Unidad 3

Estructuras repetitivas

Hasta el momento vimos que en Python el código se ejecuta de forma secuencial, es decir, se ejecuta línea por línea hasta que el intérprete no tiene más código para procesar. Sin embargo, existen casos donde vamos a necesitar que un bloque de código se repita.

El proceso de repetir bloques de código se conoce como **iteración**, **las estructuras repetitivas** (también conocidas como bucles) **se utilizan para que los programas implementen iteraciones.**

Bucle while

En Python el bucle while se utiliza para ejecutar un bloque de código mientras se cumpla una condición.

La sintaxis tiene el siguiente formato:

```
while condicion:
    #Bloque de codigo
```

Donde por ejemplo si quisiéramos que se muestre por pantalla los números de 1 hasta 5 podríamos escribir el siguiente programa:

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

Si ejecutamos el programa veremos por pantalla:

```
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5
```

En este ejemplo se ejecuta el bloque de código mientras contador sea menor o igual a 5.

Tener en cuenta lo siguiente:

- La **condición** se evalúa en cada **iteración**
- Si la **condición** nunca se vuelve falsa, el **bucle** puede ser **infinito**
- Se puede utilizar **break** para salir del bucle o **continue** para saltar a la siguiente **iteración**
- Podemos agregar un bloque de código cuando finalicen las iteraciones utilizando la cláusula **else**

Vemos un ejemplo utilizando break para salir de un bucle. Siguiendo el ejemplo anterior podríamos hacer un programa que cuente desde 1 a 5 pero cuando llegue a 3 debe salir del bucle e informarlo por pantalla.

```
contador = 1
while contador <= 5:
    print("Contador:", contador)
    contador += 1
    if contador == 3:
        print("¡finalizó el bucle!")
        break
```

Si ejecutamos el programa veremos por pantalla:

```
Contador: 1
Contador: 2
¡finalizó el bucle!
```

En este ejemplo el bucle se ejecuta hasta que el contador es igual a 3.

Ahora veamos un ejemplo utilizando continue para saltarnos la iteración. Siguiendo los ejemplos anteriores, vamos a crear un programa que cuente desde 1 a 5 pero cuando llegue a 3 debe saltarse la iteración y seguir contando hasta 5.

```
contador = 1
while contador <= 5:
    if contador == 3:
        print("¡Se salto la iteración!")
        contador += 1
        continue
    else:
        print("Contador:", contador)
        contador += 1
```

Si ejecutamos el programa veremos por pantalla:

```
Contador: 1
Contador: 2
¡Se salto la iteración!
Contador: 4
Contador: 5
```

En este ejemplo el bucle se ejecuta hasta que el contador es igual a 5 pero salta la iteración cuando el contador es igual a 3.

Por último, también podremos ejecutar un bloque de código cuando finalicen las iteraciones utilizando la cláusula else.

Ejemplo:

```
contador = 1
while contador <= 5:
    print("Contador:", contador)
    contador += 1
else:
    print("El bucle while finalizó")
```

Si ejecutamos el programa veremos por pantalla:

```
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5
El bucle while finalizó
```

En este caso cuando la condición se vuelve falsa (cuando contador es 6) se ejecuta el bloque de código de la cláusula else.

Contadores y Acumuladores

Anteriormente vimos que la variable contador incrementaba su valor en cada iteración y luego la utilizábamos para determinar si el bucle debía seguir ejecutándose. Este tipo de variables se conoce como contadores y son importantes dentro de las estructuras repetitivas, como asi también las variables de tipo acumulador.

Sus características pueden resumirse de la siguiente manera:

Contador:

- Se usa para contar cuántas veces ocurre algo.
- En general se incrementa en cada iteración.
- Ejemplo típico: contador += 1

Acumulador:

- Se usa para sumar (o acumular) valores en cada iteración.
- Ejemplo típico: acumulador += valor

¿Por qué se usan con bucles?

- Porque necesitamos llevar un control de lo que sucede en cada iteración
- Para contar las iteraciones de un bucle
- Para almacenar valores que se ingresan o se calculan
- Calcular promedios, totales, estadísticas, etc.

