (>=), y menor o igual que (<=). Estos operadores se utilizan para comparar dos valores.

Igual (==)

El operador igual se utiliza para comparar si dos valores son iguales y devuelve un valor booleano True o False. Por ejemplo:

```
x = 5
y = 5
igual = x == y
print("Igual:", igual)  # Imprime: Igual: True
```

Diferente (!=)

El operador diferente se utiliza para comparar si dos valores son diferentes y devuelve un valor booleano True o False. Por ejemplo:

```
x = 5
y = 8
diferente = x != y
print("Diferente:", diferente)  # Imprime: Diferente: True
```

Mayor que (>)

El operador mayor que se utiliza para comparar si un valor es mayor que otro y devuelve un valor booleano True o False. Por ejemplo:

```
x = 5
y = 8
mayor_que = y > x
print("Mayor que:", mayor_que)  # Imprime: Mayor que: True
```

Menor que (<)

El operador menor que se utiliza para comparar si un valor es menor que otro y devuelve un valor booleano True o False. Por ejemplo:

```
x = 5
y = 8
menor_que = x < y
print("Menor que:", menor_que)  # Imprime: Menor que: True
```

Mayor o igual que (>=)

El operador mayor o igual que se utiliza para comparar si un valor es mayor o igual que otro y devuelve un valor booleano True o False. Por ejemplo:

```
x = 5
y = 5
mayor_o_igual = y >= x
print("Mayor o igual que:", mayor_o_igual)  # Imprime: Mayor o igual que: True
```

Menor o igual que (<=)

El operador menor o igual que se utiliza para comparar si un valor es menor o igual que otro y devuelve un valor booleano True o False. Por ejemplo:

```
x = 5
y = 5
menor_o_igual = x <= y
print("Menor o igual que:", menor_o_igual)  # Imprime: Menor o igual que: True
```

Operadores lógicos

Los operadores lógicos en Python incluyen AND (and), OR (or) y NOT (not). Estos operadores se utilizan para combinar expresiones lógicas y obtener un resultado lógico.

AND (and)

El operador AND se utiliza para combinar dos expresiones lógicas y devuelve True si ambas expresiones son verdaderas, de lo contrario devuelve False. Por ejemplo:

```
verdadero = True
falso = False
conjuncion = verdadero and falso
print("AND:", conjuncion)  # Imprime: AND: False
```

OR (or)

El operador OR se utiliza para combinar dos expresiones lógicas y devuelve True si al menos una expresión es verdadera, de lo contrario devuelve False. Por ejemplo:

```
verdadero = True
falso = False
disyuncion = verdadero or falso
print("OR:", disyuncion)  # Imprime: OR: True
```

NOT (not)

El operador NOT se utiliza para negar una expresión lógica y devuelve True si la expresión es falsa, de lo contrario devuelve False. Por ejemplo:

```
verdadero = True
negacion = not verdadero
print("NOT:", negacion)  # Imprime: NOT: False
```

Estructuras de control en Python

Objetivos

El objetivo de esta sección es aprender sobre las estructuras de control en Python y cómo se utilizan para organizar y controlar el flujo de ejecución de los programas. En particular, se estudiarán las estructuras `if`, `while` y `for`.

Introducción

Las estructuras de control son fundamentales en la programación, ya que permiten controlar el flujo de ejecución de un programa. En Python, hay tres estructuras de control principales: `if`, `while` y `for`.

La estructura `if` y su uso en la programación

La estructura `if` se utiliza para tomar decisiones en un programa. La sintaxis básica de la estructura `if` es la siguiente:

```
if expresion_logica:
    # código a ejecutar si la expresión es verdadera
```

También se puede utilizar la estructura `if` con las cláusulas `elif` y `else` para tomar decisiones más complejas. La sintaxis de la estructura `if` con cláusulas `elif` y `else` es la siguiente:

```
if expresion_logica1:
    # código a ejecutar si la expresión 1 es verdadera
elif expresion_logica2:
    # código a ejecutar si la expresión 1 es falsa y la expresión 2 es verdadera
else:
    # código a ejecutar si todas las expresiones son falsas
```

Ejemplo de código:

```
edad = 18

if edad < 18:
    print("Menor de edad")
elif edad >= 18 and edad < 65:
    print("Adulto")
else:
    print("Adulto mayor")
```

Uso de la estructura `while`

La estructura `while` se utiliza para repetir una sección de código mientras una expresión lógica sea verdadera. La sintaxis básica de la estructura `while` es la siguiente:

```
while expresion_logica:
    # código a ejecutar mientras la expresión sea verdadera
```

Ejemplo de código:

```
contador = 0

while contador < 5:
    print("Contador:", contador)
    contador += 1
```

La estructura `for` y su aplicación en bucles

La estructura `for` se utiliza para iterar sobre una secuencia de valores. La sintaxis básica de la estructura `for` es la siguiente:

```
for variable in secuencia:
    # código a ejecutar para cada valor en la secuencia
```

Uso de la función `range`

La función `range` se utiliza para generar una secuencia de valores numéricos. La sintaxis de la función `range` es la siguiente:

```
range(inicio, fin, incremento)
```

Iteración sobre listas y otros objetos iterables

La estructura `for` también se puede utilizar para iterar sobre objetos iterables como listas, tuplas, conjuntos y diccionarios.

Ejemplo de código:

```
for i in range(5):
    print("Número:", i)

nombres = ["Ana", "Luis", "Carlos", "Sofía"]
for nombre in nombres:
    print("Nombre:", nombre)
```

Conclusión

En resumen, los operadores y estructuras de control son fundamentales para la programación en Python. Los operadores aritméticos, de comparación y lógicos permiten realizar cálculos y tomar decisiones en los programas, mientras que las estructuras de control `if`, `while` y `for` permiten controlar el flujo de ejecución del programa. Es importante practicar con ejercicios y proyectos para mejorar el dominio de estos conceptos.

Perspectivas de aprendizaje

Dominar los operadores, expresiones y estructuras de control es fundamental para convertirse en un programador eficiente en Python. A medida que se avanza en el aprendizaje, se pueden explorar conceptos más avanzados como funciones, módulos y clases. La práctica constante y la resolución de problemas reales ayudarán a mejorar la comprensión y habilidades en la programación en Python.