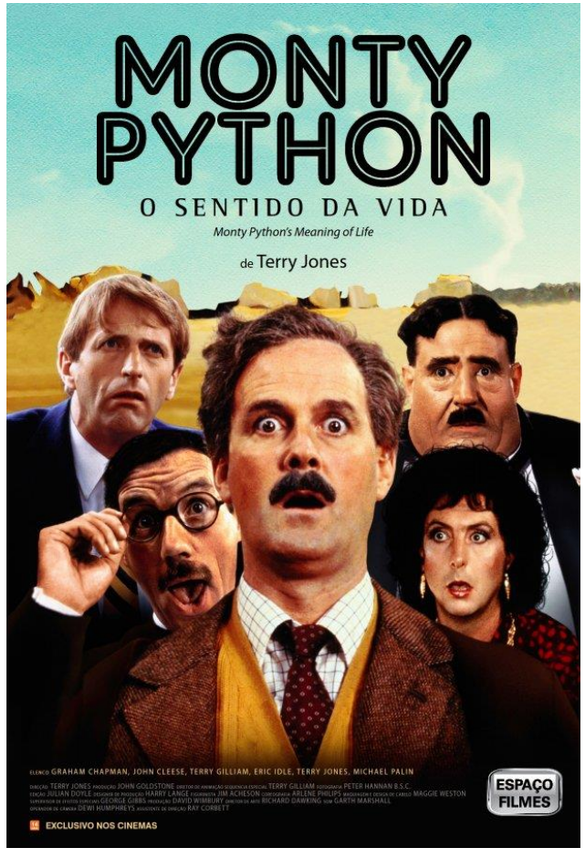
Introdução a linguagem Python

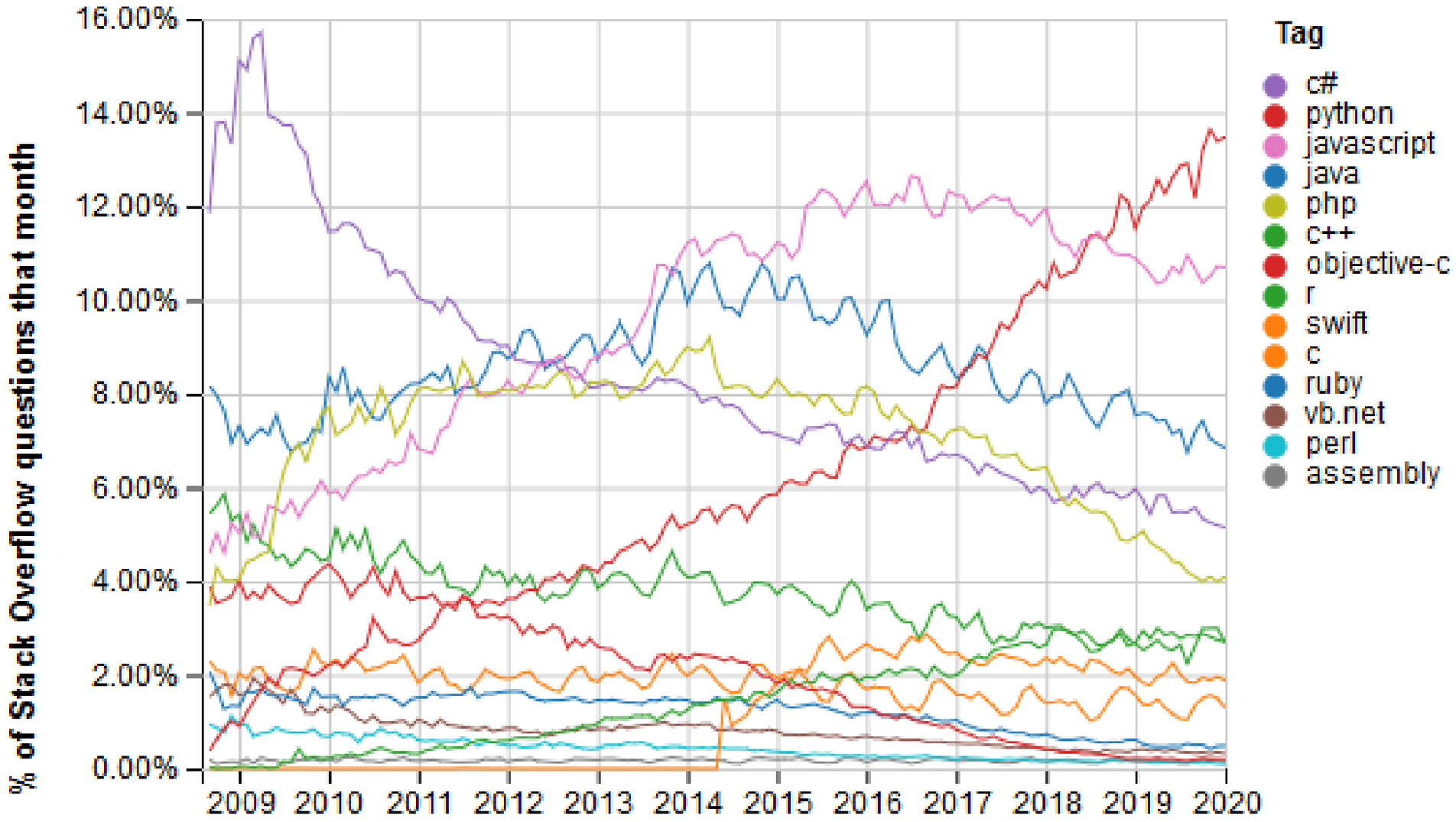
Pythonidae (Python family)

?



