

FIAP GRADUAÇÃO

ENGENHARIA DE COMPUTAÇÃO

Inteligência Artificial e Computacional

PROF. ANTONIO SELVATICI

SHORT BIO



É engenheiro eletrônico formado pelo Instituto Tecnológico de Aeronáutica (ITA), com mestrado e doutorado pela Escola Politécnica (USP), e passagem pela Georgia Institute of Technology em Atlanta (EUA). Desde 2002, atua na indústria em projetos nas áreas de robótica, visão computacional e internet das coisas, aliando teoria e prática no desenvolvimento de soluções baseadas em Machine Learning, processamento paralelo e modelos probabilísticos. Desenvolveu projetos para Avibrás, IPT, CESP e Systax.

PROF. ANTONIO SELVATICI

profantonio.selvatici@fiap.com.br

2. MACHINE LEARNING

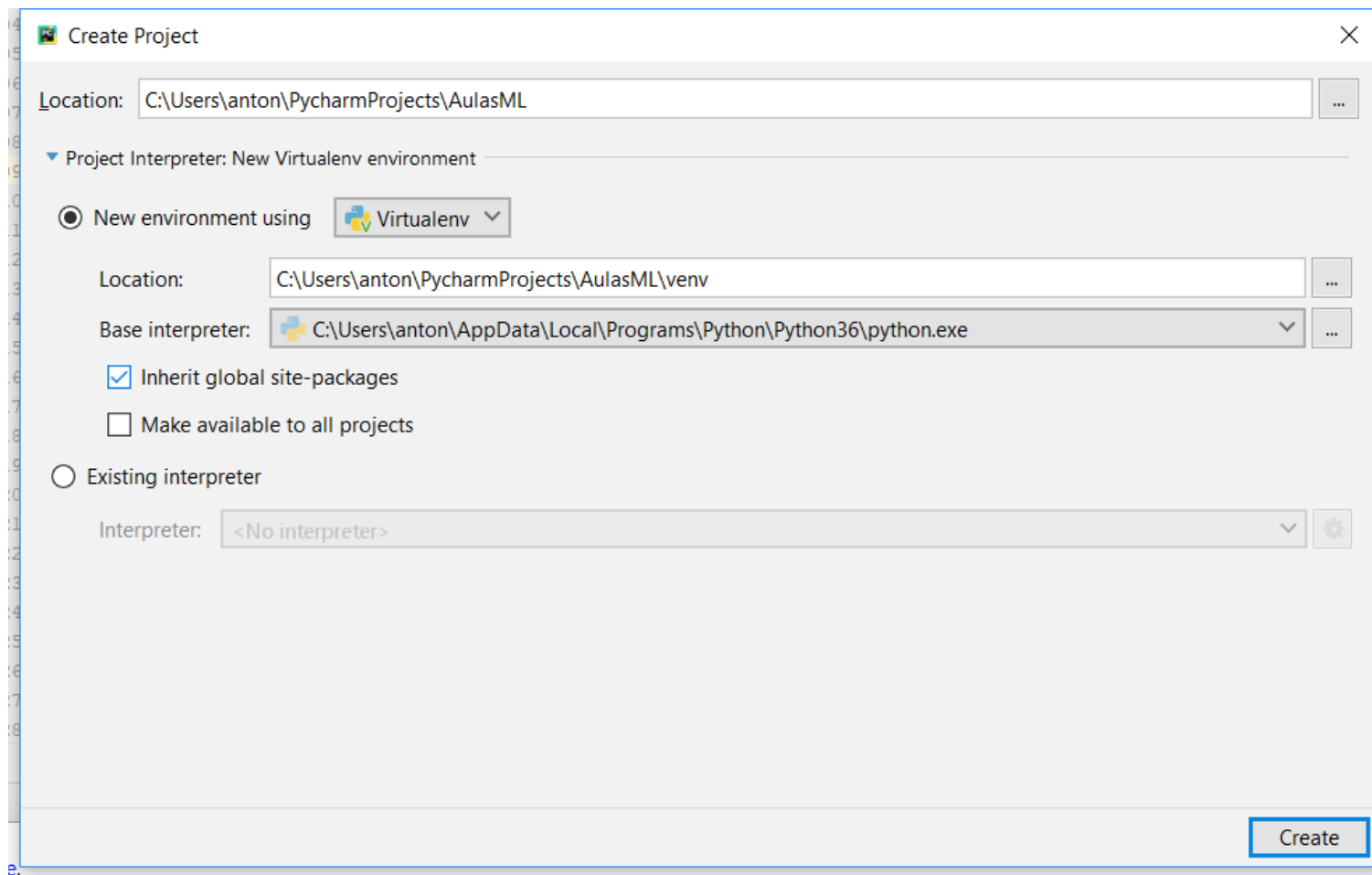
Ambiente para classificação de padrões

- As técnicas de Machine Learning que estudaremos serão executadas em Python, uma linguagem de programação interpretada, não tipada, possuindo um ambiente de execução de comandos interativo
- Sendo um projeto open source, Python possui uma grande quantidade de pacotes contribuídos pela comunidade para manipulação, análise e visualização de dados.
- Há diversas interfaces para programação e interação com programas Python:
 - PyCharm IDE
 - Spyder IDE
 - Rodeo IDE
 - IPython e Jupyter Notebook

