Introdução aos Laços em Python

Tempo estimado necessário: 10 minutos

Objetivos

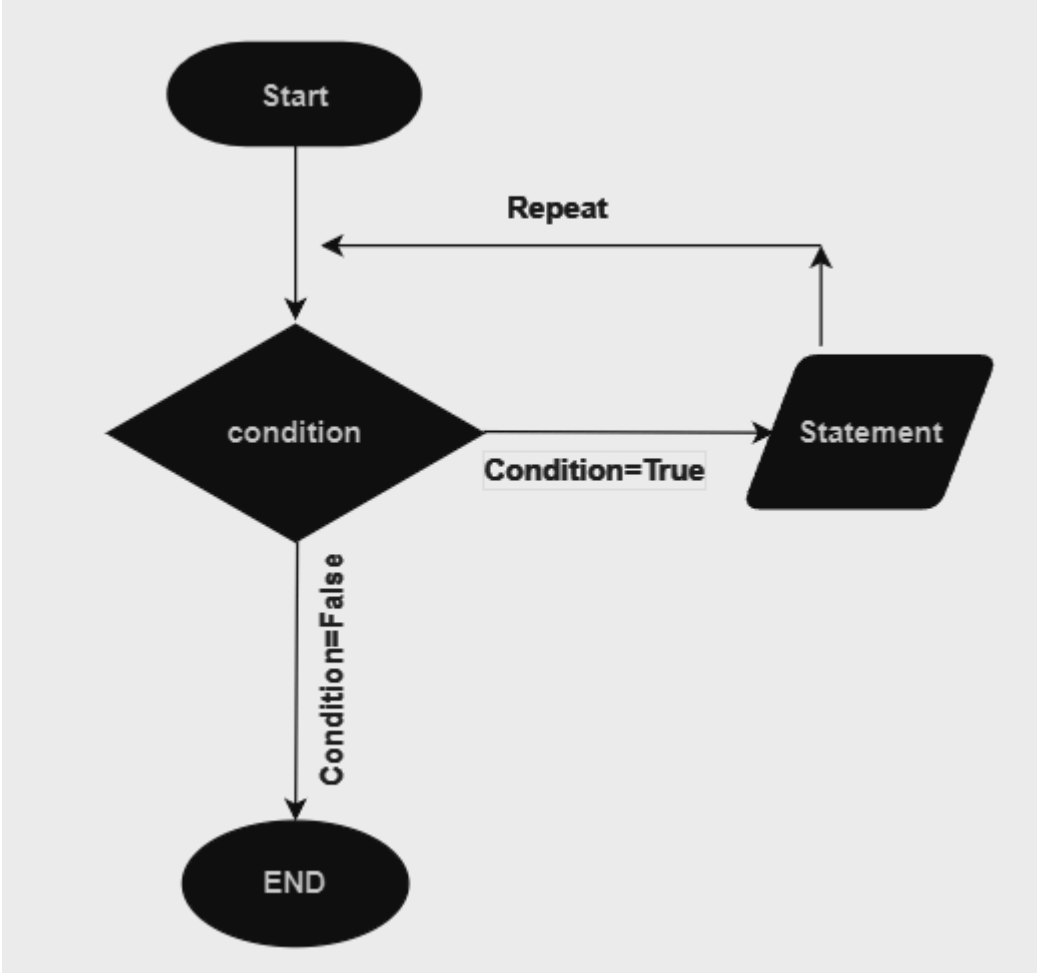
1. Compreender os loops em Python.
2. Como o loop funciona.
3. Aprender sobre as necessidades de um loop.
4. Utilizar a função Range do Python.
5. Familiarizar-se com a função enumerate do Python.
6. Aplicar loops while para tarefas condicionais.
7. Distinguir a seleção apropriada de loops.

O que é um Loop?

Na programação, um loop é como um truque de mágica que permite que um computador faça algo repetidamente. Imagine que você é o assistente de um mágico, e seu amigo mágico pede que você tire um coelho de um chapéu, mas não apenas uma vez - ele quer que você continue fazendo isso até que ele lhe diga para parar. É isso que os loops fazem para os computadores - eles repetem um conjunto de instruções quantas vezes forem necessárias.