Veamos un ejemplo donde un programa cuenta los números desde 1 hasta 5 y luego muestra por pantalla la suma de todos ellos.

```
contador = 1
acumulador = 0
while contador <= 5:
    print("Contador:", contador)
    acumulador += contador # 1 + 2 + 3 + 4 + 5 = 15
    contador += 1
else:
    print("La suma de todos los valores es: ", acumulador)
```

```
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5
La suma de todos los valores es:  15
```

En este caso el contador se utiliza para controlar cuando debe finalizar el bucle y el acumulador lo utilizamos para almacenar el valor del contador en cada iteración.

Bucle for

El bucle for se utiliza para ejecutar un bloque de código un numero determinado de veces, si bien es similar al bucle while la diferencia radica en que la cantidad de iteraciones se establece cuando se declara el bucle.

Esto quiere decir que no hay una condición que se evalúa en cada iteración, sino que existe un iterador que se incrementa de forma automática.

La estructura del bucle for es la siguiente:

```
for iterador in elemento_iterable:
    #Bloque de codigo
```

Veamos un ejemplo del programa que habíamos hecho para contar de 1 hasta 5 con un bucle while, pero esta vez lo resolveremos con un bucle for.

```
for iterador in range(1,6): #range() crea una secuencia iterable de 1 hasta 5
    print(iterador)
```

```
1
2
3
4
5
```

La funciona range() permite crear una secuencia de números ingresando un valor inicial y un valor final (que no está incluido). En este caso el elemento iterable será la secuencia de números y el iterador hace referencia al numero correspondiente a la iteración.

Tener en cuenta lo siguiente al trabajar con un bucle for:

- Recorre automáticamente los elementos de un iterable.
- La iteración finaliza cuando se terminan los elementos del iterable.
- Se puede utilizar break para salir del bucle antes de tiempo.
- Se puede utilizar continue para saltar a la siguiente iteración sin ejecutar el resto del bloque.
- Podemos agregar un bloque de código que se ejecuta si el bucle finaliza sin interrupciones, utilizando la cláusula else.

Veamos un ejemplo con break, vamos a escribir un programa que cuente desde 1 a 5 pero cuando llegue a 3 debe salir del bucle e informarlo por pantalla.

```
for iterador in range(1,6):
    if iterador == 3:
        print("Finalizo el bucle")
        break
    else:
        print("iterador: ", iterador)
```

```
iterador:  1
iterador:  2
Finalizo el bucle
```

Cuando el iterador es igual a 3 se finaliza el bucle, caso contrario se muestra el valor del iterador.

Ahora se pide contar nuevamente de 1 hasta 5 pero cuando se llegue a 3 no debe mostrarse ese valor. En este caso vamos a utilizar la instrucción continue para “saltarnos” una iteración.

```
for iterador in range(1,6):
    if iterador == 3:
        print("¡Se salto la iteración!")
        continue
    else:
        print("iterador: ", iterador)
```

```
iterador:  1
iterador:  2
¡Se salto la iteración!
iterador:  4
iterador:  5
```

Cuando el iterador es igual 3 nos saltamos la iteración, pero no finalizamos el bucle.

Por último, también podremos ejecutar un bloque de código cuando finalicen las iteraciones utilizando la cláusula else.

```
for iterador in range(1,6):
    print("iterador: ", iterador)
else:
    print("Finalizo el bucle")
```

```
iterador:  1
iterador:  2
iterador:  3
iterador:  4
iterador:  5
Finalizo el bucle
```

Tuplas

Una tupla es un tipo de dato que permite almacenar una colección ordenada de elementos, pero lo mas importante es que es **iterable** y por lo tanto podremos utilizarla en un **bucle for**.

Veamos un ejemplo.

```
tupla_1 = (1, 2, 3) # tupla de 3 elementos
tupla_2 = (1,) #tupla con 1 solo elemento
tupla_3 = (10, "Hola", True) # tupla con tipos de dato mixtos
print("tupla_1 es igual a:", tupla_1)
print("tupla_2 es igual a:", tupla_2)
print("tupla_3 es igual a:", tupla_3)
```

```
tupla_1 es igual a: (1, 2, 3)
tupla_2 es igual a: (1,)
tupla_3 es igual a: (10, 'Hola', True)
```

¿Pero que sucedería si solo quiero mostrar un elemento de la tupla? En ese caso vamos a utilizar lo que se conoce como **índices**.

Antes habíamos dicho que la **tupla** permite almacenar elementos de forma ordenada, esto es posible porque **Python asigna un índice a cada elemento de la tupla**. El primer elemento tendrá el índice 0, el segundo tendrá el índice 1, el tercero el índice 2 y así sucesivamente hasta asignar un índice a todos los elementos.