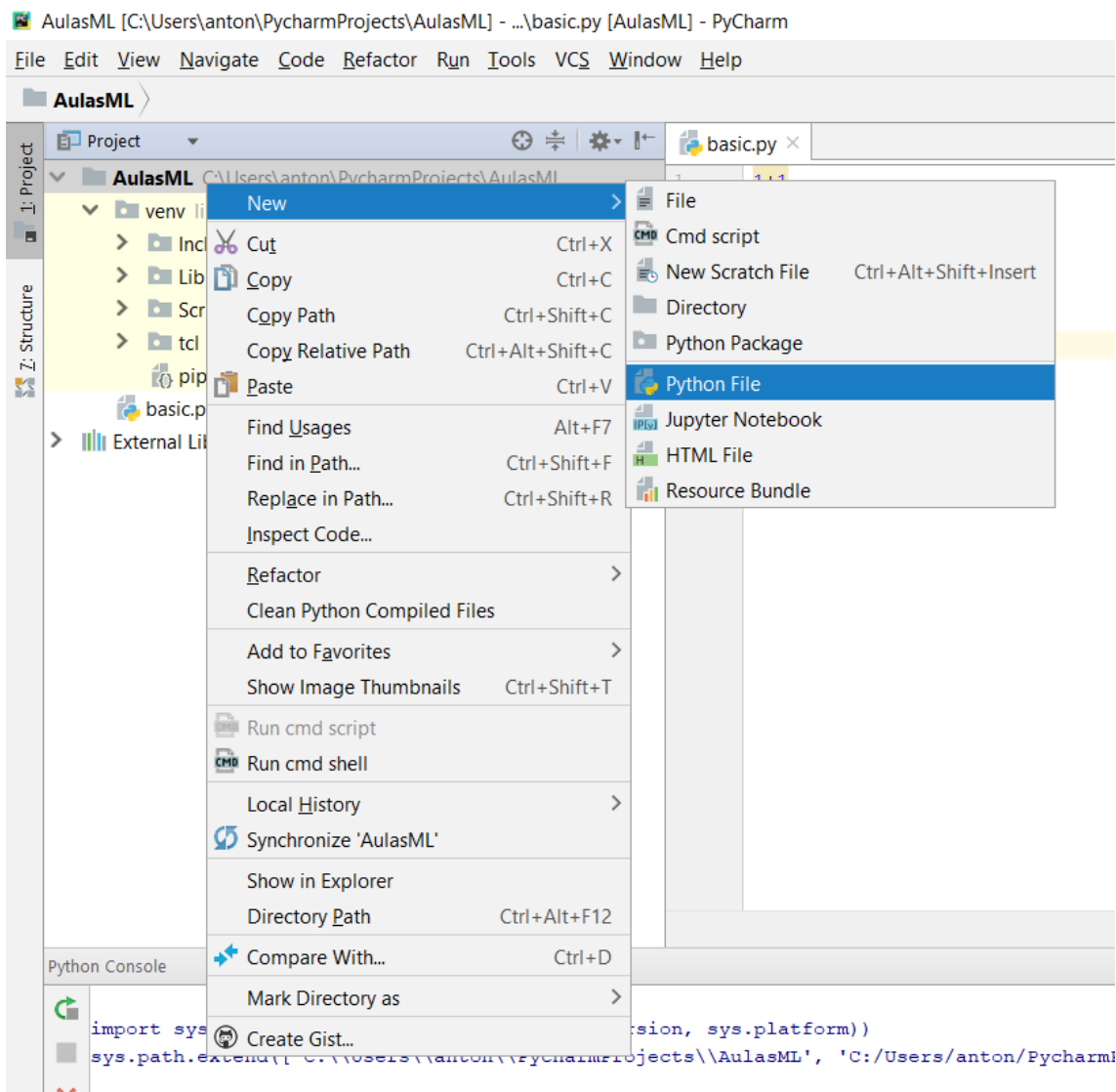
A IDE PyCharm

- É uma das mais populares para o desenvolvimento em Python
- Possui recursos para interagir com o console Python e com o kernel do IPython
- Possui recursos avançados de auto completção de código e inferência dos tipos de dados aceitos em cada função, uma vez que os tipos não são declarados

