

# ASIMOV

---

## For

Um loop **for** atua como um iterador em Python, ele passa por itens que estão em uma *sequência* ou qualquer outro item iterável. Os objetos que aprendemos até agora que podemos iterar incluem strings, listas, tuplas e até iteráveis embutidos em dicionários, como chaves ou valores.

Já vimos **for** um pouco nas palestras passadas, mas agora permitimos formalizar a nossa compreensão.

Aqui está o formato geral para um **for** loop em Python:

```
for item in objeto:    fazer algo
```

O nome da variável usado para o item fica a seu critério, você pode escolher o que quiser. Então use seu melhor julgamento para escolher um nome que faça sentido e que você poderá entender ao revisar seu código. Este nome do item pode então ser referenciado dentro de seu loop, por exemplo, se você quisesse usar instruções `if` para executar verificações.

Vamos seguir em frente e trabalhar com vários exemplos de **for** loops usando uma variedade de tipos de objetos de dados. Vamos começar com um exemplo simples e adicionar mais complexidade mais além.

## Exemplo 1

Iterando através de uma lista.

```
In [1]: # Aprenderemos a automatizar esse tipo de lista na próxima palestra  
l = [1,2,3,4,5,6,7,8,9,10]
```

```
In [2]: for num in l:  
        print(num)
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Ótimo! Espero que isso tenha feito sentido. Agora, vamos adicionar uma instrução `if` para

verificar se há números pares. Vamos primeiro apresentar um novo conceito aqui - o módulo.

## Modulo

O módulo nos permite obter o restante em uma divisão e usa o símbolo %. Por exemplo:

```
In [5]: 17 % 5
```

```
Out[5]: 2
```

Isso faz sentido, pois 17 dividido por 5 é 3 e sobra 2. Vamos ver alguns exemplos mais rápidos:

```
In [6]: # 3, sobra 1  
10 % 3
```

```
Out[6]: 1
```

```
In [9]: # 2, sobra 4  
18 % 7
```

```
Out[9]: 4
```

```
In [10]: # 2, sem sobras  
4 % 2
```

```
Out[10]: 0
```

Observe que se um número é totalmente divisível sem restante, o resultado do módulo é 0. Podemos usar isso para testar números pares, pois se um número módulo 2 for igual a 0, isso significa que é um número par!

Volte para o **for**!

## Exemplo 2

Vamos imprimir apenas os números pares dessa lista!

```
In [4]: for num in l:  
        if num % 2 == 0:  
            print(num)
```

```
2  
4  
6  
8  
10
```

Nós também poderíamos usar um else lá, também:

```
In [5]: for num in l:  
        if num % 2 == 0:  
            print(num)
```

```
else:  
    print('Número ímpar')
```

```
Número ímpar  
2  
Número ímpar  
4  
Número ímpar  
6  
Número ímpar  
8  
Número ímpar  
10
```

## Exemplo 3

Outra idéia comum durante um **for** é manter algum tipo de contagem durante os vários loops. Por exemplo, vamos criar um loop for que resume a lista:

```
In [6]: list_sum = 0  
  
for num in l:  
    list_sum = list_sum + num  
  
print(list_sum)
```

```
55
```

Ótimo! Leia sobre a célula acima e certifique-se de entender completamente o que está acontecendo. Também poderíamos ter implementado um += para a adição. Por exemplo:

```
In [7]: list_sum = 0  
  
for num in l:  
    list_sum += num  
  
print(list_sum)
```

```
55
```

## Exemplo 4

Nós usamos para loops com listas, e as strings? Lembre-se de que as strings são uma sequência, então, quando iteramos através delas, estaremos acessando cada item nessa sequência de caracteres.

```
In [8]: for letter in 'This is a string.':  
        print(letter)
```

```
T  
h  
i  
s  
  
i  
s  
  
a  
  
s
```

```
t  
r  
i  
n  
g  
.
```

## Example 5

E com tuplas?

In [16]:

```
tup = (1,2,3,4,5)  
  
for t in tup:  
    print t
```

```
1  
2  
3  
4  
5
```

## Exemplo 6

As Tuplas têm uma qualidade especial quando se trata de **for**. Se você está iterando através de uma sequência que contém tuplas, o item pode realmente ser a própria tupla, este é um exemplo de *desembalagem de tuplas*. Durante o **for**, estaremos desembalando a tupla dentro de uma sequência e podemos acessar os itens individuais dentro dessa tupla!

In [9]:

```
l = [(2,4),(6,8),(10,12)]
```

In [11]:

```
for tup in l:  
    print(tup)
```

```
(2, 4)  
(6, 8)  
(10, 12)
```

In [13]:

```
# Agora desembalando  
for (t1,t2) in l:  
    print(t1)
```

```
2  
6  
10
```

Legal! Com as tuplas em uma sequência, podemos acessar os itens dentro deles por meio de desembalagem! A razão pela qual isso é importante é porque muitos objetos entregarão seus iterables através de tuplas. Vamos começar a explorar a iteração através de Dictionaries para explorar isso ainda mais!

## Exemplo 7

In [14]:

```
d = {'k1':1,'k2':2,'k3':3}
```

```
In [15]: for item in d:
          print(item)
```

```
k3
k2
k1
```

Observe como isso produz apenas chaves. Então, como podemos obter os valores? Ou as chaves e os valores?

## items()

Você deve usar `.items()` para iterar através das chaves e valores de um dicionário. Por exemplo:

```
In [11]: # For Python 3
          for k,v in d.items():
              print(k)
              print(v)
```

```
k3
3
k2
2
k1
1
```

## Conclusão

Aprendemos a usar para loops para iterar através de tuplas, listas, strings e dicionários. Será uma ferramenta importante para nós, portanto, certifique-se de conhecê-lo bem e compreende os exemplos acima.

[Mais recursos](#)