Laços de Repetição (loop)

Introdução

Laços de reprodução, ou *loops* , são estruturas fundamentais de programação que permitem executar um bloco de código várias vezes, com base em uma condição ou uma sequência de elementos. Em Python, os principais laços são `for` e `while` . Eles são amplamente usados para analisar coleções de dados, como listas, conjuntos e arrays (geralmente representados por listas em Python).

1. Laço for

O laço `for` é usado para iterar sobre uma sequência de elementos, como listas, conjuntos, tuplas, strings ou intervalos numéricos. Ele percorre cada item da sequência automaticamente, sem a necessidade de controlar manualmente a iteração.

```
for variavel in sequencia:
    # Bloco de código a ser executado
```

Como funciona

- Assuma `variavel` , a cada iteração, o valor de um elemento da `sequencia` .
- O bloco de código dentro do laço é executado para cada elemento.
- Quando todos os elementos da sequência forem processados, o laço termina.

Exemplo 1: Iterando sobre uma lista

```
frutas = ["maçã", "banana", "laranja", "uva"]
for fruta in frutas:
    print(f"Fruta: {fruta}")
```

Saída:

```
Fruta: maçã
Fruta: banana
Fruta: laranja
Fruta: uva
```

Exemplo 2: Iterando sobre um conjunto

Conjuntos (`set`) são coleções não ordenadas de elementos únicos. O laço `for` pode ser usado para percorrer seus elementos, mas a ordem não é garantida.

```
numeros = {1, 2, 3, 4, 5}
for num in numeros:
    print(f"Número: {num}")
```

Saída (o pedido pode variar):

```
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
```

Exemplo 3: Usando `range()` para números

O `range()` é uma função que gera uma sequência de números, ideal para iterações baseadas em índices.

```
for i in range(5):  # Gera números de 0 a 4
    print(f"Índice: {i}")
```

Saída:

```
Índice: 0
Índice: 1
Índice: 2
```

```
Índice: 3
Índice: 4
```

2. Laço `while`

O laço `while` executa um bloco de código enquanto uma condição específica para verdadeira. É útil quando o número de iterações não é conhecido anteriormente.

Sintaxe

```
while condicao:
    # Bloco de código a ser executado
```

Como funciona

- A `condicao` é avaliada antes de cada iteração.
- Se a condição for `True`, o bloco de código será executado.
- Se a condição for `False`, o laço termina.
- **Cuidado**: Um laço `while` mal configurado pode resultar em um *laço infinito*.

Exemplo 1: Contagem com lista

```
numeros = [10, 20, 30, 40, 50]
indice = 0
while indice < len(numeros):
    print(f"Elemento: {numeros[indice]}")
    indice += 1
```

Saída:

```
Elemento: 10
Elemento: 20
Elemento: 30
Elemento: 40
Elemento: 50
```

Exemplo 2: Filtrando elementos de um conjunto

```
numeros = {2, 4, 6, 8, 10}
soma = 0
while numeros:
    num = numeros.pop()  # Remove e retorna um elemento do conjunto
```

```
    if num % 2 == 0:
        soma += num
    print(f"Soma parcial: {soma}")
```

Saída (o pedido pode variar):

```
Soma parcial: 2
Soma parcial: 6
Soma parcial: 12
Soma parcial: 20
Soma parcial: 30
```

3. Laços aninhados

Laços ser aninhados, ou seja, um laço dentro de outro pode. Isso é útil para trabalhar com estruturas de dados complexas, como listas de listas.

Exemplo: Matriz (lista de listas)

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for linha in matriz:
    for elemento in linha:
        print(f"Elemento: {elemento}")
```

Saída:

```
Elemento: 1
Elemento: 2
Elemento: 3
Elemento: 4
Elemento: 5
Elemento: 6
Elemento: 7
Elemento: 8
Elemento: 9
```

4. Controle de laços

Python oferece instruções para controlar o fluxo dos laços:

- `break` : Interrompe o laço imediatamente.

- `continue` : Pula para a próxima iteração, ignorando o restante do bloco atual.
- `else` : Executado quando o laço termina normalmente (sem `break`).

Exemplo com `break` e `continue`

```
numeros = [1, 2, 3, 4, 5, 6]
for num in numeros:
    if num == 4:
        print("Encontramos o 4, parando o laço!")
        break
    if num % 2 == 0:
        continue
    print(f"Número ímpar: {num}")
```

Saída:

