Flujo de control en Python

DAGDATAANALYTICS.SAS

Autor: Diego Armando Giraldo Quintero

Índice

1 …… **Flujo de control en Python**

2 …… **Estructuras secuenciales**

3 …… **Condicionales**

3.1 …… **IF**

3.2 …… **ELSE**

3.3 …… **ELIF**

4 ……**Bucles ciclos de repetición**

4.1 …… **While**

4.2 …… **For**

5 …… **Control de flujo dentro de los bucles**

5.1 …… **Break**

5.2 …… **Continúe**

5.3 …… **Else en bucles**

6 …… **Manejo de excepciones**

7 …… **Declaraciones pass y assert**

7.1 …… **Pass**

7.2 …… **Assert**

1. **Flujo de control en Python**

   El flujo de control en Python se refiere a las secuencias en las que se ejecutan las instrucciones dentro de un programa. En Python el flujo de control se organiza utilizando estructuras que permiten tomar decisiones, repetir instrucciones, o alterar el curso del programa en función de condiciones. A continuación, te detallo las principales estructuras que controlan el flujo en Python:

2. **Estructuras secuenciales**

   En el flujo de control secuencial, las instrucciones se ejecutan una tras otra en el orden que aparecen. Este es el flujo por defecto de un programa, donde no hay saltos ni decisiones.

   Ejemplo:

```
x= 5
y= 6
z= x+y
print(z)
```

3. **Condicionales (Sentencias de decisión)**

   Las sentencias condicionales permiten que el programa, ejecute bloques de código dependiendo de si se cumple o no una condición.

3.1. If

El bloque de código dentro de un if se ejecuta si la condición es verdadera.

Sintaxis:

```
if #condición
#Bloque de código que se ejecuta si la condición es verdadera
```

3.2.Else

Se ejecuta cuando la condición de if es falsa.

Sintaxis

```
if #condicion
else:
   #bloque de codigo si la condicion es falsa
```

3.3.Elif

Se usa cuando queremos comprobar múltiples condiciones. Si la primera condición es falsa se verifica la siguiente y así sucesivamente.

Sintaxis

```
if #Condicion 1:
elif #condicion 2:
else:
    #bloque de codigo si ninguna de las condiciones anteriores es verdadera
```

4. Bucles ciclos de repetición

Los bucles permiten repetir un bloque de código varias veces.

4.1.While

El bucle while ejecuta un bloque de codigo mientras la condicion sea verdadera.

Sintaxis

```
x = 0
while x<5:
   print(x)
   x+=1
#este codigo imprimira los numeros del 0 al 4
```

4.2.For

El bucle for itera sobre una secuencia (Como una lista, tupla, rango, etc)

Ejecutando el bloque de un código una vez por cada elemento.

Sintaxis

```
for i in range (5):
   print(i)
   #Este codigo imprimira los numeros del  0 al 4 range (5) genera una secuencia del 0 al 4
```

5. Control de flujo dentro de los bucles

Dentro de los bucles, existen instrucciones para modificar su comportamiento.

5.1.Break

El break se utiliza para salir de un bucle antes de que termine su ciclo completo.

Ejemplo:

```
for i in range (10):
   if i==5:
      break
print(i) # Este codigo imprimira los numeros y luego saldra del bucle cuando i sea igual a 5
```

5.2.continué

El continué se utiliza para saltarse la interacción actual y continuar con la siguiente. Esto es útil cuando queremos saltar una parte del bucle bajo ciertas condiciones.

Ejemplo:

```
for numero in numeros:
    if numero % 2 != 0:  # Si el número no es par
        continue  # Salta a la siguiente iteración
    print(f"{numero} es un número par")
```

5.3.Else en bucles

Un bloque else en bucle se ejecuta solo si el bucle se interrumpe con break.

```
notas = [4.0, 3.5, 2.0, 5.0]  # Lista de notas simuladas
for nota in notas:
    if nota >= 5:
        print("Felicidades, ha pasado el curso")
        break
    elif nota > 2.99:
        print("Felicidades, usted ha aprobado")
    else:
        print("Usted no ha aprobado")
        break
```

6. Manejo de excepciones

Permite alterar el flujo normal del programa en caso de errores.

Try:

Código que puede generar una excepción.

Except

Código a ejecutar si ocurre la excepción.

Else

Código que se ejecuta si no ocurre ninguna excepción

Finally

Código que se ejecuta siempre

```
try:
   x=10/0
except ZeroDivisionError as e:
   print("error division por cero")
else:
   print("todo bien")
finally:
   print("fin del bloque")
```

7. Declaraciones pass y assert

7.1. Pass

Se usa como un marcador de posición para bloques de códigos que aún no están implementados. Se utiliza comúnmente en lugares donde aún no se ha implementado la lógica, pero el código debe estar estructurado correctamente.

Útil en clases, funciones, o bucles que aún no tienen implementación, permitiendo al desarrollador escribir código esquelético y completarlo más tarde.

Ejemplo:

```
def funcion_a_implementar():
    pass  # Aquí eventualmente irá la implementación

class ClaseEjemplo:
    def metodo(self):
        pass  # Método aún no implementado

for i in range(5):
    pass  # Bucles en desarrollo
```

7.2.Assert

Se utiliza para pruebas rápidas, de condiciones, lanza una excepción si la condición es falsa. Se utiliza para verificar si las condiciones son verdaderas durante la ejecución de un programa.

Útil para encontrar y corregir errores de lógica durante el desarrollo, asegurando que el código se comporta como se esperaba.

```
x = 10
assert x > 5, "x debería ser mayor que 5"  # No produce error, la condición es verdadera

y = -1
assert y >= 0, "y debería ser no negativo"  # Produce AssertionError, la condición es falsa
```