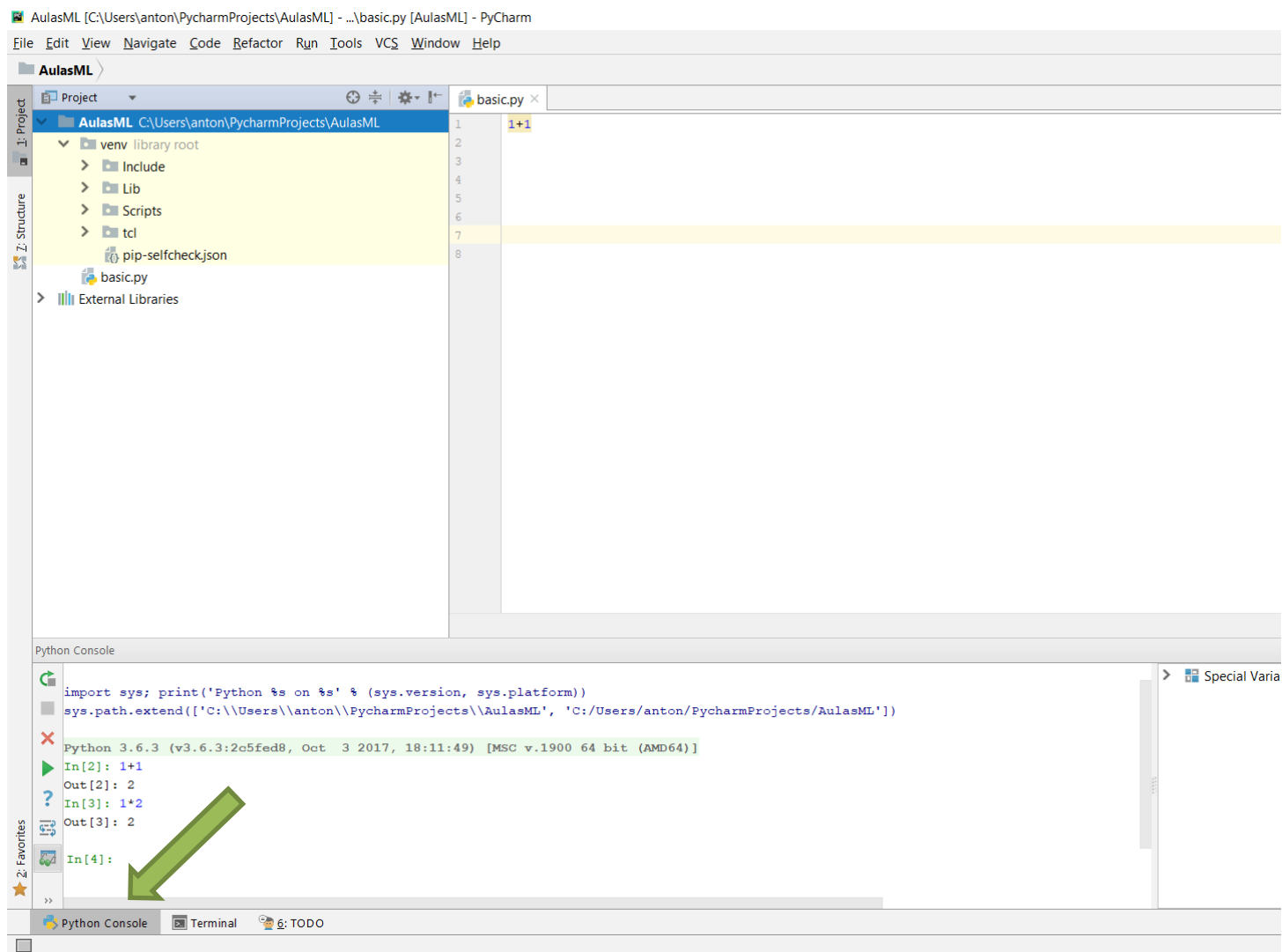
Criando um novo projeto no PyCharm



Criando um novo arquivo Python



Tela após criar um novo arquivo e ativar o console



■ Hello World!

- Executar um comando no PyCharm pela primeira vez
 - Digitar: `print("Hello World!")`
 - Pressionar `Alt + Shift + E`
- Dependendo da expressão, o sistema pode responder através de output no próprio console ou através de uma janela gráfica
- O ambiente interativo funciona como uma calculadora. Vamos executar algumas contas:
 - Ex: `5+3`; `9/2`; `4.24*3.13`; `8**3`
 - Expressões separas por ponto-e-vírgula são executadas em sequência, porém é recomendado escrever uma por linha
- Além de números e string, temos os valores lógicos `True` e `False`, além do valor nulo `None`
- Para saber o tipo de um valor em Python, usamos a função `type()`
 - Ex: `type(1)`; `type(2.3)`; `type('ABC')`;
`type(None)`; `type(True)`
- Operadores binários: `+`, `-`, `*`, `/`, `%`, `**` (exponenciação)
- Operadores de comparação: `>`, `>=`, `<`, `<=`, `==`, `!=`, `<>`

Diferentes versões

- A linguagem Python foi bastante modificada quando do lançamento da versão 3, de forma que as versões 2 e 3 não são totalmente compatíveis entre si
 - O desenvolvimento e manutenção da versão 2 continua a fim de suportar aplicações legadas, porém novos projetos devem usar a versão 3

Variáveis em Python

- Em Python, todas as variáveis são consideradas objetos, registrando em si mesma o tipo do valor, de forma similar às variáveis das diferentes linguagens de programação interpretadas
 - Não-tipadas: o tipo da variável não precisa ser declarado e é definido na atribuição de valores
 - A variável é sempre uma referência para um objeto na memória (lembrando que tudo é objeto)
 - O operador de atribuição é "="
 - Variáveis são sensíveis à caixa e não podem iniciar por números nem conter caracteres especiais
 - O padrão `__<nome-variavel>__` (com dois _ antes e depois) deve ser reservado apenas para variáveis internas do Python
- Executar:
 - `x`
 - `x = 5+3`
 - `y2 = x - 3`
 - `x + y2`

