



Clase 07 - While Loop and For Loop

Bucle While

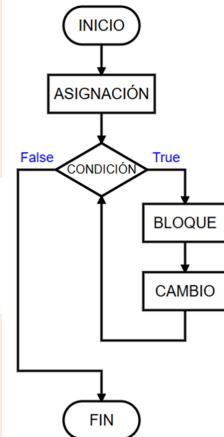
Cuando queremos hacer algo más de una vez, necesitamos recurrir a un **bucle**. En esta sección veremos las distintas sentencias en Python que nos permiten repetir un bloque de código.

La sentencia `while`

El primer mecanismo que existe en Python para repetir instrucciones es usar la sentencia `while`.

La semántica tras esta sentencia es: «Mientras se cumpla una condición haz algo».

Esta condición del bucle se conoce como **condición de parada**



```
saludar = 'S'
```

```
while saludar.upper() == 'S':
    print('Hola qué tal!')
    saludar = input('¿Quiere otro saludo? [S/N]
')
    print('Que tenga un buen día')
```



Iteración

En este contexto, llamamos **iteración** a cada «repetición» del bucle. **Iterar** significa «repetir» una determinada acción.

Romper un bucle while

Python ofrece la posibilidad de romper o finalizar un bucle antes de que se cumpla la condición de parada.

Supongamos que en el ejemplo anterior establecemos **un máximo de 4 saludos**:

```
MAX_SALUDOS = 4
num_saludos = 0
saludar = 'S'

while saludar.upper() == 'S':
    print('¡Hola qué tal!')
    num_saludos += 1

    if num_saludos >= MAX_SALUDOS:
        print('Número máximo de saludos alcanzado')
        break

    saludar = input('¿Quiere otro saludo? [S/N] ').strip()
```

```
print('Que tenga un buen día')
```

Como hemos visto en este ejemplo, `break` nos permite finalizar el bucle una vez que hemos llegado al máximo número de saludos. Pero si no hubiéramos llegado a dicho límite, el bucle habría seguido hasta que el usuario indicara que no quiere más saludos.

Solución alternativa:

```
while saludar == 'S' and num_saludos < MAX_SALUDOS:  
  
    MAX_SALUDOS = 4  
    num_saludos = 0  
    saludar = 'S'  
  
    while saludar == 'S' and num_saludos < MAX_SALUDOS:  
        print('¡Hola qué tal!')  
        num_saludos += 1  
  
        saludar = input('¿Quiere otro saludo? [S/N] ').strip().upper()  
  
        # Si ya llegamos al límite, aunque el usuario diga 'S', no seguirá  
        if num_saludos >= MAX_SALUDOS:  
            print('Número máximo de saludos alcanzado')  
  
    print('Que tenga un buen día')
```

Diferencias importantes respecto a la versión con `break`:

- En esta versión **no se usa break**
- La condición del `while` ya controla que no se pase del límite

- El mensaje "Número máximo de saludos alcanzado" aparece **solo si se alcanzó el límite**, pero el bucle ya no continuará de todas formas

Comprobar la rotura

Python nos ofrece la posibilidad de **detectar si el bucle ha acabado de forma ordinaria**, esto es, ha finalizado por no cumplirse la condición establecida. Para ello podemos hacer uso de la sentencia `else` como parte del propio bucle. Si el bucle `while` finaliza normalmente (sin llamada a `break`) el flujo de control pasa a la sentencia opcional `else`. Veamos su comportamiento siguiendo con el **ejemplo** que venimos trabajando:

```
MAX_SALUDOS = 4
num_saludos = 0
saludar = 'S'

while saludar == 'S':
    print('¡Hola qué tal!')
    num_saludos += 1
    if num_saludos == MAX_SALUDOS:
        print('Máximo número de saludos alcanzado')
        break
    saludar = input('¿Quiere otro saludo? [S/N] ').strip().upper()

else:
    print('Usted no quiere más saludos')
print("Que tenga un buen día")
```



Contexto

La sentencia `else` sólo tiene sentido en aquellos **bucles** que contienen un `break`.

Continuar un bucle while

Hay situaciones en las que, en vez de romper un bucle, nos interesa **saltar adelante hacia la siguiente iteración**. Para ello Python nos ofrece la sentencia `continue` que hace precisamente eso, descartar el resto del código del bucle y saltar a la siguiente iteración.