Professor: Alex Pereira

Porcentagem de Questões Por Mês / Linguagem



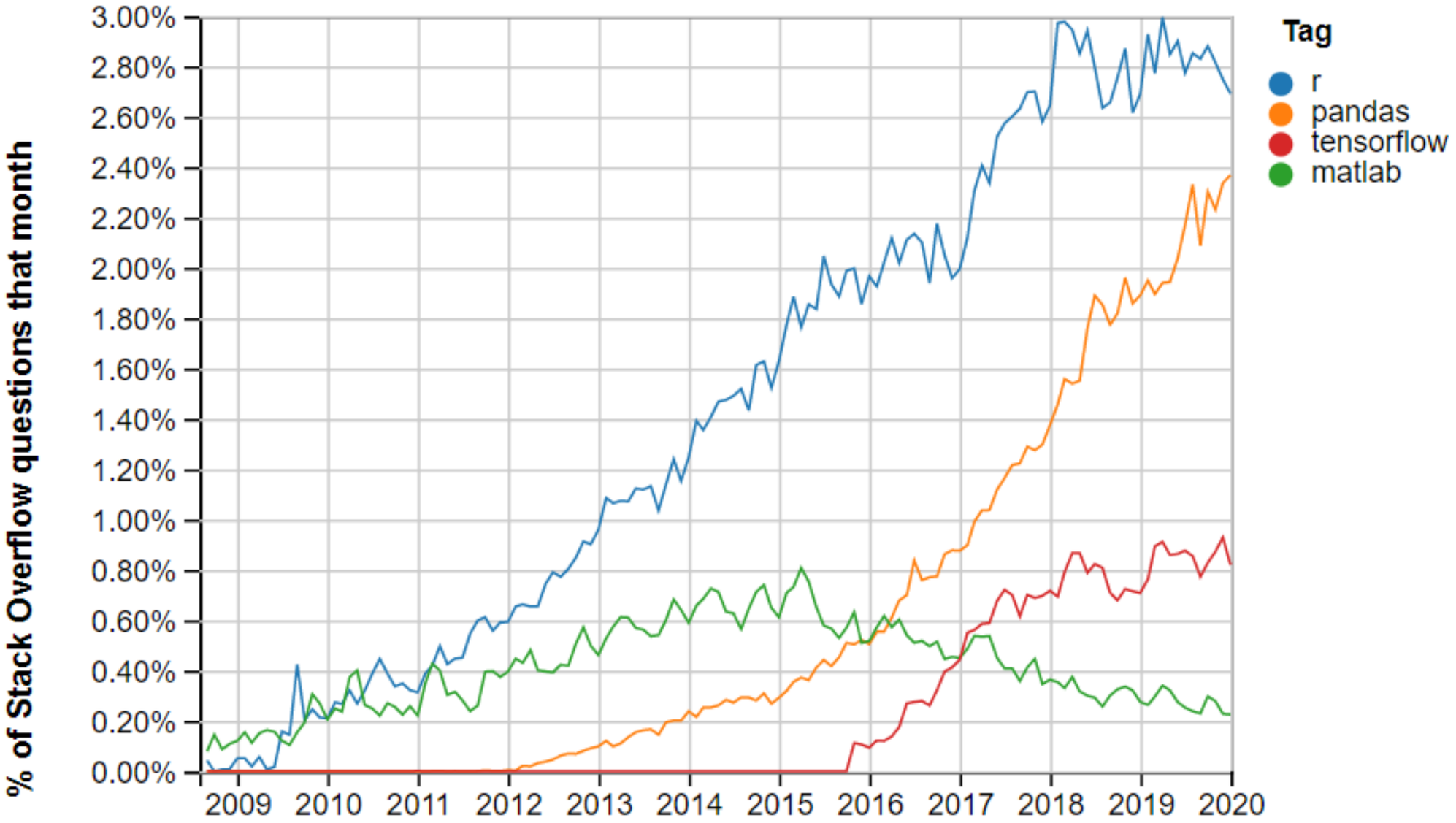
Fonte: [Stackoverflow]

TIOBE: Índice de Popularidade das linguagens

- Baseado na frequência de buscas
  - isso pode refletir o número de engenheiros hábeis, cursos e vagas de emprego no mundo todo
- Fontes de informação
  - Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube e Baidu

Feb 2020	Feb 2019	Change	Programming Language	Ratings	Change
1	1		Java	17.358%	+1.48%
2	2		C	16.766%	+4.34%
3	3		Python	9.345%	+1.77%
4	4		C++	6.164%	-1.28%

Python/Pandas vs R



Vantagens do Python

- Extenso suporte de bibliotecas
  - NumPy, Pandas, Flask e etc.
- Open source e desenvolvido em comunidade
- Fácil de aprender
- Estruturas de dados amigáveis
- Linguagem tipada dinamicamente
  - não é necessário definir previamente o tipo de dados com base no valor atribuído

Vantagens do Python

- Linguagem orientada a objetos
- Interativo
- Portável em diversos sistemas operacionais
- O código Python é significativamente menor que o código C ++ / Java equivalente.
  - Isso implica que há menos para digitar, depurar e manter.

Desvantagens do Python

- Velocidade / Desempenho computacional
  - Se precisar de alto desempenho, dê preferência para C ou C++.
    - ✓ Velocidade de execução pode não ser tão importante quanto velocidade de desenvolvimento*.
- Consumo de memória:
  - Python consome mais memória do que outras linguagens como C ou C++.
- Desenvolvimento Mobile, aplicativos embarcados e IoT:
  - Java e outras linguagens oferecem mais suporte para desenvolvimento de aplicativos de celular e IoT.

Capacitando-se em TI com eficiência

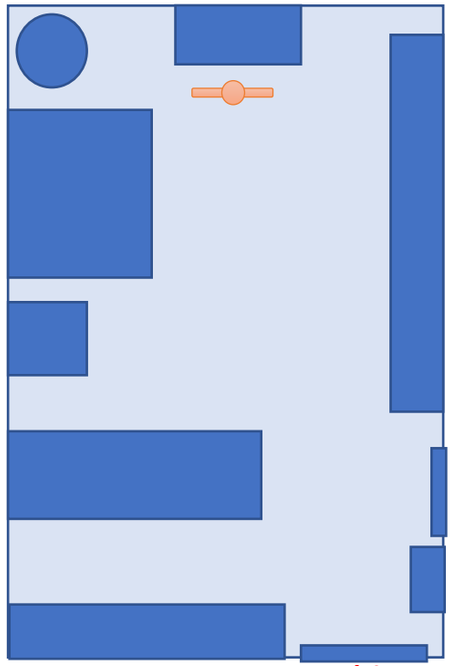
- Trade-off capacitação prévia vs Busca de informação sob demanda
  - Afiar o machado
- Como eu programo:
  - interativamente, sem focar em memorizar sintaxe/nomes,
    - ✓ e sabendo perguntar/pesquisar.
- Buscas e Stackoverflow
- Comece com pequenas iterações
- Participação ativa

Icebreaker (sleep breaker)

