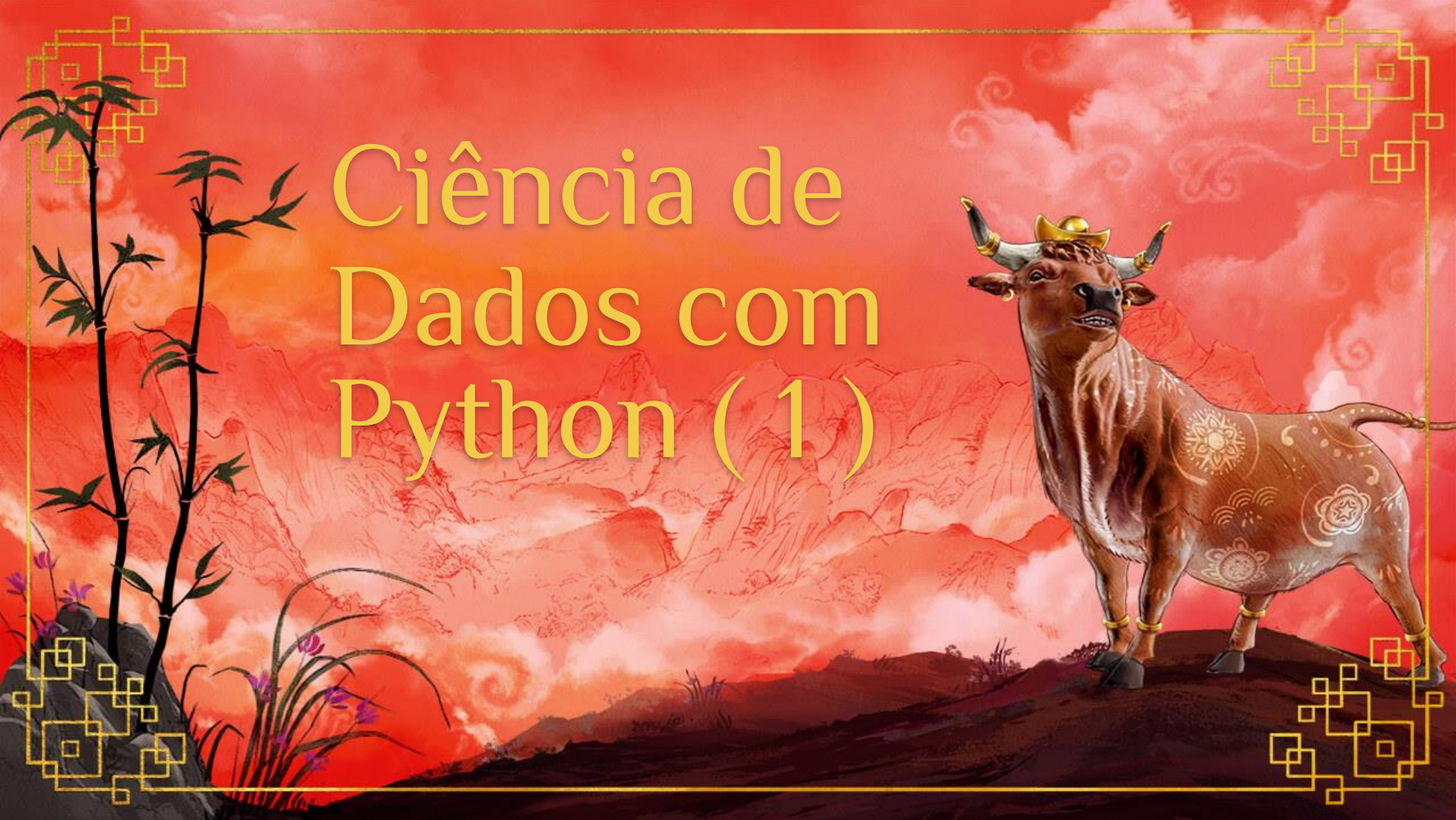
Ciência de Dados com Python (1)



“Ensinar é indicar o caminho,
mas na viagem cada um vai ver
o que quiser ver.”