Continuamos con el *ejemplo* anterior pero ahora vamos a **contar el número de respuestas válidas**:

```
saludar = 'S'  
respuestas_validas = 0  
  
while saludar == 'S':  
    print('¡Hola qué tal!')  
    saludar = input('¿Quiere otro saludo? [S/N] ').strip().upper()  
  
    if saludar not in ['S', 'N']:  
        print('No le he entendido pero le saludo')  
        saludar = 'S'  
        continue  
  
    respuestas_validas += 1  
  
print(f'{respuestas_validas} respuestas válidas')  
print('Que tenga un buen día')
```

Bucle infinito

Si no establecemos correctamente la **condición de parada** o bien el valor de alguna variable está fuera de control, es posible que lleguemos a una situación de bucle infinito, del que nunca podamos salir. Veamos un ejemplo de esto:

```
num = 1
while num != 10:
    num += 2
```

Salida:

```
KeyboardInterrupt
^CTraceback (most recent call last):
Cell In[4], line 1
  while num != 10:
  ^^^^^^^^^^^^
KeyboardInterrupt
```

El problema que surje es que la variable `num` toma los valores 1, 3, 5, 7, 9, 11, ... por lo que nunca se cumple la **condición de parada** del bucle. Esto hace que repitamos «eternamente» la instrucción de incremento.



Parar el bucle: Para abortar una situación de *bucle infinito* podemos pulsar en el teclado la combinación Ctrl+C. Se puede ver reflejado en el intérprete de Python por `KeyboardInterrupt`.

Una posible solución a este problema sería reescribir la condición de parada en el bucle:

```
num = 1
while num < 10:
    num += 2
```

Bucles infinitos como recurso

Hay ocasiones en las que un **supuesto bucle «infinito»** puede ayudarnos a resolver un problema. Si retomamos el **ejemplo** de los saludos, podríamos reescribirlo utilizando un «bucle infinito» de la siguiente manera:

Versión **con** operador morsa `:=`: Usando Versión **sin** operador morsa:
el operador morsa pedimos la entrada y
comprobamos su valor

```
while True:  
    print("Hola que tal")  
    if saludar := input('¿Quiere otro saludo? [S/N] ').strip()  
        .upper():  
        break  
    print('Que tenga un buen día')
```

```
while True:  
    print("Hola que tal")  
    saludar = input('¿Quiere otro saludo? [S/N] ').strip()  
    .upper()  
    if saludar:  
        break  
    print('Que tenga un buen día')
```

Bucle For

Iterar en Python con bucles `for`

Python permite recorrer aquellos tipos de datos que sean **iterables**, es decir, que admitan *iterar* ("recorrer") sobre ellos. Algunos ejemplos de tipos y estructuras de datos que permiten ser iteradas (*recorridas*) son: cadenas de texto, listas, diccionarios, entre otros.

La sentencia `for` nos permite realizar esta acción. A continuación, se plantea un ejemplo en el que vamos a recorrer (iterar) una cadena de texto:

```
palabra = 'Python'  
for letra in palabra:  
    print(letra)
```

Salida:

```
P  
y
```

t
h
o
n

La clave aquí está en darse cuenta de que el bucle va tomando, en cada iteración, cada uno de los elementos de la variable que especifiquemos. En este caso concreto `letra` va tomando cada una de las letras que existen en `palabra`, porque una cadena de texto está formada por elementos que son caracteres. No es necesario definir la *variable de control* antes del bucle, aunque se puede utilizar como variable de control una variable ya definida en el programa. El cuerpo del bucle se ejecuta tantas veces como elementos tenga el elemento 'recorrible' (elementos de una lista, caracteres de una cadena, entre otros).



⚠️ Importante: La variable que utilizamos en el bucle `for` para ir tomando los valores puede tener **cualquier nombre**. Al fin y al cabo, es una variable que definimos según nuestras necesidades. Tener en cuenta que se suele usar un nombre en **singular**.