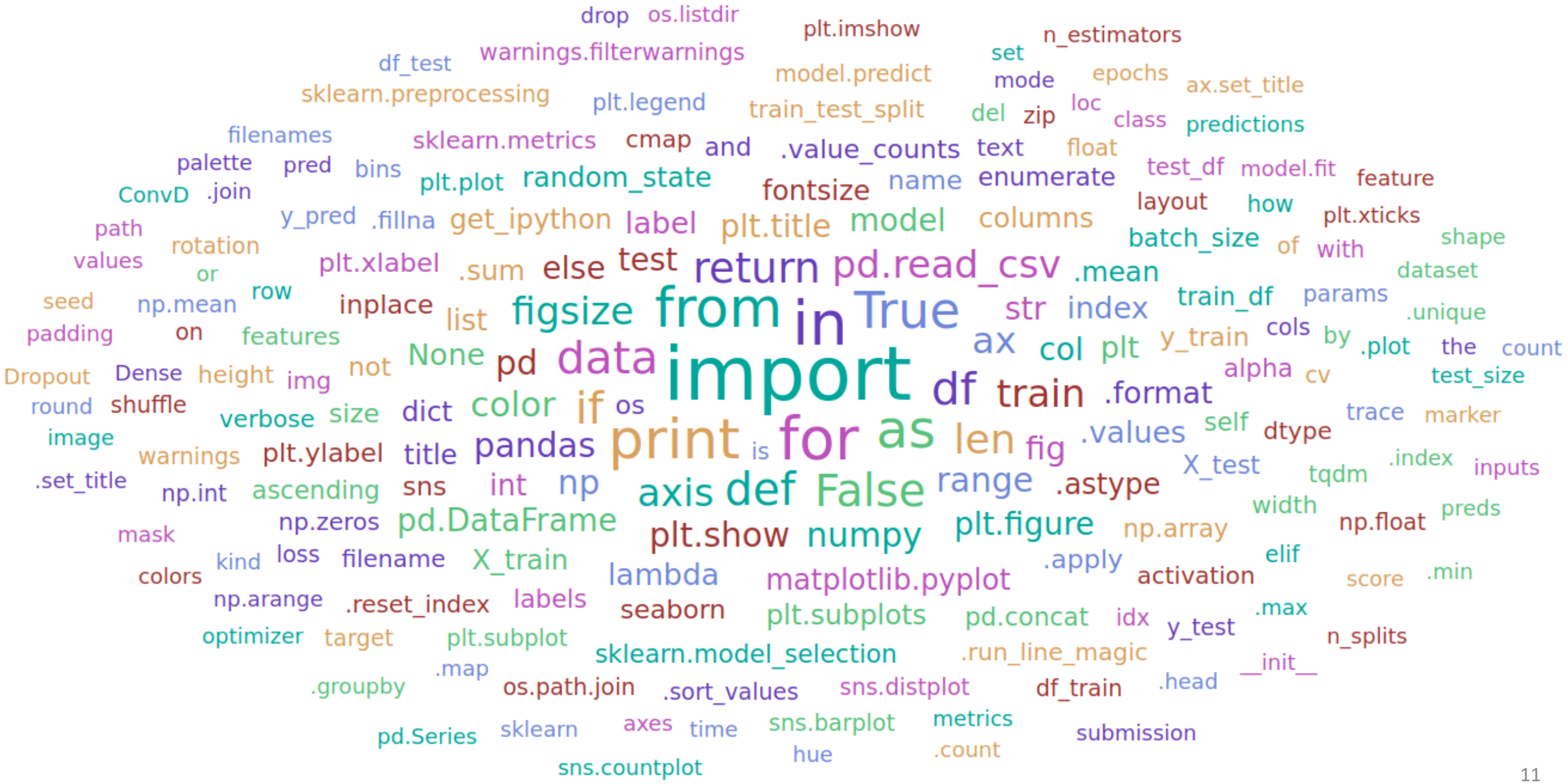
- A turma desenvolve um algoritmo para levar o professor para fora da sala;
- O professor é o interpretador (lê as instruções e as executa)
- A turma define a sintaxe (instruções)
  - Pode-se usar, por exemplo, instruções como:
    - ✓ Dar um passo (step), virar (turn), obstáculo (retorna True ou False);
  - Pode-se usar estruturas de programação, por exemplo:
    - ✓ While, if, else.
- Fazer o exercício em 3 iterações

Icebreaker (sleep breaker)

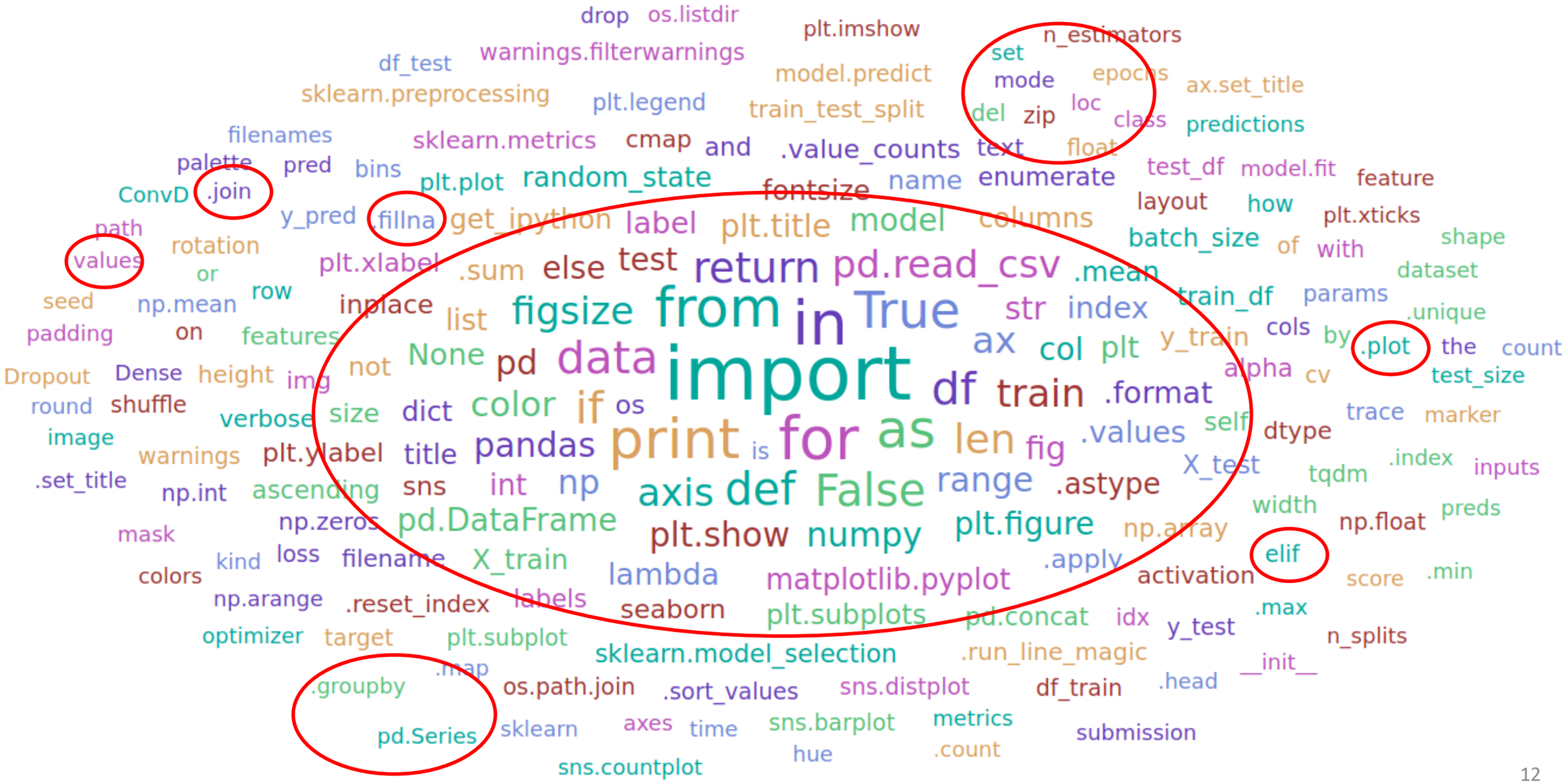
- Fazer o exercício em 3 iterações
  - 1ª: Criar algoritmo e criar novas instruções (se necessário);
    - ✓ 2 min
  - 2ª: Refazer algoritmo com novas instruções (de toda a turma) e testar; e
    - ✓ 3 min
  - 3ª: Refazer algoritmo e testar.
    - ✓ 2 min



