

Computação 1 - Python

Aula 5 - Teórica: Manipulação de Strings, Tuplas e Listas

Manipulação de Strings

- Para obter ajuda a respeito de um tipo de dado, digite `help(tipo)`.
- Por exemplo: `help(str)` para obter ajuda sobre strings, `help(int)` para ajuda sobre inteiros, etc.
- Existem várias **funções** disponíveis para executar diferentes tarefas com strings. A sintaxe para estas funções é:
`str._nomeFunção_(umaString, _parâmetros_)`

■ Exemplo

```
>>>str.upper('abcde')  
'ABCDE'
```

Manipulação de Strings

- **lower():** retorna a string com todos os caracteres maiúsculos convertidos para minúsculos.
- **upper():** retorna a string com todos os caracteres minúsculos convertidos para maiúsculos.
- **Exemplo**

```
>>> str.upper("Esperança")
ESPERANÇA
>>> str.lower("Pé de Laranja Lima")
pé de laranja lima
```

Manipulação de Strings

- **str.count(umaString, elemento, inicio, fim)**: retorna quantas vezes o elemento aparece na string, procurando-se a partir da posição **inicio** e indo até a posição **fim** - 1.
- **inicio** e **fim** são opcionais.
- **Exemplo**

```
>>> frase="macaco come banana"  
>>> str.count(frase, "a", 2, 10)  
>>> 1
```

Manipulação de Strings

- `str.index(umaString, elemento, inicio, fim)`: retorna o índice da primeira ocorrência de elemento na string, a partir da posição **inicio**, até a posição **fim - 1**.
- **inicio** e **fim** são opcionais.
- **Exemplo**

```
>>> str.index("mariana", "a")
```

```
>>> str.index("mariana", "a", 2)
```

```
>>> str.index("mariana", "a", 5, 7)
```

```
>>> str.index('Mariana', 'ana')
```

```
>>> str.index('Mariana', 'x')
```

Manipulação de Strings

- **str.index(umaString,elemento, inicio, fim)**: retorna o índice da primeira ocorrência de elemento na string, a partir da posição **inicio**, até a posição **fim - 1**.
- **inicio** e **fim** são opcionais.

■ Exemplo

```
>>> str.index("mariana", "a")
1
>>> str.index("mariana", "a", 2)
4
>>> str.index("mariana", "a",5, 7)
6
>>> str.index('Mariana', 'ana')
4
>>> str.index('Mariana', 'x')
Traceback (most recent call last):
File "<pyshell#1>", line 1, in <module>
str.index('Mariana', 'x')
ValueError: substring not found
```

Tuplas

- Uma **tupla** é uma sequência heterogênea (permite que seus elementos sejam de tipos diferentes):

```
>>> a = (1,2,3,4)
>>> b = (1.0, 2, '3', 4+0j)
>>> c = 1,2,3,4
>>> d = (1,)
```

- Valores em uma tupla podem ser distribuídos em variáveis como uma atribuição múltipla:

```
>>> x = 1, 2, 3
>>> x
(1, 2, 3)
>>> a, b, c = x
>>> a
1
>>> b
2
>>> c
3
```

Tuplas

- **Tupla Vazia:** `()`
- **Tupla unitária:** contém um único elemento, que deve ser sucedido por uma vírgula.
- Os parênteses são opcionais se não provocarem ambiguidade.
- Um valor entre parênteses sem vírgula no final é meramente uma expressão.

```
>>> (10)
10
>>> 10,
(10,)
>>> (10,)
(10,)
>>> 3*(10+3)
39
>>> 3*(10+3,)
(13, 13, 13)
```


Tuplas

- Tuplas são muito similares às strings em relação às operações.
- O tamanho de uma tupla é dado pela função `len`.

```
>>> x = (1,2,3)
>>> len(x)
3
```

- **Indexação:** começando do 0 à esquerda, ou de -1 à direita.

```
>>> x[0]
1
```

- **Fatiamento:** idêntico às strings.

```
>>> x[0:2]
(1,2)
```

Tuplas

- Tuplas são muito similares às strings em relação às operações.
- O tamanho de uma tupla é dado pela função `len`.

```
>>> x = (1,2,3)
>>> len(x)
3
```

- **Indexação:** começando do 0 à esquerda, ou de -1 à direita.

```
>>> x[0]
1
```

- **Fatiamento:** idêntico às strings.

```
>>> x[0:2]
(1,2) -> NOVA TUPLA
```

Tuplas

■ Concatenação e Replicação

```
>>> x*2
(1,2,3,1,2,3)
>>> x + (5,4)
(1,2,3,5,4)
```

■ Imutabilidade : uma vez criada, uma tupla não pode ser alterada !

```
>>> x[0] = 9
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[0]=9
TypeError: 'tuple' object does not support item assignment
```

Tuplas

Joãozinho quer comprar o maior número de bombons possível com o dinheiro que tem. Faça funções para:

- a. calcular o número de bombons e o troco, dados o dinheiro e o preço de um bombom.

