O que é um laço de repetição?

Um laço de repetição é uma estrutura que permite executar um bloco de código várias vezes. Isso é útil quando você quer fazer algo repetidamente, como percorrer todos os itens de uma lista ou repetir uma ação várias vezes.

Como o `for` funciona?

Imagine que você tem uma lista de coisas e quer olhar cada uma delas, uma por uma. O `for` em Python ajuda você a fazer isso de maneira automática.

Exemplo simples:

Vamos usar um exemplo de uma lista de frutas:

```
frutas = ["maçã", "banana", "laranja"]
```

Agora, queremos imprimir cada fruta da lista. Podemos fazer isso manualmente:

```
print(frutas[0])  # maçã
print(frutas[1])  # banana
print(frutas[2])  # laranja
```

Mas, se a lista tiver 100 frutas? Fazer isso manualmente seria muito trabalhoso. É aqui que entra o `for` .

Usando o `for` :

```
for fruta in frutas:
    print(fruta)
```

Como funciona:

1. **`frutas`** : Esta é a nossa lista de frutas.
2. **`for fruta in frutas:`** : Aqui, estamos dizendo ao Python: "Para cada coisa (que chamamos de `fruta`) na lista `frutas` ".
3. **`print(fruta)`** : Dentro do laço, queremos imprimir cada `fruta` .

O que acontece:

- O Python pega o primeiro item da lista `frutas` (que é “maçã”) e coloca na variável `fruta`.
- Ele executa o bloco de código dentro do `for` (neste caso, `print(fruta)`), então imprime “maçã”.
- Depois, ele passa para o próximo item da lista (que é “banana”), coloca na variável `fruta`, e repete o bloco de código, imprimindo “banana”.
- Ele continua fazendo isso até que todos os itens da lista tenham sido processados.

Visualizando:

Pense no laço `for` como uma linha de montagem onde cada item da lista passa pelo mesmo processo um por um:

1. Primeiro item (maçã) -> Processa (imprime)
2. Segundo item (banana) -> Processa (imprime)
3. Terceiro item (laranja) -> Processa (imprime)

E assim por diante, até terminar a lista.

Resumindo:

O laço `for` é uma maneira de repetir ações automaticamente para cada item em uma lista (ou em qualquer sequência de itens). Você não precisa saber quantos itens há na lista ou fazer a repetição manualmente, o `for` faz isso para você!

O que é o `range`?

O `range` é uma função em Python que gera uma sequência de números. Você pode pensar nele como uma “lista” de números que o Python cria automaticamente para você.

Como funciona o `range`?

Exemplo simples:

Vamos ver o que acontece quando usamos o `range`:

```
numeros = range(5)
print(list(numeros))
```

Resultado:

```
[0, 1, 2, 3, 4]
```

O que isso faz:

- **`range(5)`**: Cria uma sequência de números começando de 0 até 4 (5 números no total, mas não inclui o 5).
- **`list(numeros)`**: Converte o `range` para uma lista para que possamos ver os números.

Diferentes formas de usar o `range`:

1. **`range(n)`**: Gera números de 0 até n-1.

   ```
   range(5)  # [0, 1, 2, 3, 4]
   ```

2. **`range(start, stop)`**: Gera números começando do `start` até `stop-1`.

   ```
   range(2, 6)  # [2, 3, 4, 5]
   ```

3. **`range(start, stop, step)`**: Gera números começando do `start`, até `stop-1`, incrementando de `step` em `step`.

   ```
   range(1, 10, 2)  # [1, 3, 5, 7, 9]
   ```

Usando `range` com `for`:

Agora vamos usar o `range` com o `for` para repetir ações várias vezes.

Exemplo simples:

Vamos imprimir os números de 0 a 4:

```
for numero in range(5):
    print(numero)
```

O que acontece:

- **`range(5)`** : Cria uma sequência de números de 0 a 4.
- **`for numero in range(5):`** : Aqui, estamos dizendo ao Python: “Para cada número na sequência de 0 a 4”.
- **`print(numero)`** : Imprime o número atual.

Passo a passo:

1. O Python cria a sequência de números [0, 1, 2, 3, 4] usando `range(5)` .
2. Para cada número na sequência:
    - Primeiro, `numero` é 0 -> imprime 0.
    - Depois, `numero` é 1 -> imprime 1.
    - E assim por diante, até `numero` ser 4 -> imprime 4.

Outro exemplo com intervalo diferente:

Vamos imprimir os números de 2 a 5:

```
for numero in range(2, 6):
    print(numero)
```

O que acontece:

- **`range(2, 6)`** : Cria uma sequência de números de 2 a 5 (não inclui 6).
- **`for numero in range(2, 6):`** : Para cada número na sequência de 2 a 5.
- **`print(numero)`** : Imprime o número atual.

Passo a passo:

1. O Python cria a sequência de números [2, 3, 4, 5] usando `range(2, 6)` .
2. Para cada número na sequência:
    - Primeiro, `numero` é 2 -> imprime 2.
    - Depois, `numero` é 3 -> imprime 3.

- E assim por diante, até `numero` ser 5 -> imprime 5.

