Bucles en Python: Repetición y Control de Flujo

Los bucles son una de las herramientas fundamentales en la programación que permiten ejecutar un bloque de código repetidamente. Esta capacidad de automatizar tareas repetitivas es esencial para la eficiencia y la escalabilidad de cualquier programa. Python ofrece dos tipos principales de bucles: el bucle `for` y el bucle `while`.

I. El Bucle `for`: Iteración sobre Secuencias

El bucle `for` en Python se utiliza principalmente para **iterar** sobre los elementos de una secuencia (como listas, tuplas, cadenas, diccionarios o conjuntos) o sobre cualquier objeto que sea *iterable*. Es ideal cuando se conoce de antemano el número de repeticiones (por ejemplo, el número de elementos en una lista).

1. Sintaxis Básica

La sintaxis del bucle `for` es clara y concisa:for variable_de_iteracion in secuencia:

```
    # Bloque de código a ejecutar en cada iteración
    instruccion_1
    instruccion_2
    # ...
```

- `variable_de_iteracion`: Es una variable temporal que toma el valor de cada elemento de la secuencia en cada paso del bucle.
- `secuencia`: Es el objeto iterable sobre el cual se desea recorrer.

2. Ejemplos de Uso del Bucle `for`

A. Iteración sobre una Lista

```
frutas = ["manzana", "banana", "cereza"]

for fruta in frutas:
    print(f"Me gusta comer {fruta}.")
```

B. Iteración sobre una Cadena de Texto

Una cadena es una secuencia de caracteres.palabra = "Python"

for letra in palabra:

    print(letra)

C. Uso de la Función `range()`

La función `range()` es una herramienta muy común con el bucle `for`, ya que genera una secuencia de números.

- **`range(n)`**: Genera números desde 0 hasta *n-1*.
- **`range(inicio, fin)`**: Genera números desde *inicio* hasta *fin-1*.
- **`range(inicio, fin, paso)`**: Genera números desde *inicio* hasta *fin-1*, incrementando en el valor de *paso*.

Ejemplo 1: Contar de 0 a 4

for i in range(5):

    print(f"Contador: {i}")

Ejemplo 2: Contar de 2 a 8

for num in range(2, 9):

    print(num)

Ejemplo 3: Contar de 10 a 0 de 2 en 2 (descendente)

for j in range(10, -1, -2):

    print(j)

D. Iteración sobre un Diccionario

Al iterar sobre un diccionario, se itera por defecto sobre sus claves.estudiante = {"nombre": "Ana", "edad": 20, "curso": "Programación"}

Iterar sobre las claves

for clave in estudiante:

```
    print(f"Clave: {clave}")

# Iterar sobre los valores

for valor in estudiante.values():

    print(f"Valor: {valor}")

# Iterar sobre claves y valores (ítems)

for clave, valor in estudiante.items():

    print(f"{clave}: {valor}")
```

3. La Cláusula `else` en el Bucle `for`

Python permite añadir una cláusula `else` al final de un bucle `for`. El bloque de código dentro de `else` **se ejecuta solo si el bucle finaliza su iteración completamente** (es decir, no fue interrumpido por una instrucción `break`).numeros = [1, 2, 3, 4, 5]

```
for n in numeros:

    print(n)

else:

    print("El bucle ha terminado sin interrupciones.")
```

II. El Bucle `while`: Repetición Condicional

El bucle `while` se utiliza para ejecutar un bloque de código **mientras una condición determinada sea verdadera**. Es ideal cuando el número de repeticiones no se conoce de antemano y depende de que se cumpla una condición.

