TECHMIMO
Real Solutions

Curso de Python Básico

Criado por Rafael Pereira da Silva

2a ed, São José dos campos, junho de 2020

Contatos:

- techmimo.solutions@gmail.com
- http://www.linkedin.com/in/rafael-pereira-da-silva-23890799 (http://www.linkedin.com/in/rafael-pereira-da-silva-23890799)

Seção 1 - Introdução ao curso

1.1 Iniciando os estudos

Por que Python?

- Linguagem de alto nível e sintaxe fácil
- Comunidade grande e com muitas bibliotecas prontas
- Open source

Sobre

- Linguagem orientada a objetos
- Distribuidos nas versões 2.n e 3.n

Distribuições recomendadas:

- https://www.anaconda.com/products/individual (https://www.anaconda.com/products/individual)
- https://sourceforge.net/projects/winpython/files/WinPython_3.8/ (https://sourceforge.net/projects/winpython/files/WinPython_3.8/)

1.2 IDEs

- *Integrated Development Environment*
- Exemplos de IDE: Idle, Ipython, Pycharm, Spyder, Jupyter notebook.
- Procurar por: idle.exe, ipython3.exe, Spyder3.exe
- Os executáveis estão na pasta Scripts ou WinPython-64bit.

- o Pycharm deve ser instalado separadamente: https://www.jetbrains.com/pycharm/download/#section=windows (https://www.jetbrains.com/pycharm/download/#section=windows)

In [2]:

```
print('Olá mundo')
```

```
Olá mundo
```

In [3]:

```
1+1
```

Out[3]:

```
2
```

Seção 2 - Introdução ao Python

Essa aula é inspirada em:

- https://docs.python.org/3/tutorial/introduction.html (https://docs.python.org/3/tutorial/introduction.html)
- https://docs.python.org/3/library/stdtypes.html#iterator-types (https://docs.python.org/3/library/stdtypes.html#iterator-types)

2.1 Iniciando os estudos

Tipo de variáveis (*built-in types*):

- int = número inteiro
- float = número real
- complex = número complexo
- str = texto

Estes tipos de valores podem ser armazenados dentro de variáveis.

Primeiras funções

Função	Descrição
print()	exibe um dado input
input()	retorna uma **string** dada pelo usuário
int(), float(), str()	conversão de tipo
type()	retorna o tipo da variável

In [1]:

```
variavel_a = 'Olá mundo'

print(variavel_a)
```

```
Olá mundo
```

In [4]:

```
variavel_b = input('Digite um número')

type(variavel_b)
```

```
Digite um número10
```

Out[4]:

```
str
```

In [11]:

```
numero_str = '5.'

numero = float(numero_str)

type(numero)
```

Out[11]:

```
float
```

2.2 Números

Operações básicas:

Símbolo	Descrição
(+)	soma
(-)	subtração
(*)	multiplicação
(/)	divisão
(//)	parte inteira da divisão
(**)	exponencial
+=	adiciona para a variável
*=	multiplica pela variável

- LEE, Kent D. Python Programming Fundamentals. Second Edition. Springer - Verlag London 2014.
- https://link.springer.com/chapter/10.1007%2F978-1-4471-6642-9_8 (https://link.springer.com/chapter/10.1007%2F978-1-4471-6642-9_8)
- https://link.springer.com/chapter/10.1007%2F978-1-4471-6642-9_9 (https://link.springer.com/chapter/10.1007%2F978-1-4471-6642-9_9)

In [29]:

```
a = 5
b = a + 5.

type(b)
```

Out[29]:

```
float
```

In [33]:

```
c = 10
c -= 5
c
```

Out[33]:

```
5
```

In [34]:

```
5**2
```

Out[34]:

```
25
```

In [38]:

```
e = 5
f = 2

type(e//f)
```

Out[38]:

```
int
```

In [46]:

```
g = 1j

g**2
```

Out[46]:

```
(-1+0j)
```

2.3 String

- Strings são variáveis de texto
- Alguns caracteres, como as aspas, não podem ser adicionados diretamente. Então é necessário \
- \n representa uma nova linha
- Algumas operações básicas funcionam para manipular strings

In [53]:

```
print('Olá " mundo')
```

```
Olá " mundo
```

In [61]:

```
a = str(10)

b = 'Python ' + a

b
```

Out[61]:

```
'Python 10'
```

2.4 Listas

Listas são conjuntos de dados que podem conter dados de diferentes tipos. Elas possuem valores indexados a partir de zero.

In [70]:

```
lista_a = [5,6,['python',2],8,9]
type(lista_a)
```

Out[70]:

```
list
```

In [75]:

```
lista_a[4]
```

Out[75]:

```
9
```

In [80]:

```
lista_b = ['c','a','e']

lista_c = lista_b + lista_b

lista_c
```

Out[80]:

```
['c', 'a', 'e', 'c', 'a', 'e']
```

In [83]:

```
a = len(lista_c)
a + a
```

Out[83]:

```
12
```

In [87]:

```
lista_d = ['c', 'a', 'e', 'c', 'a', 'e']

lista_d[2:]

# slicing
```

Out[87]:

```
['e', 'c', 'a', 'e']
```

2.5 Similaridades entre listas e strings

- Ambos são indexáveis com indices a partir de [0]
- Strings, assim como listas, podem ser fatiadas
- A funcão len(*x*) retorna o número de elementos de uma lista e o número de caractéres de uma string
- A operação de soma funciona de maneira parecida
- Ambas podem ser multiplicadas por um valor inteiro

Mas atenção, não são do mesmo tipo!

