Estructura de Repetición While

Algoritmos y Programación Básica (CC2005) - 2026

Estructuras de Repetición While

Semestre 02, 2026

Problema

- A veces se necesita repetir una acción muchas veces.
- Hacerlo línea por línea no es práctico.
- El número de repeticiones puede depender de los datos.

Ejemplo

Se deben imprimir los números del 1 al 100.

¿Se escribirían 100 líneas de `print`?

Se necesita una forma de automatizar la repetición.

Definición

Una estructura de repetición permite ejecutar un bloque de código múltiples veces.

Esto hace que los programas sean más cortos y más potentes.

Tipos

Existen 2 tipos de estructuras de repetición:

- Por condición → se repite mientras una condición sea verdadera.
- Por cantidad → se repite un número fijo de veces.

Variable de control

Ambos tipos usan una variable de control.

- Guarda el valor de la iteración actual.
- Se evalúa en la condición.
- Debe actualizarse dentro del ciclo.

El ciclo While

El ciclo `while` se repite mientras la condición sea verdadera.

Se detiene cuando la condición se vuelve falsa.

¿Cuándo se usa?

Se usa `while` cuando no se sabe de antemano cuántas repeticiones se necesitan.

- El usuario decide cuándo parar.
- Se espera un evento.
- Se valida una entrada.

Sintaxis

```
while condicion:
    # bloque de código
```

- La condición se evalúa antes de cada iteración.
- Si la condición es falsa desde el inicio, el bloque nunca se ejecuta.

Ejemplo 1: Contador básico

Imprimir los números del 1 al 5.

```
x = 1

while x <= 5:
    print("El valor de x es:", x)
    x = x + 1
```

Tabla de ejecución

| n | x | Condición | Pantalla | | --- | --- | --- | --- | | 1 | 1 | 1 <= 5 ✓ | 1 | | 2 | 2 | 2 <= 5 ✓ | 2 | | 3 | 3 | 3 <= 5 ✓ | 3 | | 4 | 4 | 4 <= 5 ✓ | 4 | | 5 | 5 | 5 <= 5 ✓ | 5 | | 6 | 6 | 6 <= 5 ✗ | — |

Pregunta

¿Qué pasa si nunca se escribe `x = x + 1`?

- La condición siempre es verdadera.
- El ciclo nunca termina.
- Se llama ciclo infinito.

Ejemplo 2: Cuenta regresiva

Contar desde 10 hasta 1 y mostrar "¡Despegue!".

```
contador = 10

while contador > 0:
    print(contador)
    contador = contador - 1

print("¡Despegue!")
```

Resultado

```
10
9
8
...
1
¡Despegue!
```

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

Ejemplo 3: Suma acumulada

Sumar todos los números del 1 al 100.

```
numero = 1
total = 0

while numero <= 100:
    total = total + numero
    numero = numero + 1

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

¿Qué hace?

- `total` acumula la suma en cada iteración.
- `numero` avanza de 1 en 1.
- Al final, `total` contiene la suma de 1 a 100.

Resultado: `La suma es: 5050`

Ejemplo 4: Entrada del usuario

Pedir un número positivo al usuario.

```
numero = int(input("Ingresa un número positivo: "))

while numero <= 0:
    print("El número debe ser positivo.")
    numero = int(input("Ingresa un número positivo: "))

print("Número aceptado:", numero)
```

¿Por qué es útil?

- No se sabe cuántas veces el usuario va a equivocarse.
- El ciclo repite la petición hasta que el dato sea válido.
- Es un patrón muy común en programas reales.

Ejemplo 5: Menú interactivo

Mostrar un menú hasta que el usuario elija salir.

```
opcion = 0

while opcion != 3:
    print("1. Saludar")
    print("2. Despedirse")
    print("3. Salir")
    opcion = int(input("Elige una opción: "))

    if opcion == 1:
        print("¡Hola!")
    elif opcion == 2:
        print("¡Adiós!")

print("Programa terminado.")
```

¿Qué ocurre?

- El menú se muestra en cada iteración.
- El ciclo continúa mientras la opción no sea 3.
- Al elegir 3, el ciclo termina y se imprime el mensaje final.

Ejemplo 6: Adivina el número

El usuario intenta adivinar un número secreto.

