# FUNDAMENTOS DE PROGRAMAÇÃO

## SUMÁRIO

## LISTAS

Uma lista é um conjunto ordenado de valores, onde cada valor é identificado por um índice. Os valores que compõem uma lista são chamados elementos. Listas são similares a strings, que são conjuntos ordenados de caracteres, com a diferença que os elementos de uma lista podem possuir qualquer tipo. Listas e strings XXX e outras coisas que se comportam como conjuntos ordenados XXX são chamadas sequências.

Valores da lista

Existem várias maneiras de criar uma nova lista; a mais simples é envolver os elementos em colchetes [ e ]:

```
>>> [10, 20, 30, 40]
```

```
>>> ['spam', 'bungee', 'swallow']
```

No primeiro exemplo a uma lista de quatro inteiros. O segundo é uma lista de três strings. Os elementos de uma lista não necessitam ser do mesmo tipo.

## LISTA DENTRO DE LISTA?

A lista a seguir contém uma string, um valor float, um valor inteiro, e uma outra lista:

```
>>> ['alo', 2.0, 5, [10,20]]
```

Uma lista dentro de outra lista é dita estar aninhada.

Listas que contém inteiros consecutivos são comuns, então Python fornece uma maneira simples de criá-los através do método range:

```
>>> list(range(1,5))
```

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

A função range pega dois argumentos e devolve uma lista que contém todos os inteiros do primeiro até o segundo, incluindo o primeiro, mas não incluindo o segundo!

Existem outras formas de range. Com um argumento simples, ela cria uma lista que inicia em 0:

```
>>> list(range(10))
```

```
[0,1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Se existe um terceiro argumento, ele especifica o espaço entre os valores sucessivos, que é chamado de tamanho do passo. Este exemplo conta de 1 até 10 em passos de 2:

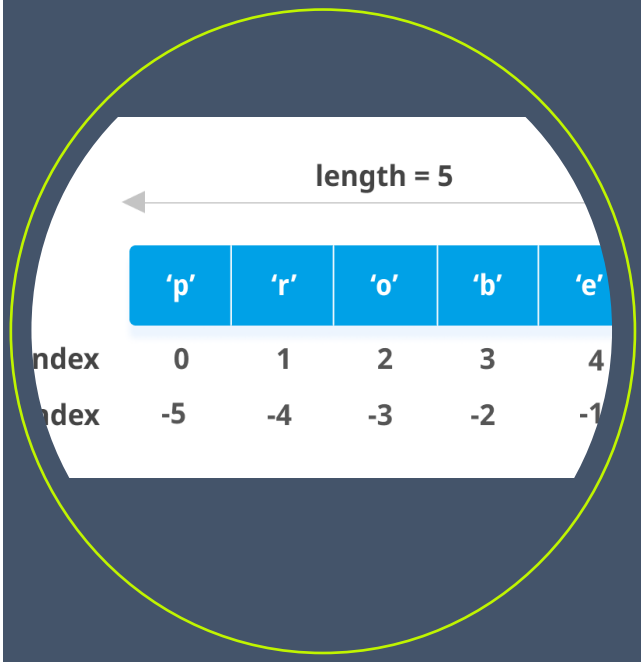
```
>>> list(range(1, 10, 2))

[1, 3, 5, 7, 9]
```

Finalmente, existe uma lista especial que não contém elementos. Ela é chamada lista vazia, e sua notação é [].

Com todas estas formas de criar listas, seria decepcionante se não pudéssemos atribuir valores de listas a variáveis ou passar listas como parâmetros a funções. Felizmente, podemos.

```
>>> vocabulario =
['melhorar', 'castigar', 'defenestrar']

>>> numeros = [17, 123]

>>> vazio = []

>>> print (vocabulario, numeros, vazio)

['melhorar', 'castigar', 'defenestrar']
[17, 123] []
```

Um terceiro argumento especifica o espaço entre os valores de uma lista numérica.




## ACESSANDO ELEMENTOS

A sintaxe para acessar os elementos de uma lista é a mesma que a sintaxe para acessar os caracteres de uma string XXX o operador colchete ([]). A expressão dentro dos colchetes especifica o índice. Lembre-se que os índices iniciam em 0:

```
>>> print (numeros[0])
```

```
>>> numeros[1]= 5
```

O operador colchete pode aparecer em qualquer lugar em uma expressão. Quando ele aparece no lado esquerdo de uma atribuição, ele modifica um dos elementos em uma lista, de forma que o um-ésimo elemento de numeros, que era 123, é agora 5.

Qualquer expressão inteira pode ser utilizada como um índice:

```
>>> numeros[3-2]
5
>>> numeros[1.0]
TypeError: sequence index must be integer
```

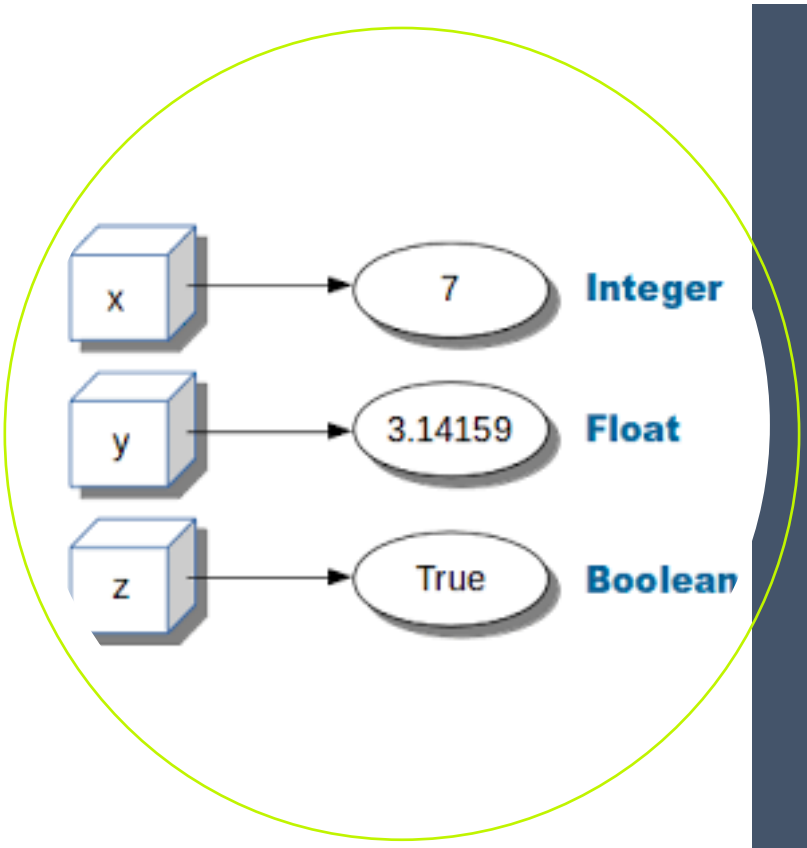
Se você tentar ler ou escrever um elemento que não existe, você recebe um erro de tempo de execução (runtime error):

```
>>> numeros[2]=5
IndexError: list assignment index out of range
```

Se um índice possui um valor negativo, ele conta ao contrário a partir do final da lista:

```
>>> numeros[-1]
5
>>> numeros[-2]
17
>>> numeros[-3]
IndexError: list index out of range
```

numeros[-1] é o último elemento da lista, numeros[-2] é o penúltimo e numeros[-3] não existe.




## TAMANHO DE LISTAS

A função len devolve o comprimento de uma lista. É uma boa ideia utilizar este valor como o limite superior de um laço ao invés de uma constante. Desta forma, se o tamanho da lista mudar, você não precisará ir através de todo o programa modificando todos os laços; eles funcionarão corretamente para qualquer tamanho de lista:

```
>>> cavaleiros = ['guerra', 'fome', 'peste', 'morte']

print(len(cavaleiros))

4
```

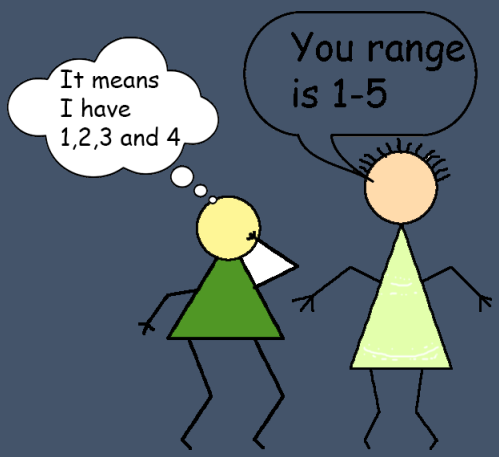
## MEMBROS DE UMA LISTA

in é um operador lógico que testa se um elemento é membro de uma sequência.