In [100]:

```
lista_a = ['PA','YA','TA','H','O','N']

string_a = 'pAyAtAhon'

lista_a + listas
```

Out[100]:

```
['PA', 'YA', 'TA', 'H', 'O', 'N', 'PA', 'YA', 'TA', 'H', 'O', 'N']
```

2.6 Valores Booleanos (*Boolean Values*)

O que é

As variáveis do tipo *bool* armazenam constantes do tipo *True* ou *False*

In [106]:

```
a = False
type(a)
```

Out[106]:

```
bool
```

Comparações

Comparações retornam variáveis do tipo *bool*

Comparação	Descrição
>	menor que

Comparação	Descrição
<=	menor ou igual a
>	maior que
>=	maior ou igual a
==	igual a
!=	diferente de

Fonte: https://docs.python.org/3/library/stdtypes.html#iterator-types (https://docs.python.org/3/library/stdtypes.html#iterator-types)

In [111]:

```
a = 5
b = 10
variavel_bool =  a < b

variavel_bool
```

Out[111]:

```
True
```

Seção 3 - Condionais e loops

Palavras chave: *Control flow* ou *Control structures*

Essa aula é inspirada em: https://docs.python.org/3/tutorial/controlflow.html (https://docs.python.org/3/tutorial/controlflow.html)

3.1 Condicional *if*

Construções

- **if** condição**:**
- **elif** condição**:**
- **else:**

Operações booleanas

Além dos comparadores, também podemos fazer operações booleanas. Elas são úteis em estruturas condicionais. Duas delas são: **or** e **and**.

Acesse: https://docs.python.org/3/library/stdtypes.html# (https://docs.python.org/3/library/stdtypes.html)

In [18]:

```
x = -6

if x > 5 or x < -5:
    print('OK')
```

OK

3.2 Loop *for*

- **for** variável **in** o_que_será_iterado**:**

Há uma série de tipos que podem ser utilizado como iteradores, como:

- range --> seus elementos são inteiros
- list --> seus elementos são os que estão contidos na lista
- enumerate --> seus elementos são tuplas que contem inteiros e valores

In [24]:

```
for variavel_i in range(2,10):
    print(variavel_i**2)
```

```
4
9
16
25
36
49
64
81
```

In [29]:

```
lista_vogais = ['a','e','i','o','u']

for i in range(len(lista_vogais)):
    print(i)
```

```
0
1
2
3
4
```

In [35]:

```
for i in enumerate(lista_vogais):
    print(i[1])
```

```
a
e
i
o
u
```

3.3 Loop *while*

- **while** condição:

Enquanto a condição for verdadeira, o programa executará os comandos

Uma noção sobre tipos booleanos é interessante para executar esse comando

In [45]:

```
x = 0

while x < 10:
    x += 2
    print(x)
```

```
2
4
6
8
10
```

In [48]:

```
x = 4
fact_x = 1

while x > 0:
    fact_x *= x
    x -= 1

fact_x
```

Out[48]:

```
24
```

Seção 4 - Estrutura de dados

Essa aula é inspirada em:

- https://docs.python.org/3/tutorial/datastructures.html (https://docs.python.org/3/tutorial/datastructures.html)
- LEE, Kent D. Python Programming Fundamentals. Second Edition. Springer - Verlag London 2014.

4.1 Métodos para listas

Método	Descrição
.append(*variável*)	Adiciona a variável no final
.extend(*outra_lista*)	Adiciona os elementos ao final
.remove(*variável*)	Remove o primeiro elemento com valor *variável*
.count(*variável*)	Conta o número de elementos com o valor *variável*
.sort()	Reordena elementos em ordem numérica ou alfabética
.reverse()	Inverte a ordem dos elementos
.copy()	Retorna uma cópia da lista
.index(*variável*)	Retorna o índice da primeira *variável* da lista
.pop(*índice*)	Remove o elemento da lista e retorna seu valor

In [6]:

```
lista_inteiros = [1,2,3,4,5]

lista_inteiros.append(6)

lista_2 = [7,8,9]

lista_inteiros.append(lista_2)

lista_inteiros[6]
```

Out[6]:

```
[7, 8, 9]
```

In [11]:

```
lista_letras = ['a','s','d','a','s','a']

lista_letras.sort()

lista_letras
```

Out[11]:

```
['a', 'a', 'a', 'd', 's', 's']
```

In [ ]:

```
```

4.2 Formas de trabalhar com listas

Função	Descrição
range(*inteiro*)	retorna um iterável do tipo range
lambda x: *função_x*	Retorna uma função
map(*função*,*lista*)	Retorna um iterável do tipo map
list(*iterável*)	Cria uma lista a partir de um iterável

Atenção, este não é um tutorial sobre programação funcional! Quem se interessar pode pesquisar por: *Functional Programming* ou Programação Funcional.