Como o Loop funciona?

Aqui está como funciona em Python:



- **Início:** O loop for começa com a palavra-chave for, seguida de uma variável que assumirá cada valor em uma sequência.
- **Condição:** Após a variável, você especifica a palavra-chave in e uma sequência, como uma lista ou um intervalo, que o loop irá iterar.
- Se **Condição Verdadeira**:

1. O loop pega o primeiro valor da sequência e o atribui à variável.
2. O bloco de código indentado que segue o cabeçalho do loop é executado usando esse valor.
3. O loop então passa para o próximo valor na sequência e repete o processo até que todos os valores tenham sido utilizados.

- **Declaração:** Dentro do bloco indentado do loop, você escreve as declarações que deseja repetir para cada valor na sequência.
- **Repetir:** O loop continua a repetir o bloco de código para cada valor na sequência até que não haja mais valores restantes.
- Se **Condição Falsa**:

1. Uma vez que todos os valores na sequência tenham sido processados, o loop termina automaticamente.
2. O loop completa sua execução e o programa continua para a próxima declaração após o loop.

A Necessidade de Laços

Pense em quando você precisa contar de 1 a 10. Fazer isso manualmente é fácil, mas e se você tivesse que contar até **um milhão**? Digitar todos esses números um por um seria um pesadelo! É aqui que os laços se tornam úteis. Eles ajudam os computadores a repetir tarefas de forma rápida e precisa, sem se cansar.

Principais Tipos de Laços

Laços For

Os laços for são como uma lista de verificação de um super-herói. Um laço for na programação é uma estrutura de controle que permite a execução repetida de um conjunto de instruções para cada item em uma sequência, como elementos em uma lista ou números em um intervalo, possibilitando iteração eficiente e automação de tarefas.

Sintaxe do loop for

```
for val in sequence:
      # statement(s) to be executed in sequence as a part of the loop.
```

Aqui está um exemplo de loop For.

Imagine que você é um pintor e quer pintar um lindo arco-íris com sete cores. Em vez de pegar cada cor uma por uma e pintar o arco-íris, você poderia pedir a um assistente mágico de pintura para fazer isso por você. Isso é o que um loop for básico faz na programação.

Temos uma lista de cores.

```
colors = ["red", "orange", "yellow", "green", "blue", "indigo", "violet"]
```

Vamos imprimir o nome da cor em uma nova linha usando um loop for.

```
for color in colors:
    print(color)
```

Neste exemplo, o loop for seleciona cada cor da lista de cores e a imprime na tela. Você não precisa escrever o mesmo código para cada cor - o loop faz isso automaticamente!

Às vezes, você não quer pintar um arco-íris, mas *você quer contar o número de passos para alcançar seu objetivo.* Um loop for baseado em intervalo é como ter um contador de passos amigável que ajuda você a atingir sua meta. Aqui está como você pode usar um loop for para contar de 1 a 10:

```
for number in range(1, 11):
    print(number)
```

Aqui, o **range(1, 11)** gera uma sequência de 1 a 10, e o loop for percorre cada número nessa sequência, imprimindo-o. É como dar 10 passos, e você é guiado pelo loop!

Função range

A função range em Python gera uma sequência ordenada que pode ser usada em loops. Ela aceita um ou dois argumentos:

- Se fornecido um argumento (por exemplo, range(11)), ela gera uma sequência começando de 0 até (mas não incluindo) o número dado.

  ```
  for number in range(11):
      print(number)
  ```

- Se forem fornecidos dois argumentos (por exemplo, range(1, 11)), ele gera uma sequência começando do primeiro argumento até (mas não incluindo) o segundo argumento.

  ```
  for number in range(1, 11):
      print(number)
  ```

O Laço For Enumerado

Você já precisou acompanhar tanto o item quanto sua posição em uma lista? Um laço for enumerado vem em seu auxílio. É como ter um assistente pessoal que não apenas lhe entrega o item, mas também lhe diz onde encontrá-lo.

