Python - para loops

Python para Loop

O loop **for** em Python tem a capacidade de iterar sobre os itens de qualquer sequência, como uma lista, tupla ou string.

Sintaxe

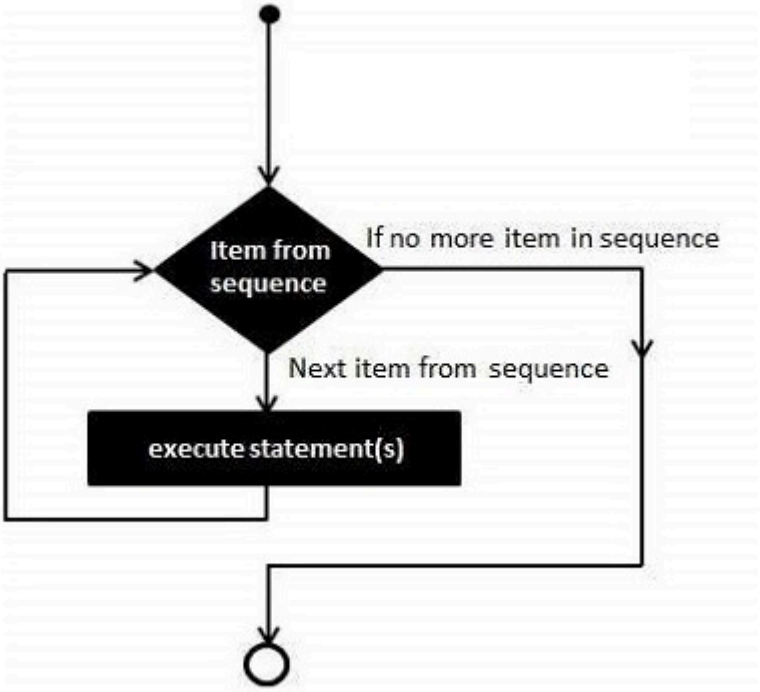
```
for iterating_var in sequence:
   statements(s)
```

Se uma sequência contiver uma lista de expressões, ela será avaliada primeiro. Então, o primeiro item (no índice 0) da sequência é atribuído à variável iterativa iterating_var.

A seguir, o bloco de instruções é executado. Cada item da lista é atribuído a iterating_var e o bloco de instruções é executado até que toda a sequência se esgote.

Fluxograma

O diagrama de fluxo a seguir ilustra o funcionamento do loop **for** -



Como o loop é executado para cada elemento membro em uma sequência, não há necessidade de verificação explícita da expressão booleana que controla o loop (como no loop **while**).

Os objetos de sequência, como lista, tupla ou string, são chamados **de iteráveis** , pois o loop for itera pela coleção. Qualquer objeto iterador pode ser iterado pelo loop **for** .

Os objetos de visualização retornados pelos métodos items(), keys() evalues() do dicionário também são iteráveis, portanto, podemos executar um loop **for** com esses métodos.

A função range() integrada do Python retorna um objeto iterador que transmite uma sequência de números. Também podemos executar um loop for com range.

para Loop com Strings

Uma string é uma sequência de letras Unicode , cada uma com um índice posicional. O exemplo a seguir compara cada caractere e exibe se não é uma vogal ('a', 'e', 'I', 'o' ou 'u')

Exemplo

```
zen = '''
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
'''
for char in zen:
   if char not in 'aeiou':
      print (char, end='')
```

Saída

Ao ser executado, este código produzirá a seguinte saída -

```
Btfl s bttr thn gly.
Explct s bttr thn mplct.
Smpl s bttr thn cmplx.
Cmplx s bttr thn cmplctd.
```

for Loop com Tuplas

O objeto tupla do Python também é uma sequência indexada e, portanto, podemos percorrer seus itens com um loop **for** .

Exemplo

No exemplo a seguir, o loop **for** percorre uma tupla contendo inteiros e retorna o total de todos os números.

```
numbers = (34,54,67,21,78,97,45,44,80,19)
total = 0
for num in numbers:
   total+=num
print ("Total =", total)
```

Saída

Ao ser executado, este código produzirá a seguinte saída -

```
Total = 539
```

for Loop com Listas

O objeto de lista do Python também é uma sequência indexada e, portanto, podemos percorrer seus itens com um loop **for** .

