Procesos repetitivos

1) while (mientras)

En general, en Python, un ciclo se puede representar de la siguiente manera:

while expresión_condicional:

instrucción_uno

instruccion_dos

instrucción_tres

:

:

instrucción_n

Ahora, es importante recordar que:

• Si deseas ejecutar más de una declaración dentro de un while, debes (como con if) poner sangría a todas las instrucciones de la misma manera.

• Una instrucción o conjunto de instrucciones ejecutadas dentro del while se llama el cuerpo del ciclo.

• Si la condición es False (igual a cero) tan pronto como se compruebe por primera vez, el cuerpo no se ejecuta ni una sola vez (ten en cuenta la analogía de no tener que hacer nada si no hay nada que hacer).

• El cuerpo debe poder cambiar el valor de la condición, porque si la condición es True al principio, el cuerpo podría funcionar continuamente hasta el infinito. Observa que hacer una cosa generalmente disminuye la cantidad de cosas por hacer.

Ejercicios resueltos

1. Desarrollar una aplicación que muestre los 10 primeros números enteros positivos.

```
# Repetitivo 01
c = 1
while c<=10:
    print(c,end=" ")
    c=c+1
```

2. Hacer un programa para mostrar los numeros : 20 25 30 35 65 70

```
c = 20
while c<=80:
    print(c,end=" ")
    c=c+5
```

3. Hacer un programa para mostrar los numeros : 100 98 96… 64 62 60

```
c = 20
while c<=80:
    print(c,end=" ")
    c=c+5
```

4. Ingresar varios numeros hasta que se ingrese el valor -1 y reportar el mayor de ellos.

```
# Almacenaremos el número más grande actual aquí

numeroMayor = -999999999

numero = int(input ("Introduzca un número o escriba -1 para detener:"))

while numero != -1:
    if numero > numeroMayor:
        numeroMayor = numero
        numero = int (input("Introduce un número o escribe -1 para detener:"))
    print("El número más grande es:", numeroMayor)
```

5. Desarrollar una aplicación que calcule la suma de los numeros hasta que el número ingresado sea negativo.

```
numeroMayor = -999999999

numero = int(input ("Introduzca un número o escriba -1 para detener:"))

while numero != -1:
    if numero > numeroMayor:
        numeroMayor = numero
    numero = int (input("Introduce un número o escribe -1 para detener:"))
print("El número más grande es:", numeroMayor)
```

6. Desarrollar una aplicación que calcule el producto (multiplicación) entre un conjunto de números reales hasta que el número ingresado sea cero.

```
producto=1
numero=float(input('Ingrese numero o escribe un cero para detener: '))
while numero !=0:
    producto = producto*numero
    numero = float (input('Introduce un número o escribe un cero para detener:')
print('La producto de los numeros es:', producto)
```

7. Ingresar n numeros enteros y reportar la cantidad de pares e impares.

```
c ,cp, ci = 1, 0, 0
n = int(input('Cantidad de numeros: '))
while c<=n:
    x=int(input('Ingrese numero entero:'))
    if x % 2 ==0:
        cp=cp+1
    else:
        ci=ci+1
    c=c+1
print('La cantidad de pares es: ',cp)
print('La cantidad de impares es: ',ci)
```

2. Proceso repetitivo for

```
for i in range (10):
    print(i,end=" ")
```

El codigo imprime los numeros desde el 0 hasta el 9

- La palabra reservada *for* abre el ciclo `for`; nota - No hay condición después de eso; no tienes que pensar en las condiciones, ya que se verifican internamente, sin ninguna intervención.
- Cualquier variable después de la palabra reservada *for* es la **variable de control** del ciclo; cuenta los giros del ciclo y lo hace automáticamente.
- La palabra reservada *in* introduce un elemento de sintaxis que describe el rango de valores posibles que se asignan a la variable de control.
- La función `range()` (esta es una función muy especial) es responsable de generar todos los valores deseados de la variable de control; en nuestro ejemplo, la función creará (incluso podemos decir que **alimentará** el ciclo con) valores subsiguientes del siguiente conjunto: 0, 1, 2 .. 9
- nota: en este caso, la función `range()` comienza su trabajo desde 0 y lo finaliza un paso (un número entero) antes del valor de su argumento.

