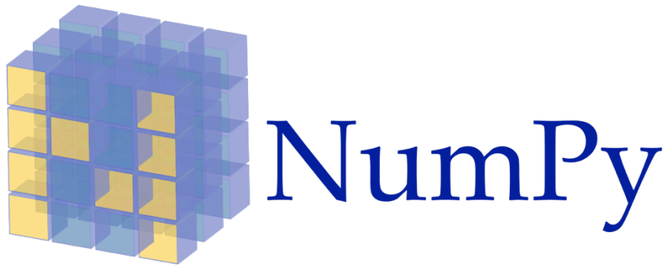
Numpy & Matplotlib

Camila Laranjeira, Jefersson A. dos Santos
{camilalaranjeira, jefersson}@dcc.ufmg.br

Biblioteca

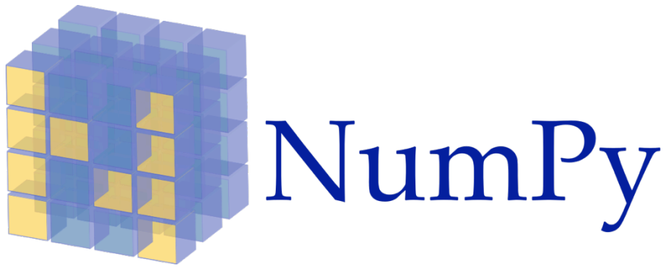
- Contém um conjunto de tarefas relacionadas entre si
- Permite estender o núcleo básico do Python.
- Nessa aula conheceremos:

    1. Numpy: Processamento Vetorial
    2. Matplotlib: Visualização de Dados



Numpy & Matplotlib

- Nativos do Anaconda
- Devem ser instalados separadamente no Python básico




```
pip install numpy
conda install -c anaconda numpy
```


```
pip install matplotlib
conda install -c conda-forge matplotlib
```

Pacotes Anaconda

Para listar os pacotes instalados no Anaconda:

- No terminal

```
$ conda list
```

- No ambiente Python

```
>>> print(help("modules"))
```

Relembrando

- Propriedades de Listas

```
# Cria uma lista heterogênea
>>> xs = [3, 1, 'minas']
# Acessa elementos
>>> print(xs[0], xs[-1]) #imprime “3 minas”
# Funções de lista
>>> xs.append('gerais')
>>> x = xs.pop()
>>> print(x, xs) #imprime “gerais [3, 1, 'minas']”
```

Relembrando

- Fatiamento de listas

```
>>> nums = list(range(5)) # [0, 1, 2, 3, 4]
# Fatia do índice 2 ao 4
>>> print(nums[2:4]) # [2, 3]
# Omitindo valores
>>> print(nums[2:], [:2], [:])
>>> print(nums[:-1])
# Atribui sublista à fatia
>>> nums[2:4] = [8, 9]
>>> print(nums) # imprime "[0, 1, 8, 9, 4]"
```

Relembrando

- **Exercício 1**

- Dada a seguinte lista:

```
lista1 = [0, 1, 2, 3, 2, 1, 0]
```

- Determine o fatiamento que produza a lista `[2, 3, 2]`

```
>>>
```

Relembrando

- **Exercício 1**

- Dada a seguinte lista:

```
lista1 = [0, 1, 2, 3, 2, 1, 0]
```

- Determine o fatiamento que produza a lista `[2, 3, 2]`

```
>>> lista1[2:5]
```

Ponteiros Implícitos

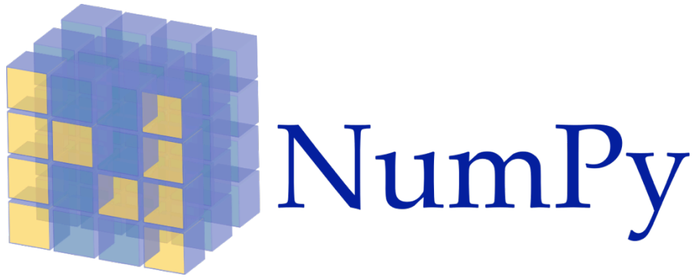
- Atribuições entre listas fazem com que ambas as listas apontem para o mesmo conteúdo

```
>>> lista1 = [1,2,3,4]
>>> lista2 = lista1
>>> lista2.append(10)
>>> print(lista1) # imprime [1,2,3,4,10]
>>> print(lista2) # imprime [1,2,3,4,10]
```

Ponteiros Implícitos

- É possível realizar uma cópia do conteúdo da lista, mantendo ambas as listas independentes.

```
>>> lista1 = [1,2,3,4]
>>> lista2 = list(lista1)
>>> lista2.append(10)
>>> print(lista1) # imprime [1,2,3,4]
>>> print(lista2) # imprime [1,2,3,4,10]
```



- O primeiro passo para usar qualquer biblioteca é importá-la no seu programa
- A literatura convenciona o import do **`numpy`** com o apelido **`np`**

```
import numpy as np
```

Numpy

- Processamento vetorial **multidimensional homogêneo**
- **rank:** Número de dimensões do array

```
>>> a = np.array([1, 2, 3])   # array de rank 1
>>> print("Tipo", type(a))    # <class 'numpy.ndarray'>
>>> print("Shape", a.shape)   # (3,)
>>> print(a)                  # [1, 2, 3]
>>> a[0] = 5
>>> print(a[0], a[1], a[2])   # 5 2 3
```

Numpy

- Processamento vetorial **multidimensional homogêneo**
- **rank:** Número de dimensões do array

```
# array de rank 2
>>> a = np.array([[1,2,3],[4,5,6]])
>>> print("Tipo", type(a))    # <class 'numpy.ndarray'>
>>> print("Shape", a.shape)   # (2,3)
>>> print(b[0, 0], b[0, 1], b[1, 0]) # 1 2 4
```

Numpy

- Processamento vetorial **multidimensional homogêneo**
- **rank:** Número de dimensões do array

```
# array rank 2 tipo float64
>>> x = np.array([[1,2],[3,4]], dtype=np.float64)
>>> print(x) # [[1., 2.][3., 4.]]

# array rank 2 tipo int64
>>> y = np.array([[1,2],[3,4]], dtype=np.int64)
```