💡 Ejemplo: recorrer una lista

```
print('Comienzo')
for i in [0, 1, 2]:
    print("Hola ", end="")
```

Salida:

```
Comienzo
Hola Hola Hola
```

⚠ Si la lista está vacía, el bucle no se ejecuta

```
print('Comienzo')
for i in []:
    print('Hola', end=' ')
print()
print('Final')
```

Salida:

Comienzo

Final

⟳ Lo importante es la cantidad de elementos, no su valor

En el ejemplo anterior, los valores que toma la variable no son importantes, lo que importa es que la lista tiene tres elementos y por tanto el bucle se ejecuta tres veces. El siguiente programa produciría el mismo resultado que el anterior:

```
print("Comienzo")
for i in [1, 1, 1]:
    print("Hola ", end=' ')
print()
print('Final')
```

Salida:

Comienzo
Hola Hola Hola
Final

— Uso de `_` si no necesitamos la variable

Si la variable de control no se va a utilizar en el cuerpo del bucle, como en los ejemplos anteriores, se puede utilizar el guion bajo (`_`) en vez de un nombre de variable. Esta notación no tiene ninguna consecuencia con respecto al funcionamiento del programa, pero **ayuda a la persona que esté leyendo el código fuente**, que sabe así que los valores no se van a utilizar.

Por ejemplo:

```
print("Comienzo")
for _ in [0, 1, 2]:
    print("Hola ", end="")
print()
print("Final")
```

Salida:

```
Comienzo
Hola Hola Hola
Final
```

El indicador puede incluir cualquier número de guiones bajos (`_, __, ___, _____, ...`).

Los más utilizados son uno o dos guiones bajos (`_` o `__`).

✓ Uso de la variable de control dentro del bucle

En los ejemplos anteriores, la variable de control `i` no se utilizaba en el bloque de instrucciones, pero en muchos casos **sí que se utiliza**. Cuando se utiliza, hay que tener en cuenta que la variable de control va tomando los valores del elemento recorrible.

Por ejemplo:

```
print('Comienzo')
for i in [3, 4, 5]:
```

```
print(f'Hola. Ahora i vale {i} y su cuadrado {i ** 2}')
```

```
print('Final')
```

Salida:

```
Comienzo
```

```
Hola. Ahora i vale 3 y su cuadrado 9
```

```
Hola. Ahora i vale 4 y su cuadrado 16
```

```
Hola. Ahora i vale 5 y su cuadrado 25
```

```
Final
```

La lista puede contener cualquier tipo de elementos

La lista puede contener cualquier tipo de elementos, no sólo números. El bucle se repetirá siempre tantas veces como elementos tenga la lista y la variable irá tomando los valores de uno en uno. Por ejemplo:

```
print('Comienzo')
```

```
for i in ['Alba', 'Benito', 27, True, 2.58]:
```

```
    print(f'Hola. Ahora i vale {i}')
```

```
print('Final')
```

Salida:

```
Comienzo
```

```
Hola. Ahora i vale Alba
```

```
Hola. Ahora i vale Benito
```

```
Hola. Ahora i vale 27
```

```
Hola. Ahora i vale True
```

```
Hola. Ahora i vale 2.58
```

```
Final
```

La variable de control puede existir antes del bucle

La *variable de control* puede ser una variable empleada antes del bucle. El valor que tuviera la variable no afecta a la ejecución del bucle, pero cuando termina el bucle, la variable de control conserva el último valor asignado:

```
i = 10
print(f"El bucle no ha comenzado. Ahora i vale {i}")

for i in [0, 1, 2, 3, 4]:
    print(f"{i} * {i} = {i ** 2}")

print(f"El bucle ha terminado. Ahora i vale {i}")
```

Salida:

```
El bucle no ha comenzado. Ahora i vale 10
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
El bucle ha terminado. Ahora i vale 4
```



¿Qué pasa si modificamos la variable dentro del bucle?

La variable de control se puede modificar en el interior del bucle, pero en cada iteración Python asigna a la variable de control el valor que le corresponde de acuerdo con el elemento iterable que define el bucle:

```
i = 10
print(f"El bucle no ha comenzado. Ahora i vale {i}")

for i in [0, 1, 2]:
    print(f"Ahora i vale {i}")
    i = 100
```

```
print(f"Ahora i vale {i}")

print(f"El bucle ha terminado. Ahora i vale {i}")
```

Salida:

```
El bucle no ha comenzado. Ahora i vale 10
Ahora i vale 0
Ahora i vale 100
Ahora i vale 1
Ahora i vale 100
Ahora i vale 2
Ahora i vale 100
El bucle ha terminado. Ahora i vale 100
```