La invocación de la función `range()` puede estar equipada con dos argumentos, no solo uno:

```
for i in range (2, 8):
    print("El valor de i es actualmente", i)
```

La salida ser:

El valor de i es actualmente 2
El valor de i es actualmente 3
El valor de i es actualmente 4
El valor de i es actualmente 5
El valor de i es actualmente 6
El valor de i es actualmente 7

La funcion range con 3 argumentos

La función range() genera una secuencia de números. Acepta enteros y devuelve objetos de rango. La sintaxis de range() tiene el siguiente aspecto: range(start, stop, step), donde:

• start es un parámetro opcional que especifica el número de inicio de la secuencia (0 por defecto).

• stop es un parámetro opcional que especifica el final de la secuencia generada (no está incluido).

• y step es un parámetro opcional que especifica la diferencia entre los números en la secuencia es (1 por defecto).

Código de ejemplo:

```
for i in range(3):
    print(i, end=" ")
```

La salida sera : 0 1 2

```
for i in range(6, 1, -2):
    print(i, end=" ")
```

La salida sera : 6 4 2

8. Ingresar n numeros y reportar el promedio de los positivos, el promedio de los negativos y la cantidad de ceros.

```
cp, cn,sp,sn = 0,0,0,0
n = int(input('Cantidad  de numeros: '))
for i in range(0, n):
    x=float(input('Ingrese numero:'))
    if x>0:
        sp=sp+x;
        cp=cp+1;
    elif x<0:
        sn=sn+x;
        cn=cn+1;
if cp>0:
    pp=sp/cp
    print('El promedio de positivos:',pp)
else:
    print('No se ingresaron positivos')

if cn>0:
    pn=sn/cn
    print('El promedio de negativos:',pn)
else:
    print('No se ingresaron negativos')
```

Las declaraciones break y continue

• Break: Sale del ciclo inmediatamente, e incondicionalmente termina la operación del ciclo; el programa comienza a ejecutar la instrucción más cercana después del cuerpo del ciclo.

• Continue: Se comporta como si el programa hubiera llegado repentinamente al final del cuerpo; el siguiente turno se inicia y la expresión de condición se prueba de inmediato.

Ambas palabras son palabras clave reservadas.

```
# break - ejemplo

print("La instrucción de ruptura:")
for i in range(1,6):
    if i == 3:
        break
    print("Dentro del ciclo.", i)
print("Fuera del ciclo.")
```

```
# continue - ejemplo

print("\nLa instrucción continue:")
for i in range(1,6):
    if i == 3:
        continue
    print("Dentro del ciclo.", i)
print("Fuera del ciclo.")
```

Ejercicio 1:

Diseña un programa que use un ciclo while y le pida continuamente al usuario que ingrese una palabra a menos que ingrese "bye" como la palabra de salida secreta, en cuyo caso el mensaje "¡Has dejado el ciclo con éxito". Debe imprimirse en la pantalla y el ciclo debe terminar.
No imprimas ninguna de las palabras ingresadas por el usuario. Utiliza el concepto de ejecución condicional y la declaración break.

```
palabra=input('Ingrese una palabra: ')
while True:
    if palabra=='bye':
        print('!Has dejado el ciclo con exito')
        break
    palabra=input('Ingrese una palabra:')
```

Ejercicio 2

Hacer un programa que

• Peda al usuario que ingrese una palabra.

• Utiliza palabra= palabra.upper() para convertir la palabra ingresada por el usuario a mayúsculas;
• Utiliza la ejecución condicional y la instrucción continue para "comer" las siguientes vocales A , E , I , O , U de la palabra ingresada.

• Imprime las letras no consumidas en la pantalla, cada una de ellas en una línea separada

```
palabra=input('Ingrese una palabra: ')
palabra=palabra.upper()

for letra in palabra:
    if letra in ['A','E','I','O','U']:
        continue
    print(letra)
```

El while y la opción else

Ambos ciclos, while y for, tienen una característica interesante (y rara vez se usa).
Te mostraremos cómo funciona: intenta juzgar por ti mismo si es utilizable.
En otras palabras, trata de convencerte si la función es valiosa y útil, o solo es azúcar sintáctica.
Echa un vistazo al fragmento en el editor. Hay algo extraño al final: la palabra clave else.
Como pudiste haber sospechado, los ciclos también pueden tener la rama else, como los if.
La rama else del ciclo siempre se ejecuta una vez, independientemente de si el ciclo ha entrado o no en su cuerpo .

```
i = 5
while i < 5:
    print(i)
    i += 1
else:
    print("else:", i)
```

El siguiente codigo imprimira 5.

