BUCLES O CICLOS REPETITIVOS

Un ciclo infinito, también denominado ciclo sin fin, es una secuencia de instrucciones en un programa que se repite indefinidamente (ciclo sin fin).

Bucle while

Este es un ejemplo de un ciclo que no puede finalizar su ejecución:

```
while True:
    print ("Estoy atrapado dentro de un ciclo")
```

Este ciclo imprimirá infinitamente "Estoy atrapado dentro de un ciclo" en la pantalla.

Propósito de los ciclos while.

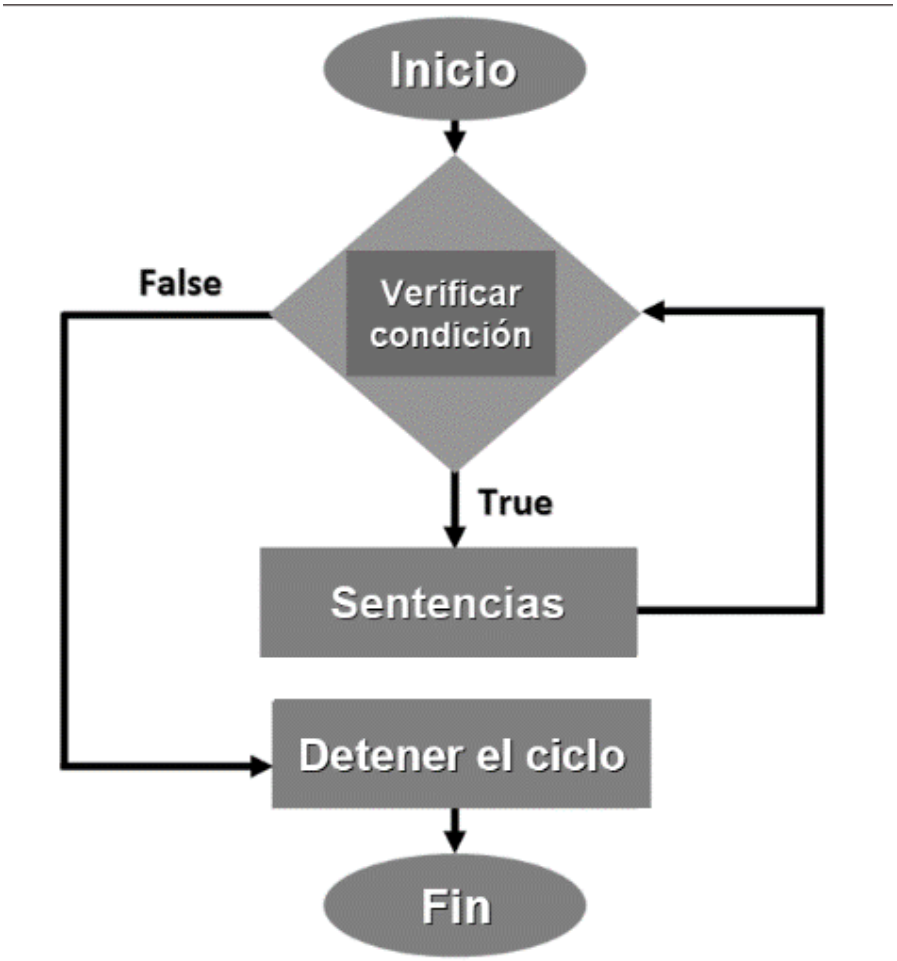
Dicho bucle se usa para repetir una secuencia de instrucciones o sentencias un número indefinido de veces. Este tipo de ciclo se ejecuta **mientras** una condición dada es **True** (verdadera) y solo se detiene si la condición es **False** (falsa).

Cuando escribimos un ciclo **while** (la palabra **while** significa "*mientras*" en español.), no definimos explícitamente cuántas iteraciones serán completadas, solo escribimos una condición que debe ser verdadera (**True**) para continuar el proceso y falsa (**False**) para detenerlo.
Si la condición del ciclo **while** nunca es falsa (**False**), entonces tendremos un ciclo *infinito*, el cual es un ciclo que en teoría nunca se detiene sin que sea interrumpido de forma externa. Por ejemplo, si se interrumpe con un atajo de teclado.

Cómo funcionan los ciclos while

Ahora que ya sabes para qué se usan los ciclos **while**, veamos su lógica principal y cómo funcionan detrás de escenas cuando se ejecuta el código.

Aquí tenemos un diagrama:



Si analizamos dicho diagrama podemos concluir:

- El proceso inicia cuando se encuentra un ciclo **while** durante la ejecución del programa.
- Se evalúa la condición para determinar si es verdadera o falsa (**True** o **False**).
- Si la condición es verdadera (**True**), la secuencia de instrucciones o sentencias que pertenecen al ciclo **while** se ejecutan.
- Nuevamente se evalúa y verifica la condición del ciclo **while**.
- Si la condición es verdadera (**True**) nuevamente, la secuencia de instrucciones en el cuerpo del ciclo **while** se ejecutan y el proceso se repite.
- Cuando la condición es falsa (**False**), el ciclo se detiene y el programa continúa más allá del ciclo **while**.