```
Número ímpar: 1
Número ímpar: 3
Encontramos o 4, parando o laço!
```

Exemplo com `else`

```
numeros = [1, 3, 5, 7]
for num in numeros:
    if num % 2 == 0:
        print("Encontramos um número par!")
        break
else:
    print("Nenhum número par encontrado.")
```

Saída:

```
Nenhum número par encontrado.
```

5. Exercícios

Exercício 1: Soma de números positivos Escreva um programa que peça ao usuário para inserir números inteiros (um por vez) até digitar 0. Use um laço while para somar apenas os números positivos inseridos e armazene-os em uma lista. Mostre a soma e a lista final. Exemplo:

```
Digite um número (0 para parar): 5
Digite um número (0 para parar): -3
```

```
Digite um número (0 para parar): 10
Digite um número (0 para parar): 0
Saída: Soma dos positivos: 15, Lista: [5, 10]
```

Exercício 2: Maior número com limite Peça ao usuário para inserir números inteiros até digitar um número negativo. Use um laço while para encontrar o maior número inserido, mas ignore números maiores que 100. Se nenhum número válido for inserido, informe. Exemplo:

```
Digite um número (negativo para parar): 50
Digite um número (negativo para parar): 120
Digite um número (negativo para parar): 30
Digite um número (negativo para parar): -1
Saída: Maior número válido: 50
```

```
Entrada:
Digite um número (negativo para parar): 150
Digite um número (negativo para parar): -1
Saída: Nenhum número válido inserido.
```

Exercício 3: Tabuada personalizada Peça ao usuário para inserir um número inteiro entre 1 e 10. Use laços for aninhados para gerar a tabuada desse número, mas apenas para multiplicadores ímpares (1, 3, 5, 7, 9). Valide a entrada com if.

Exemplo:

```
Digite um número (1 a 10): 5

5 x 1 = 5
5 x 3 = 15
5 x 5 = 25
5 x 7 = 35
5 x 9 = 45
```

```
Entrada: Digite um número (1 a 10): 12
Saída: Número inválido! Digite entre 1 e 10.
```

Exercício 8: Contagem com salto Peça ao usuário para inserir um número inteiro positivo. Use um laço while para contar de 1 até esse número, mas pule números que são múltiplos de 4 (usando continue). Se o número for inválido (não positivo), informe. Exemplo: Entrada: Digite um número positivo: 10 Saída: 1, 2, 3, 5, 6, 7, 9, 10

Entrada: Digite um número positivo: 0 Saída: Número inválido! Deve ser positivo.

Exercício 9: Procurar elemento com validação Peça ao usuário para inserir uma lista de números inteiros (separados por espaço) e um número para buscar. Use um laço for com break para encontrar a primeira ocorrência do número. Use if para verificar se o número está na lista e exibir sua posição (baseada em 1). Se não estiver, use else. Exemplo: Entrada: Digite números separados por espaço: 10 20 30 40 Digite o número a buscar: 30 Saída: Número 30 encontrado na posição 3

Entrada: Digite números separados por espaço: 10 20 40 Digite o número a buscar: 50 Saída: Número 50 não encontrado.

Exercício 10: Somar até atingir limite ou esgotar Peça ao usuário para inserir uma lista de números inteiros (separados por espaço). Use um laço while para somar os números até que a soma ultrapasse 100 ou a lista acabe. Inclua apenas números positivos e pares (usando if). Mostre a soma final e quantos números foram usados. Exemplo: Entrada: Digite números separados por espaço: 10 20 -5 30 60 Saída: Soma: 120, Números usados: 4

Entrada: Digite números separados por espaço: 2 3 5 Saída: Soma: 2, Números usados: 1

Dicas para resolução

Use input() para capturar entradas do usuário. Converta strings para inteiros com int() ou listas com split() e map(). Exemplo: numeros = list(map(int, input("Digite números: ").split())). Valide entradas com if para garantir que os dados são válidos (ex.: números positivos, intervalo correto). Use try-except se quiser lidar com entradas inválidas (ex.: letras em vez de números). Para conjuntos, use set() para criar a coleção a partir da entrada. Teste os programas com diferentes casos, incluindo entradas vazias ou inválidas. Use break e continue para controlar o fluxo dos laços quando necessário. Para maior clareza, exiba mensagens informativas antes de pedir entradas.

Esses exercícios aumentam a complexidade ao integrar input e if, mas mantêm a dificuldade média-baixa, ideal para praticar laços de repetição com interação e lógica condicional.