```
>>> cavaleiros = ['guerra', 'fome', 'peste', 'morte']

>>> 'peste' in cavaleiros

True
```

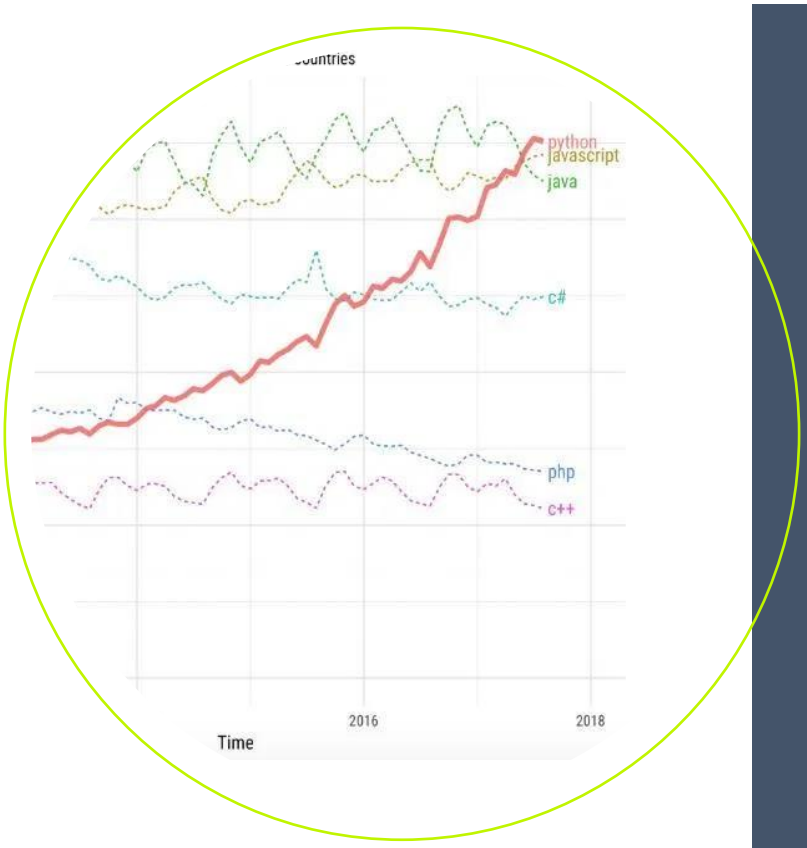
## ADICIONANDO ELEMENTOS

Para adicionar elementos no final da lista, utilize a função `.append( )`, que tem por finalidade adicionar um dado no final da lista.




Abaixo um exemplo de aplicação deste método:

```
lista = [1,2,3,4]

lista.append(5)

print( lista)  # [1,2,3,4,5]
```




## FOR

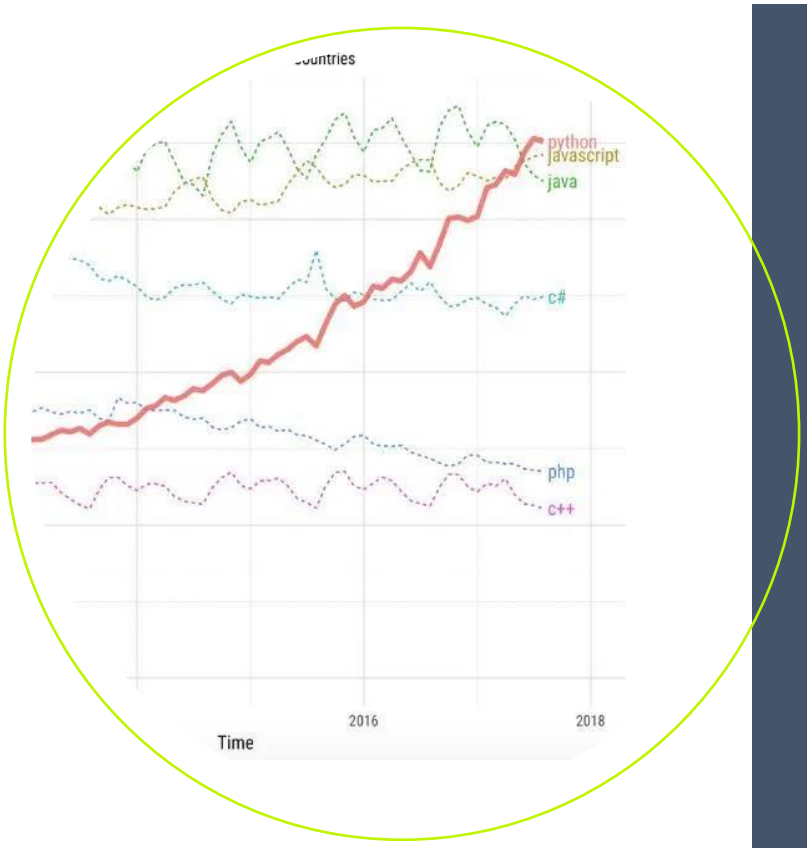
Várias computações envolvem o processamento de uma string, um caractere de cada vez. Muitas vezes elas começam com o primeiro, selecionam um de cada vez, fazem alguma coisa com ele, e continuam até o fim. Este padrão de processamento é chamado de uma travessia (traversal, com a ideia de "percorrimento").

Este loop, por exemplo, pode percorrer a string e exibir cada letra em sua própria linha. A condição do loop é indice < len(palavra), assim, quando índice é igual ao comprimento da string, a condição se torna falsa, e o corpo do loop não é executado. O último caractere acessado é aquele com o índice len(palavra)-1, que vem a ser o último caractere da string.

Usar um índice para percorrer um conjunto de valores é tão comum que Python oferece uma sintaxe alternativa simplificada - o loop for:

```
for char in fruta:

    print (char)
```

Sempre que soubermos quantas vezes queremos percorrer o loop, utilizamos o FOR.




## FOR COM ÍNDICE

Se for necessário um índice numérico, utilize a função interna enumerate.