In [15]:

```
x = []

for i in range(10):
    x.append(i)

x
```

Out[15]:

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

In [18]:

```
y = list(range(10))
type(y)
```

Out[18]:

```
list
```

In [24]:

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

f = lambda x:x**2

list(map(f,lista))
```

Out[24]:

```
[1, 4, 9, 16]
```

In [27]:

```
quadrados = [x for x in range(1,5)]

quadrados
```

Out[27]:

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

In [ ]:

```

```

4.3 Tuplas, conjuntos e dicionários

Tipo	Descrição	Sintaxe
tuple	tupla: uma sequência de dados que é imutável	vogais = ('a','e','i','o','u')
set	conjunto: elementos não possuem ordem e não se repetem	alg_decimais = {0,1,2,3,4,5,6,7,8,9}
dict	dicionário: são indexaxos por uma chave *keys*	alg_romanos = {'I':1,"II":2,'III':3,'IV':4,'V':5,'X':10}

In [42]:

```
vogais = ('a','e','i','o','u')
```

In [52]:

```
inteiros = {0,1,2,3,4,5,6,7,8,9,11}

len(inteiros)
```

Out[52]:

```
11
```

In [54]:

```
dicionario_1 = {'A':1,'B':2,'C':3}
dicionario_1['B']
```

Out[54]:

```
2
```

In [59]:

```
vogais = ['a','e','i','o','u']

for i in enumerate(vogais):
    print(i)
```

```
(0, 'a')
(1, 'e')
(2, 'i')
(3, 'o')
(4, 'u')
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

4.4 Métodos para dicionários

Método	Descrição
.items()	Retorna os itens do dicionário
.keys()	Retorna as chaves do dicionário
.values()	Retorna as chaves do dicionário
.pop(*key*)	Remove a chave especificada e retorna o valor do item
.copy()	Retorna uma cópia
.clear()	Remove os itens
.get(*key*)	Retornas a váriável em *key*

Disponível em: https://link.springer.com/chapter/10.1007%2F978-1-4471-6642-9_12 (https://link.springer.com/chapter/10.1007%2F978-1-4471-6642-9_12)

In [65]:

```
alg_romanos = {'I':1 , 'II':2,'III':3, 'IV':4}
II =  alg_romanos.pop('II')

II
```

Out[65]:

```
2
```

In [74]:

```
(list(alg_romanos.items()))[0][1]
```

Out[74]:

1

Seção 5 - Funções e módulos

5.1 Funções

Definindo e chamando funções

def minha_funcao(argumentos)**:**

```
        funcao_aqui
    minha_funcao(argumentos) ##para chamar a função
```

Declaração *return*

É usada quando se deseja retornar uma variável específica.

def minha_funcao(argumentos)**:**

```
     resultado = operações

     **return** resultado
```

In [3]:

```
def minha_funcao(p_nome,s_nome):
    print('Meu nome é ' + p_nome + s_nome)
    print(s_nome)
    print(p_nome)


p = 'Rafael '
s = 'Pereira'

minha_funcao(p,s)
```

```
Meu nome é Rafael Pereira
Pereira
Rafael
```

In [10]:

```
def soma_quadrados(n_1,n_2):
    resposta = a**2 + b**2
    return resposta

a = 3
b = 4.

a = soma_quadrados(a,b)

type(a)
```

Out[10]:

```
float
```

5.2 Módulos

O que é

Módulos são arquivos Python (extensão *.py*) que contém conjuntos de funções, objetos, variáveis, entre outros. Essas funcionalidades podem ser chamadas das seguintes maneiras:

import Nome_do_modulo

import Nome_do_modulo **as** abreviação

from Nome_do_modulo **import** algum_objeto

Referência

Lista de módulos do Python: https://docs.python.org/3.8/py-modindex.html (https://docs.python.org/3.8/py-modindex.html)

In [21]:

```
from numpy import pi,cos

a = cos(0)
a
```

Out[21]:

```
1.0
```

In [25]:

```
from MEDIA import g

g
```

Out[25]:

```
9.8
```

In [ ]:

```
```

5.3 Lendo e escrevendo arquivos (extra)

- **Estrutura para ler:**

with open(caminho_string**) as f:**

    variavel = **f.read()**

    f.close

- **Estrutura para escrever:**

with open(nome_novo_arquivo,**'w') as f:**

    f.write('nova_string')

    f.close

In [2]:

```
caminho = 'D:/Programas/Python/WinPython-64bit-3.6.1.0Qt5/notebooks/Python básico/Arquivo_1

with open(caminho) as f:
    texto_1 = f.read()
    f.close

texto_1
```

Out[2]:

```
'A\nB\nC\nD\nE'
```

In [4]:

```
caminho_2 = 'D:/Programas/Python/WinPython-64bit-3.6.1.0Qt5/notebooks/Python básico/Arquivo

texto_2 = texto_1 + '\nF\nG\nH'

with open(caminho_2,'w') as f:
    f.write(texto_2)
    f.close
```

In [ ]:

```
```