Numpy

- **Exercício 2**

a) Crie um array do numpy com 3 linhas e 2 colunas (com valores que você escolher)

b) Imprima o array

c) Imprima o formato (shape) do array

d) Imprima o rank do array

e) Imprima o primeiro elemento da primeira linha e o último elemento da última linha

Numpy

- Outras funções para criação de arrays

```
>>> a = np.zeros((2,2))
>>> b = np.ones((1,2))
>>> c = np.full((2,2), 7)
>>> d = np.eye(2)
>>> e = np.random.random((2,3)) #intervalo [0.0, 1.0)
```

Numpy

- Criando e preenchendo arrays vazios

```
# rank 1 tipo float
>>> vetor = np.empty((0), dtype=np.float64)
>>> vetor = np.append(vetor, [1., 2., 3.])
```

Numpy

- Criando e preenchendo arrays vazios

```
# rank 1 tipo float
>>> vetor = np.empty((0), dtype=np.float64)
>>> vetor = np.append(vetor, [1., 2., 3.])
```

- Note que:

```
>>> vetor.append([1., 2., 3.])  # incorreto
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'numpy.ndarray' object has no attribute 'append'
```

Numpy

- Criando e preenchendo arrays vazios

```
# rank 2 tipo float
>>> vetor = np.empty((0, 3), dtype=np.float64)
>>> vetor = np.append(vetor, [1., 2., 3.])
# acrescenta várias linhas de uma vez
>>> matriz = np.append(matriz, [[3., 4., 5.], [6.,
7., 8.], [9., 10., 11.]], axis=0)
```

Numpy

- Fatiamento **multidimensional**

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Como fatiar para obter os elementos das primeiras
duas linhas, das colunas 1 e 2?
>>>
```

Numpy

- Fatiamento **multidimensional**

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Como fatiar para obter os elementos das primeiras
duas linhas, das colunas 1 e 2?
>>> slice = a[:2, 1:3]
```

Numpy

- Fatiamento **multidimensional**

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Como fatiar para obter os elementos das primeiras
duas linhas, das colunas 1 e 2?
>>> slice = a[:2, 1:3]
# Qual o subarray resultante?
>>> print(slice)
```

Numpy

- Fatiamento **multidimensional**

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Como fatiar para obter os elementos das primeiras
duas linhas, das colunas 1 e 2?
>>> slice = a[:2, 1:3]
# Qual o subarray resultante?
>>> print(slice)  # [[2 3] [6 7]]
```

Numpy

- Fatiamento **multidimensional**

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
>>> slice = a[:2, 1:3]

# Modificações na fatia são refletidas no array original
>>> slice[0, 0] = 77
>>> print(a) #[[1,77,3,4], [5,6,7,8], [9,10,11,12]]
>>> print(slice) #[[77, 3] [6, 7]]
```

Numpy

- Fatiamento **multidimensional**

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
>>> slice = a[:2, 1:3]

# Modificações na fatia são refletidas no array original
>>> slice[0, 0] = 77
>>> print(a) #[[1,77,3,4], [5,6,7,8], [9,10,11,12]]
>>> print(slice) #[[77, 3] [6, 7]]
# A posição [0,0] do slice equivale a posição [0,1] do array
```

Numpy

- Fatiamento **multidimensional**

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Similar às listas, é possível realizar a cópia independente
>>> slice = np.array(a[:2, 1:3])
>>> slice[0, 0] = 77
>>> print(a) #[[1,2,3,4], [5,6,7,8], [9,10,11,12]]
>>> print(slice) #[[77, 3] [6, 7]]
```

Numpy

- **Exercício 3**
- Dado o array

```
ar = np.array([[0, 0, 0, 0],
               [0, 0, 0, 0],
               [0, 1, 2, 0],
               [0, 3, 4, 0],
               [0, 0, 0, 0]])
```

- Qual fatiamento produz o subarray `[[1 ,2] [3, 4]]` ?

```
>>>
```

Numpy

- **Exercício 3**
- Dado o array

```
ar = np.array([[0, 0, 0, 0],
               [0, 0, 0, 0],
               [0, 1, 2, 0],
               [0, 3, 4, 0],
               [0, 0, 0, 0]])
```

- Qual fatiamento produz o subarray `[[1 ,2] [3, 4]]` ?

```
>>> ar[2:4, 1:3]
```

Numpy

- Indexação inteira com fatiamento
  - ❏ Você também pode usar indexação inteira com fatiamento, mas o subarray produzido **terá rank menor que o original.**

```
>>> a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
# Subarray da linha 2 com rank 1
>>> linha_r1 = a[1, :]
# Subarray da linha 2 com rank 2
>>> linha_r2 = a[1:2, :]
```

Numpy

- Indexação booleana
  - ❏ Você também pode selecionar elementos com base em alguma **condição**

```
>>> a = np.array([[21,-6], [43, 14], [-5, 36]])
# Vamos zerar* todos os elementos negativos da matriz.
>>> a[a < 0] = 0
>>> print(a)
# [[21,0], [43, 14], [0, 36]]
```

Numpy

- Operações matemáticas básicas operam elemento a elemento nos arrays
- Array e Escalar

```
>>> x = np.array([[1,2],[3,4]])
>>> soma = x + 10 #[[11,12],[13,14]]
>>> subt = np.subtract(x, 1)
#[[0,1],[2,3]]
>>> mult = x * 10 #[[10,20],[30,40]]
>>> divi = np.divide(x, 2) #[[0,1],[1,2]]
```

+	np.add()
-	np.subtract()
*	np.multiply()
/	np.divide()

Numpy

- Operações matemáticas básicas operam elemento a elemento nos arrays
- Array e Array

