

Introdução a Python

IPL 2021



Strings

- Representam sequência de caracteres
 - sequência → *tipo composto*
- Marcadas por aspas duplas (") ou simples (')

Operações em strings

 concatena strings

 concatena cópias da string

Extraindo elementos (*indexando*)

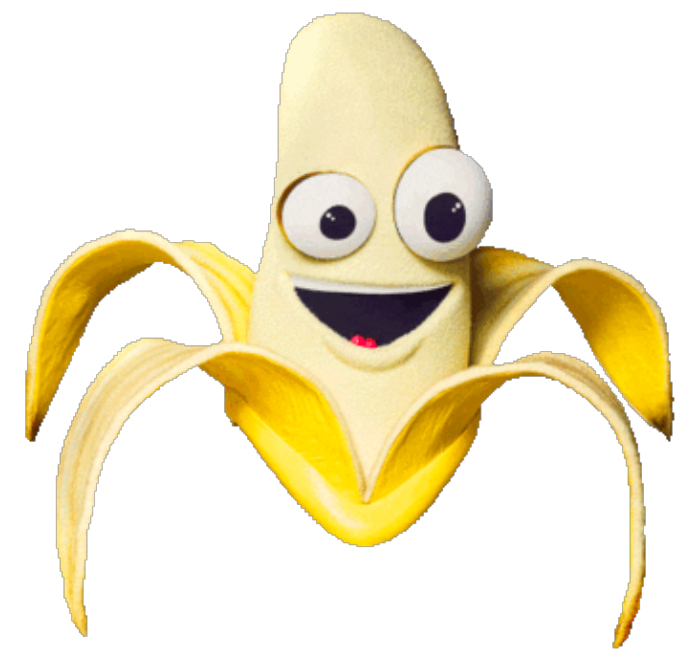
 retorna o caractere em um dado *índice*

Python usa **0-indexing**:
contagem começa em 0, não 1

2

```
1 a = "esta é uma string"
2 b = 'esta também é uma string'
3
4 print(a) # esta é uma string
5 print("a") # a
6 print(a + "!!!") # esta é uma string!!!
7 print("banana" * 2) # bananabanana
8 print(a[2]) # t
9
```

0	1	2	3	4	5
b	a	n	a	n	a
-6	-5	-4	-3	-2	-1



Iteração: *for* loops

- Permitem percorrer uma sequência
 - Um elemento por vez
 - **letter** assume “a”, “e”, “i”, “o”, “u”
 - **letter** mantém o último valor, “u”
- O *corpo* é repetido para cada elemento

range

retorna uma sequência de números, útil para repetir o corpo de um loop *for* um número específico de vezes

range(n)

$0, 1, 2, 3, \dots, n-2, n-1$

começa em 0, para antes do número n

range(s, n)

$s, s+1, s+2, \dots, n-2, n-1$

começa em s , para antes do número n

range(s, n, i)

$s, s+i, s+2i, \dots, (n-2), (n-1)$

começa em s , para antes do número n , pulando i números entre cada um




```
1 vowels = "aeiou"
2
3 for letter in vowels:
4     print(letter)
5
6 print(letter)
7 # imprime, em ordem: a e i o u u
8
```

```
1 for num in range(6):
2     print(num)
3
4 print(num)
5 # imprime, em ordem: 0 1 2 3 4 5 5
6
```


Tuplas

- Também são sequências, mas podem conter *qualquer tipo de objeto* (até outras tuplas!)
- Marcadas com **parênteses** e **vírgulas**


Operações em tuplas

-  concatena tuplas (forma uma única tupla)
-  concatena cópias da tupla
-  compara duas tuplas: iguais se tiverem mesmo tamanho e elementos correspondentes também são iguais (==)

4

```
1 x = (7, -7.8, "blue")
2 y = 7, -7.8, "blue" # parênteses são subentendidos
3
4 print(x == y) # True: x e y são objetos tuple
5 # diferentes em princípio, mas com mesmo valor
6 print(x + (1, 2, 3)) # (7, -7.8, "blue", 1, 2, 3)
7
8 for e in y:
9     print(e)
10
11 print(e) # imprime 7 -7.8 blue blue (em ordem)
12
```

Extraindo elementos (*indexando*)


-  retorna o elemento em um dado *índice*

Python usa **0-indexing**:
contagem começa em 0, não 1

Listas

- Quase idênticas a tuplas:
 - Sequência que contêm objetos de qualquer tipo
- Diferença: *listas são mutáveis* (podem ser alteradas depois de criadas)
- Marcadas com **colchetes** e **vírgulas**

Extraindo elementos

 retorna o elemento em um dado *índice*

Python usa **0-indexing**:
contagem começa em 0, não 1

```
1 x = (7, -7.8, "blue") # tupla
2 y = [7, -7.8, "blue"] # lista
3
4 print(x == y) # False: x e y têm os mesmos elementos,
5 # mas possuem tipos diferentes (tuple e list)
6 print(y + [1, 2, 3]) # [7, -7.8, "blue", 1, 2, 3]
7
8 for e in y:
9     print(e)
10
11 print(e) # imprime 7 -7.8 blue blue (em ordem)
12
13 z = tuple(y) # casting a lista para uma tupla
14 print(z == x) # True: z é uma tupla igual a x
15 |
```


Listas: mutabilidade

Novas operações em listas

`x[i] = y`

Modifica o elemento no index i da lista x , inserindo y em seu lugar

```
>>> a = (1,2,3)
```

Geraria um erro em tuplas!

```
>>> a[2] = 10
```

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

```
a[2] = 10
```

TypeError: 'tuple' object does not support item assignment

`x.append(y)`

Adiciona o elemento y no final da lista x (sem substituição, lista fica maior)

Modifica a lista atual, não gera uma cópia!