Resumindo:

- **`range`** : Cria uma sequência de números.
- **`for` com `range`** : Permite que você repita uma ação para cada número na sequência criada pelo `range` .

O `range` é muito útil quando você sabe quantas vezes quer repetir algo ou quando precisa de uma sequência de números específica. Usando `for` com `range` , você pode facilmente controlar quantas vezes um bloco de código é executado.

O que é a variável de descarte (_)?

Em Python, o underscore (_) é frequentemente usado como uma variável “especial” que significa “não me importa o valor desta variável”. É uma maneira de indicar que você não vai usar essa variável para nada.

Por que usar a variável de descarte?

Às vezes, você precisa de um valor ou precisa percorrer um laço de repetição, mas não se importa com o valor real. Você só quer que algo aconteça um certo número de vezes ou está interessado em outro aspecto do seu código.

Exemplo sem o underscore:

Vamos dizer que você quer imprimir “Olá!” 5 vezes:

```
for i in range(5):
    print("Olá!")
```

Neste código:

- Estamos usando `i` como a variável do laço.
- Mas não estamos realmente usando `i` dentro do laço. Só queremos repetir algo 5 vezes.

Usando o underscore (_):

Se você não precisa do valor de `i`, pode usar o underscore para deixar claro que não se importa com essa variável:

```
for _ in range(5):
    print("Olá!")
```

O que acontece aqui:

- **`range(5)`** : Cria uma sequência de 0 a 4.
- **`for _ in range(5):`** : Estamos dizendo ao Python: “Para cada número na sequência de 0 a 4, mas eu não me importo com o valor, apenas quero repetir isso 5 vezes”.
- **`print("Olá!")`** : Imprime “Olá!” cinco vezes.

Visualizando:

Pense no underscore como um “nome de variável” que diz ao Python (e a qualquer pessoa lendo seu código) que o valor não é importante:

1. O Python cria a sequência de números [0, 1, 2, 3, 4].
2. Para cada número na sequência:
    - Primeiro, `_` é 0 -> imprime “Olá!”.
    - Depois, `_` é 1 -> imprime “Olá!”.
    - E assim por diante, até `_` ser 4 -> imprime “Olá!”.

Outro uso do underscore:

O underscore pode ser usado em várias situações onde você quer descartar valores. Aqui estão alguns exemplos:

Descartando valores em desempacotamento:

```
valores = (1, 2, 3)
a, _, c = valores
print(a)  # 1
print(c)  # 3
```

Neste exemplo:

- Temos uma tupla `valores` com três números.
- Queremos apenas `a` e `c`, então usamos `_` para ignorar o segundo valor.

Resumindo:

- `_` : Usado como uma variável de descarte em Python.
- **Para laços `for`** : Indica que não nos importamos com o valor da variável do laço.
- **Para desempacotamento**: Ignora valores que não são necessários.

Usar o underscore desta maneira torna seu código mais limpo e deixa claro que certos valores não são importantes para a lógica do seu programa.

Lista de Exercícios com `for`

1 - Quantidade de vogais

Solicite ao usuário que informe uma frase qualquer. A partir dessa frase que foi digitada pelo usuário, verifique quantas letras são vogais e informe o texto para o seu usuário da seguinte maneira: `A frase " possui x vogais!"

> Obs: Você não deve considerar vogais as que possuírem acentuação gráfica como é, ã, ô, ê, etc.

2 - Filtrar Palavras Longas

Solicite ao usuário que informe uma frase qualquer. A partir dessa frase que foi digitada pelo usuário, verifique quantas palavras têm mais que 4 letras e informe para o usuário a quantidade.

3 - Quadrados e Cubos

Solicite para o usuário 5 números, informe o quadrado do número se ele for par e o cubo do número se ele for ímpar.

4 - Range e Múltiplos

Utilizando a função `range` e `for` , gere um intervalo de 0 até 150. Apenas imprima os números que forem ao mesmo tempo múltiplos de 3 e 5.

5 - Fatorial

Solicite para o usuário um número e descubra qual o fatorial dele. O símbolo que representa o fatorial é a exclamação (`!`), um número que possuir a exclamação será um número fatorial `Ex: 5!`

Para resolver o fatorial, você deve realizar a multiplicação de todos os números anteriores a ele até chegar em 1. No exemplo fornecido de `5!` o resultado será `120`

Resposta: `5!= 5x4x3x2x1 = 120`

6 - Sequência de Fibonacci

Teoria

A sequência de Fibonacci é uma série de números onde cada número é a soma dos dois números anteriores. A sequência começa com 0 e 1, e continua indefinidamente. Os primeiros números na sequência de Fibonacci são:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
```

Aqui está como a sequência é formada:

- O primeiro número é 0.
- O segundo número é 1.
- O terceiro número é a soma dos dois primeiros: 0 + 1 = 1.
- O quarto número é a soma do segundo e do terceiro: 1 + 1 = 2.
- O quinto número é a soma do terceiro e do quarto: 1 + 2 = 3.
- E assim por diante.

Sabendo disso... Escreva um programa em Python que gere a sequência de Fibonacci até o n-ésimo termo. O programa deve pedir ao usuário um número inteiro n e exibir os primeiros n termos da sequência de Fibonacci.