```
>>> x = np.array([[1.,2.],[3.,4.]])
>>> y = np.array([[5.,6.],[7.,8.]])
>>> soma = x + y
#[[6.0,8.0],[10.0,12.0]]
>>> subt = np.subtract(x, y)
#[[-4.0,-4.0],[-4.0,-4.0]]
>>> mult = x * y #[[5.0, 12.0],[21.0, 32.0]]
>>> divi = np.divide(x, y) #[[0.2, 0.43],[0.34,0.5]]
```

+	np.add()
-	np.subtract()
*	np.multiply()
/	np.divide()

Numpy

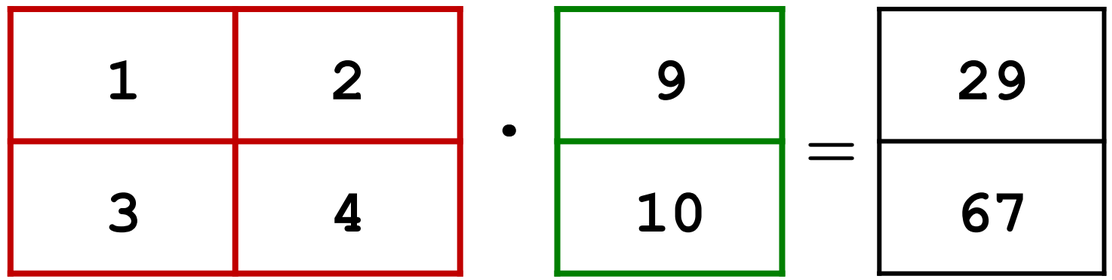
- Operações matemáticas básicas operam elemento a elemento nos arrays
- Array e Array

```
>>> x = np.array([[1.,2.],[3.,4.]])
>>> y = np.array([[5.,6.],[7.,8.]])
# Note que essa é uma multiplicação elemento a elemento
# Diferente da multiplicação de matrizes
>>> mult = x * y
```

Numpy

- Para multiplicação de matrizes, o numpy possui a função **dot**

```
>>> x = np.array([[1,2],[3,4]])
>>> v = np.array([[9],[10]])

>>> x.dot(v)
>>> np.dot(x, v)
```



Numpy

- O Numpy fornece várias funções para realizar computações em arrays, uma das mais úteis é a função **sum**:

```
>>> x = np.array([[1,2, 3, 4],[10, 20, 30, 40]])
>>> print("Soma de todos elementos de x: ")
>>> print(np.sum(x))
>>> print("Soma dos elementos de cada coluna:")
>>> print(np.sum(x, axis=0))
>>> print("Soma dos elementos de cada linha:")
>>> print(np.sum(x, axis=1))
```

Numpy

- **Exercício 4**

Numpy

- Matriz transposta

```
>>> a = np.array([[1,2], [3, 4], [5, 6]]) # (3, 2)
>>> print("matriz a transposta:")
>>> print(a.T) # (2, 3)
```

1	2
3	4
5	6

1	3	5
2	4	6

Numpy e Arquivos

- É possível realizar a leitura de arquivos com o Numpy

```
>>> arr = np.loadtxt('dados.csv',delimiter=',',dtype=np.float64)
```

- **Os dados já serão lidos no formato definido pelo `dtype`**

- Biblioteca de visualização de dados
- Módulo **`pyplot`** é o mais importante da biblioteca
- A literatura convenciona o import do **`matplotlib.pyplot`** com o apelido **`plt`**

```
import matplotlib.pyplot as plt
```


```
import matplotlib.pyplot as plt
```

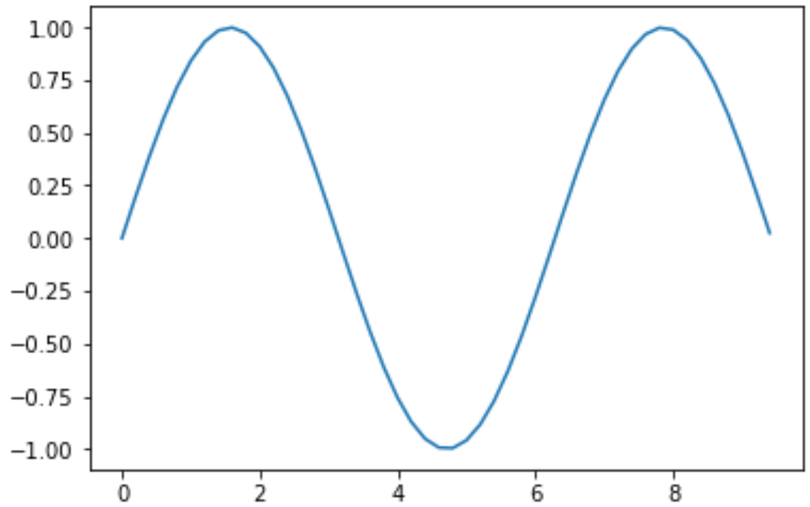
```
%matplotlib inline
```

- Comando do ambiente Jupyter para plotar os gráficos logo após a célula onde o gráfico foi instanciado

Matplotlib

- Plot 2D

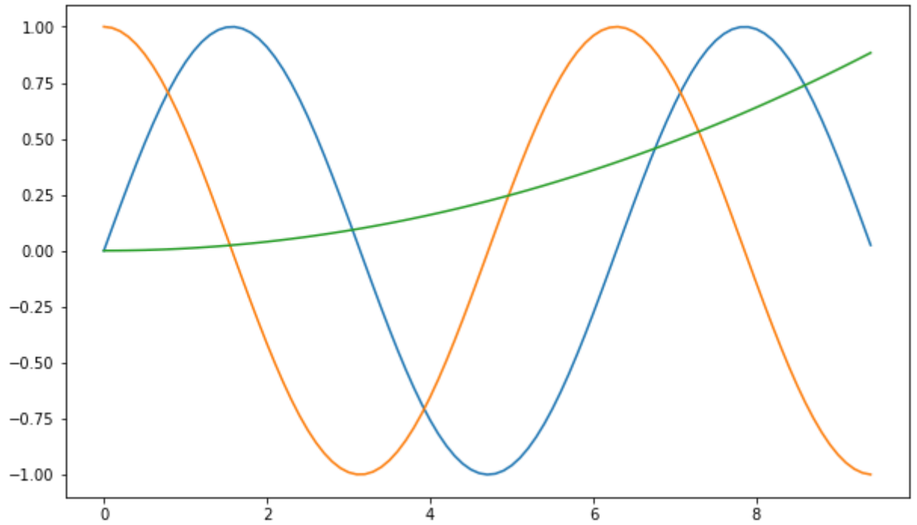
```
# Plotando uma senoide
>>> x = np.arange(0, 3 * np.pi, 0.2)
>>> y = np.sin(x)
>>> plt.plot(x, y)
```



Matplotlib

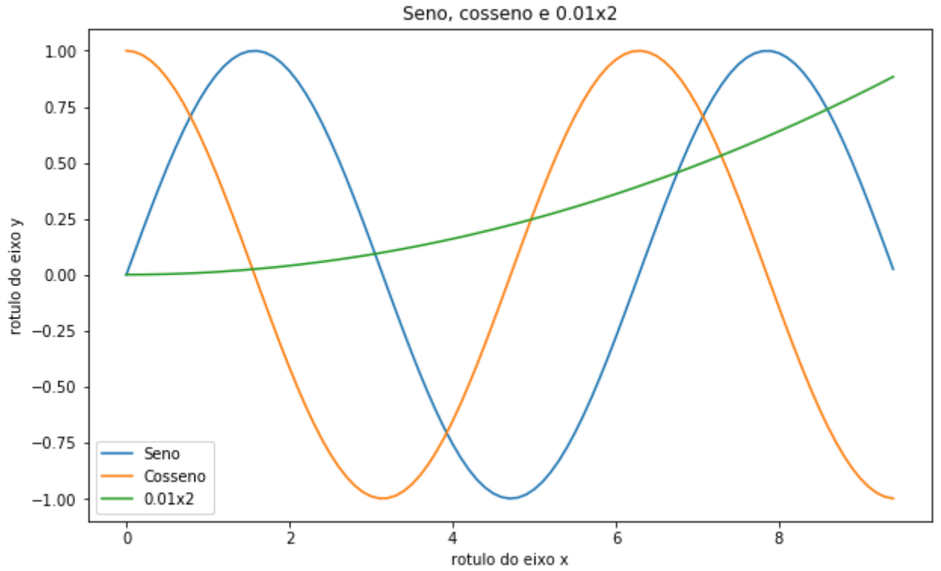
- Múltiplos plots e Propriedades