La forma de acceder a un índice especifico es la siguiente:

```
tupla_1 = (1, 2, 3) # tupla de 3 elementos
print(tupla_1[0]) #Muestro el valor almacenado en el indice 0
print(tupla_1[1]) #Muestro el valor almacenado en el indice 1
print(tupla_1[2]) #Muestro el valor almacenado en el indice 2
```

```
1
2
3
```

Tener en cuenta que la cantidad de índices es la misma que la cantidad de elementos de la tupla, con lo cual no podremos acceder un índice mayor que el de la cantidad de elementos.

Ejemplo:

```
tupla_1 = (1, 2, 3) # tupla de 3 elementos
print(tupla_1[4]) #Esto dara error de indice fuera de rango "tuple index out of range"
```

Otra característica de la tupla es que no podremos modificar ninguno de sus elementos. Por lo tanto, una tupla es **inmutable**.

```
tupla_1 = (1, 2, 3) # tupla de 3 elementos
tupla_1[0] = 4 # Esto dara error de asignacion
```

Con lo visto anteriormente, podemos resumir sus características de la siguiente manera:

- Es ordenada → mantiene el orden de los elementos.
- Es inmutable → no se puede modificar una vez creada.
- Permite elementos repetidos.
- Puede contener distintos tipos de datos.
- Se puede acceder a los elementos por índice.

¿Cuándo es conveniente usar tuplas?

- Cuando necesitemos ordenar datos de forma ordenada que no van a modificarse
- Cuando necesitemos una estructura de datos simple

Veamos un ejemplo de cómo utilizar una tupla como iterable en un bucle for.

```
tupla = (10, "Hola", 30, "Adios")  #Tupla de 4 elementos
for iterador in tupla:
    print("iterador:", iterador) # Muestro el elemento de la tupla
```

```
iterador: 10
iterador: Hola
iterador: 30
iterador: Adios
```

Listas

Una lista en Python es una estructura de datos que permite almacenar una colección ordenada de elementos, al igual que una tupla también utiliza índices, pero la principal diferencia es que sus elementos pueden modificarse, por lo que es de tipo mutable.

```
lista_1 = [1, 2, 3] # lista de 3 elementos
lista_2 = [1] #lista con 1 solo elemento
lista_3 = [10, "Hola", True] # lista con tipos de dato mixtos
print("lista_1 es igual a:", lista_1)
print("lista_2 es igual a:", lista_2)
print("lista_3 es igual a:", lista_3)
```

```
lista_1 es igual a: [1, 2, 3]
lista_2 es igual a: [1]
lista_3 es igual a: [10, 'Hola', True]
```

En este caso si podremos modificar un elemento de la lista a través de su índice.

```
lista_1 = [1, 2, 3] # lista de 3 elementos
lista_1[0] = "hola" # Reemplazamos el primero elemento por "hola"
print("lista_1 es igual a:", lista_1)
```

```
lista_1 es igual a: ['hola', 2, 3]
```

También podremos agregar elementos a una lista utilizando el operador + .

```
lista_1 = [1, 2, 3] # lista de 3 elementos
lista_2 = ["Hola"] # Lista de 1 elemento
lista_1 = lista_1 + lista_2
print("lista_1 es igual a:", lista_1 )
```

```
lista_1 es igual a: [1, 2, 3, 'Hola']
```

O remover un elemento de la lista utilizando el índice y la función pop().

```
lista_1 = [1, 2, 3] # lista de 3 elementos
lista_1.pop(0) # Elimino el elemento del indice 0
print("lista_1 es igual a:", lista_1)
```

```
lista_1 es igual a: [2, 3]
```

Las principales características de una lista son las siguientes:

- Es ordenada → mantiene el orden de los elementos.
- Es mutable → se puede modificar una vez creada.
- Permite elementos repetidos.
- Puede contener distintos tipos de datos.
- Se puede acceder a los elementos por índice.

¿Cuándo usar listas?

- Cuando necesitemos ordenar datos de forma ordenada pero que puedan modificarse
- Cuando necesitemos una estructura de datos flexible

Una lista es iterable entonces también la podremos utilizar en un bucle for.

```
lista = [10, "Hola", 30, "Adios"]  #Lista de 4 elementos
for iterador in lista:
    print("iterador:", iterador) # Muestro el elemento de la Lista
```

```
iterador: 10
iterador: Hola
iterador: 30
iterador: Adios
```