  Una de las características más importantes de los ciclos while es que las variables usadas en la condición del ciclo no se actualizan automáticamente.

  Nosotros tenemos que actualizar sus valores explícitamente en nuestro código para asegurarnos de que el ciclo se detendrá en algún momento, cuando la condición sea falsa (**False**).

  La sintaxis del ciclo **while:**

  ***While** condición:*
  Instrucción

  Estos son los elementos principales (en orden):

- La palabra clave **while** (seguida de un espacio).

- Una condición que determina si el ciclo continuará su ejecución o no en base a su valor (**True** o **False**).
- Dos puntos (**:**) al final de la primera línea.
- La secuencia de instrucciones o sentencias que se repetirán. A este bloque de código se le denomina el "cuerpo" del ciclo y debe estar indentado. Si una línea de código no está indentada, no se considerará parte del ciclo.

  Veamos un ejemplo:

```
i = 4

while i < 8:
    print(i)
    i += 1
```

Resultado:

```
4
5
6
7
```

En el siguiente cuadro se visualizará las iteraciones que realizará el bucle **while**

Iteración	i	i < 8
1	4	True
2	5	True
3	6	True
4	7	True
5	8	False

- **Iteración 1:** inicialmente, el valor de **i** es **4**, así que la condición **i < 8** evalúa a **True** y el ciclo inicia su ejecución. El valor de **i** se muestra (4) y este valor se incrementa en 1. El ciclo inicia nuevamente.
- **Iteración 2:** ahora el valor de **i** es **5**, así que la condición **i < 8** evalúa a **True**. El cuerpo del ciclo se ejecuta, el valor de **i** se muestra (5) y el valor de **i** se incrementa en 1. El ciclo inicia nuevamente.
- **Iteraciones 3 y 4:** el mismo proceso se repite para la tercera y la cuarta iteración, así que los números **6** y **7** se muestran en el terminal.
- Antes de iniciar la quinta iteración, el valor de **i** es **8**. Ahora la condición del ciclo **while i < 8** evalúa a **False** y el ciclo **while** se detiene inmediatamente.

Datos para la condición de un ciclo while

La condición del ciclo while tiene un rol muy importante en su funcionalidad y resultado. Se debe tener mucho cuidado al momento de escoger un operador de comparación porque esta es una fuente muy común de errores en el código (bugs).

Por ejemplo, estos son errores comunes:

- Usar < (menor que) en lugar de <= (menor o igual que) (o viceversa).
- Usar > (mayor que) en lugar de >= (mayor o igual que) (o viceversa).
  Esto puede cambiar el número de iteraciones del ciclo e incluso su resultado y efecto en el programa.

Veamos un ejemplo:

Si escribimos este ciclo while con la condición i < 9:

```
i = 6

while i < 9:
    print(i)
    i += 1
```

Resultado:

```
6
7
8
```

El ciclo completa tres iteraciones y luego se detiene cuando i es igual a 9.
Esta tabla muestra lo que ocurre detrás de escenas cuando se ejecuta el código:

Iteración	i	i < 9
1	6	True
2	7	True
3	8	True
4	9	False

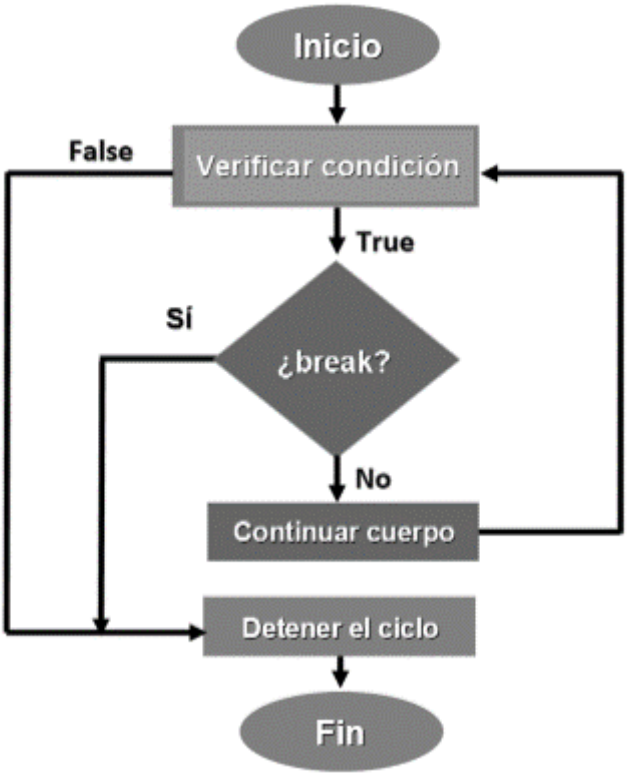
- Antes de la primera iteración del ciclo, el valor inicial de **i** es **6**, así que la condición **i < 9** es verdadera (**True**) y el ciclo inicia su ejecución. Se muestra el valor de **i** y luego se incrementa en 1.
- En la segunda iteración del ciclo, el valor de **i** es **7**, así que la condición **i < 9** es verdadera (**True**). El cuerpo del ciclo se ejecuta, se muestra el valor de **i** y luego se incrementa este valor en 1.
- En la tercera iteración del ciclo, el valor de **i** es **8**, así que la condición **i < 9** es verdadera (**True**). El cuerpo del ciclo se ejecuta, se muestra el valor de i y luego se incrementa este valor en 1.
- La condición se verifica nuevamente antes de que inicie una cuarta iteración, pero ahora el valor de **i** es **9**, así que **i < 9** es falsa (**False**) y el ciclo se detiene.

