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 repetir?

Ejemplo

Imprimir los números del 1 al 10 usando `while`:

```
x = 1

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

- Funciona, pero requiere inicializar, evaluar y actualizar manualmente.
- Es repetitivo para algo tan simple.

La solución

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

- Número de repeticiones conocido de antemano.
- La variable de control se gestiona automáticamente.
- El código es más corto y fácil de leer.

El ciclo For

El ciclo `for` repite un bloque de código para 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` — puede ser un `range`, una lista, o cualquier iterable.
- El bloque se ejecuta una vez por cada valor.

Diferencia con while

| Característica | while | for | | --- | --- | --- | | ¿Cuándo se usa? | Condición desconocida | Cantidad conocida | | Variable de control | Manual | Automática | | Riesgo de ciclo infinito | Sí | No |

La función range()

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

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

Formas de uso

```
range(fin)             # 0, 1, 2, ..., fin-1
range(inicio, fin)     # inicio, inicio+1, ..., fin-1
range(inicio, fin, paso)  # avanza de "paso" en "paso"
```

El valor `fin` nunca se incluye en la secuencia.

Ejemplos de range()

| Expresión | Secuencia generada | | --- | --- | | `range(5)` | 0, 1, 2, 3, 4 | | `range(1, 6)` | 1, 2, 3, 4, 5 | | `range(0, 10, 2)` | 0, 2, 4, 6, 8 | | `range(10, 0, -1)` | 10, 9, 8, ..., 1 |

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?

- `range(1, 101)` genera los valores del 1 al 100.
- `total` acumula la suma en cada iteración.
- Al finalizar, `total` contiene 5050.

Resultado: `La suma es: 5050`

Ejemplo 3: Cuenta regresiva

Contar desde 10 hasta 1 y mostrar "¡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.
- 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 puede recorrer 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.

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.
- Al salir, calcula el promedio dividiendo entre 5.

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.

¿Por qué no usar break?

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

Existe, 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 oculta 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 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.

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

Casos de uso

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

| Situación | ¿for o while? | | --- | --- | | Repetir exactamente n veces | `for` | | Recorrer una lista | `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.

Antes de escribir el ciclo

Preguntarse:

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

Si la respuesta es sí, usar `for`.