```
for key, value in enumerate(["p", "y", "t", "h", "o", "n"]):

    print (key, value)
```

Podemos utilizar o laço for com outras funções como por exemplo o range

```
for i in range(5):

    print(i)
```

A cada vez através do loop, o próximo caractere da string é atribuído à variável char. O loop continua até que não reste mais caracteres.

O exemplo seguinte mostra como usar concatenação e um loop for para gerar uma série abecedário.

"Abecedário" se refere a uma série ou lista na qual os elementos aparecem em ordem alfabética. Por exemplo, no livro de Robert McCloskey's Make Way for Ducklings, os nomes dos "ducklings" são Jack, Kack, Lack, Mack, Nack, Ouack, Pack e Quack. O loop seguinte, produz como saída aqueles nomes, em ordem:

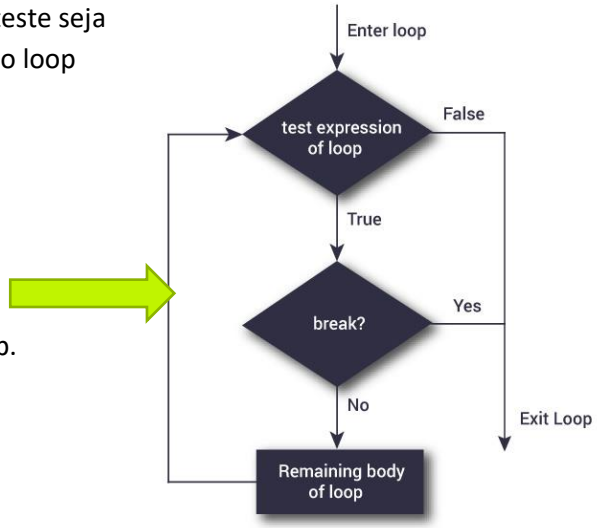
```
prefixos = "JKLMNOPQ"

sufixo = "ack"

for letra in prefixos:

    print (letra + sufixo)
```

# CONTINUE E BREAK

Em Python, as instruções break e continue podem alterar o fluxo de um loop normal.

Os loops iteram sobre um bloco de código até que a expressão de teste seja falsa, mas às vezes desejamos terminar a iteração atual ou mesmo o loop inteiro sem verificar a expressão de teste.

As instruções break e continue são usadas nesses casos.

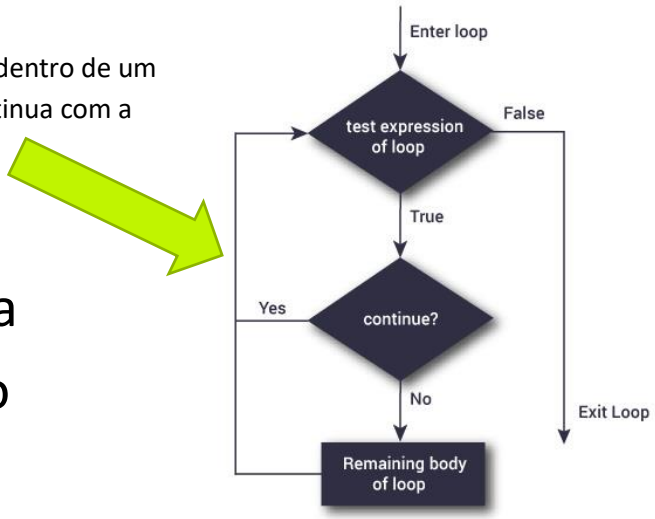



**Declaração de quebra em Python**

A instrução break finaliza o loop que a contém. O controle do programa flui para a instrução imediatamente após o corpo do loop. Se a instrução break estiver dentro de um loop aninhado (loop dentro de outro loop), o  break encerrará o loop mais interno.

**Declaração de continue em Python**

A instrução continue é usada para ignorar o restante do código dentro de um loop apenas para a iteração atual. O loop não termina, mas continua com a próxima iteração.




- Para iniciar imediatamente a próxima volta do loop, use o comando continue.
- Para encerrar imediatamente o loop, use o comando break