Estructura de Repetición For

Algoritmos y Programación Básica (CC2005) - 2026

Estructuras de Repetición For

Semestre 02, 2026

El problema de repetir n veces

En el ciclo `while` la repetición depende de una condición.

¿Qué pasa cuando se sabe exactamente cuántas veces hay que repetir?

Con while

Imprimir los números del 1 al 10:

```
x = 1

while x <= 10:
    print(x)
    x = x + 1
```

- Funciona, pero hay que inicializar, evaluar y actualizar a mano.
- Son tres responsabilidades repartidas en tres líneas distintas.
- Olvidar la última produce un ciclo infinito.

La solución

El ciclo `for` fue diseñado exactamente para este caso.

- El número de repeticiones se conoce de antemano.
- La variable de control se gestiona sola.
- El código es más corto y más fácil de leer.
- No hay riesgo de ciclo infinito cuando se recorre un `range`.

El ciclo For

El ciclo `for` repite un bloque de código una vez por cada valor de una secuencia.

La variable de control toma cada valor automáticamente en cada iteración.

Sintaxis

```
for variable in secuencia:
    # bloque de código
```

- `variable` — toma el valor de cada elemento de la secuencia.
- `secuencia` — un `range`, una lista, un string o cualquier iterable.
- El bloque indentado se ejecuta una vez por cada valor.

Una diferencia importante

En otros lenguajes el `for` cuenta números.

En Python el `for` recorre los elementos de una secuencia, en el orden en que aparecen.

Contar del 1 al 10 es solo un caso particular: se recorre la secuencia que genera `range`.

Diferencia con while

Característica	while	for
Cuándo se usa	Cantidad desconocida	Cantidad conocida
Variable de control	Manual	Automática
Condición de parada	La escribe el programador	La define la secuencia
Riesgo de ciclo infinito	Sí	No con `range`

La función range()

`range()` genera una secuencia de números enteros.

Es la forma más común de controlar cuántas veces se repite un `for`.

Se puede llamar de tres formas, según cuánta información se le dé.

Las tres firmas

```
range(fin)                # arranca en 0, avanza de uno en uno
range(inicio, fin)        # arranca donde se indique, avanza de uno en uno
range(inicio, fin, paso)  # arranca donde se indique, avanza de paso en paso
```

El fin nunca se incluye

Las tres formas comparten una regla. Conviene leer `fin` como el primer valor que no se quiere.

- `range(1, 10)` genera del 1 al 9, no al 10.
- Para llegar hasta 10, se escribe `range(1, 11)`.

Esta es la fuente número uno de errores al empezar con `for`.

range con un argumento

```
range(fin)
```

El inicio es cero y el paso es uno aunque no se escriban. Lo único que hay que decir es dónde parar.

Ejemplo

```
for i in range(5):
    print(i)
```

Imprime 0, 1, 2, 3 y 4 — cinco vueltas exactas.

range con dos argumentos

```
range(inicio, fin)
```

Ahora el inicio es explícito. El fin se comporta igual, así que el último valor de la secuencia es el anterior.

Ejemplo

```
for i in range(2, 8):
    print(i)
```

Imprime 2, 3, 4, 5, 6 y 7.

La cantidad de valores es la resta del fin menos el inicio. Ocho menos dos da seis vueltas.

range con tres argumentos

```
range(inicio, fin, paso)
```

El paso dice de cuánto es cada salto.

Ejemplo

```
for i in range(1, 10, 3):
    print(i)
```

Imprime 1, 4 y 7.

El siguiente salto caería en 10, que es el fin y queda fuera.

El fin marca hasta dónde se puede llegar, no un valor que la secuencia deba alcanzar.

El paso puede ser negativo

```
range(10, 0, -2)
```

Genera 10, 8, 6, 4 y 2.

Con paso negativo la secuencia va hacia atrás y el inicio es mayor que el fin. El cero tampoco entra.

El paso debe apuntar hacia el fin

`range(10, 1, 1)` genera una secuencia vacía y el ciclo no da ni una vuelta.

No hay mensaje de error, el programa simplemente no hace nada.

Lo correcto es `range(10, 1, -1)`.

Resumen de range()

Expresión	Secuencia generada
`range(5)`	0, 1, 2, 3, 4
`range(2, 8)`	2, 3, 4, 5, 6, 7
`range(1, 10, 3)`	1, 4, 7
`range(10, 0, -2)`	10, 8, 6, 4, 2
`range(1, 1)`	vacía — el ciclo no se ejecuta

range no construye la lista

`range` no guarda todos los números en memoria.

Solo recuerda el inicio, el fin y el paso, y calcula cada valor cuando el ciclo lo pide.

Por eso `range(1, 1000000)` ocupa lo mismo que `range(1, 5)`.

Ejemplo 1: Contador básico

Imprimir los números del 1 al 5.

```
for x in range(1, 6):
    print("El valor de x es:", x)
```

Tabla de ejecución

Iteración	x	Pantalla
1	1	El valor de x es: 1
2	2	El valor de x es: 2
3	3	El valor de x es: 3
4	4	El valor de x es: 4
5	5	El valor de x es: 5

El ciclo termina solo — no hay que actualizar `x`.

Ejemplo 2: Suma acumulada

Sumar todos los números del 1 al 100.

```
total = 0

for numero in range(1, 101):
    total = total + numero

print("La suma es:", total)
```

Qué hace

- `total` arranca en cero porque todavía no se ha sumado nada.

- `range(1, 101)` genera los valores del 1 al 100.
- En cada vuelta `total` toma su valor anterior y le agrega el número actual.
- Al finalizar, `total` contiene 5050.

Resultado: `La suma es: 5050`

El acumulador va fuera del ciclo

La variable que acumula se crea antes del `for` y se imprime después.