Considere este exemplo:

```
fruits = ["apple", "banana", "orange"]
for index, fruit in enumerate(fruits):
    print(f"At position {index}, I found a {fruit}")
```

Com este loop, você não apenas obtém a fruta, mas também sua posição na lista. É como se você tivesse um guia mágico apontando a localização de cada fruta!

Laços While

Laços while são como uma noite sem sono na casa de um amigo. Imagine que você e seus amigos continuam contando histórias de fantasmas até que alguém decida que é hora de dormir. Enquanto ninguém disser: “Vamos dormir”, você continua contando histórias.
Um laço while funciona de maneira semelhante - ele repete uma tarefa enquanto uma determinada condição for verdadeira. É como dizer: “Ei computador, continue fazendo isso até eu dizer para parar!”

Sintaxe básica do loop While.

```
while condition:
    # Code to be executed while the condition is true
    # Indentation is crucial to indicate the scope of the loop
```

Por exemplo, aqui está como você pode usar um loop while para contar de 1 a 10:

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

aqui está uma explicação do código acima.

1. Há uma variável chamada **count** inicializada com o valor 1.

2. O loop while é usado para executar repetidamente um bloco de código enquanto uma determinada condição for verdadeira. Neste caso, a condição é **count <= 10**, o que significa que o loop continuará enquanto count for menor ou igual a 10.

3. Dentro do loop:
    - A instrução **print(count)** exibe o valor atual da variável count.
    - A instrução **count += 1** incrementa o valor de count em 1. Este passo garante que o loop eventualmente terminará quando count se tornar maior que 10.
4. O loop continuará executando enquanto a condição count <= 10 for satisfeita.
5. O loop imprimirá os números de 1 a 10 em ordem consecutiva, uma vez que a instrução print está dentro do bloco do loop e é executada durante cada iteração.
6. Uma vez que count atinja 11, a condição count <= 10 será avaliada como Falsa, e o loop será encerrado.
7. A saída do código será os números de 1 a 10, cada um impresso em uma linha separada.

O Fluxo do Loop

Tanto os loops `for` quanto os `while` têm seus movimentos especiais, mas seguem um padrão:

- **Inicialização:** Você configura coisas como um ponto de partida ou condições.
- **Condição:** Você decide quando o loop deve continuar e quando deve parar.
- **Execução:** Você realiza a tarefa dentro do loop.
- **Atualização:** Você faz alterações no seu ponto de partida ou condições para avançar.
- **Repetir:** O loop volta ao passo 2 até que a condição não seja mais verdadeira.

Quando Usar Cada Um

Laços For: Use laços for quando você souber o número de iterações com antecedência e quiser processar cada elemento em uma sequência. Eles são mais adequados para iterar sobre coleções e sequências onde o comprimento é conhecido.

Laços While: Use laços while quando você precisar realizar uma tarefa repetidamente enquanto uma determinada condição for verdadeira. Laços while são particularmente úteis para situações onde o número de iterações é incerto ou onde você está aguardando que uma condição específica seja atendida.

Resumo

Nesta aventura na programação, exploramos loops em Python - ferramentas especiais que nos ajudam a fazer coisas repetidamente sem nos cansar. Conhecemos dois tipos de loops: **"loops for"** e **"loops while."**

Loops For eram como ajudantes que nos faziam repetir tarefas em ordem. Pintamos cores, contamos números e até conseguimos um ajudante para nos dizer onde estavam as coisas em uma lista. Os loops for tornaram nosso trabalho mais fácil e nosso código mais limpo.

Loops While eram como detetives que continuavam fazendo algo enquanto uma regra fosse verdadeira. Eles nos ajudaram a dar passos, adivinhar números e trabalhar até ficarmos cansados. Os loops while eram como assistentes inteligentes que não paravam até que dissessemos.

Autor(es)

Akansha Yadav

Changelog

Data	Versão	Alterado por	Descrição da Alteração
2023-21-08	1.0	Akansha Yadav	Criado um arquivo de leitura