Nuvem de Palavras de Arquivos Python do Kaggle

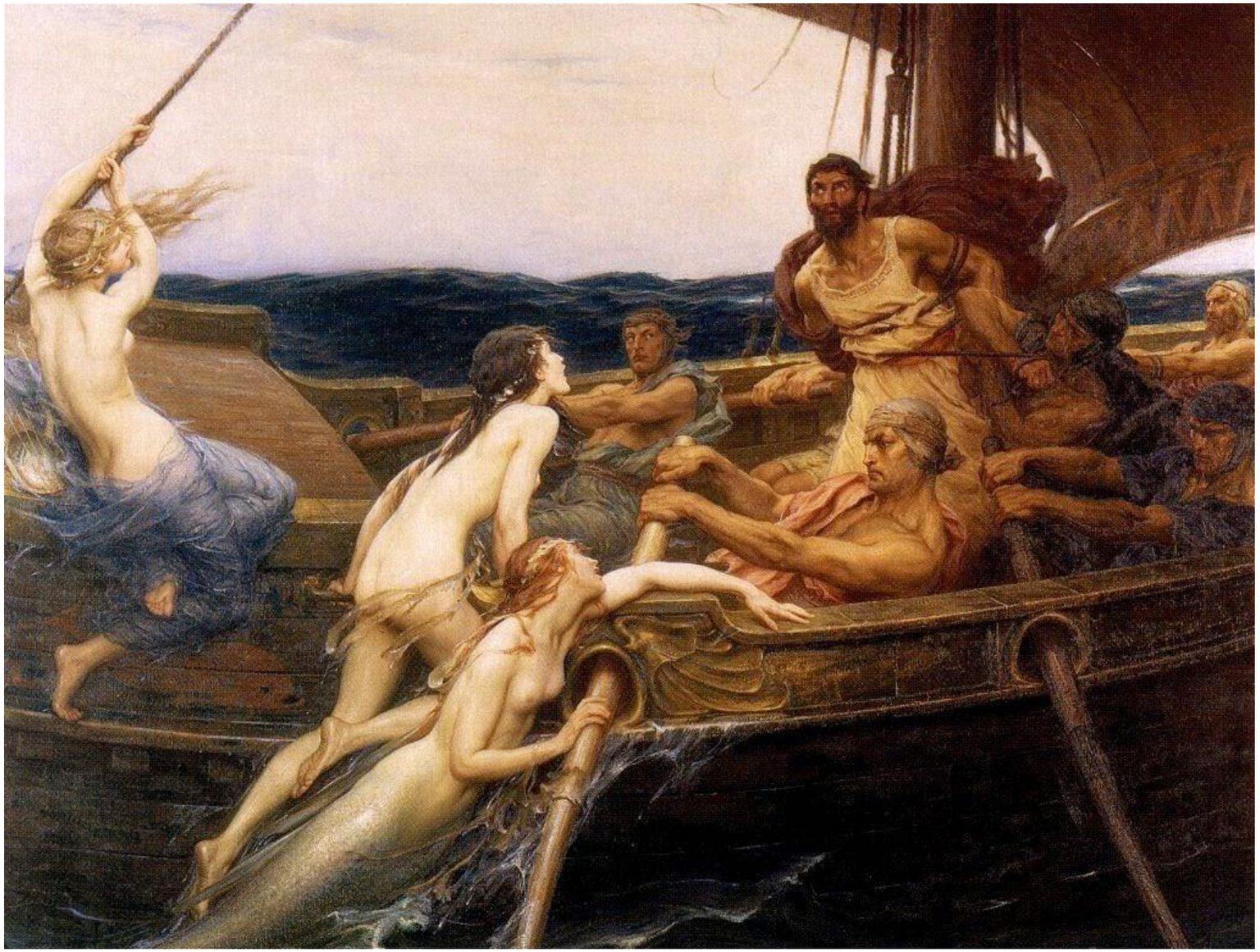


Nuvem de Palavras de Arquivos Python do Kaggle



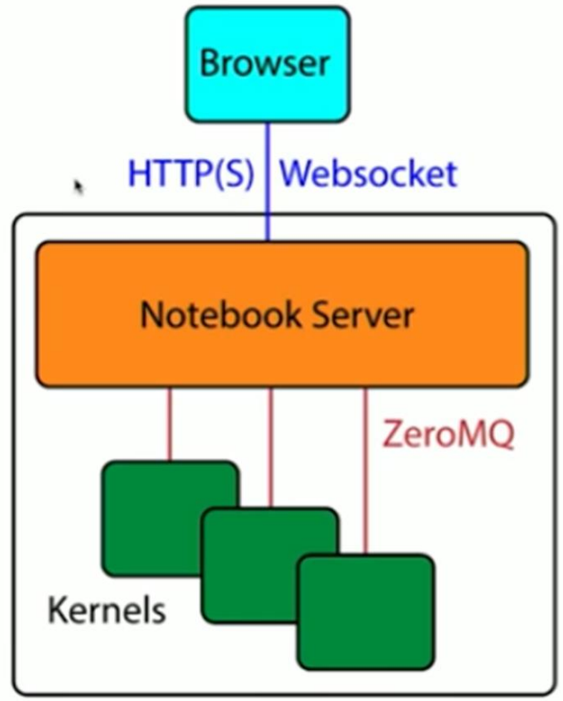
Concentração na Aula – Durante Aula Prática

