Python

Estructuras de control

CertiDevs

Índice de contenidos

1. Estructuras de control

Las **estructuras de control** en Python permiten modificar el flujo de ejecución del programa, tomando decisiones y repitiendo bloques de código según sea necesario.

2. Condicionales

Los **condicionales** permiten tomar decisiones en función de una condición dada.

2.1. if

La declaración `if` evalúa una condición y ejecuta un bloque de código si la condición es verdadera (`True`).

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

2.2. elif

La declaración `elif` (abreviatura de "else if") se utiliza junto con `if` para verificar múltiples condiciones.

```
edad = 16

if edad >= 18:
    print("Eres mayor de edad")
elif edad >= 16:
    print("Eres menor de edad pero puedes obtener un permiso de conducir")
```

2.3. else

La declaración `else` se utiliza junto con `if` y `elif` para proporcionar una opción por defecto cuando todas las condiciones anteriores son falsas (`False`).

```
edad = 15

if edad >= 18:
    print("Eres mayor de edad")
elif edad >= 16:
    print("Eres menor de edad pero puedes obtener un permiso de conducir")
else:
    print("Eres menor de edad")
```

2.4. Expresiones condicionales (operador ternario)

Las **expresiones condicionales** permiten asignar un valor a una variable basándose en una condición, en una sola línea de código.

Determinar el valor absoluto de un número:

```
numero = -5
valor_absoluto = numero if numero >= 0 else -numero
print(valor_absoluto)
```

3. Bucles

Los **bucles** permiten repetir bloques de código mientras se cumpla una condición o por cada elemento de una secuencia.

3.1. while

El bucle `while` repite un bloque de código mientras se cumpla una condición.

```
contador = 0
while contador < 5:
    print(f"Contador: {contador}")
    contador += 1
```

3.2. for

El bucle `for` itera sobre una secuencia (**lista**, **tupla**, **cadena de caracteres**, etc.) y ejecuta un bloque de código para cada elemento.

```
frutas = ["manzana", "naranja", "uva"]
for fruta in frutas:
    print(fruta)
```

Puedes usar la función `range()` para generar una secuencia de números y usarla en un bucle `for`.

```
for i in range(5):
    print(i)
```

4. Control de bucles

Python proporciona dos declaraciones para controlar el flujo de ejecución dentro de un bucle: `break`

y `continue`.

4.1. break

La declaración `break` se utiliza para salir de un bucle antes de que se cumpla la condición de terminación.

```
for numero in range(1, 11):
    if numero == 6:
        break
    print(numero)
```

En este ejemplo, el bucle se detendrá cuando numero sea igual a 6.

4.2. continue

La declaración `continue` se utiliza para saltar la iteración actual del bucle y continuar con la siguiente.

```
for numero in range(1, 11):
    if numero % 2 == 0:
        continue
    print(numero)
```

En este ejemplo, el bucle imprimirá solo los números impares, ya que la iteración se salta cuando numero es par.

5. Estructuras de control avanzadas en Python

5.1. Comprensiones de listas

Las **comprensiones de listas** (list comprehension) son una forma concisa y eficiente de crear nuevas listas a partir de secuencias o iterables existentes.

En lugar de utilizar bucles `for` y **condicionales** para crear listas, las comprensiones de listas permiten lograr lo mismo en una sola línea de código.

5.1.1. Ejemplo 1: Crear una lista de números del 1 al 10

```
numeros = [x for x in range(1, 11)]
print(numeros)
```

En este ejemplo, se crea una lista de números del 1 al 10 utilizando una comprensión de lista.

La función `range()` genera una secuencia de números del 1 al 10, y la variable x toma cada uno de estos números en la comprensión de la lista.

5.1.2. Ejemplo 2: Crear una lista de cuadrados de números del 1 al 10

```
cuadrados = [x**2 for x in range(1, 11)]
print(cuadrados)
```

En este ejemplo, se crea una lista de cuadrados de números del 1 al 10.

La función `range()` genera una secuencia de números del 1 al 10, y la variable x toma cada uno de estos números y calcula su cuadrado (x**2) en la comprensión de la lista.

5.1.3. Ejemplo 3: Crear una lista de números pares del 1 al 20

```
pares = [x for x in range(1, 21) if x % 2 == 0]
print(pares)
```

En este ejemplo, se crea una lista de números pares del 1 al 20 utilizando una comprensión de lista con una condición. La función range() genera una secuencia de números del 1 al 20, y la variable x toma cada uno de estos números solo si el número es par (x % 2 == 0).

5.1.4. Ejemplo 4: Crear una lista de palabras en mayúsculas a partir de una lista de palabras

```
palabras = ["manzana", "naranja", "pera", "banana"]
mayusculas = [palabra.upper() for palabra in palabras]
print(mayusculas)
```

En este ejemplo, se crea una lista de palabras en mayúsculas a partir de una lista de palabras existente.

La variable `palabra` toma cada una de las palabras en la lista palabras, y el método `upper()` convierte la palabra en mayúsculas en la comprensión de la lista.

5.1.5. Ejemplo 5: Filtrar elementos de una lista

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9]
impares = [x for x in numeros if x % 2 != 0]
print(impares)
```

6. Funciones generadoras

Las **funciones generadoras** son funciones que utilizan la palabra clave `yield` en lugar de `return`.

Esto permite que la función devuelva una secuencia de valores de manera "perezosa", es decir, se produce un valor a la vez en lugar de calcular todos los valores de una vez. Esto puede resultar en un mejor rendimiento y menor uso de memoria en ciertos casos.

6.1. Ejemplo: Generador de números de Fibonacci

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b

for num in fibonacci(10):
    print(num)
```

En este ejemplo, la función generadora fibonacci produce la secuencia de números de Fibonacci hasta n. El bucle for itera sobre los números generados por la función generadora.