⚡ Iterar sobre una cadena en lugar de una lista

En vez de una lista se puede escribir una cadena, en cuyo caso la variable de control va tomando como valor cada uno de los caracteres:

```
for i in "AMIGO":
    print(f'Dame una {i}')
print('¡Amigo!')
```

Salida:

```
Dame una A
Dame una M
Dame una I
Dame una G
Dame una O
¡Amigo!
```



Romper un bucle `for`

Una sentencia `break` dentro de un `for` rompe el bucle, igual que para los bucles `while`. Veamos un ejemplo: En este caso vamos a recorrer una cadena de texto y pararemos el bucle cuando encontremos una letra `t` minúscula:

```
palabra = 'Python'  
for letra in palabra:  
    if letra == 't':  
        print()  
        print(f"el valor de letra es {letra} por lo tanto se  
ejecuta el break")  
        break  
    print(f"el valor de letra es {letra}")  
    print("por lo tanto no entró al if")  
print()  
print('Fin')
```

✓ Salida:

```
el valor de letra es P  
Por lo tanto no entró al if  
el valor de letra es y  
Por lo tanto no entró al if  
  
el valor de letra es t por lo tanto se ejecuta el break  
  
Fin
```

 **Ejercicio resuelto #1.** Escriba un programa en el que se le pide a un usuario introducir una palabra (`string`) y luego imprima el total de vocales encontradas.

Solución:

```

VOCALES = 'aeiou'
cadena = input('Introduzca una palabra: ')
nro_vocales = 0 # Este es un contador
# Usaremos para el input la cadena: 'Supercalifragilisticoesp
ialidoso'

for letra in cadena.lower():
    if letra in VOCALES:
        nro_vocales += 1

print(f'El numero de vocales contadas es {nro_vocales}')
print()
print('Fin')

```

✓ Salida:

Introduzca una palabra: Supercalifragilisticoespialidoso
 El numero de vocales contadas es 15

Fin

1 2 3 4 Secuencias de números

Es muy habitual hacer uso de secuencias de números en bucles. Python aporta una función `range()` que devuelve un **flujo de números** en el rango especificado. Una de las grandes ventajas es que la "lista" generada por la función `range()` no se construye explícitamente, sino que cada valor se genera bajo demanda. Esta técnica mejora el consumo de recursos, especialmente en términos de memoria. La técnica para la generación de secuencias de números es muy similar a la utilizada en los *slices* de cadenas de texto. En este caso disponemos de la función:

`range(start, stop, step)`

- **start**: Es *opcional* y tiene valor por defecto `0`.
 - **stop**: Es *obligatorio* (el rango llega hasta `stop - 1`).
 - **step**: Es *opcional* y tiene valor por defecto `1`.
-

Uso de `range()`

`range()` devuelve un objeto iterable, así que iremos obteniendo los valores paso a paso con una sentencia `for ... in`.

Veamos diferentes ejemplos de uso:

Ejemplo: `range(0, 3)`

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

Salida:

```
0  
1  
2
```

Ejemplo: `range(3)`

```
for i in range(3): # No hace falta indicar el inicio si es 0  
    print(i)
```

Salida:

```
0  
1  
2
```



Ejemplo: `range(1, 6, 2)`

```
for i in range(1, 6, 2):
    print(i)
```

Salida:

```
1
3
5
```



Ejemplo: `range(2, -1, -1)`

```
for i in range(2, -1, -1):
    print(i)
```

Salida:

```
2
1
0
```



Ejercicio resuelto #2 . Escriba un programa en el que se le pide a un usuario introducir un número entero y luego que imprima en pantalla si dicho número es primo o no. Excluir `0` y `1` en el rango.

Solución:

```
numero = int(input('Introduzca un numero entero positivo que
no sea ni cero ni uno: '))
for i in range(2, numero):
    if numero % i == 0:
```

```
        print('No es primo')
        break
else:
    print('Tenemos un numero primo')
```

✓ Salida:

```
Introduzca un numero entero positivo que no sea ni cero ni un
o: 2
Tenemos un numero primo
```

↳ Bucles anidados

El **anidamiento** es una técnica por la que incluimos distintos niveles de encapsulamiento de sentencias, unas dentro de otras, con mayor nivel de profundidad. En el caso de los bucles también es posible hacer anidamiento.