Estruturas de dados fundamentais

- Em Python podemos armazenar conjuntos de dados em listas, tuplas e dicionários
- Uma **lista** é uma sequência imutável de valores separados por vírgulas dentro de colchetes
 - `lista1 = [1, 2.0, False]`
 - `outra_lista = ['3', 2.0, 1]`
 - `lista_simples = ['A']`
- Os valores podem ser de tipos distintos, incluído ainda outras estruturas complexas
- Para saber o comprimento da lista: `len(lista)`
- Acessando elementos singulares de uma tupla através de índices inteiros:
 - `dois_ponto_zero = outra_lista[1]` # começa em 0
 - `um = outra_lista[-1]` # conta a partir do último
- Acessando sub-listas de elementos a partir de slices
 - Notação: `lista[start : stop : step]`
 - `start`: índice de início (inclusive); 0 por omissão
 - `stop`: índice de parada (exclusive); `len(lista)` por omissão
 - `step`: opcional, incremento do índice
 - Exemplos:
 - `um_dois = lista1[0:2]` # tomas os índices 0 e 1
 - `lista_reversa = lista1[::-1]` # inverte a lista
 - `um_tres = outra_lista[::2]` # toma só os índices pares

Estruturas de dados fundamentais

- Além da lista, a **tupla** também é uma sequência de valores separados por vírgulas, porém não precisam de delimitadores.
 - Opcionalmente, os elementos ficam dentro de parênteses
 - Exemplos:
 - `tupla1 = 1, 2.0, False`
 - `outra_tupla = ('3', 2.0, 1)`
 - `tupla_simples = ('A',)` # necessario ter a vírgula no final
- Para saber o comprimento da tupla: `len(tupla)`
- Os elementos da tupla usam a mesma notação de índice e slices que a lista
- A diferença entre tuplas e listas é que as primeiras são imutáveis

Dicionários

- Dicionários são sequências de itens rotulados
 - Armazenam pares de chave-valor
 - Cada chave é separada de seu valor por dois pontos
 - os itens são separados por vírgulas
 - o dicionário é delimitado por chaves
 - um dicionário vazio é definido por `{ }`.
- As chaves de um dicionário são exclusivas, enquanto os valores podem não ser.
- Os valores de um dicionário podem ser de qualquer tipo, mas as chaves devem ser de um tipo de dados imutável, como strings, números ou tuplas
- Exemplos:
 - `dic1 = {'um': 1, 2.0: [2], False: None}`
 - `dic2 = {(0,0): 'a', (0,1): 'b', (1,0): 'c'}`
- Para acessar os elementos de um dicionário usamos a chave no lugar do índice:
 - `dic1['um']`
 - `dic2[0,1]`

Funções

- Funções em Python são definidas de acordo com a seguinte sintaxe:

```
def nome_funcao(arg0, arg1, ..., argopt0=val0, argopt1=val1, ...):  
    <Corpo da função>  
    return valor_retorno
```

- Exemplo:

```
def soma2(a, b=2):  
    return a + b
```

- A definição de função, bem como as demais definições em blocos, é delimitada através da *indentação* das linhas de código, e não por caracteres delimitadores
 - Os caracteres usados na indentação das linhas devem ser sempre os mesmos, seja o caractere de tabulação, seja um certo número de caracteres de espaço
- Os argumentos `arg0, arg1, ...` são obrigatórios
- Os argumentos `argopt0, argopt1, ...` são opcionais
- A invocação de uma função é semelhante a outras linguagens, exceto que os argumentos podem ser identificados por sua posição ou pelo identificador do argumento.
 - `var_retorno = nome_funcao(val0, arg1=val1, argopt4=val2)`