Exemplo

No exemplo a seguir, o loop for percorre uma lista contendo inteiros e imprime apenas aqueles que são divisíveis por 2.

```
numbers = [34,54,67,21,78,97,45,44,80,19]
total = 0
for num in numbers:
   if num%2 == 0:
      print (num)
```

Saída

Ao ser executado, este código produzirá a seguinte saída -

```
34
54
78
44
80
```

for Loop com objetos Range

A função range() integrada do Python retorna um objeto range. O objeto range do Python é um iterador que gera um número inteiro a cada iteração. O objeto contém números inteiros do início ao fim, separados pelo parâmetro step.

Sintaxe

A função range() tem a seguinte sintaxe -

```
range(start, stop, step)
```

Parâmetros

- **Start** - Valor inicial do intervalo. Opcional. O padrão é 0
- **Stop** - O intervalo vai até stop-1
- **Step** - Inteiros no intervalo são incrementados pelo valor do passo. Opção, o padrão é 1.

Valor de retorno

A função range() retorna um objeto range. Ele pode ser analisado em uma sequência de lista.

Exemplo

```
numbers = range(5)
'''
start is 0 by default,
step is 1 by default,
range generated from 0 to 4
'''
print (list(numbers))
# step is 1 by default, range generated from 10 to 19
numbers = range(10,20)
print (list(numbers))
# range generated from 1 to 10 increment by step of 2
numbers = range(1, 10, 2)
print (list(numbers))
```

Saída

Ao ser executado, este código produzirá a seguinte saída -

```
[0, 1, 2, 3, 4]
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[1, 3, 5, 7, 9]
```

Exemplo

Depois de obter o intervalo, podemos usar o loop **for** com ele.

```
for num in range(5):
 print (num, end=' ')
print()
for num in range(10,20):
 print (num, end=' ')
print()
for num in range(1, 10, 2):
 print (num, end=' ')
```

Saída

Ao ser executado, este código produzirá a seguinte saída -

```
0 1 2 3 4
10 11 12 13 14 15 16 17 18 19
1 3 5 7 9
```


for Loop com índices de sequência

Para iterar sobre uma sequência, podemos obter a lista de índices usando a função range()

```
Indices = range(len(sequence))
```

Podemos então formar um loop **for** da seguinte maneira:

```
numbers = [34,54,67,21,78]
indices = range(len(numbers))
for index in indices:
   print ("index:",index, "number:",numbers[index])
```

Ao ser executado, este código produzirá a seguinte **saída** -

```
index: 0 number: 34
index: 1 number: 54
index: 2 number: 67
index: 3 number: 21
index: 4 number: 78
```


for Loop com dicionários

Ao contrário de uma lista, tupla ou string, o tipo de dados de dicionário em Python não é uma sequência, pois os itens não possuem um índice posicional. No entanto, percorrer um dicionário ainda é possível com diferentes técnicas.

A execução de um loop for simples sobre o objeto de dicionário percorre as chaves usadas nele.

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers:
   print (x)
```

Ao ser executado, este código produzirá a seguinte **saída** -

```
10
20
30
40
```

Assim que conseguirmos obter a chave, seu valor associado pode ser facilmente acessado usando o operador de colchetes ou com o método **get()** . Dê uma olhada no exemplo a seguir -

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers:
   print (x,":",numbers[x])
```

Ele produzirá a seguinte **saída** -

```
10 : Ten
20 : Twenty
30 : Thirty
40 : Forty
```

Os métodos items(), chaves() e valores() da classe dict retornam os objetos de visualização dict_items, dict_keys e dict_values respectivamente. Esses objetos são iteradores e, portanto, podemos executar um loop for sobre eles.

O objeto dict_items é uma lista de tuplas de valores-chave sobre as quais um loop for pode ser executado da seguinte forma -

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers.items():
   print (x)
```

Ele produzirá a seguinte **saída** -

```
(10, 'Ten')
(20, 'Twenty')
(30, 'Thirty')
(40, 'Forty')
```

Aqui, "x" é o elemento tupla do iterador dict_items. Podemos descompactar ainda mais esta tupla em duas variáveis diferentes. Verifique o seguinte código -

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x,y in numbers.items():
   print (x,":", y)
```

Ele produzirá a seguinte **saída** -

```
10 : Ten
20 : Twenty
30 : Thirty
40 : Forty
```

Da mesma forma, a coleção de chaves no objeto dict_keys pode ser iterada. Dê uma olhada no exemplo a seguir -

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers.keys():
   print (x, ":", numbers[x])
```

Ele produzirá a mesma **saída** -

```
10 : Ten
20 : Twenty
30 : Thirty
40 : Forty
```