Concentração na Aula – Ulisses e o canto das sereias

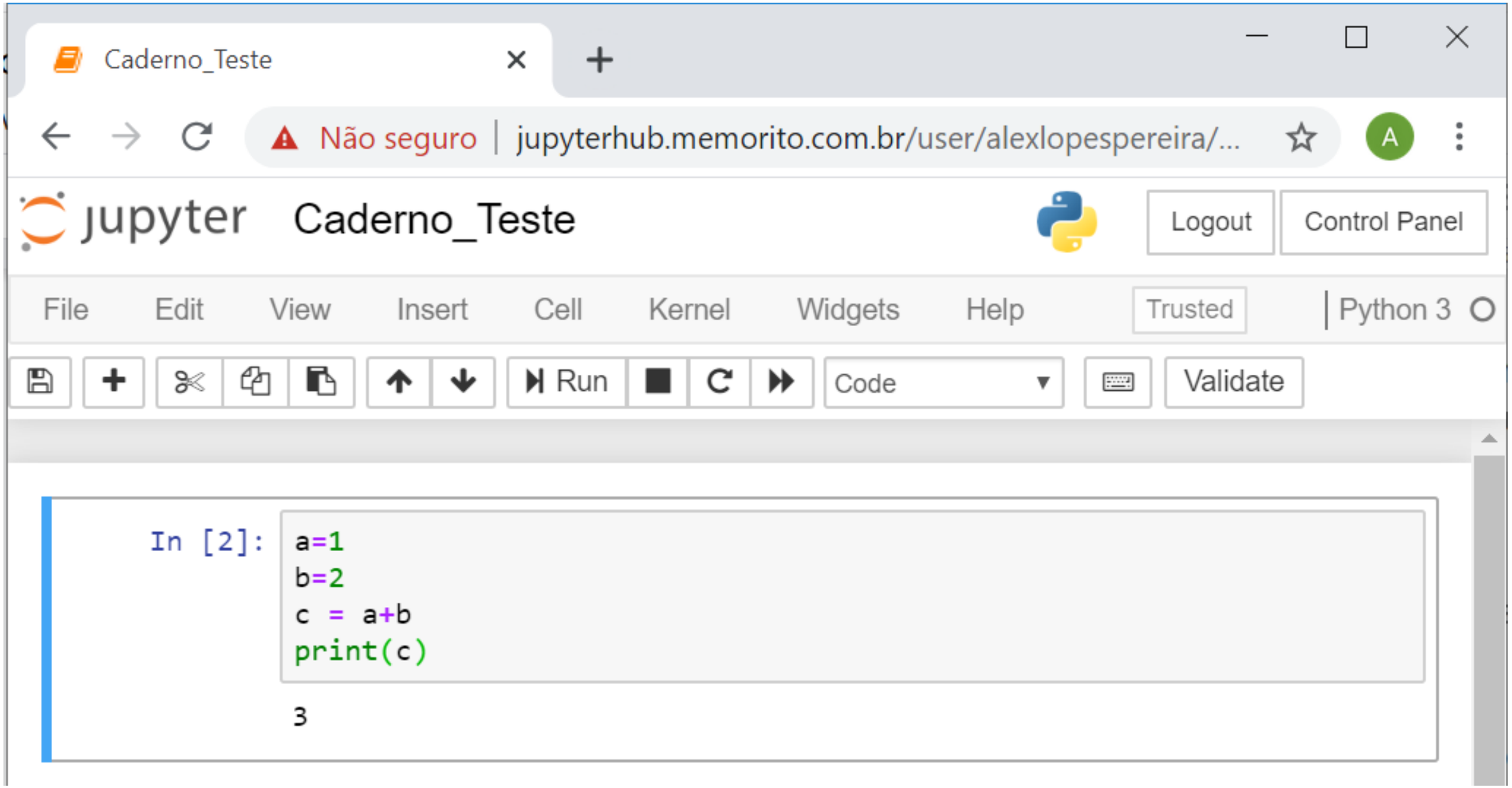


Introdução ao Jupyter Notebook

- O que é
  - Ferramenta de programação no navegador;
  - Códigos, instruções e resultados são mostrados juntos;
  - Útil para escrever códigos que contam uma história; e
  - Utilizado por estudantes, cientistas e pesquisadores.
- Como é implementado
  - É um servidor web local.
  - Abre uma página no navegador.
  - Suporta diversas linguagens de programação
    - ✓ Entre elas o Python.
- Boa prática de programação no Jupyter notebook
  - Programar iterativamente: validando cada pequeno resultado