```
>>> x = np.arange(0, 3 * np.pi, 0.1)
>>> y_sin = np.sin(x)
>>> y_cos = np.cos(x)
>>> y_001x2 = 0.01 * x**2
>>> plt.plot(x, y_sin)
>>> plt.plot(x, y_cos)
>>> plt.plot(x, y_001x2)
```



Matplotlib

- Múltiplos plots e Propriedades

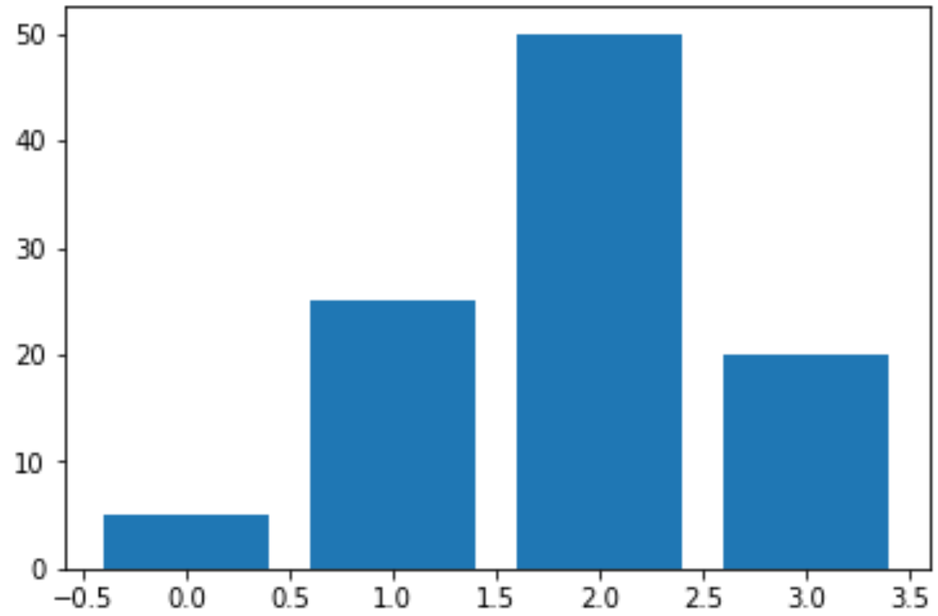



```
>>> plt.plot(x, y_sin)
>>> plt.plot(x, y_cos)
>>> plt.plot(x, y_001x2)
>>> plt.xlabel('rotulo do eixo x')
>>> plt.ylabel('rotulo do eixo y')
>>> plt.title('Seno, cosseno e 0.01x2')
>>> plt.legend(['Seno', 'Cosseno', '0.01x2'])
```

Matplotlib

- Gráfico de Barras (Vertical)

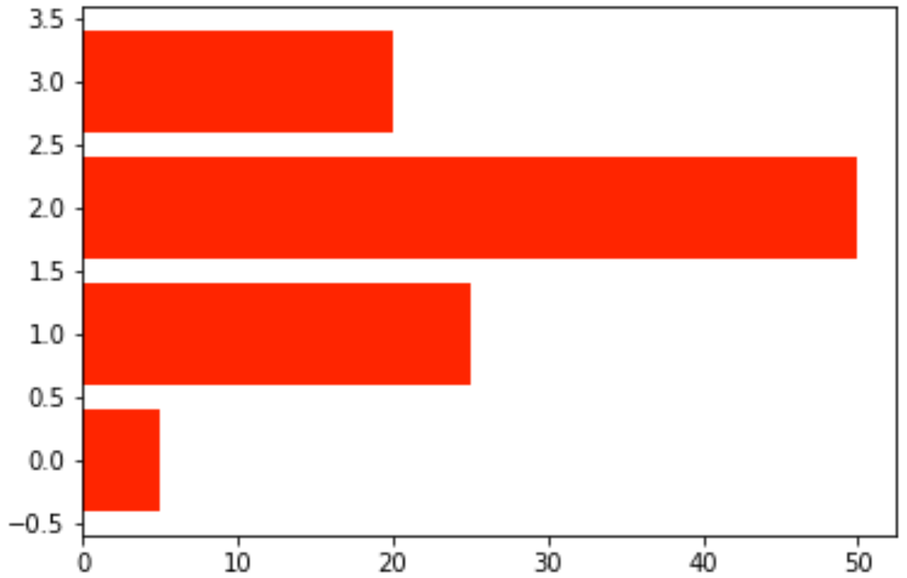
```
>>> data = [5., 25., 50., 20.]
>>> plt.bar(range(len(data)), data)
```



Matplotlib

- Gráfico de Barras (Horizontal)