(Plotino)


jupyter
python
ANACONDA

1.Introdução

Let´s understand

Ciência de Dados

- Área interdisciplinar voltada para o estudo e a análise de dados econômicos, financeiros e sociais
- Dados estruturados e não-estruturados
- Visa a extração de conhecimento, detecção de padrões e/ou obtenção de insights para possíveis tomadas de decisão


Nerdologia
Nº40
BIG DATA


Verdelegia
Nº 131
DILÚVIO
DE DADOS

Python

- É uma linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.
- Foi lançada por Guido van Rossum em 1991.
- Muito popular nas áreas da tecnologia relacionadas à análise de dados, pesquisa, desenvolvimento de algoritmos e IA

2.Ambiente de Trabalho

Let´s prepare

Anaconda

- É uma distribuição das linguagens de programação Python e R para computação científica.
- Visa simplificar o gerenciamento e implantação de pacotes.
- A distribuição inclui pacotes de ciência de dados adequados para Windows, Linux e macOS.

https://www.anaconda.com

Jupyter

- É uma interface gráfica que permite a edição de notebooks em um navegador web.
- O nome Jupyter é um acrônimo criado a partir das linguagens de programação que inicialmente foram aceitas pelo Projeto Jupyter: Julia, Python e R.
- Criado por Fernando Pérez e Brian Granger em 2015

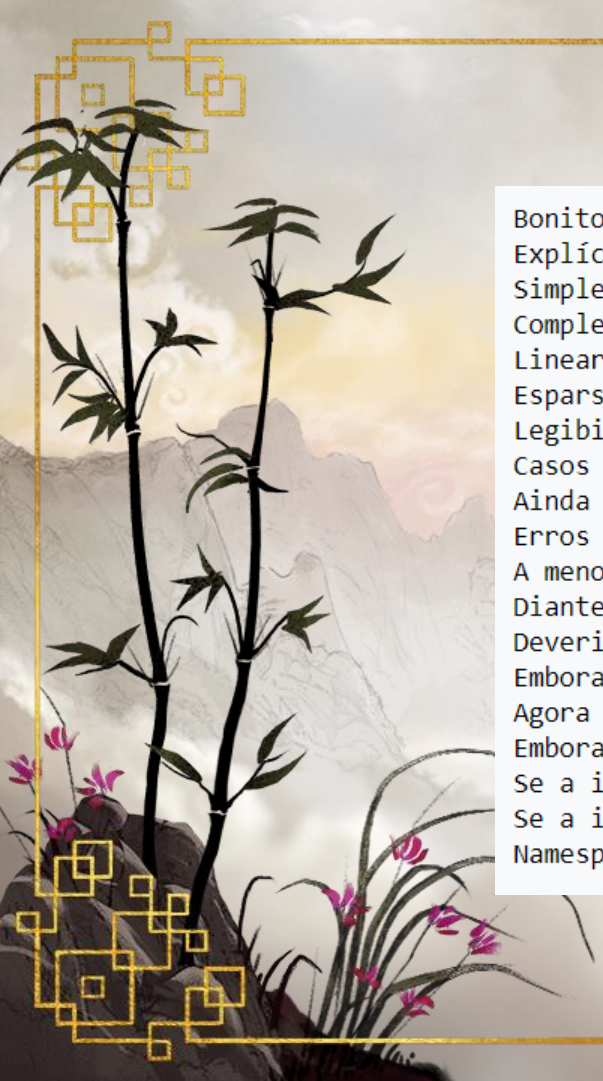
https://jupyter.org


3.Python
Let´s code – Tipos de Dados

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```



```
Bonito é melhor que feio
Explícito é melhor que implícito
Simples é melhor que complexo
Complexo é melhor que complicado
Linear é melhor do que aninhado
Esparso é melhor que denso
Legibilidade conta
Casos especiais não são especiais o bastante para quebrar as regras.
Ainda que praticidade vença a pureza
Erros nunca devem passar silenciosamente.
A menos que sejam explicitamente silenciados
Diante da ambiguidade, recuse a tentação de adivinhar
Deveria haver um — e preferencialmente apenas um — modo óbvio para fazer algo.
Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês
Agora é melhor que nunca
Embora nunca freqüentemente seja melhor que já
Se a implementação é difícil de explicar, é uma má ideia
Se a implementação é fácil de explicar, pode ser uma boa ideia
Namespaces são uma grande ideia — vamos ter mais dessas!
```

Operações matemáticas

```
In [8]: 2 + 3
Out[8]: 5

In [9]: 3 - 1
Out[9]: 2

In [10]: 4 * 3
Out[10]: 12

In [11]: 9 / 2
Out[11]: 4.5

In [12]: 9 % 2
Out[12]: 1

In [7]: 2 ** 4
Out[7]: 16
```

