

Estructuras Repetitivas en Python

¿Qué son?

Las **estructuras repetitivas** o **bucles** permiten que un conjunto de instrucciones se ejecute **más de una vez**, de forma automática, sin necesidad de escribir el mismo código varias veces.

En la vida real, repetimos acciones todo el tiempo: contar del 1 al 10, revisar mensajes, intentar una contraseña varias veces, etc. En programación, los bucles cumplen exactamente esa función.

Python tiene dos estructuras repetitivas principales:

- **while:** repite mientras una **condición sea verdadera**
- **for:** repite un bloque de código una **cantidad determinada** de veces

Bucle For

¿Qué hace?

El bucle **for** se utiliza cuando queremos **repetir instrucciones una cantidad exacta de veces**, o cuando queremos **recorrer una secuencia de elementos**. Como aún no trabajamos con listas ni arrays, vamos a usar una función incorporada de Python llamada **range()**, que genera valores de a uno para repetir código.

Sintaxis:

```
for variable in range(...):  
    # instrucciones que se ejecutan
```

La función **range()** le indica al bucle cuántas veces repetir y qué valores usar en cada iteración.

Formas de usar `range()`

1. `range(fin)`

Genera una secuencia que **comienza en 0** y termina antes del valor `fin`.

```
for i in range(5):  
    print("i vale:", i)
```

✦ Salida:

```
i vale: 0  
i vale: 1  
i vale: 2  
i vale: 3  
i vale: 4
```

☒ 2. range(inicio, fin)

Genera una secuencia que empieza en `inicio` y termina **antes de** `fin`.

```
for i in range(3, 6):  
    print("i vale:", i)
```

✦ Salida:

```
i vale: 3  
i vale: 4  
i vale: 5
```

☒ 3. range(inicio, fin, paso)

Permite controlar **de cuánto en cuánto** se incrementa o disminuye el valor en cada vuelta.

```
for i in range(10, 0, -2):  
    print("i vale:", i)
```

✦ Salida:

```
i vale: 10  
i vale: 8  
i vale: 6  
i vale: 4  
i vale: 2
```

💡 `range()` no incluye el valor final, sin importar si va en forma ascendente o descendente.

Recorriendo texto con `For`

El bucle `for` también se puede usar para **recorrer un string**, es decir, leer una letra por vez.

```
mensaje = "Hola"
for letra in mensaje:
    print(letra)
```

✚ Salida:

H
o
l
a

Bucle While

☒ ¿Qué hace?

El bucle `while` permite **repetir instrucciones mientras se cumpla una condición lógica**. A diferencia del `for`, en el que sabemos cuántas veces se ejecuta, el bucle `while` **se repite un número indefinido de veces**, siempre que la condición se mantenga verdadera.

✚ Sintaxis:

```
while condición:
    # instrucciones
```

¿Cuándo se entra al bucle?

La condición se evalúa antes de cada vuelta. Si es verdadera, se ejecuta el bloque de código. Si es falsa, el bucle se detiene.

¿Cómo evitar un bucle infinito?

Un **bucle infinito** ocurre cuando la condición **nunca deja de ser verdadera**. Esto hace que el programa quede atrapado repitiendo el mismo código para siempre.

Para evitar esto, es muy importante que **dentro del bloque del bucle modifiques alguna variable que intervenga en la condición**.

Por ejemplo, si usás una variable como contador, tenés que cambiar su valor dentro del `while` para que en algún momento la condición se vuelva falsa y el bucle termine.

✓ Ejemplo correcto de uso de `while`:

```
contador = 1
while contador <= 3:
    print("Contando:", contador)
    contador += 1
```

✦ Salida:

```
Contando: 1
Contando: 2
Contando: 3
```

🔍 Explicación paso a paso:

1. `contador` empieza en 1
2. La condición `contador <= 3` es verdadera → se ejecuta el bloque
3. Dentro del bloque se imprime el número y luego se suma 1 a `contador`
4. Se repite el proceso hasta que `contador` vale 4, y la condición ya no se cumple
5. El bucle se detiene

✓ Como el valor de `contador` cambia dentro del bucle, **no hay riesgo de bucle infinito**.