```
>>> data = [5., 25., 50., 20.]
>>> plt.barh(range(len(data)), data, color='r')
```



Matplotlib

- Propriedade: **color**

```
>>> data = [5., 25., 50., 20.]
>>> plt.barh(range(len(data)), data, color='r')
```

- Cores padrão

```
{ 'b': blue, 'g': green, 'r': red, 'c': cyan, 'm': magenta, 'y':
yellow, 'k': black, 'w': white }
```

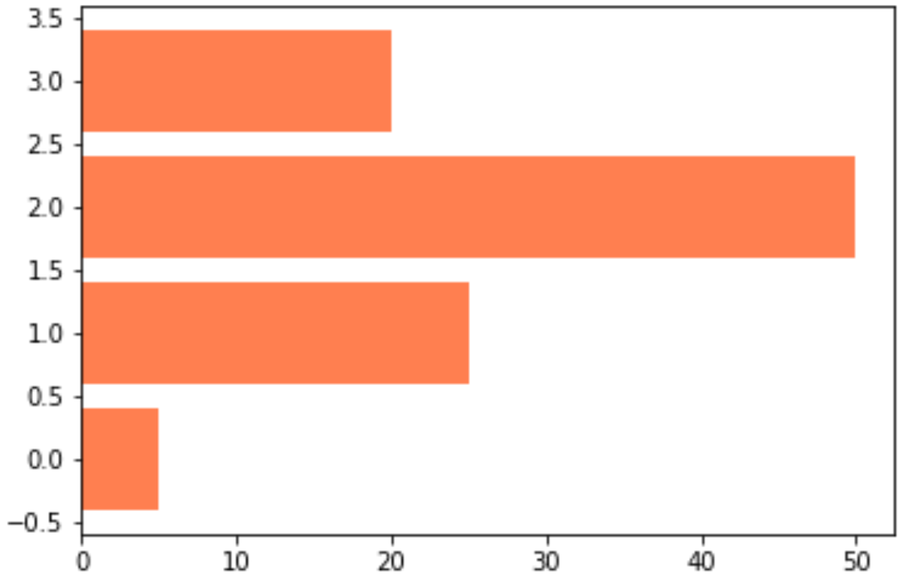
Matplotlib

- Propriedade: **color**

```
>>> data = [5., 25., 50., 20.]
>>> plt.barh(range(len(data)), data, color='#FF7F50')
```

- Cores hexadecimal

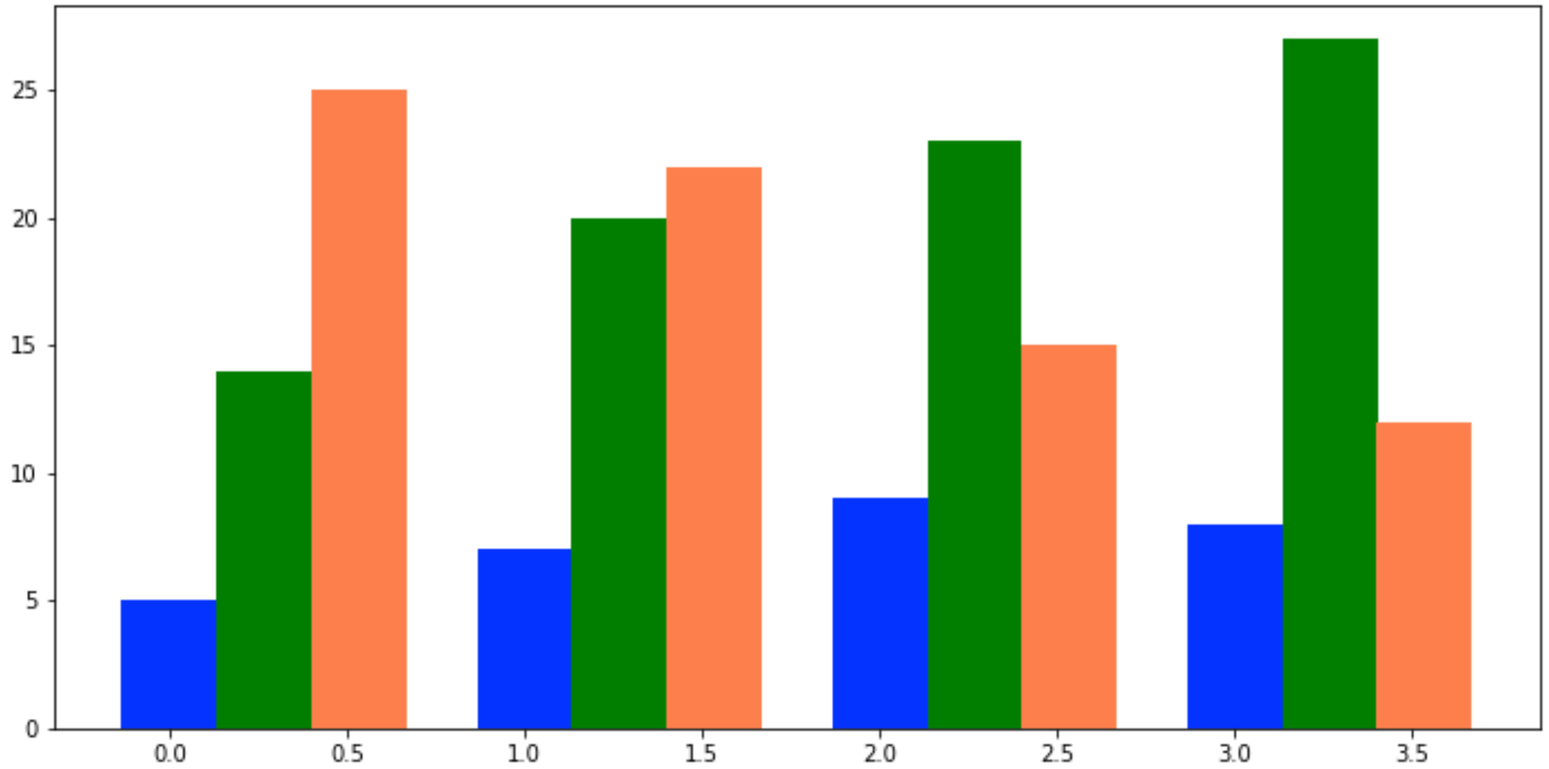
  #RRGGBB



Matplotlib

- Múltiplas barras