1. Sintaxis Básica

```
while condicion_logica:

    # Bloque de código a ejecutar

    instruccion_1
```

```
    instruccion_2

    # Es crucial que una de las instrucciones modifique la condición,

    # para evitar un bucle infinito.
```

2. Ejemplos de Uso del Bucle `while`

A. Ejemplo Básico con Contador

```
contador = 0

while contador < 5:

    print(f"El contador es: {contador}")

    contador += 1 # Es necesario incrementar la variable para que la condición cambie
```

B. Solicitud de Entrada del Usuario

```
entrada = ""

while entrada != "salir":

    entrada = input("Escribe algo ('salir' para terminar): ").lower()

    if entrada != "salir":

        print(f"Escribiste: {entrada}")
```

3. Bucles Infinitos

Un bucle infinito ocurre si la condición del bucle `while` nunca llega a ser falsa. Esto es generalmente un error, a menos que el programa esté diseñado para correr indefinidamente (como un servidor).

```
# PELIGRO: Esto es un bucle infinito

# while True:

#     print("Ejecutándose...")

#     # Se necesita un mecanismo de salida, como 'break' o modificar la condición
```

4. La Cláusula `else` en el Bucle `while`

Similar al `for`, el `while` también puede tener una cláusula `else`. El bloque `else` se ejecuta cuando la condición del `while` se vuelve falsa, **a menos que el bucle se haya terminado prematuramente con `break`**.i = 0

while i < 3:

    print(i)

    i += 1

else:

    print("El bucle while terminó porque la condición se volvió falsa.")

III. Control de Flujo de Bucles: `break` y `continue`

Dentro de cualquier bucle (`for` o `while`), se pueden utilizar dos sentencias especiales para controlar el flujo de ejecución: `break` y `continue`.

1. La Sentencia `break`

La sentencia `break` detiene inmediatamente la ejecución del bucle actual y el control del programa salta a la instrucción que sigue inmediatamente después del bucle.for numero in range(10):

    if numero == 5:

        print("Encontrado el 5, terminando el bucle.")

        break

    print(numero) # Imprime 0, 1, 2, 3, 4

print("Fuera del bucle.")

Nota importante: Si se usa `break`, la cláusula `else` del bucle (si existe) **no** se ejecuta.

2. La Sentencia `continue`

La sentencia `continue` salta el resto del código dentro del cuerpo del bucle para la iteración actual, e inmediatamente comienza la siguiente iteración del bucle.for i in range(5):

```
    if i % 2 == 0: # Si el número es par

        continue   # Omite la impresión para este número

    print(f"Número impar: {i}") # Imprime 1, 3
```

IV. Bucles Anidados

Es posible colocar un bucle dentro de otro bucle. Esto se conoce como **bucles anidados**. Son comúnmente utilizados para trabajar con estructuras de datos bidimensionales (como matrices) o para generar combinaciones.

Ejemplo de Bucles `for` Anidados

El bucle exterior se ejecuta una vez por cada elemento. El bucle interior se ejecuta *completamente* por cada iteración del bucle exterior.# Imprimir una tabla de multiplicar básica (3x3)

```
for i in range(1, 4):  # Bucle exterior (Filas)

    for j in range(1, 4):  # Bucle interior (Columnas)

        resultado = i * j

        print(f"{i} * {j} = {resultado}", end="\t") # '\t' añade una tabulación

    print() # Salto de línea después de cada fila
```

Salida esperada:1 * 1 = 1 1 * 2 = 2 1 * 3 = 3

2 * 1 = 2 2 * 2 = 4 2 * 3 = 6

3 * 1 = 3 3 * 2 = 6 3 * 3 = 9

V. Buenas Prácticas y Consideraciones

Aspecto	Consideración
Elección del Bucle	Usar `for` cuando se itera sobre una colección o se conoce el número de repeticiones (`range`). Usar `while` cuando la repetición depende de una condición dinámica o de entrada del usuario.
Bucles Infinitos	En bucles `while`, asegurar que la condición se modifique dentro del cuerpo del bucle para evitar ejecuciones infinitas no deseadas.
Índices vs. Iteradores	En Python, es preferible iterar directamente sobre los elementos (`for item in lista:`) en lugar de usar índices (`for i in range(len(lista)):`), a menos que sea estrictamente necesario modificar la lista o acceder al índice. Si se necesita el índice, usar `enumerate()`.
`enumerate()`	Para obtener tanto el índice como el valor al iterar sobre una secuencia, usar `for indice, valor in enumerate(secuencia):`.
Comprensiones de Lista	Para tareas sencillas de construcción de listas basadas en iteración, las comprensiones de lista (`[expresion for item in iterable]`) son más concisas y eficientes que los bucles `for` tradicionales.