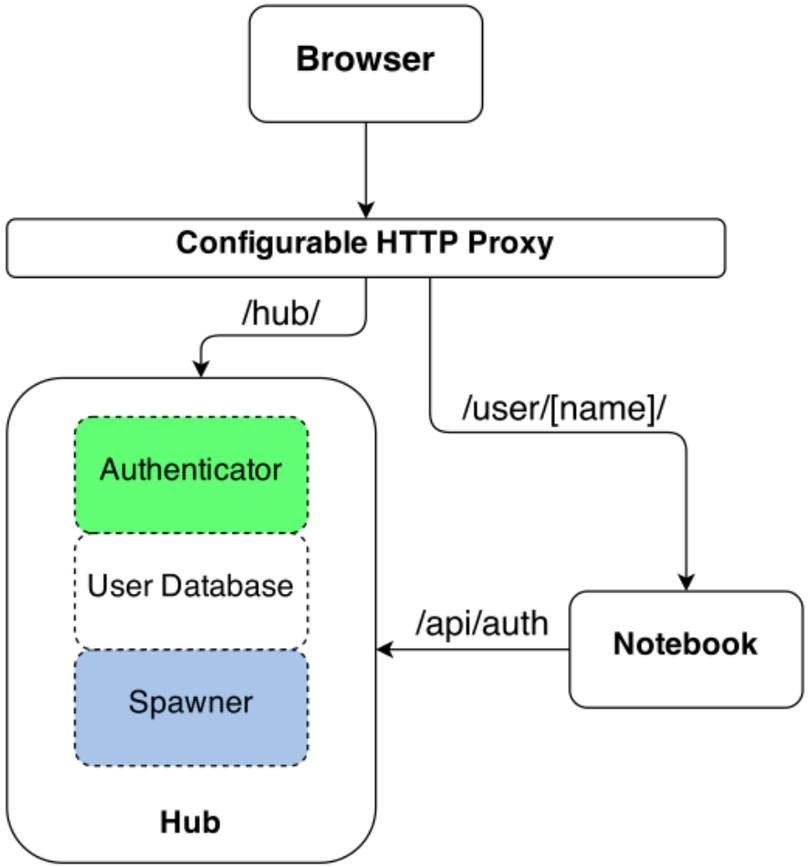


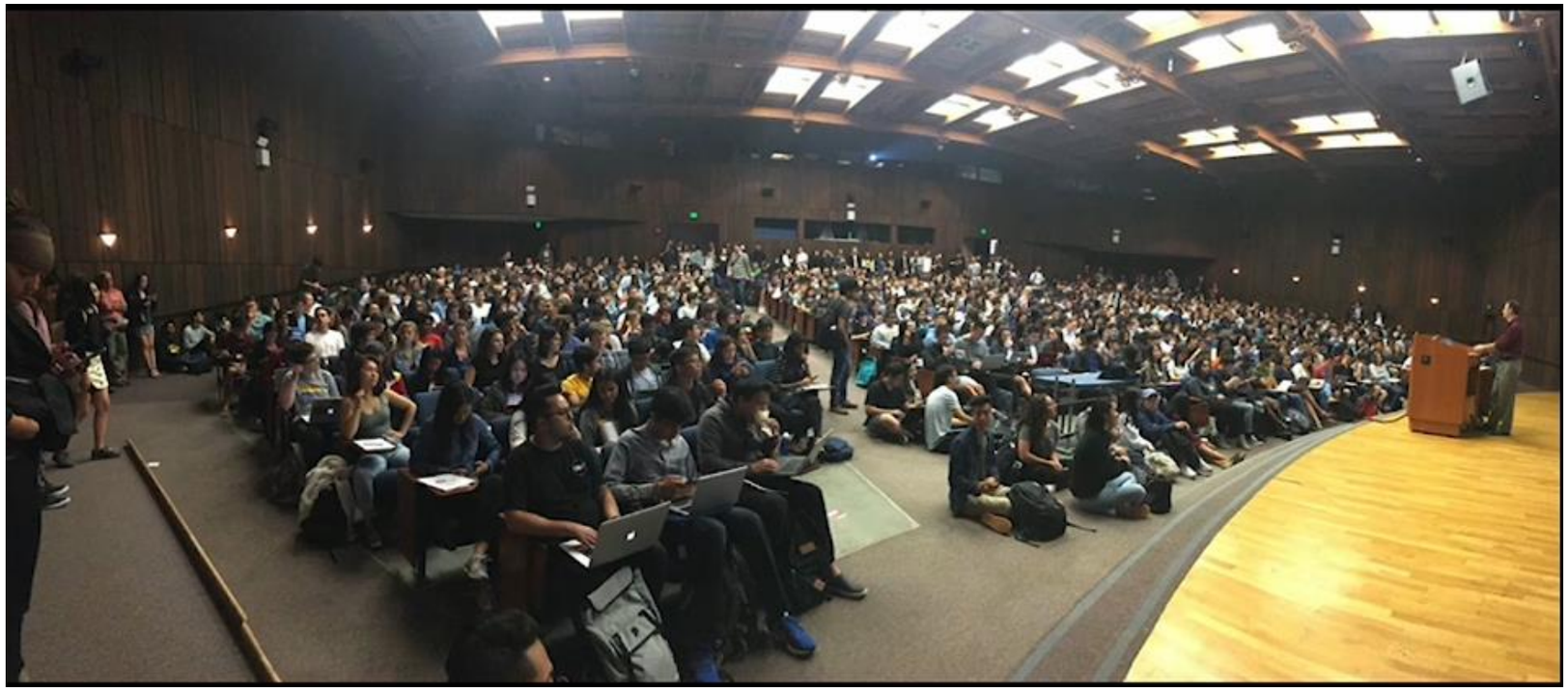
Jupyter Notebook



Jupyter Hub

- Ferramenta para criar e gerenciar
  - Múltiplas instâncias de Jupyter Notebooks na nuvem





University of California, Berkeley: curso usando o Jupyter Hub.
Fonte: https://www.youtube.com/watch?v=ivswAxysfTk

Programming Aptitude Tests

- Fatores avaliados em testes de aptidão para programação
  - Habilidade com lógica,
  - Interpretação de especificações,
  - Atenção a detalhes, e
  - Acurácia e raciocínio com simbolos.
- *The tests assessed the participant's ability to follow a detailed procedural instruction.*
  - Tukiainen, Markku & Mönkkönen, Eero. (2002). Programming aptitude testing as a prediction of learning to program.
  - Wolfe, J. M. (1971) Perspectives on testing for programming aptitude. In the proceedings of the 1971 Annual Conference of the ACM , 268-277.
  - Winrow, B. (1999) The Walden programmer analyst aptitude test. Dr. Dobb's Journal, Fall 1999, Software Careers. (http://www.ddj.com/documents/ s=894/ddj9914b/9914b.htm).

Jupyter Notebook: Como usar?

- **Preste atenção** na demonstração (*live coding*) do professor.
  - Você terá tempo para praticar sozinho.
- Interação básica com o Jupyter Notebook
- Clicar em *Play* ou **tecle SHIFT+ENTER** para executar uma célula
  - Os números entre colchetes indicam a ordem de execução dos comandos.
  - Um asterisco entre colchetes indica que o código está sendo executado.
- Se você **reiniciar** o notebook o conteúdo das variáveis é **perdido**.
- **Leia** as mensagens de **log de erro** (elas são úteis).
- https://jupyter.enap.gov.br

Atalhos de Teclado muito úteis

- ESC
  - Sai do modo de edição e entra no modo de comandos
    - ✓ Pode-se sair do modo de edição clicando fora das células
- SHIFT+ENTER
  - Executa a célula atual e passa o cursor para a próxima célula;
- CTRL+ENTER
  - Executa a célula atual e mantém o cursos na mesma célula;
- B (**B**elow) / A (**A**bove) – no modo de comando
  - Adiciona uma célula abaixo/acima da célula selecionada
- ↓ / ↑
  - Move células para baixo / cima

Jupyter Hub e NBGrader

- Jupyter Hub
  - Login com sua conta do Github
  - 1 container na nuvem por aluno, com
    - ✓ 1 processador
    - ✓ E até 4GB de memória RAM
- NBGrader
  - Ferramenta de automação de correção
    - ✓ De cadernos/notebooks Jupyter
  - Etapas de uso do Jupyter auxiliado pelo NBGrader
    - ✓ Fetch: recebe um notebook publicado;
    - ✓ Submit: submete notebook para correção/revisão; e
    - ✓ Fetch Feedback: recebe feeback de uma submissão (se existir).