Estruturas de controle de fluxo

- Para executar um código iterando sobre os elementos de uma sequência (listas, tuplas, etc.):

```
for i in seq:  
    <Código a ser executado>
```
- Para criar uma sequência de inteiros, usamos uma das versões da função range:
 - range(stop): cria uma sequência de 0 a stop - 1,
 - range(start, stop): cria uma sequência iniciando em start e terminando em stop - 1,
 - range(start, stop, step): cria uma sequência de start (inclusive) a stop (exclusive), com passo de incremento de step
- Exemplos:

```
for i in range(0, 20, 2):  
    print(i) # imprime os números pares  
soma = 0.0  
for n in [2.0, 3.5, -1, 4, 5]:  
    soma += n #soma os valores da lista
```
- Estrutura de desvio condicional

```
if <condição>:  
    <Código a ser executado>  
elif <outra-condição>: #opcional, quantas vezes for necessario  
    <Código a ser executado>  
else: #opcional  
    <Código a ser executado>
```
- Exemplo:

```
n = int(input('Entre com um número natural: '))  
if n % 2 == 0:  
    print("O número {} é par".format(n))  
else:  
    print("O número {} é ímpar".format(n))
```
- Para executar um código com condição de parada:

```
while <condição>:  
    <Código a ser executado>
```

Exercícios

- Crie uma função que receba um número natural $n > 0$ e retorne o n -ésimo número da sequência de Fibonacci $F(n)$, onde
 - $F(1) = 1$
 - $F(2) = 1$
 - $F(n) = F(n-1) + F(n-2)$, se $n > 2$
 - Dica: Python aceita funções recursivas
- Crie um programa que pergunte um número natural e imprima uma lista com a sequência de Fibonacci até a n -ésima posição, usando a função anterior.

Python notebooks

- Uma forma bastante profícua para interação com a disciplina é através dos notebooks
- O programa IPython fornece uma interface interativa para o Python, de forma que outros programas (IDEs) possam incorporar os resultados das chamadas, sejam na forma numérica, de texto ou de gráfico, apresentando-os de modo elegante
- Um desses programas é o Jupyter, que fornece uma interface web para a entrada de comandos e acessar os resultados
 - Além do mais, Jupyter entende a formatação de texto markdown, permitindo a criação de apostilas e relatórios de um modo elegante e interativo
- Para iniciar o Jupyter, usamos a linha de comando:
 - `jupyter notebook`
- Na janela do navegador que foi aberta, navegar entre as pastas até encontrar os notebooks fornecido pelo professor:
 - IntroPython0.ipynb, IntroPython1.ipynb e IntroPython2.ipynb

Pacotes e módulos

- Módulos são coleções de objetos do Python, em geral definidos dentro de um arquivo ou diretório
 - Objetos do Python são as funções, classes, variáveis, etc.
 - Todo arquivo fonte do Python pode ser considerado um módulo
- Um diretório que contenha um módulo deve possuir o arquivo `__init__.py`, mesmo que esteja vazio
 - Um diretório que também é módulo pode conter submódulos
- Um módulo pode corresponder também a uma biblioteca nativa (`.dll` ou `.SO`) contendo objetos já compilados em C ou C++ em geral
- Os pacotes são arquivos contendo conjuntos de módulos que podem ser facilmente instalados no repositório de módulos do ambiente Python

Importando módulos para o arquivo

- Importa o módulo `modulo` para o programa atual:
 - `import modulo`
- Importa o submódulo `submod` dentro do módulo `modulo` para ser invocado como `mod` no programa
 - `import modulo.submod as mod`
- Importa o objeto `myfunc` do módulo `modulo`
 - `from modulo import myfunc`
- Importa todos os objetos do submódulo `submod` dentro do módulo `modulo`
 - `from modulo.submod import *`

O pacote `numpy`

- Pacote que contém a estrutura básica de operações com vetores homogêneos em Python: `ndarray`
 - Trata-se de uma classe que representa arrays multidimensionais de elementos do mesmo tipo
 - O conteúdo do array pode ser de qualquer tipo reconhecido pelo Python, ou mesmo de tipos definidos pelo pacote `numpy`
- Os arrays mais usados são os vetores (1D) e matrizes (2D)
- Operações realizadas sobre arrays em Python são muito mais eficientes do que o uso de estruturas de repetição
- Diversas operações matemáticas podem ser realizadas em arrays, de forma que elas são aplicadas a cada um dos elementos:
 - Soma, multiplicação, divisão, etc. de dois arrays do mesmo tamanho
 - Soma, multiplicação, divisão, etc. de um array com uma constante
 - Comparação de valores em dois arrays do mesmo tamanho
 - Comparação de valores de array com uma constante
 - Funções matemáticas aplicadas aos elementos do array
- Há uma grande flexibilidade no acesso de dados dentro do array, com a possibilidade de:
 - Uso de slices
 - Índices booleanos
 - Vetores de índices