```
secreto = 42
intento = int(input("Adivina el número: "))

while intento != secreto:
    if intento < secreto:
        print("Muy bajo. Intenta de nuevo.")
    else:
        print("Muy alto. Intenta de nuevo.")
    intento = int(input("Adivina el número: "))
```

```
print("¡Correcto! El número era", secreto)
```

Análisis

- El ciclo no tiene un número fijo de intentos.
- Depende completamente del usuario.
- El `while` es ideal para este tipo de situaciones.

Ciclo infinito

Un ciclo infinito ocurre cuando la condición nunca se vuelve falsa.

El programa queda atascado y hay que detenerlo manualmente.

Ejemplo de ciclo infinito

```
x = 1

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

- `x` nunca cambia.
- La condición `x <= 10` siempre es `True`.
- El programa imprime `1` indefinidamente.

¿Cómo evitarlo?

- Siempre actualizar la variable de control dentro del ciclo.
- Verificar que la condición eventualmente se vuelva falsa.

- Revisar la lógica antes de ejecutar.

continue

¿Qué es continue?

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

Ejemplo

```
x = 0

while x < 10:
    x = x + 1
    if x % 2 == 0:
        continue
    print(x)
```

Resultado: se imprimen solo los números impares del 1 al 9.

¿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 condición.

- La condición del `while` se vuelve irrelevante.
- El flujo del programa se vuelve difícil de seguir.

- Es más difícil entender cuándo y por qué termina el ciclo.

Siempre hay una mejor alternativa

En lugar de esto:

```
x = 1

while True:
    if x == 5:
        break
    print(x)
    x = x + 1
```

La alternativa correcta

```
x = 1

while x < 5:
    print(x)
    x = x + 1
```

- La condición de parada está en el `while`.
- El ciclo es claro, predecible y fácil de leer.
- No se necesita `break`.

Regla

Si se siente la necesidad de usar `break`, hay que preguntarse:

¿Se puede expresar esta condición de parada directamente en el `while`?

La respuesta casi siempre es sí.

Errores comunes

Error 1: olvidar actualizar la variable

```
x = 1

while x <= 5:
    print(x)
    # nunca se actualiza x
```

Ciclo infinito garantizado.

Error 2: condición incorrecta

```
x = 10

while x <= 5:
    print(x)
    x = x + 1
```

El ciclo nunca se ejecuta porque la condición es falsa desde el inicio.

Error 3: comparar tipos distintos

```
respuesta = input("¿Continuar? (s/n): ")

while respuesta != "n":
    respuesta = input("¿Continuar? (s/n): ")
```

Funciona, pero si se escribe `"N"` con mayúscula, no se detecta.

Mejor: `respuesta.lower() != "n"`

Casos de uso

¿Cuándo se usa `while`?

| Situación | ¿Usar while? | | --- | --- | | No se sabe cuántas veces repetir | ✓ | | Se espera una condición externa | ✓ | | Se valida entrada del usuario | ✓ | | Se sabe exactamente cuántas veces | ✗ |

Para un número fijo de repeticiones, existe el ciclo `for`.

Buenas prácticas

- Siempre actualizar la variable de control.
- Asegurarse de que la condición eventualmente sea falsa.
- Usar nombres de variables descriptivos.
- Evitar lógica muy compleja dentro del ciclo.

Recomendación

Antes de escribir el ciclo, es útil preguntarse:

- ¿Cuál es la condición de parada?
- ¿Qué cambia en cada iteración?
- ¿Puede la condición volverse falsa?

Actividad práctica

Situación

Un cajero automático permite al usuario retirar dinero.

El saldo inicial es `Q500.00`.

Instrucciones

1. Mostrar el saldo disponible.
2. Pedir al usuario cuánto desea retirar.
3. Si el monto es mayor al saldo, mostrar un mensaje de error.
4. Si el monto es válido, restar del saldo.
5. Repetir hasta que el usuario ingrese `0` o el saldo sea `0`.
6. Al final, mostrar el saldo restante.

Preguntas

- ¿Cuál es la condición del `while`?
- ¿Qué variable de control se necesita?
- ¿Qué pasa si el usuario intenta retirar más de lo disponible?