Teste com o NBGrader

- 1) Logar com sua conta no JupyterHub
  - Endereço: jupyter.enap.gov.br
- 2) Fetch do Assignment Aula1_Exercicios.ipynb
- 3) Resolução do caderno Aula1_Exercicios.ipynb
- 4) Submit do caderno Aula1_Exercicios.ipynb
- 5) Fetch feedback do Assignment Aula1_Exercicios.ipynb

Sintaxe do Python

- Dois pontos sinaliza o início de uma sentença composta
  - O conteúdo das sentenças compostas é aninhado com espaços/tabs
    - ✓ Sem chaves
- Ponto e virgula para finalizar uma sentença é opcional

```
x = 2
if x > 0:
      # adiciona 1 a x
      x = x + 1;
      print(x)
      print('x maior que zero')
else:
      print('x menor ou igual a zero')
```

```
3
x maior que zero
```

Sintaxe do Python

- Referências para objetos não possuem um tipo associado
  - Não há problema em reatribuir uma referência a outro tipo
    - ✓ Por exemplo:

```
x = 2
print(type(x))
x = 'carro'
print(type(x))
```

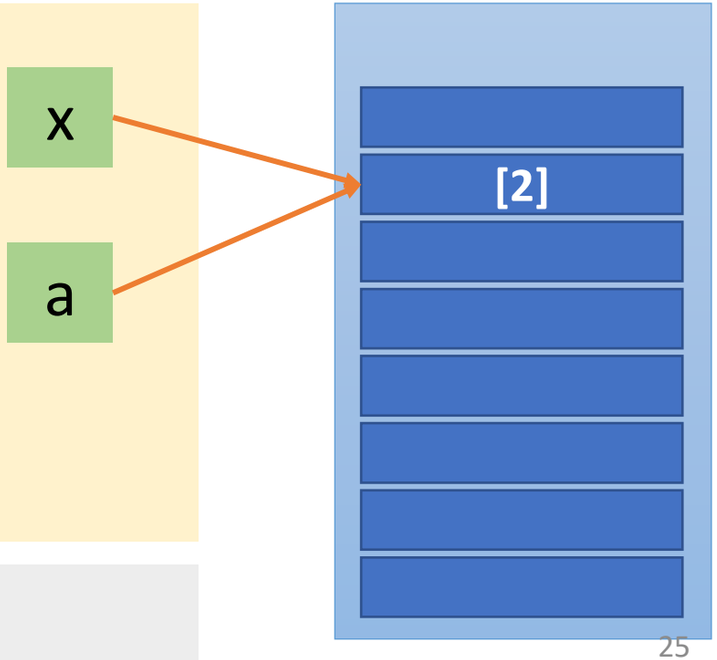
```
<class 'int'>
<class 'str'>
```

Sintaxe do Python

- Funções são declaradas com a palavra-chave
  - def
- Argumentos são passados por referência, e não por valor
  - Na chamada de funções

```
def insert1(a):
      a.append(1)
b = [2]
insert_1(b)
print(b)
```

```
[2, 1]
```



Dados do tipo Escalar

- Guardam valores únicos (single) ou simples
- None
  - A representação de valor nulo em Python
- str
  - Conjunto de caracteres. Guarda strings codificadas com UTF-8
- bytes
  - bytes ASCII (ou bytes codificados como Unicode)
- float
  - Número em formato ponto flutuante de 64-bit (Exemplo: 3.1290)
- bool
  - Um valor True ou False
- int
  - Um número inteiro

Formando uma string (texto)

- Strings podem ser definidas usando aspas simples ou duplas
  - `'aluno'` ou `“aluno”` são válidos

```
a = 4.5
b = 2
frase = 'a={0}, b={1}'.format(a, b)
print(frase)
```

```
a=4.5, b=2
```

Funções Populares de String

- join
  - cria uma string concatenando uma lista de objetos com um serparador (vírgula, por exemplo)
- strip
  - remove caracteres (em branco*) do início e do fim de uma string
- upper / lower
  - Transforma os caracteres e maiúsculos / minúsculos
- replace
  - substitui todas as ocorrências de uma string por outra
- split
  - divide uma string em várias outras. O ponto de divisão é o local do separador especificado

Imports

- Um módulo é um arquivo de extensão .py
  - Útil para reusabilidade de código

```
# modulo_exemplo.py
def insert1(a):
     a.append(1)
```

```
from modulos.modulo_exemplo import insert1
b = [2]
insert1(b)
print(b)
```

```
[2, 1]
```

Operadores Binários

Operação	Descrição
a + b	**Soma a e b**
a - b	**Subtrai a de b**
a * b	**Multiplica a por b**
a / b	**Divide a por b**
a // b	Divisão inteira de a por b. Desconsidera o resto.
a ** b	Eleva a à potência b
a & b	**True se a e b são True**
a | b	**True se a ou b são True**

*** Em negrito os mais importantes**

Operadores Binários

Operação	Descrição
a == b	**`True` se `a` é igual a `b`**
a != b	**`True` se `a` não é igual a `b`**
a < b, a <= b	**`True` se `a` é menor (ou menor ou igual) a `b`**
a > b, a >= b	**`True` se `a` é maior (ou maior ou igual) a `b`**
a is b	**`True` se `a` e `b` referenciam o mesmo objeto Python**
a is not b	**`True` se `a` e `b` referenciam objetos Python distintos**
a ^ b	Operação de XOR. True se a e b são True, mas não ambos.

*** Em negrito os mais importantes**

Conversão de Tipos

- `str`, `bool`, `int`, e `float`
  - também são funções para converter valores para esses tipos

```
s = '3.14159'
fval = float(s)
ival = int(fval)
print(fval)
print(ival)
```

```
3.14159
3
```

Laços (loops) do tipo for

- Uma maneira de iterar sobre uma coleção
- Usa-se o keyword **`in`** para referenciar a coleção.
  - **`val`** foi um termo escolhido pelo programador
  - **`break`** interrompe a iteração

```
sequence = [1, 2, 0, 4, 6, 5, 2, 1]
total_until_5 = 0
for val in sequence:
    if val == 5:
        break
    total_until_5 = total_until_5 + val
print(total_until_5)
```

```
13
```

Estruturas de Dados do Python (Nativas)

list (Lista)

- Sequência de tamanho variável e conteúdo mutável (alterável)
  - Assim como as tuplas, pode conter objetos de vários tipos
  - Define-se uma lista com colchetes [ ]
  - append (inserir elementos), pop (remover elementos pelo índice)

```
al = [2, 4, 0, None]
print(al)
al.append(9)
print(al)
al.pop(1) # Remover pelo índice
print(al)
al.remove(0) # Use remove para remover pelo valor
print(al)
```

```
[2, 4, 0, None]
[2, 4, 0, None, 9]
[2, 0, None, 9]
[2, None, 9]
```

Combinando Listas

- Use o operador + ou a função extend
  - A função extend é mais rápida do que o operador +

```
al = [2, 4, 0, None]
al_plus = al + ['a', 'b']
al.extend(['a', 'b'])
comp = al_plus == al
print(comp)
```

```
True
```

Slicing (fatiar)

- Use intervalos entre colchetes para fatiar sequências

```
al = [2, 4, 0, 3, 7, 10, 4, 5]
print(al[0:3])  # De zero a 3
print(al[:4])   # De zero a 4
print(al[2:])   # De 2 até o último
print(al[-1])   # O último element
print(al[::2])  # A cada dois elementos
print(al[::-1]) # Reverter/espelhar os elementos
al[1:2] = [8, 8]
print(al)
```