✗ Ejemplo de bucle infinito:

```
contador = 1
while contador <= 3:
    print("Contando:", contador)
```

⚠ En este ejemplo, `contador` **nunca cambia**, por lo tanto la condición siempre es verdadera. El programa imprimiría “Contando: 1” para siempre.

+ Cláusula `else` en `while`

La cláusula `else` se ejecuta **si el bucle terminó normalmente**, sin ser interrumpido por un `break`.

```
contador = 1
while contador <= 3:
    print("Contando:", contador)
    contador += 1
else:
    print("Fin del conteo.")
```

✚ Salida:

```
Contando: 1
Contando: 2
Contando: 3
Fin del conteo.
```

⊖ Palabras reservadas: `break` y `continue`

◆ `break`

La palabra `break` se usa para **interrumpir un bucle antes de que termine por sí solo**. Puede usarse tanto en `for` como en `while`.

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

✚ Salida:

```
1
2
```

🔍 Se imprime hasta el 2, luego se ejecuta `break` y se detiene el bucle.

⚠ Aunque `break` puede ser útil, **no es recomendable usarlo como regla general para controlar bucles**, ya que puede dificultar la lectura del código. Es mejor definir condiciones claras desde el inicio.

◆ `continue`

`continue` se usa para **saltar el resto del código en una vuelta del bucle**, y continuar con la siguiente iteración.

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

✚ Salida:

1
2
4
5

🔍 Cuando `i` vale 3, no se ejecuta el `print()` y se pasa directamente al siguiente valor.

👁 Se recomienda usar `continue` con moderación, ya que puede volver el código difícil de seguir si se abusa de esta técnica.

☑ Buenas prácticas al usar bucles

- Usá `for` cuando **sabés exactamente cuántas veces** se debe repetir el código.
- Usá `while` cuando **dependés de una condición** que puede cambiar durante el programa.
- Siempre asegurate de que las condiciones se actualicen correctamente, para evitar **bucles infinitos**.
- Evitá abusar de `break` y `continue`: si bien son útiles, es mejor escribir condiciones claras y bien estructuradas.
- Usá nombres de variables significativos: `contador`, `intento`, `letra`, etc.
- Probá tus bucles en papel antes de ejecutarlos: pensá qué valor tiene la variable en cada vuelta.

🔧 Ejercicios resueltos paso a paso

1. Mostrar los números del 1 al 10 usando `for`

```
for i in range(1, 11):  
    print(i)
```

🔍 Explicación:


- `range(1, 11)` genera los números del 1 al 10.
- La variable `i` toma esos valores uno por uno, y se imprime.

✚ Salida:

```
1
2
...
10
```

2. Pedir una contraseña hasta que sea "python"

```
clave = ""
while clave != "python":
    clave = input("Ingresá la contraseña: ")
print("¡Contraseña correcta!")
```

 Explicación:

- Se define la variable `clave` como texto vacío.
 - Se entra al `while` porque `"python"` no es igual a `""`.
 - Se vuelve a pedir hasta que el usuario escribe exactamente `"python"`.
-

3. Contar de 10 a 1 con `for`

```
for i in range(10, 0, -1):
    print(i)
```

 Explicación:

- Comienza en 10 y se va restando de 1 en 1.
- Termina cuando llega a 1 (el 0 no se incluye).

 Salida:

```
10
9
...
1
```

4. Mostrar del 1 al 5, pero omitir el 3

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

 Explicación:

- Cuando `i` vale 3, se salta esa vuelta gracias a `continue`.

📌 Salida:

1
2
4
5

5. Mostrar del 1 al 100, pero cortar en 15

```
for i in range(1, 101):  
    if i == 15:  
        break  
    print(i)
```

🔍 Explicación:

- El bucle se corta con `break` cuando llega a 15, por lo tanto no se imprime ese valor.

📌 Salida:

1
2
...
14