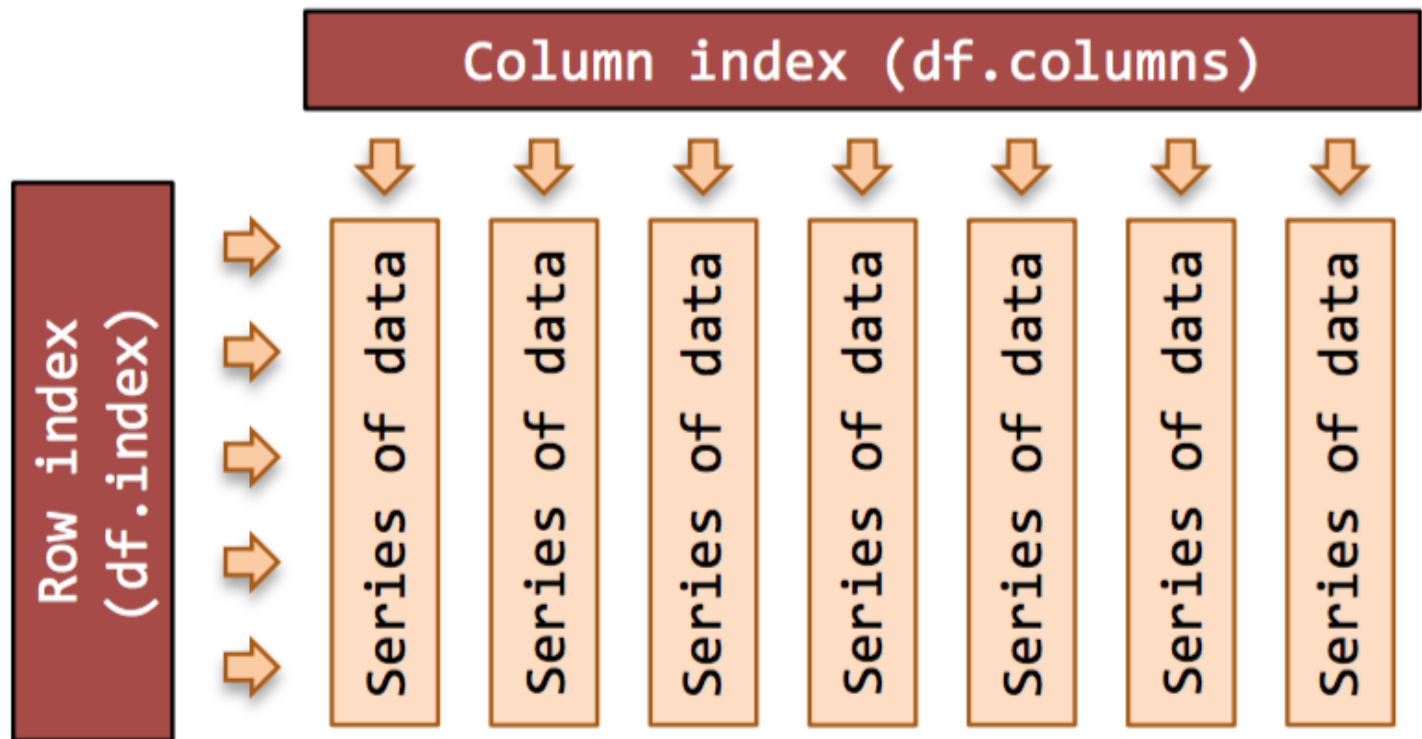
I O pacote `matplotlib`

- O pacote `matplotlib` confere ao ambiente Python a capacidade de gerar gráficos a partir de dados numéricos
 - Em especial, o submódulo `pyplot` emprega uma notação semelhante aos comandos de plotagem do Matlab e Octave
- Com ele podemos criar gráficos interativos e até animações para visualizar e interagir com os dados
- Vamos manipular as estruturas de dados apresentadas através do notebook `IntroPython1.ipynb`