*dict (Dicionário) *** + IMPORTANTE ****

- Uma coleção de key-value (chave-valor) de tamanho flexível
  - O key e o value podem ser (quase) qualquer objeto python

```
empty_dict = {} # Cria um dicionário vazio
d1 = {'tipo' : 'carro', 'ano':2020, 'ocup' : [1, 2]}
d1['cor'] = 1 # Cria um novo par chave-valor
d1['fabric'] = 3 # Cria um novo par chave-valor
print(d1)
print(d1['ocup'])
```

```
{'ocup': [1, 2], 'ano': 2020, 'fabric': 3, 'tipo': 'carro', 'cor': 1}
[1, 2]
```

*dict (Dicionário) *** + IMPORTANTE ****

- Outros métodos relacionados a dicionários

```
d1 = {'tipo' : 'carro', 'ano':2020, 'ocup' : [1, 2]}
print('tipo' in d1) # Testa: 'tipo' está em d1.keys()?
del d1['ano'] # Remove o par identificado por 'ano'
ret = d1.pop('ocup') # Remove do dict e retorna o valor
print(list(d1.keys()))
d1.update({'motor' : '18c', 'dono' : 12}) #atualiza
print(d1)
```

```
True
['tipo']
{'motor': '18c', 'dono': 12, 'tipo': 'carro'}
```

Prática no Jupyter Notebook (10 min)

- Faça os exercícios da aula (referente ao conteúdo já visto); ou
- Revise o notebook de teoria.

NumPy
(Numerical Python)

Pacote Python para computação numérica

Recursos do Numpy

- Computação eficiente com array multi-dimensional
  - armazena dados numa região contínua de memória;
  - As funções numpy escritas em C podem operar diretamente na memória.
- Funções matemáticas eficientes/rápidas em arrays
  - sem a necessidade de escrever loops (laços)
- Ferramentas para ler e escrever arrays do disco
- Álgebra linear, geração de números aleatórios, entre outros
- API C para integração com Código escrito em C/C++/Fortran

Operações vetorizadas em ndarrays

- Funções e operações em várias dimensões sem loops
  - Mais eficientes (preferíveis)

```
import numpy as np
data = np.random.randn(2, 3) # Gerar dados aleatórios. 2L x 3C
print(data)
print(data * 10)
print(data + data)
```

```
[[ 0.44591129 -0.48318509  0.09357558]
 [ 0.97525852  0.51980318  0.15236606]]
[[ 4.45911285 -4.83185091  0.93575582]
 [ 9.75258516  5.19803182  1.52366062]]
[[ 0.89182257 -0.96637018  0.18715116]
 [ 1.95051703  1.03960636  0.30473212]]
```

Instruções inline (na mesma linha)

```
a = 2

# Sintaxe: valor_se_verdadeiro if condicao else valor_se_falso

b = 0 if a > 10 else 1

print(b)

# [ OPERACAO_VAL for VAL in SEQUENCIA ]

resultado1 = [x for x in range(10)] # list comprehension

resultado2 = [x * x for x in range(10)]

print(resultado1)

print(resultado2)
```

```
1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Comparação de Desempenho: Numpy array vs list

```
# import numpy as np  => Importar se já não tiver importado
my_arr = np.arange(10000000) #cria array de números sequenciais
my_list = list(range(10000000))
# %time mede o tempo tomado pela execução da linha
%time my_arr2 = my_arr * 2
%time my_list2 = [x * 2 for x in my_list]
print(my_arr2[1:5])
print(my_list2[1:5])
```

```
Wall time: 32 ms
Wall time: 1.15 s
[2 4 6 8]
[2, 4, 6, 8]
```

Criando ndarrays

```
data1 = [6, 7.5, 8, 0, 1]

arr1 = np.array(data1)

print(arr1)

data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]

arr2 = np.array(data2)

print(arr2)

print(np.zeros(5)) # Função que cria um array de zeros
```

```
[6.  7.5 8.  0.  1. ]
[[1 2 3 4]
 [5 6 7 8]]
[0. 0. 0. 0. 0.]
```

Atributos do ndarrays (alguns)

```
# O tipo do dado é inferido. Mas também pode ser especificado
arr1 = np.array([6, 7.5, 8, 0, 1])
print(arr1.dtype) # tipo do dado
arr2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]], dtype=np.int32)
print(arr2.shape) # Forma/dimensões do array
print(arr2.ndim) # Quantidade de dimensões
```

```
float64
(2L, 4L)
2
```

Numpy data types (tipos de dados)

Tipo básico	Tipo Numpy disponív.	Comentários
Boolean	bool	Tamanho de 1 byte
Integer	int8, int16, int32, int64, int128, int	int tem o tamanho do int padrão da plataforma C
Unsigned Integer	uint8, uint16, uint32, uint64, uint128, uint	uint tem o tamanho do uint padrão da plataforma C
Float	float32, float64, float, longfloat	Float é sempre de precisão dupla (64 bits). longfloat é um float maior cujo tamanho depende da plataforma.
Complex	complex64, complex128, complex	A parte real e a imaginaria de um complex64 ocupam cada um 32 bits
String	str, Unicode	Unicode é sempre UTF32
Object	Object	Representa itens em um array de objetos Python

Conversão de tipos (cast)

```
num_str = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
arr_float = num_str.astype(float) # Converte para float
print(arr_float)
arr1 = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
arr1_int = arr1.astype(np.int32) # Converte para int32
print(arr1_int)
```

```
[ 1.25 -9.6  42. ]
[ 3 -1 -2  0 12 10]
```

Aritmética com NumPy Arrays

```
arr = np.array([1., 2., 3.])
arr2 = np.array([4., 5., 6.])
print(arr * arr)
print(arr – arr)
print(1 / arr)
print(arr ** 0.5)
print(arr2 > arr)
```

```
[1. 4. 9.]
[0. 0. 0.]
[1.      0.5     0.33333333]
[1.      1.41421356 1.73205081]
[ True  True  True]
```

Slicing (fatiar) NumPy Arrays

- Nas listas os slices são cópias;
- Nos arrays numpy os slices são views (visualizações)
  - para copiar, use a função: .copy() , exemplo: arr[1:4].copy()

```
arr = np.arange(10)
print(arr)
arr_slice = arr[5:8]
arr_slice[1] = 12345
print(arr)
li2 = list(range(10)) # O equivalente com uma lista
list_slice = li2[5:8]
list_slice[1] = 12345
print(li2)
```

Slicing (fatiar) de arrays bidimensionais

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d.shape)
print(arr2d[2])
print(arr2d[0][2]) # Tanto faz
print(arr2d[0, 2]) # Tanto faz
```

```
(3L, 3L)
[7 8 9]
3
3
```