Retorna **None**, não y nem uma versão de x

```
1 x = [1, 2, 3]
2 x[2] = 100
3 print(x) # [1, 2, 100]
4
5 y = [4, 5, 6]
6 z = y
7 a = z.append(x)
8 print(a) # None
9
10 y[0] = 12
11 print(x) # [1, 2, 100]
12 print(y) # [12, 5, 6, [1, 2, 100]]
13 print(z) # [12, 5, 6, [1, 2, 100]]
14 |
```

Problema comum: aliasing

Os nomes y e z apontam para a mesma lista, então mudar y muda z também e vice-versa, mesmo que as variáveis pareçam diferentes

Sequências: *slicing*

`x[i:j]`

Retorna a subsequência de x que começa no índice i e termina no índice $j - 1$

`x[i:]`

Retorna a subsequência de x que começa no índice i e vai até o final de x

`x[:j]`

Retorna a subsequência de x do começo de x até o índice $j - 1$

`x[i:j:s]`

Retorna a subsequência de x que começa no índice i e termina no índice $j - 1$, pulando s elementos por vez

```
1 a = [0, 1, 2, 3, 4, 5, 6]
2
3 print(a[2:5]) # [2, 3, 4]
4 print(a[:4]) # [0, 1, 2, 3]
5 print(a[3:]) # [3, 4, 5, 6]
6 print(a[1::2]) # [1, 3, 5]
7 print(a[:]) # [0, 1, 2, 3, 4, 5, 6]
8 print(a[::-1]) # [6, 5, 4, 3, 2, 1, 0]
9 |
```

`x[:]`

Retorna uma cópia da sequência x : uma solução simples para aliasing!

`x[::-1]`

Truque simples para obter uma cópia de ordem reversa de uma sequência

Alguns padrões comuns

Construindo listas relacionadas

```
1 lista_original = [15, 12, 14, 0, -2, 1, 4]
2 nova_lista = []
3
4 for num in lista_original:
5     if num % 2:
6         nova_lista.append(num)
7
8 print(nova_lista) # ??? [15, 1]
9
10 nova_lista_2 = [num for num in lista_original if num % 2]
11 print(nova_lista == nova_lista_2) # True
12 |
```

Python List Comprehension

Jeito mais *pythônico* de criar
listas associadas: mais eficiente
e conciso que loop *for*

Exemplos: <https://www.vooo.pro/insights/tutorial-compreensao-de-listas-python-com-exemplos/>

Modelo para apenas com if

```
novo = []
for i in items:
    if i == y:
        novo.append(i)
```

if no final

```
novo = [i for i in items if i == y]
```

Modelo contendo if/else

```
novo = []
for i in items:
    if i == y:
        novo.append(i)
    else:
        novo.append(2 * i)
```

if / else no começo

```
novo = [i if i == y else 2 * i for i in items]
```


Alguns padrões comuns

Loop sobre índices

len(x)

Retorna o número de items em uma sequência (tupla, lista, string) x

range(len(x)) então dá todos os índices possíveis de x , começando em 0

```
for i in range(len(items)):
    v = items[i]
    ...
```

```
for i, v in enumerate(items):
    ...
```

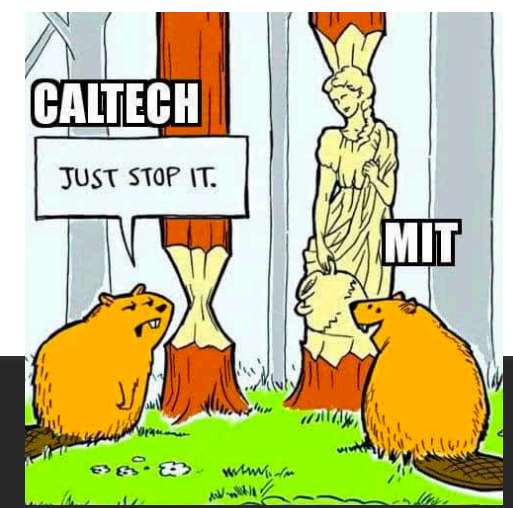
enumerate(x)

Fornece pares da forma (índice, objeto), um por vez, de uma sequência x

Tuple unpacking

```
1 animais = ["gato", "esquilo", "ornitorrinco", "castor"]
2 for a in animais:
3     print(a) # imprime gato esquilo ornitorrinco castor (em ordem)
4
5 for i in range(len(animais)):
6     print(i) # imprime 0 1 2 3 (em ordem)
7     print(animais[i]) # imprime gato esquilo ornitorrinco castor (em ordem)
8
9 for i, a in enumerate(animais):
10    print(a) # imprime gato esquilo ornitorrinco castor (em ordem)
11    print(i) # imprime 0 1 2 3 (em ordem)
12
```

```
1 a, b = 10, 20
2 print(a) # 10
3 print(b) # 20
4
```



Simply put, por exemplo, **enumerate(["a", "b", "c"])** forneceria, em ordem:

1. (0, "a")
2. (1, "b")
3. (2, "c")