■ O pacote **pandas**

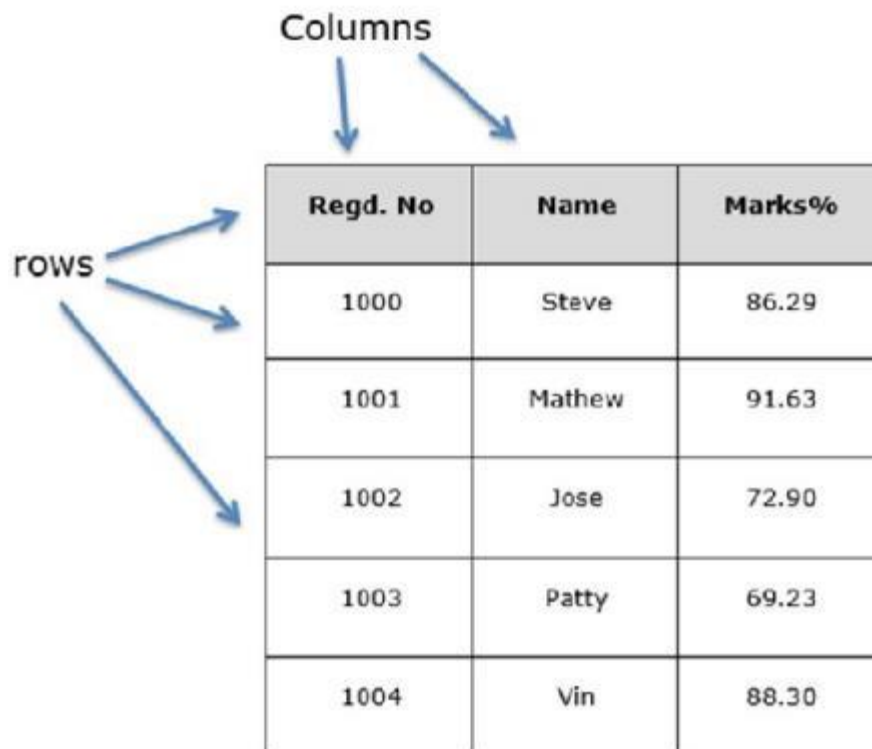
- `pandas` é uma ferramenta para manipulação de dados em alto nível.
- A estrutura de dados principal desse pacote é o `DataFrame`, que se assemelha a uma tabela de um banco de dados:
 - As colunas possuem dados de mesma natureza, e é implementada através da classe `Series` do `pandas`, que se assemelha ao array do `numpy`
 - Cada linha corresponde a uma entrada de dados
- Aprenderemos a manipular um `DataFrame` de acordo com as necessidades da disciplina
 - Vamos manipular as estruturas de dados apresentadas através do notebook `IntroPython2.ipynb`

Representação de um DataFrame



<https://www.kaggle.com/timolee/a-home-for-pandas-and-sklearn-beginner-how-tos>

Exemplo de um DataFrame



The diagram shows a table representing a DataFrame. The columns are labeled 'Regd. No', 'Name', and 'Marks%'. The rows are labeled with student IDs: 1000, 1001, 1002, 1003, and 1004. Blue arrows point from the labels 'Columns' and 'ROWS' to their respective parts of the table.

Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

REFERÊNCIAS

- <https://www.learnpython.org/>
- https://www.tutorialspoint.com/python_pandas/index.htm





Copyright © 2018 Prof. Antonio Selvatici

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).