Tuplas

Joãozinho quer comprar o maior número de bombons possível com o dinheiro que tem. Faça funções para:

- calcular o número de bombons e o troco, dados o dinheiro e o preço de um bombom.

```
def bombom(dinheiro,preco):  
    return dinheiro // preco , dinheiro % preco
```

Tuplas

Joãozinho quer comprar o maior número de bombons possível com o dinheiro que tem. Faça funções para:

- calcular o número de bombons e o troco, dados o dinheiro e o preço de um bombom.

```
def bombom(dinheiro,preco):  
    return dinheiro // preco , dinheiro % preco
```

- calcular quanto Joãozinho terá que pedir para sua mãe para comprar um bombom a mais, dados o dinheiro que ele tem e o preço de um bombom. Utilize a função definida em a.

Tuplas

Joãozinho quer comprar o maior número de bombons possível com o dinheiro que tem. Faça funções para:

- calcular o número de bombons e o troco, dados o dinheiro e o preço de um bombom.

```
def bombom(dinheiro,preco):  
    return dinheiro // preco , dinheiro % preco
```

- calcular quanto Joãozinho terá que pedir para sua mãe para comprar um bombom a mais, dados o dinheiro que ele tem e o preço de um bombom. Utilize a função definida em a.

```
def maisbombom(dinheiro,preco):  
    return preco - bombom(dinheiro,preco)[1]
```

Testes: `bombom(10,3)` e `maisbombom(10,3)`

Tuplas

- a. Escreva uma função que recebe uma tupla e retorna **True** se o primeiro elemento for igual ao último elemento da tupla.

Tuplas

- a. Escreva uma função que recebe uma tupla e retorna **True** se o primeiro elemento for igual ao último elemento da tupla.

```
# Início da tupla é igual ao final?  
# tupla → bool  
def igual_if(tup):  
    return tup[0] == tup[-1]
```

Tuplas

- a. Escreva uma função que recebe uma tupla e retorna **True** se o primeiro elemento for igual ao último elemento da tupla.

```
# Início da tupla é igual ao final?  
# tupla → bool  
def igual_if(tup):  
    return tup[0] == tup[-1]
```

- b. Escreva uma função *inverte* que recebe uma tupla de três elementos e retorna uma nova tupla com os elementos na ordem reversa.

Tuplas

- b. Escreva uma função *inverte* que recebe uma tupla de três elementos e retorna uma nova tupla com os elementos na ordem reversa.

```
# Inverte elementos de uma tupla de tamanho 3
# tupla_tamanho_3 → tupla_tamanho_3
def inverte(tup):
    return tup[2], tup[1], tup[0]
```

```
# Inverte elementos de uma tupla de tamanho 3
# tupla_tamanho_3 → tupla_tamanho_3
def inverte(tup):
    return tup[::-1]
```

Tuplas

Exercícios

- c. Escreva a função *intercala* que recebe duas tuplas de três elementos cada e retorna uma tupla de seis elementos intercalando as duas tuplas.
- d. Escreva a função *opera* que recebe uma tupla com uma string e dois números; se a string for 'SOMA', retorna a soma dos dois números, se for 'MULT', retorna a multiplicação, se for 'DIV', retorna a divisão, se for 'SUB', retorna a subtração, se não for nenhuma das anteriores retorna *None*.

Listas

- Tipo de dados mais versátil do Python.
- Uma lista é representada como uma sequência de valores entre colchetes e separados por vírgula.
- Os elementos de uma lista podem ser de tipos de dados diferentes.
- Listas são **mutáveis** !!!

Exemplo

```
>>> lista1 = ['calculo', 'fisica', 'computacao']
>>> lista2 = ['notas', 5.4, 'aprovado']
>>> lista2[1] = 6
>>> lista2
['notas', 6, 'aprovado']
```

- **Atenção:** *Uma lista vazia não contém nenhum elemento*

Exemplo

```
>>> lista3 = [ ]
>>> lista3[0]
Traceback (most recent call last):
File "<pyshell#18>", line 1, in <module>
lista3[0]
IndexError: list index out of range
```

Listas

Nome da sequência (c)

c[0]	-45	c[-12]
c[1]	6	c[-11]
c[2]	0	c[-10]
c[3]	72	c[-9]
c[4]	1543	c[-8]
c[5]	-89	c[-7]
c[6]	0	c[-6]
c[7]	62	c[-5]
c[8]	-3	c[-4]
c[9]	1	c[-3]
c[10]	6453	c[-2]
c[11]	78	c[-1]