Veamos un ejemplo de 2 bucles anidados en el que generamos todas las tablas de multiplicar:

```
for tabla_numeros in range(1, 10):
    for factor_multiplicacion in range(1, 10):
        resultado = tabla_numeros * factor_multiplicacion
        print(f'{tabla_numeros} * {factor_multiplicacion} =
{resultado}' )
```



Testigos, contadores y acumuladores

En muchos programas se necesitan variables que indiquen si simplemente ha ocurrido algo (**testigos**), o que cuenten cuántas veces ha ocurrido algo (**contadores**), o que acumulen valores (**acumuladores**). Las situaciones pueden ser muy diversas, por lo que en este apartado se ofrecen unos ejemplos para mostrar la idea.

✓ Testigo (Bandera/Flag)

Se entiende por **testigo** una variable que indica simplemente si una condición se ha cumplido o no. Es un caso particular de contador, pero se suele hacer con variables lógicas en vez de numéricas.

En este ejemplo, la variable testigo es `sacaste_cinco`.

```
import random

print('Comienzo')
sacaste_cinco = False

for i in range(3):
    dado = random.randrange(1, 7)
    print(f'Tirada {i + 1}: {dado}')
    if dado == 5:
        sacaste_cinco = True

if sacaste_cinco:
    print('Ha salido al menos un 5 ')
else:
    print("No ha salido ningun 5. ")

print('Fin del Programa')
```

Salida (ejemplo):

```
Comienzo
Tirada 1: 4
Tirada 2: 3
Tirada 3: 3
No ha salido ningun 5.
Fin del Programa
```

Detalles importantes:

- Antes del bucle se debe dar un valor inicial al testigo (`False`).

- En cada iteración, el programa comprueba si `dado` es 5.
- El testigo se modifica la primera vez que `dado` es 5.
- El testigo no cambia a `False` una vez ha cambiado a `True`.
- Se podría haber utilizado un `range(1, 4)` para escribir directamente `{i}` en vez de `{i + 1}`.

1 2 3 4 Contador

Se entiende por **contador** una variable que lleva la cuenta del número de veces que se ha cumplido una condición. En el siguiente programa, la variable que hace de contador es `cuenta_cincos`.

```
import random

print("Comienzo")
cuenta_cincos = 0

for i in range(3):
    dado = random.randrange(1, 7)
    print(f"Tirada {i + 1}: {dado}")
    if dado == 5:
        cuenta_cincos += 1

print(f"En total ha(n) salido {cuenta_cincos} cinco(s).")
print("Final")
```

Salida (ejemplo):

```
Comienzo
Tirada 1: 6
Tirada 2: 2
Tirada 3: 6
```

```
En total ha(n) salido 0 cinco(s).
```

```
Final
```

Detalles importantes:

- Antes del bucle se debe dar un valor inicial al contador (`0`).
- En cada iteración, el programa comprueba si `dado` es 5.
- El contador se incrementa solo si `dado` es cinco.
- El contador va aumentando de uno en uno.
- Para adaptar la frase final según la cantidad, se podría usar `if...elif...else`:
 - "No se han obtenido cincos"
 - "Se ha obtenido un cinco"
 - "Se han obtenido XX cincos"



Acumulador

Se entiende por **acumulador** una variable que acumula el resultado de una operación.

El ejemplo siguiente es un programa con acumulador (en este caso, la variable que hace de acumulador es la variable `total`):

```
import random

print("Comienzo")
total = 0

for i in range(3):
    dado = random.randrange(1, 7)
    print(f"Tirada {i + 1}: {dado}")
    total += dado
```

```
print(f"En total ha obtenido {total} punto(s).")
print("Final")
```

Salida (ejemplo):

```
Comienzo
Tirada 1: 6
Tirada 2: 1
Tirada 3: 1
En total ha obtenido 8 punto(s).
Final
```

Detalles importantes:

- El acumulador se modifica en cada iteración del bucle (en este caso, el valor de `dado` se añade al acumulador `total`).
- Antes del bucle se debe dar un valor inicial al acumulador (en este caso, `0`).
- Si la frase indicando el número de puntos obtenidos se quisiera adaptar al número de puntos obtenidos (por ejemplo, "sólo se ha obtenido un punto" o "se han obtenido XX puntos") se debería utilizar una estructura `if ... elif ... else`.