```
[[5., 7., 9., 8.],
[14., 20., 23., 27.],
[25., 22., 15., 12.]]
```



Matplotlib

- Múltiplas barras

```
>>> dados = np.array([[5., 7., 9., 8.],
                      [14., 20., 23., 27.],
                      [25., 22., 15., 12.]])
>>> X = np.arange(4)
>>> plt.bar(X + 0.00, dados[0], color = 'b', width = 0.267)
>>> plt.bar(X + 0.267, dados[1], color = 'g', width = 0.267)
>>> plt.bar(X + 0.533, dados[2], color = '#FF7F50', width = 0.267)
```

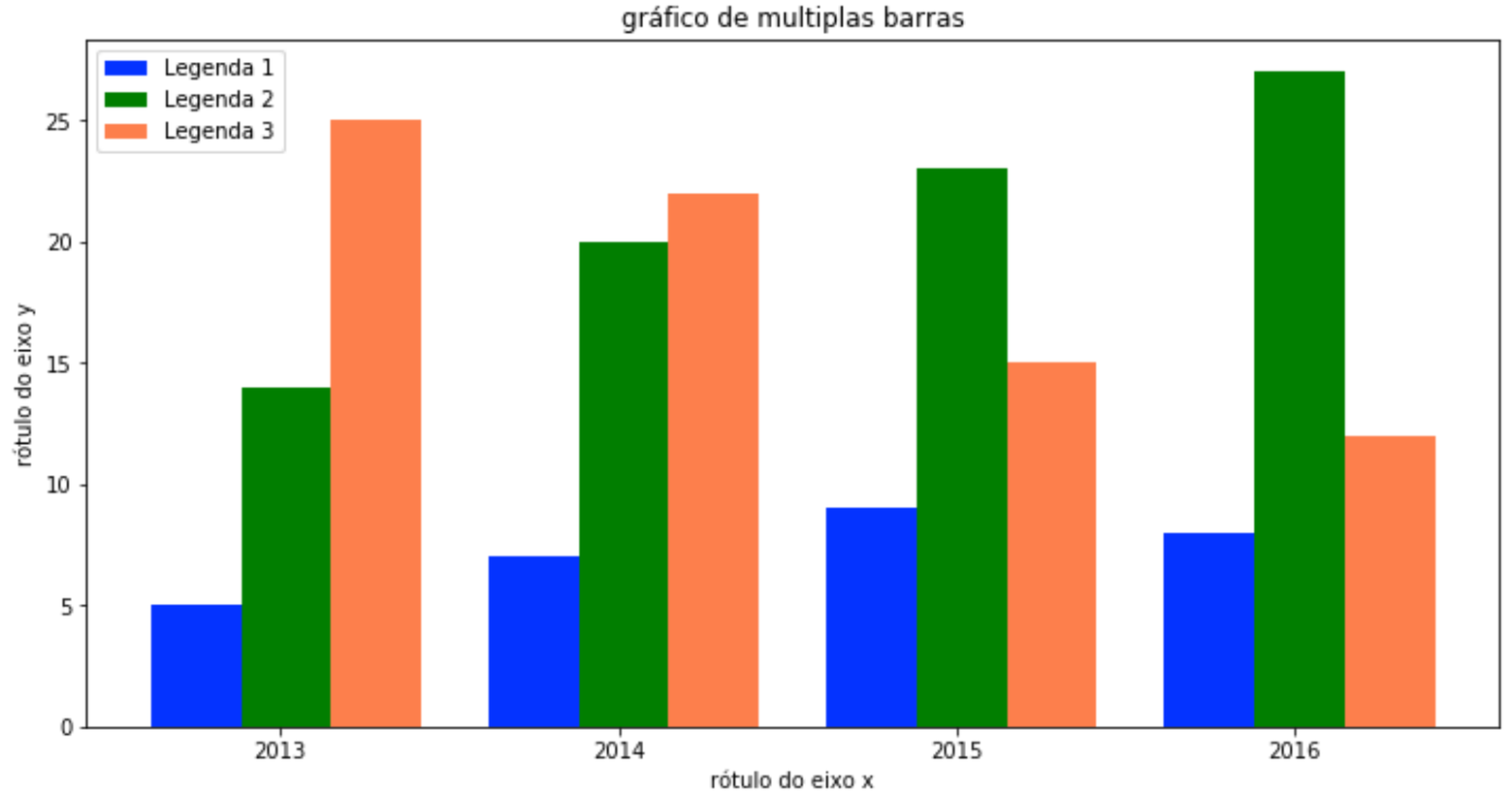
Matplotlib

- Propriedades

