FOR

Um laço `for` é utilizado para a iteração através de uma sequência (isto é, quer uma lista, uma tupla, um dicionário, um conjunto ou uma cadeia).

Isso é menos parecido com a palavra-chave for em outras linguagens de programação e funciona mais como um método iterador, como encontrado em outras linguagens de programação orientadas a objetos.

Com o loop for , podemos executar um conjunto de instruções, uma vez para cada item em uma lista, tupla, conjunto etc.

Exemplo:

Imprima cada fruta em uma lista de frutas:

```
frutas = ["maçã", "banana", "cereja"]
for x in frutas:
 print(x)

#Resultado:
maçã
banana
cereja
```

O loop for não requer uma variável de indexação para definir de antemão.

Looping através de uma string

Até mesmo strings são objetos iteráveis, eles contêm uma sequência de caracteres.

Exemplo:

```
for x in "banana":
 print(x)
```

```
#Resultado:
b
a
n
a
n
a
```

A declaração break

Com a instrução `break` , podemos interromper o loop antes que ele percorra todos os itens.

Exemplo:

```
frutas = ["maçã","banana","cereja"]
for x in frutas:
 print(x)
 if x == "banana":
  break

#Resultado:
maçã
banana
```

Válido mostrar um detalhe interessante, quando a impressão é realizado pós-avaliação do if, o resultado é diferente.

```
frutas = ["maçã", "banana", "cereja"]
for x in frutas:
 if x == "banana":
  break
 print(x)

#Resultado:
maçã
```

A declaração continue

Com a instrução `continue` , podemos interromper a iteração atual do loop e continuar com a próxima.

Exemplo:

```
frutas = ["maçã", "banana", "cereja"]
for x in frutas:
 if x == "banana":
  continue
 print(x)

#Resultado:
maçã
cereja
```

A função range ()

Para percorrer um conjunto de código um determinado número de vezes, podemos usar a função `range()`.

A função range() retorna uma sequência de números, começando em 0 por padrão, e incrementos em 1 (por padrão), e termina em um número especificado.

Exemplo:

```
for x in range(6):
 print(x)

#Resultado:
0
1
2
3
4
5
```

O padrão da função range() tem como valor inicial 0, no entanto, é possível especificar o valor inicial adicionando um parâmetro: range (2,6), que significa valores de 2 a 6 (mas não incluindo 6).

Exemplo:

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

#Resultado:
2
3
```

```
4
5
```

O padrão da função range() é incrementar a sequência em 1, no entanto, é possível especificar o valor do incremento adicionando um terceiro parâmetro: range (2, 6, 2), significa valores de 2 a 6(mas não incluindo 6), incrementando 2.

Exemplo:

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

#Resultado:
2
4
```

Loop For com else

A palavra-chave `else` em um loop `for` especifica um bloco de código a ser executado quando o loop for concluído.

Exemplo:

```
for x in range(6):
 print(x)
else:
 print("O laço terminou!")

#Resultado:
1
2
3
4
5
O laço terminou!
```

Nota: O `else` NÃO será executado se o loop for interrompido por uma instrução `break`.

Exemplo:

```
for x in range(6):
 if x == 3: break
 print(x)
else:
 print("O laço terminou!")

#Resultado:
0
1
2
```

Loops aninhados

Um loop aninhado é um loop dentro de um loop. O "loop interno" será executado uma vez para cada iteração do "loop externo".

Exemplo:

```
adjetivos = ["vermelha", "grande", "gostosa"]
frutas = ["maçã", "banana", "cereja"]
for x in frutas:
 for y in adjetivos:
  print(x, y)

#Resultado:
maçã vermelha
maçã grande
maçã gostosa
banana vermelha
banana grande
banana gostosa
cereja vermelha
cereja grande
cereja gostosa
```

A declaração pass

Assim como nas outras estruturas, os laços `for` não podem estar vazios, mas se, por algum motivo, você tiver um `for` loop sem conteúdo, insira a `pass` instrução para evitar um erro.

Exemplo:

```
for x in [0, 1, 2]:
 pass
```