Temas:

- **Bucle while**
- **Contadores y acumuladores**
- **Bucle for**
- **Tuplas y Listas**

Bucle While

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

Por ejemplo si quisiéramos contar de 1 hasta 5 podríamos escribir el siguiente programa:

```
contador = 1
while contador <= 5: #Mientras el contador sea menor o igual a 5
    print("Contador:", contador) #Muestro el valor del contador
    contador += 1 #Actualizo el valor del contador sumando 1
```

Por pantalla veremos lo siguiente:

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

Bucle While

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**

Bucle While - Break

Es una instrucción que se utiliza para cortar la ejecución del bucle.

Ejemplo:

```
contador = 1
while contador <= 5: #Mientras el contador sea menor o igual a 5
    print("Contador:", contador) #Muestro el valor del contador
    contador += 1 #Actualizo el valor del contador sumando 1
    if contador == 3: #Si el contador es igual a 3
        print("¡finalizó el bucle!")
        break #Se corta el bucle
```

Salida por pantalla:

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

Bucle While - Continue

Es una instrucción que se utiliza para “saltar” a la siguiente iteración. **Ejemplo:**

```
contador = 1
while contador <= 5: #Mientras el contador sea menor o igual a 5
    if contador == 3: #Si el contador es igual a 3 no lo muestro
        print("¡Se salto la iteración!")
        contador += 1
        continue #Finalizo la iteración, pero no el bucle
    else: #Si el contador no es igual a 3 entonces lo muestro
        print("Contador:", contador)
        contador += 1 #Actualizo el valor del contador sumando 1
```

Salida por pantalla:

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

Bucle While - Else

También podremos ejecutar un bloque de código cuando finalicen las iteraciones utilizando la cláusula else.

Ejemplo:

```
contador = 1
while contador <= 5: #Mientras el contador sea menor o igual a 5
    print("Contador:", contador) #Muestro el valor del contador
    contador += 1 #Actualizo el valor del contador sumando 1
else: #Si el contador no es menor o igual a 5 entonces el bucle finaliza
    print("El bucle while finalizó")
```

Salida por pantalla:

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

Contadores y Acumuladores

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.

Contadores y Acumuladores

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
```

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.

```
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).

Bucle for

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.

Bucle for - Break

Es una instrucción que se utiliza para cortar la ejecución del bucle.

Ejemplo:

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

Salida por pantalla:

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

Bucle for - Continue

Es una instrucción que se utiliza para “saltar” a la siguiente iteración. **Ejemplo:**

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

Salida por pantalla:

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

Bucle for - Else

También podremos ejecutar un bloque de código cuando finalicen las iteraciones utilizando la cláusula else.

Ejemplo:

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

Salida por pantalla:

```
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 una estructura repetitiva.

```
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)
```

Salida por pantalla:

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

Tuplas - Indices

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.

Ejemplo:

```
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
```

Salida por pantalla:

```
1
2
3
```

Tuplas - Índices

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"
```

Salida por pantalla:

```
print(tupla_1[4])
          ~~~~~~~^^^
IndexError: tuple index out of range
```

Tuplas - Inmutable

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

Ejemplo:

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

Salida por pantalla:

```
tupla_1[0] = 4
    ~~~~~~~^^^
TypeError: 'tuple' object does not support item
assignment
```

Tuplas - Caracteristicas

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

Tuplas – Bucle for

Como una tupla es un tipo de dato iterable entonces podremos utilizarla en un bucle for.

Ejemplo:

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

Salida por pantalla:

```
elemento: 10
elemento: Hola
elemento: 30
elemento: Adios
```

Listas

Una lista es un tipo de dato 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**.

Ejemplo:

```
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)
```

Salida por pantalla:

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

Listas - Índices

Al igual que con las tuplas también podremos acceder a un elemento especifico mediante su indice.

Ejemplo:

```
lista = ["Hola", "Pedro", "Adios"] # lista de 3 elementos
print(lista[0])
print(lista[1])
print(lista[2])
```

Salida por pantalla:

```
Hola
Pedro
Adios
```

Listas - Mutable

En una lista si podremos modificar un elemento a través de su índice.

Ejemplo:

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

Salida por pantalla:

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

Listas - Mutable

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

Ejemplo:

```
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 )
```

Salida por pantalla:

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

Listas - Características

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

Listas – Bucle for

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

Ejemplo:

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

Salida por pantalla:

```
elemento: 10
elemento: Hola
elemento: 30
elemento: Adios
```