La sentencia break

Esta sentencia se usa para detener un ciclo **while** inmediatamente. Debes considerarla como una señal de tránsito roja de "**stop**" (detenerse) que puedes usar en tu código para controlar la funcionalidad del ciclo while.

La sentencia **break** termina el bucle **FOR** o **WHILE** más anidado.

Este diagrama muestra la lógica básica de una sentencia break:



Esta es la lógica básica de una sentencia break:

- El ciclo **while** solo inicia si la condición es verdadera (**True**).
- Si se encuentra una sentencia **break** durante la ejecución del ciclo, el ciclo se detiene inmediatamente.
- Si la sentencia **break** no se encuentra, el ciclo continúa su ejecución normal y se detiene cuando la condición es falsa (**False**).

  Podemos usar break para detener un ciclo **while** cuando una condición se cumple en un momento en particular de su ejecución, así que normalmente la encontrarás en un condicional.

  Este es un ejemplo:

```
while True:
    # Código
    if <condición>:
        break
    # Código
```

Esto detiene el ciclo inmediatamente si la condición del condicional es verdadera (**True**).

Bucles for

La palabra reservada **for** abre el ciclo **for**.

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.

Python utiliza bucles **for** para iterar sobre una lista de elementos. Se utiliza el bucle for para cambiar un valor (una variable) en cada iteración y así poder acceder a los elementos de un arreglo (array) utilizando ese valor.

En Python los bucles **for** iteran sobre estructuras de datos basadas en colecciones como **listas**, **tuplas** y/o **diccionarios**.

La sintaxis básica es:

```
for valor in lista_de_valores:
  # puedes usar la variable valor dentro de este bloque de codigo

#for, in -->> son palabras reservadas del bucle for
#se puede leer de la siguiente forma:
#por cada <valor> en la <lista_de_valores>
```

Diferentes formas de usar bucles For:

Iterar sobre la función range() – rango

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 al 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.

Veamos un ejemplo:

```
for i in range(10):
    print(i)
```

En lugar de ser una función, **range()** en realidad es un tipo de secuencia inmutable.

La secuencia resultante tendrá una lista desde el límite inferior,

Por ejemplo:

Cero, hasta el límite superior menos uno. Por defecto el límite inferior o índice inicial será cero.

Resultado:

```
range(10)
>
0
1
2
3
4
5
6
7
8
9
>
```

Adicionalmente, es posible especificar el límite inferior de la secuencia e incluso los pasos de la numeración añadiendo un segundo y tercer parámetro.

```
for i in range(4,10,2): #Secuencia del 4 al 9 , de 2 en 2
    print(i)
```

Resultado:

```
>
4
6
8
>
```

Iterar sobre los valores de una lista o tupla

```
A = ["hola", 1, 65, "gracias", [2, 3]]
for value in A:
    print(value)
```

Resultado:

```
>
hola
1
65
gracias
[2, 3]
>
```

Iterar sobre claves en un diccionario

```
fruta_a_color = {"manzana": "#ff0000",
                 "lima": "#ffff00",
                 "naranja": "#ffa500"}

for key in fruta_a_color:
    print(key, fruta_a_color[key])
```

Resultado:

```
>
manzana #ff0000
lima #ffff00
naranja #ffa500
>
```

Contador

Un contador es una variable entera que la utilizamos para contar cuando ocurre un suceso. Un contador:

- Se **inicializa** a un valor inicial.

```
cont = 0;
```

- Se **incrementa**, cuando ocurre el suceso que estamos contado se le suma 1.

```
cont = cont + 1;
```

- Se **incrementa**, cuando ocurre el suceso que estamos contado se le suma 1.

```
cont += 1
```

Acumulador

Un acumulador es una variable numérica que permite ir acumulando operaciones. Me permite ir haciendo operaciones parciales.

Un acumulador:

- Se **inicializa** a un valor inicial según la operación que se va a acumular: a 0 si es una suma o a 1 si es un producto.

- Se **acumula** un valor intermedio.

```
acum = acum + num;
```