```
>>> colunas = ['2013', '2014', '2015', '2016']
>>> plt.xlabel('rótulo do eixo x')
>>> plt.ylabel('rótulo do eixo y')
>>> plt.title('gráfico de multiplas barras')
>>> plt.legend(['Legenda 1', 'Legenda 2', 'Legenda 3'])
>>> plt.xticks(np.arange(0, 4)+0.25, colunas)
```

Matplotlib

- Múltiplas barras



Matplotlib

- Múltiplas barras (em loop)

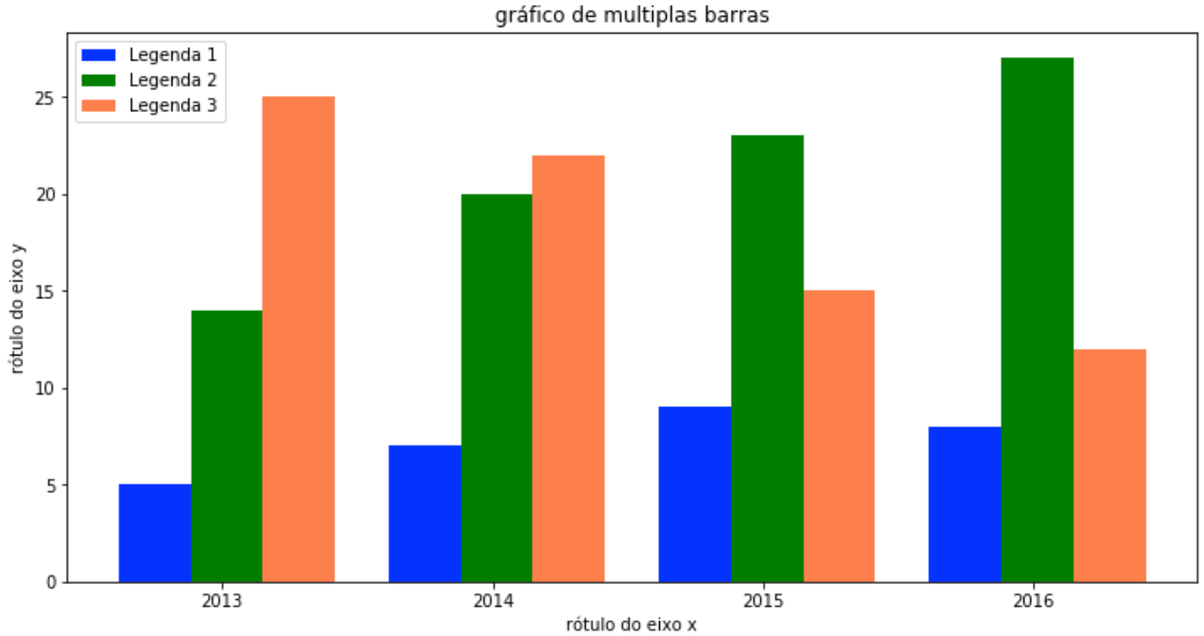
```
>>> X = np.arange(dados.shape[1])
>>> cores = ['b', 'g', '#FF7F50']
>>> compr = 0.8/(dados.shape[0])
>>> for i in range(0, dados.shape[0]):
>>>     plt.bar(X + compr*i, dados[i], color=cores[i],
width=compr)
```

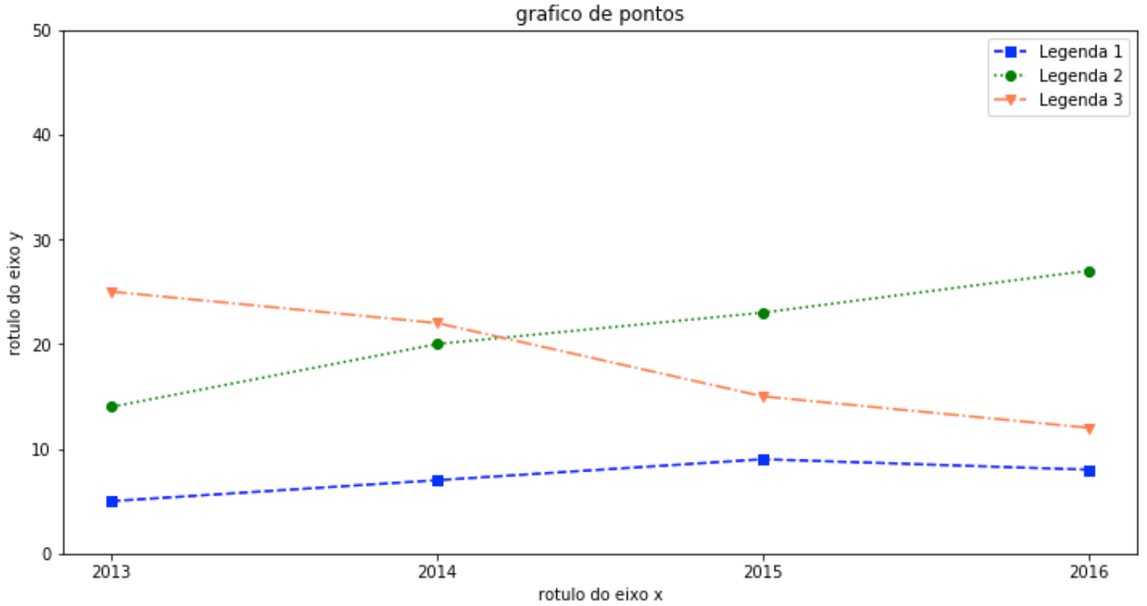
Matplotlib

- **Exercício 5**

Matplotlib

- Os mesmos dados podem ser plotados de diferentes formas!





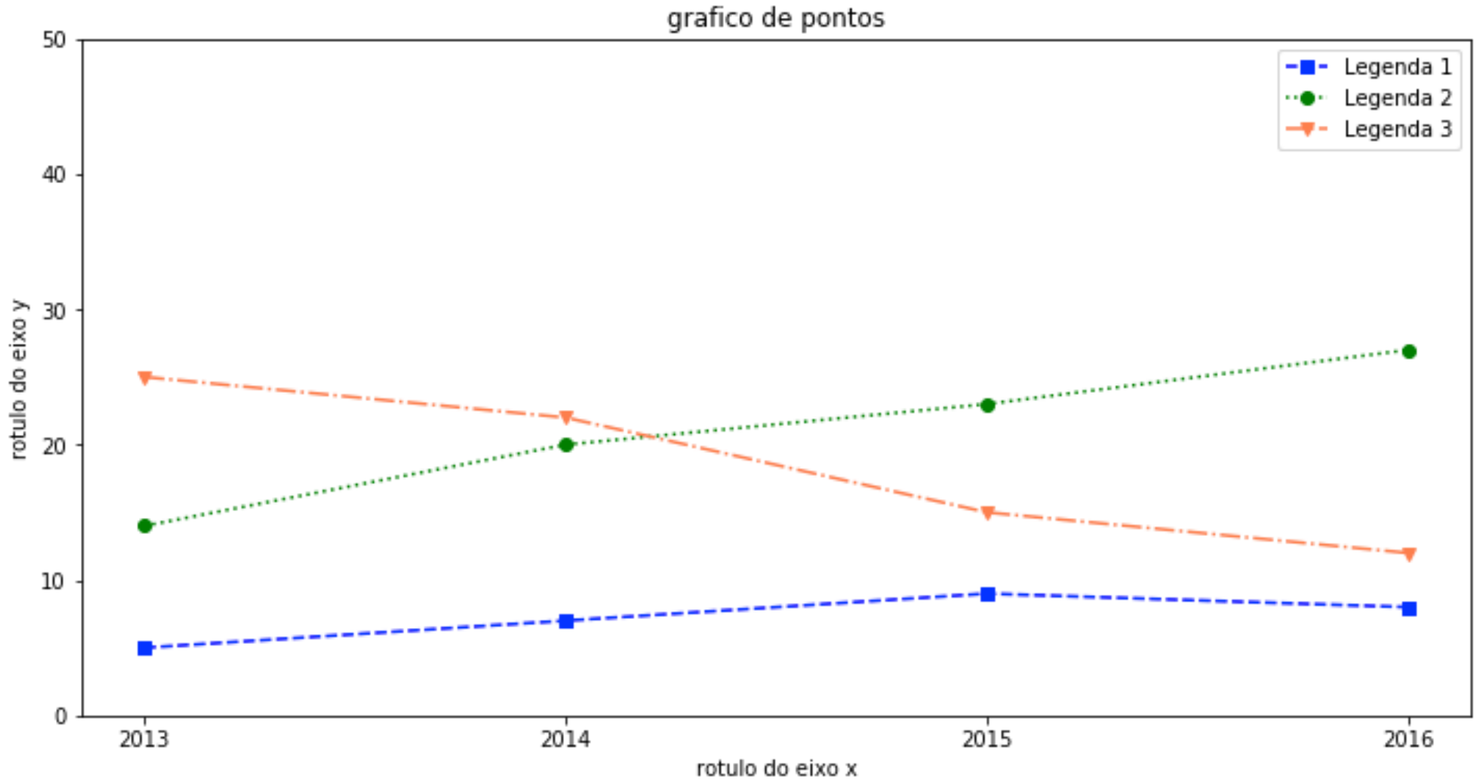
Matplotlib

- Gráfico de Pontos