Operadores Lógicos

```
In [45]: 1 > 2
Out[45]: False

In [46]: 1 < 2
Out[46]: True

In [47]: 1 >= 1
Out[47]: True

In [48]: 1 <= 4
Out[48]: True

In [49]: 1 == 1
Out[49]: True

In [37]: 'oi' == 'tchau'
Out[37]: False
```

```
In [51]: (1 > 2) and (2 < 3)
Out[51]: False

In [52]: (1 > 2) or (2 < 3)
Out[52]: True

In [53]: (1 == 2) or (2 == 3) or (4 == 4)
Out[53]: True
```

Definição de Variáveis

```
In [13]: # comentários
         anoAtual = 2021
         anoNascimento = 1969
         nome = 'Lucília'
         idade = anoAtual - anoNascimento
```

```
In [14]: print(nome)
         print(idade)
```

```
Lucília
51
```

```
In [16]: print('Meu nome é {}, e tenho {} anos'.format(nome,idade))
```

```
Meu nome é Lucília, e tenho 51 anos
```

4.Python

Outros Tipos de Dados – Let's code

Listas

```
In [20]: [1,2,3]
Out[20]: [1, 2, 3]

In [1]: ['Oi',1,[1,2]]
Out[1]: ['Oi', 1, [1, 2]]

In [12]: minha_lista = ['a','b','c']

In [13]: minha_lista.append('d')

In [14]: minha_lista
Out[14]: ['a', 'b', 'c', 'd']

In [15]: minha_lista[0]
Out[15]: 'a'

In [16]: minha_lista[1]
Out[16]: 'b'
```

```
In [17]: minha_lista[1:]
Out[17]: ['b', 'c', 'd']

In [18]: minha_lista[:1]
Out[18]: ['a']

In [28]: minha_lista[0] = 'NOVO'

In [29]: minha_lista
Out[29]: ['NOVO', 'b', 'c', 'd']

In [3]: lista2 = [1,2,3,[4,5,['nome']]]

In [4]: lista2[3]
Out[4]: [4, 5, ['nome']]

In [5]: lista2[3][2]
Out[5]: ['nome']

In [6]: lista2[3][2][0]
Out[6]: 'nome'
```

Dicionários

```
In [7]: nome = 'Lucília'

In [9]: d = {'nome':nome,'idade':51, 'hobby':'codar', 'filhos':['Bruna', 'André']}

In [10]: d
Out[10]: {'nome': 'Lucília',
          'idade': 51,
          'hobby': 'codar',
          'filhos': ['Bruna', 'André']}

In [11]: d['hobby']
Out[11]: 'codar'

In [12]: d['filhos'][0]
Out[12]: 'Bruna'
```

Tuplas

```
In [40]: t = (1,2,3)

In [13]: l = [1, 2, 3]

In [41]: t[0]
Out[41]: 1

In [14]: l.append(4)

In [15]: l
Out[15]: [1, 2, 3, 4]

In [16]: l[0] = 'novo'

In [17]: l
Out[17]: ['novo', 2, 3, 4]
```

```
In [18]: t.append(4)
---------------------
NameError
<ipython-input-18-a
----> 1 t.append(4)

NameError: name 't'
```

```
In [42]: t[0] = 'NEW'
---------------------
TypeError
<ipython-input-42-9
----> 1 t[0] = 'NEW'

TypeError: 'tuple'
```

5.Python

Estruturas de Controle – Let's code

Estruturas Condicionais

```
In [20]: if (1 < 2):
             print('Sim!')
```

```
Sim!
```

```
In [21]: if (1 < 2):
             print('Primeiro')
         else:
             print('Último')
```

```
Primeiro
```

```
In [19]: if (1 > 2) and (5 <4):
             print('Primeiro')
         else:
             print('Último')
```

```
Último
```

```
In [22]: if (1 == 2):
             print('Primeiro')
         elif (3 == 3):
             print('Meio')
         else:
             print('Último')
```

```
Meio
```

Laços For

```
: seq = [1,2,3,4,5]
```

```
: for item in seq:
      print(item)
```

```
1
2
3
4
5
```

```
: for item in seq:
      print('!')
```

```
!
!
!
!
!
```

```
a = list(range(5, 10))
```

```
print(a)
```

```
[5, 6, 7, 8, 9]
```

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

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

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

Laços While

```
i = 1
while i < 5:
    print('i = {}'.format(i))
    i = i+1
```

```
i = 1
i = 2
i = 3
i = 4
```

List Comprehension

```python
x = [1,2,3,4]
```

```python
out = []
for item in x:
    out.append(item**2)
print(out)
```

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

```python
[item**2 for item in x]
```

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

```python
resultado = [numero for numero in range(20) if numero % 2 == 0]
print(resultado)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

```python
resultado = [numero for numero in range(100) if numero % 5 == 0 if numero % 6 == 0]
print (resultado)
```