El ciclo for y la rama else

Los ciclos for se comportan de manera un poco diferente: echa un vistazo al fragmento en el editor y ejecútalo.

```
i = 111
for i in range(2, 1):
    print(i)
else:
    print("else:", i)
```

La salida sera: else: 111
El cuerpo del ciclo no se ejecutará aquí en absoluto. Nota: hemos asignado la variable i antes del ciclo.
Cuando el cuerpo del ciclo no se ejecuta, la variable de control conserva el valor que tenía antes del ciclo.

```
for i in range(3, 10):
    print(i)
else:
    print("else:", i)
```

La salida sera:
3
4
5
6
7
8
9
else: 9

1. Hay dos tipos de ciclos en Python: `while` y `for`:

- El ciclo `while` ejecuta una sentencia o un conjunto de declaraciones siempre que una condición booleana especificada sea verdadera, por ejemplo:

```
# Ejemplo 1
while True:
    print("Atascado en un ciclo infinito")

# Ejemplo 2
contador = 5
while contador > 2:
    print(contador)
    contador -= 1
```

- El ciclo `for` ejecuta un conjunto de sentencias muchas veces; se usa para iterar sobre una secuencia (por ejemplo, una lista, un diccionario, una tupla o un conjunto; pronto aprenderás sobre ellos) u otros objetos que son iterables (por ejemplo, cadenas). Puedes usar el ciclo `for` para iterar sobre una secuencia de números usando la función incorporada `range`. Mira los ejemplos a continuación:

```
# Ejemplo 1
palabra = "Python"
for letter in palabra:
    print(letter, fin = "*")

# Ejemplo 2
for i in range(1, 10):
    if i % 2 == 0:
        print(i)
```

2. Puedes usar las sentencias `break` y `continue` para cambiar el flujo de un ciclo:

- Utiliza `break` para salir de un ciclo, por ejemplo:

```
texto = "OpenEDG Python Institute"
for letter in texto:
    if letter == "P":
        break
    print(letter, end= "")
```

- Utiliza `continue` para omitir la iteración actual, y continuar con la siguiente iteración, por ejemplo:

```
text = "pyxpyxpyx"
for letter in text:
    if letter == "x":
        continue
    print(letter, end= "")
```

3. Los ciclos `while` y `for` también pueden tener una cláusula `else` en Python. La cláusula `else` se ejecuta después de que el ciclo finalice su ejecución siempre y cuando no haya terminado con `break`, por ejemplo:

```
n = 0

while n != 3:
    print(n)
    n += 1
else:
    print(n, "else")

print()

for i in range(0, 3):
    print(i)
else:
    print(i, "else")
```

4. La función `range()` genera una secuencia de números. Acepta enteros y devuelve objetos de rango. La sintaxis de `range()` tiene el siguiente aspecto: `range(start, stop, step)`, donde:

- `start` es un parámetro opcional que especifica el número de inicio de la secuencia (`0` por defecto).
- `stop` es un parámetro opcional que especifica el final de la secuencia generada (no está incluido).

- y `step` es un parámetro opcional que especifica la diferencia entre los números en la secuencia es (`1` por defecto).

Código de ejemplo:

```
for i in range(3):
    print(i, end=" ") # salidas: 0 1 2

for i in range(6, 1, -2):

    print(i, end=" ") # salidas: 6, 4, 2
```

Ejercicios propuestos

1. Se desea calcular independientemente la suma de los pares e impares de los numero desde el 1 hasta el 50.
2. Calcular el factorial de un número entero mayor o igual que cero.
3. Ingresar n números. Se pide calcular el promedio de ellos
4. Sea n un entero positivo. Si n es par, divídalo entre 2, sino lo es, multiplíquelo por 3 y súmele 1. Realice este proceso hasta que el número que alcance sea 1. Realice un programa en C que implemente dicho proceso. Imprima los números que van obteniendo. Por Ejemplo:
   Para n = 10 la sucesión generada es: 10 5 16 8 4 2 1
5. Calcular la sumatoria:
   s= 1 + x + x2/2! + x3/3! + x4/4! + ... + xn/n!
   Se debe ingresar x real y n entero positivo