Sentencias Iterativas en Python: Repetir para Resolver

¿Qué es una sentencia iterativa y por qué se necesita?

En la resolución de problemas mediante programación, una necesidad común es **repetir una acción varias veces** hasta que se cumpla cierta condición o se haya recorrido una colección completa de datos. Este patrón de repetición se conoce como **iteración**, y Python lo resuelve con estructuras llamadas **sentencias iterativas**.

Una sentencia iterativa permite ejecutar **un bloque de código múltiples veces**. Esto es esencial para tareas como recorrer listas, contar elementos, validar datos repetidamente o generar secuencias de información. En Python, las estructuras más comunes para realizar iteraciones son: `while` y `for`.

El bucle `while`: Iterar mientras se cumple una condición

La sentencia `while` permite ejecutar un bloque de código **mientras una condición sea verdadera**. Es útil cuando no sabemos cuántas veces necesitaremos repetir una acción, ya que depende de una condición lógica que puede cambiar dentro del ciclo.

Estructura general:

```
while condición:
    # bloque de código a ejecutar
```

Ejemplo práctico:

```
contador = 0
while contador < 5:
    print("Contador:", contador)
    contador += 1
```

Este código imprimirá los valores del contador desde 0 hasta 4. La condición se evalúa antes de cada iteración, y el ciclo se detiene cuando ya no se cumple.

Blandskron

El bucle `for`: Iterar sobre colecciones

La sentencia `for` permite recorrer directamente los elementos de una **colección iterable**, como listas, diccionarios, cadenas o sets. Es la forma preferida en Python para recorrer estructuras de datos sin preocuparse del control del índice.

Estructura general:

```
for elemento in colección:
    # bloque de código
```

Ejemplo con lista:

```
frutas = ["manzana", "plátano", "pera"]
for fruta in frutas:
    print("Me gusta la", fruta)
```

Iterando una cadena:

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

En ambos ejemplos, Python accede a cada elemento de la colección y lo asigna a la variable del bucle en cada iteración.

La función `range()`: Generar secuencias numéricas

Cuando queremos recorrer una secuencia de números (por ejemplo, del 0 al 9), usamos la función `range()`, que genera automáticamente un rango de valores enteros. Es ideal para ciclos `for` basados en conteo.

Ejemplo básico:

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

Este ciclo imprimirá los números del 0 al 4.

También podemos personalizar el inicio, fin y paso:

```
for i in range(2, 10, 2):
    print(i)  # Imprime: 2, 4, 6, 8
```

Iterando Listas y Diccionarios

Listas:

```
nombres = ["Ana", "Luis", "Carlos"]
for nombre in nombres:
    print("Hola", nombre)
```

Diccionarios:

Al iterar un diccionario, se puede acceder a:

- Solo las claves: `for clave in diccionario`
- Claves y valores: `for clave, valor in diccionario.items()`

```
persona = {"nombre": "Ana", "edad": 30}
for clave, valor in persona.items():
    print(clave, ":", valor)
```

Aplicación Pedagógica: Iteración para resolver un problema

Supongamos que queremos calcular el promedio de una lista de números:

```
numeros = [10, 20, 30, 40]
suma = 0

for numero in numeros:
    suma += numero

promedio = suma / len(numeros)
print("El promedio es:", promedio)
```

Este ejemplo muestra cómo usar `for` para **acumular valores de una lista** y luego aplicar una operación matemática.

Conclusión: Repetición como estrategia para automatizar

Las sentencias iterativas permiten que un programa **automatice tareas repetitivas** de forma eficiente y precisa. La elección entre `while` y `for` depende del tipo de repetición que se desea lograr:

- `while`: se usa cuando **la cantidad de repeticiones no es fija**
- `for`: se usa cuando **hay una secuencia definida o colección a recorrer**

Dominar estas estructuras es un paso esencial en la construcción de programas dinámicos y resolutivos, especialmente cuando se trabaja con grandes volúmenes de datos o flujos de decisión repetitivos.