



1. ¿Qué es el bucle FOR?

El **for** es una estructura de **repetición** (o bucle) que permite ejecutar un bloque de instrucciones **una vez por cada elemento** de algo que se pueda recorrer. A ese "algo que se puede recorrer" se lo llama **iterable**.

A diferencia del **while**, que repite mientras se cumple una condición lógica (y nosotros tenemos que controlar cuándo cortar), el **for** ya sabe cuándo terminar: se detiene solo cuando terminó de recorrer todos los elementos del iterable. Por eso decimos que el **for** es un bucle **acotado** o **definido**, mientras que el **while** es más abierto.

Definición

Iterable: es cualquier objeto de Python que se puede recorrer elemento por elemento. Ejemplos: un rango de números (range), una cadena de texto, una lista, una tupla, y más adelante también diccionarios y conjuntos.

2. Sintaxis general

```
for variable in iterable:  
    # bloque de instrucciones (cuerpo del for)
```

Elementos de la sintaxis:

Elemento	¿Qué es?
for	Palabra clave que inicia el bucle.
variable	Nombre que elegimos nosotros; en cada vuelta (iteración) toma el valor de un elemento del iterable.
in	Palabra clave que conecta la variable con el iterable.
iterable	El conjunto de elementos a recorrer (un range, una cadena, una lista, etc.).
:	Indica que a continuación comienza el bloque de instrucciones.
bloque indentado	Las instrucciones que se repiten en cada iteración. Van con sangría (4 espacios).

Importante

La **indentación** no es estética: en Python define qué instrucciones pertenecen al bucle. Todo lo que esté indentado debajo del **for** se repite; en cuanto una línea vuelve al margen anterior, ya quedó fuera del bucle.

Ejemplo mínimo:



```
for numero in range(3):  
    print('Vuelta numero:', numero)  
# Salida:  
# Vuelta numero: 0  
# Vuelta numero: 1  
# Vuelta numero: 2
```



3. ¿Para qué se utiliza?

Se usa siempre que necesitamos **repetir una acción una cantidad determinada de veces**, o **procesar, uno por uno, todos los elementos de un conjunto de datos**. Algunos ejemplos típicos:

- Mostrar los números del 1 al 10.
- Saludar a cada alumno de una lista de nombres.
- Sumar todos los valores de una serie de datos.
- Recorrer las letras de una palabra para contar vocales.
- Repetir 5 veces la misma pregunta en un juego.

4. El argumento más común: range()

Cuando no tenemos una lista, tupla o cadena para recorrer, pero necesitamos repetir algo una cantidad determinada de veces, usamos la función **range()**, que genera una secuencia de números enteros.

range() puede recibir 1, 2 o 3 argumentos

- **range(stop)**: desde 0 hasta stop-1.
- **range(start, stop)**: desde start hasta stop-1.
- **range(start, stop, paso)**: desde start hasta stop-1, saltando de a "paso".

```
for i in range(5):           # 0, 1, 2, 3, 4
    print(i)
for i in range(2, 6):        # 2, 3, 4, 5
    print(i)
for i in range(0, 10, 2):    # 0, 2, 4, 6, 8
    print(i)
for i in range(10, 0, -1):   # 10, 9, 8, ..., 1 (cuenta regresiva)
    print(i)
```

Advertencia

El valor de **stop** **nunca se incluye** en el resultado. Por ejemplo, **range(5)** genera 0, 1, 2, 3 y 4 — llega hasta 4, no hasta 5. Es un error muy común olvidar esto y pensar que la última vuelta es "5".



5. ¿Qué otras cosas se pueden recorrer con FOR?

El for no es exclusivo de las listas y tuplas (esas las vamos a ver en detalle en el próximo apunte).
Cualquier **iterable** sirve:

```
# Una cadena de texto (recorre caracter por caracter)
for letra in 'Python':
    print(letra)
# Un rango de numeros
for numero in range(3):
    print(numero)
# Una lista
for elemento in [10, 20, 30]:
    print(elemento)
# Una tupla
for elemento in (1, 2, 3):
    print(elemento)
```

Para tener en cuenta

Más adelante en la materia vamos a ver que también se puede usar for con diccionarios y conjuntos (sets), que tienen su propia forma de recorrido.

6. Dos instrucciones que modifican el comportamiento del FOR

break

Corta el bucle **inmediatamente**, aunque falten elementos por recorrer.

```
for numero in range(10):
    if numero == 5:
        break          # corta al llegar a 5
    print(numero)
# Salida: 0 1 2 3 4
```

continue

Salta directamente a la siguiente iteración, sin ejecutar el resto del bloque para ese elemento.

```
for numero in range(10):
    if numero % 2 == 0:
        continue      # se salta los pares
    print(numero)
# Salida: 1 3 5 7 9
```



Resumen

- El for recorre un iterable y repite el bloque una vez por cada elemento.
- Sintaxis: **for variable in iterable:** seguido de un bloque indentado.
- range() genera secuencias numéricas cuando no hay una lista/tupla/cadena para recorrer.
- break corta el bucle; continue salta a la siguiente vuelta.
- En el próximo apunte vamos a aplicar todo esto al recorrido de listas y tuplas.



7. Actividades prácticas

Resolvé los siguientes ejercicios en Python usando `for`, `range()` y las cadenas de texto. Todavía no uses listas ni tuplas: eso lo vamos a ver en el próximo apunte. Escribí el código completo y probá cada programa antes de entregarlo.

Actividad 1 — Contar con `range()`

Usando un `for` con `range()`, mostrar en pantalla los números del 1 al 20 (incluido el 20).

Actividad 2 — Múltiplos

Usando `range()` con el argumento de "paso", mostrar todos los múltiplos de 3 entre 0 y 30 (incluido el 30).

Actividad 3 — Recorrido de una cadena

Pedirle al usuario que ingrese una palabra con `input()`. Recorrerla con `for` y contar cuántas vocales (a, e, i, o, u) tiene. Mostrar el resultado.

Actividad 4 — Cuenta regresiva con `break`

Usando `range()` en orden descendente (de 10 a 1), mostrar la cuenta regresiva. Al llegar a 5, cortar el bucle con `break` y mostrar el mensaje "¡Despegue anticipado!".

Actividad 5 — Saltear con `continue`

Usando un `for` con `range(1, 21)` (del 1 al 20), mostrar únicamente los números que **no** son múltiplos de 3, usando `continue` para saltar los que sí lo son.