Si el `print` estuviera indentado, el programa mostraría cien resultados parciales.

Ejemplo 3: Cuenta regresiva

Contar desde 10 hasta 1 y mostrar el despegue.

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

print("¡Despegue!")
```

Resultado

```
10
9
8
...
1
¡Despegue!
```

- `range(10, 0, -1)` genera 10, 9, 8, ..., 1.
- Con paso negativo el fin también queda fuera, por eso el 0 no aparece.

- El código después del `for` se ejecuta cuando el ciclo termina.

Ejemplo 4: Tabla de multiplicar

Mostrar la tabla de multiplicar del 7.

```
numero = 7

for i in range(1, 11):
    resultado = numero * i
    print(numero, "x", i, "=", resultado)
```

Resultado

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 10 = 70
```

El valor de `numero` no cambia — solo cambia `i` en cada iteración.

Ejemplo 5: Iteración sobre una lista

El `for` también recorre los elementos de una lista directamente.

```
frutas = ["manzana", "naranja", "uva", "mango"]

for fruta in frutas:
    print("Fruta:", fruta)
```

Resultado

```
Fruta: manzana
Fruta: naranja
Fruta: uva
Fruta: mango
```

- No se necesita `range()`.
- `fruta` toma automáticamente el valor de cada elemento.
- El ciclo da tantas vueltas como elementos tenga la lista.

Las listas se estudian a fondo más adelante.

También recorre un string

Un string es una secuencia de caracteres, así que el `for` lo recorre letra por letra.

```
for letra in "Python":
    print(letra)
```

Ejemplo 6: Promedio de calificaciones

Calcular el promedio de 5 calificaciones ingresadas por el usuario.

```
total = 0

for i in range(1, 6):
    nota = float(input("Ingresa la nota " + str(i) + ": "))
    total = total + nota

promedio = total / 5
print("Promedio:", promedio)
```

Qué hace

- El ciclo se repite exactamente 5 veces.
- En cada iteración pide una nota y la acumula.
- La variable `i` se usa solo para numerar la pregunta en pantalla.
- Al salir, calcula el promedio dividiendo entre 5.

Ciclos anidados

Un `for` puede contener otro `for` dentro de su bloque.

Por cada vuelta del ciclo externo, el interno se ejecuta completo.

Ejemplo

Las tablas del 1 al 3:

```
for tabla in range(1, 4):
    for i in range(1, 4):
        print(tabla, "x", i, "=", tabla * i)
    print("---")
```

Cuántas vueltas da

- El ciclo externo da 3 vueltas.
- El interno da 3 vueltas por cada una de ellas.
- En total se imprimen 9 resultados.

La variable de cada ciclo debe tener un nombre distinto.

continue

`continue` salta a la siguiente iteración sin terminar el ciclo.

Ejemplo

Imprimir solo los números pares del 1 al 10.

```
for x in range(1, 11):
    if x % 2 != 0:
        continue
    print(x)
```

Resultado: 2, 4, 6, 8, 10.

Cuándo se usa

- Cuando se quiere omitir ciertos valores sin interrumpir el ciclo.
- Útil para filtrar elementos dentro de una secuencia.

La instrucción break

`break` detiene un ciclo de forma abrupta desde adentro.

Existe en Python, pero en este curso no se usará.

El problema

Cuando se usa `break`, el ciclo ya no se controla por su secuencia.

- La lógica de terminación queda escondida dentro del bloque.
- El flujo del programa es más difícil de seguir.

Siempre hay una mejor alternativa

En lugar de esto:

```
for i in range(1, 11):
    if i == 5:
        break
    print(i)
```

La alternativa correcta

```
for i in range(1, 5):
    print(i)
```

- La condición de parada queda explícita en el `range`.
- El ciclo es claro, predecible y fácil de leer.

Errores comunes

Error 1: confundir el límite de range

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

Imprime del 0 al 9, no del 0 al 10.

`range(10)` genera 0, 1, 2, ..., 9 — el 10 no se incluye.

Error 2: modificar la variable de control

```
for i in range(1, 6):
    print(i)
    i = i + 10  # no tiene efecto
```

- Modificar `i` dentro del ciclo no cambia la secuencia.
- En la siguiente iteración, `i` toma el siguiente valor del `range`.

Error 3: usar range con el paso equivocado

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

No imprime nada.

El paso es positivo pero el inicio es mayor que el fin, así que la secuencia queda vacía.

Correcto: `range(10, 1, -1)`

Error 4: recorrer un número

```
for i in 5:
    print(i)
```

Produce `TypeError: 'int' object is not iterable`.

Un número no es una secuencia. Lo correcto es `range(5)`.

Error 5: olvidar la indentación

```
for i in range(1, 4):
    print(i)
print("Fin")
```

`print("Fin")` está fuera del ciclo y se ejecuta una sola vez.

La indentación es lo único que marca qué pertenece al bloque.

Casos de uso

¿Cuándo se usa `for` y cuándo `while`?

Situación	for o while
Repetir exactamente n veces	`for`
Recorrer una lista o un string	`for`
Repetir mientras una condición	`while`
Validar entrada del usuario	`while`
No se sabe cuántas repeticiones	`while`

Buenas prácticas

- Usar `range(inicio, fin)` cuando se conocen los límites exactos.
- No modificar la variable de control dentro del ciclo.
- Usar nombres descriptivos: `for estudiante in lista` es más claro que `for x in lista`.
- Preferir `for` sobre `while` cuando el número de repeticiones es fijo.
- Nombrar `_` a la variable cuando su valor no se usa dentro del bloque.

Antes de escribir el ciclo

Preguntarse:

- ¿Sé exactamente cuántas veces se repite?
- ¿Recorro una secuencia de valores?

Si la respuesta es sí, usar `for`.