```
[0, 30, 60, 90]
```

```python
resultado = ['1' if numero % 5 == 0 else '0' for numero in range(16)]
print (resultado)
```

```
['1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1']
```

5.Python

Funções – Let´s create

Funções

```
def minhaFuncao(param):
    print(param)
```

```
minhaFuncao
```

```
<function __main__.minhaFuncao(param)>
```

```
param1 = 'novo parametro'
minhaFuncao(param1)
```

```
novo parametro
```

```
def square(x):
    return x**2
```

```
out = square(2)
```

```
 print(out)
```

```
4
```

Expressões lambda

Uma *expressão lambda* permite escrever funções anônimas/sem-nome usando apenas uma linha de código

```
def vezes2(var):
    return var*2
```

```
vezes2(2)
```

```
4
```

```
(lambda var: var*2)(4)
```

```
8
```

```
(lambda x,y: x + y)(3,4)
```

```
7
```

```
soma = lambda x,y: x + y
soma(1,3)
```

```
4
```

Map

Aplicando uma função *func* a uma seqüência *seq*, o resultado é sempre uma lista cujo os elementos são obtidos aplicando-se individualmente cada elemento de seq a função func.

```
list(map(abs, [-1, -2, -3]))
```

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

```
list(map(str, [2, 4, 6]))
```

```
['2', '4', '6']
```

```
list(map(hex, (10, 11, 12)))
```

```
['0xa', '0xb', '0xc']
```

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

```
list(map(lambda var: var ** 2,seq))
```

```
[1, 4, 9, 16, 25]
```

```
list(map(lambda x, y: x*y, [1, 3, 5], [2, 4, 6]))
```

```
[2, 12, 30]
```

```
list(map(lambda a, b, x: a*x+b, [1, 3, 5], [2, 4, 8], [0, 0, 0]))
```

```
[2, 4, 8]
```

```
list(map(pow, [2, 4, 6], [0, 1, 2]))
```

```
[1, 4, 36]
```

Reduce

A função reduce aplica acumuladamente os ítens de uma seqüência de entrada seq (da esquerda para a direita) a uma função func de dois parâmetros até reduzir esse cálculo a um único valor de resposta. Opcionalmente pode-se atribuir um valor inicial como parâmetro

```
from functools import reduce
reduce(lambda x, y: x+y, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
55
```

```
reduce(lambda x, y: x+y, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 100)
```

```
155
```

```
reduce(lambda x, y: x * y, seq)
```

```
120
```

Filter

A função *filter* retorna uma seqüência *seq* cujos valores são os elementos da seqüência de entrada *seq* que respeitam determinado critério. Caso uma função *func* esteja definida, o valor de retorno da função é utilizado como valor verdade e apenas esses elementos vão fazer parte da seqüência de retorno.

```
list(filter(lambda item: item % 2 == 0,seq))
```

```
[2, 4]
```

```
list(filter(lambda x: x > 3, [0, 1, 2, 3, 4, 5]))
```

```
[4, 5]
```

```
list(filter(lambda s: s > 'm', 'python r0cks!'))
```

```
['p', 'y', 't', 'o', 'n', 'r', 's']
```

Métodos

```
st = 'Olá, me chamo Lucília'
```

```
st.lower()
```

```
'olá, me chamo lucília'
```

```
st.upper()
```

```
'OLÁ, ME CHAMO LUCÍLIA'
```

```
st.split()
```

```
['Olá,', 'me', 'chamo', 'Lucília']
```

```
tweet = 'Dale Ners! #Nerds'
```

```
tweet.split('#')
```

```
['Dale Ners! ', 'Nerds']
```

```
tweet.split('#')[1]
```

```
'Nerds'
```

```
d
```

```
{'nome': 'Lucília',
 'idade': 51,
 'hobby': 'codar',
 'filhos': ['Bruna', 'André']}
```

```
d.keys()
```

```
dict_keys(['nome', 'idade', 'hobby', 'filhos'])
```

```
d.items()
```

```
dict_items([('nome', 'Lucília'), ('idade', 51), ('hobby', 'codar'), ('filhos',
['Bruna', 'André'])])
```

```
lst = [1,2,3]
```

```
lst.pop()
```

```
3
```

```
lst
```

```
[1, 2]
```

```
'x' in [1,2,3]
```

```
False
```

```
'x' in ['x','y','z']
```

```
True
```

谢Perguntas?

professora@lucilia.com.br

Créditos

- Template: SlidesCarnival
- Ilustração de fundo: Alex Monge
- PythonBrasil
- Nerdologia