Número da posição do elemento dentro da sequência c

```
>>> c = [-45, 6, 0, 72, 1543, -89, 0, 62, -3, 1, 6453, 78]
```

```
>>> c[3]
72
```

```
>>> c[9]==c[-3]
True
```

```
>>> len(c)
12
```

Listas

```
>>> [1,2] + [3]
```

```
>>> [1,2] + [[3]]
```

```
>>> [[1,2]] + [[3]]
```

```
>>> [1,2] * 3
```


Listas

```
>>> [1,2] + [3]
[1, 2, 3] (Concatenando Listas)
```

```
>>> [1,2] + [[3]]
[1, 2, [3]]
```

```
>>> [[1,2]] + [[3]]
[[1, 2], [3]]
```

```
>>> [1,2] * 3
[1, 2, 1, 2, 1, 2] (Equivale a [1,2]+[1,2]+[1,2])
```

Listas

```
>>> [1,2] * [3]
```

```
>>> [1,2] - [3]
```

Listas

```
>>> [1,2] * [3]
Traceback (most recent call last):
File "<pyshell#35>", line 1, in <module>
[1,2]*[3]
TypeError: can't multiply sequence by non-int of type 'list'

>>> [1,2] - [3]
Traceback (most recent call last):
File "<pyshell#37>", line 1, in <module>
[1,2]-[2]
TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

Como retirar um elemento de uma lista?
Aguarde

Listas

Faça uma função que receba duas listas como entrada e retorne a concatenação destas listas.

Listas

Faça uma função que receba duas listas como entrada e retorne a concatenação destas listas.

```
# Função que dadas duas listas,  
# retorna a concatenação das listas  
# list,list → list  
def concatenaListas(Lista1,Lista2):  
    return Lista1+Lista2
```

```
>>> concatenaListas([1,2,3],[4,5,6])  
[1,2,3,4,5,6]
```

Listas

Faça uma função que dado um número inteiro como entrada, retorne uma lista com todos os números pares entre 1 e o número dado, inclusive.

Listas

- A função `range(...)` pode ter 1, 2 ou 3 argumentos:
 - `range(numero)`: retorna uma lista contendo uma sequência de valores de `0` a `numero-1`

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```
 - `range(inf,sup)`: retorna uma lista contendo uma sequência de valores de `inf` a `sup-1`

```
>>> range(3, 8)
[3, 4, 5, 6, 7]
```
 - `range(inc, sup, inc)`: retorna uma lista contendo uma sequência de valores de `inf` a `sup-1` com incremento de `inc`

```
>>> range(3, 8, 2)
[3, 5, 7]
```

Listas

- **ATENÇÃO:** A função `range(...)` começa com **zero**

- São equivalentes:

```
range(10)
```

```
range(0,10)
```

```
range(0,10,1)
```

- Exemplos:

```
>>> range(3)
```

```
>>> range(2,5,2)
```

```
>>> range(5,2,-2)
```


Listas

- **ATENÇÃO:** A função `range(...)` começa com **zero**
- São equivalentes:

```
range(10)
```

```
range(0,10)
```

```
range(0,10,1)
```

- Exemplos:

```
>>> range(3)
```

```
[0, 1, 2]
```

```
>>> range(2,5,2)
```

```
[2, 4]
```

```
>>> range(5,2,-2)
```

```
[5, 3]
```

Listas

Faça uma função que dado um número inteiro como entrada, retorne uma lista com todos os números pares entre 1 e o número dado, inclusive.

Listas

Faça uma função que dado um número inteiro como entrada, retorne uma lista com todos os números pares entre 1 e o número dado, inclusive.

```
# Função que dado um número inteiro,  
# retorna uma lista com todos os números  
# pares entre 1 e o número dado, inclusive  
# int → list  
def lista(n):  
    if n%2==0:  
        return range(2,n+1,2)  
    else:  
        return range(2,n,2)
```

```
>>> lista(5)  
[2,4]  
>>> lista(6)  
[2,4,6]
```

Listas - Exercícios

1. Faça uma função que dada uma lista com 5 notas, retorne a média das notas.
2. Faça uma função que, dados dois inteiros x e y , retorna uma lista com todos os valores entre x e y (inclusive), funcionando tanto para $x \leq y$ como para $x > y$.

Exemplos

$x = 2, y = 6, \text{ resultado} = [2, 3, 4, 5, 6]$

$x = 10, y = 7, \text{ resultado} = [10, 9, 8, 7]$

3. Faça uma função que dadas duas listas de três elementos com números inteiros, retorna uma lista onde cada elemento é a soma dos elementos de mesma posição nas duas primeiras listas.

Exemplo

Lista1 = [1,4,6]

Lista2 = [2,4,3]

Lista resultante = [3,8,9]

Computação 1 - Python

Aula 5 - Teórica: Manipulação de Strings, Tuplas e Listas