```
>>> X = np.arange(dados.shape[1])
>>> cores = ['b', 'g', '#FF7F50']
>>> marcadores = ['s', 'o', 'v']
>>> linhas = ['--', ':', '-.']
>>> for i in range(0, dados.shape[0]):
>>>     plt.plot(X, dados[i], marker=marcadores[i],
                linestyle=linhas[i], color=cores[i])
>>> plt.ylim((0, 50))
```

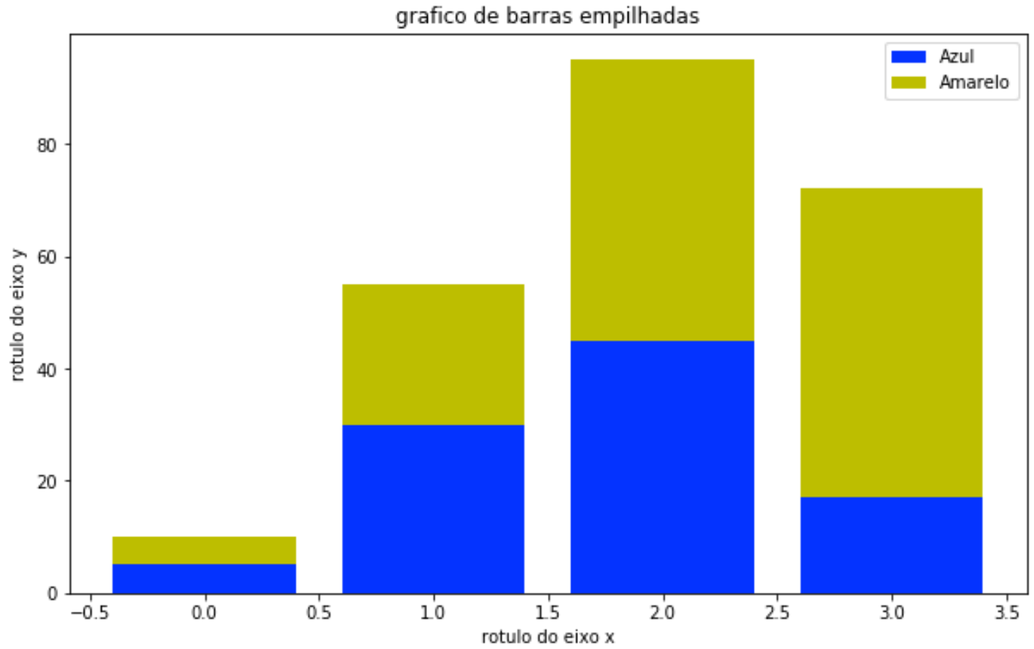
Matplotlib

- Gráfico de Pontos



Matplotlib

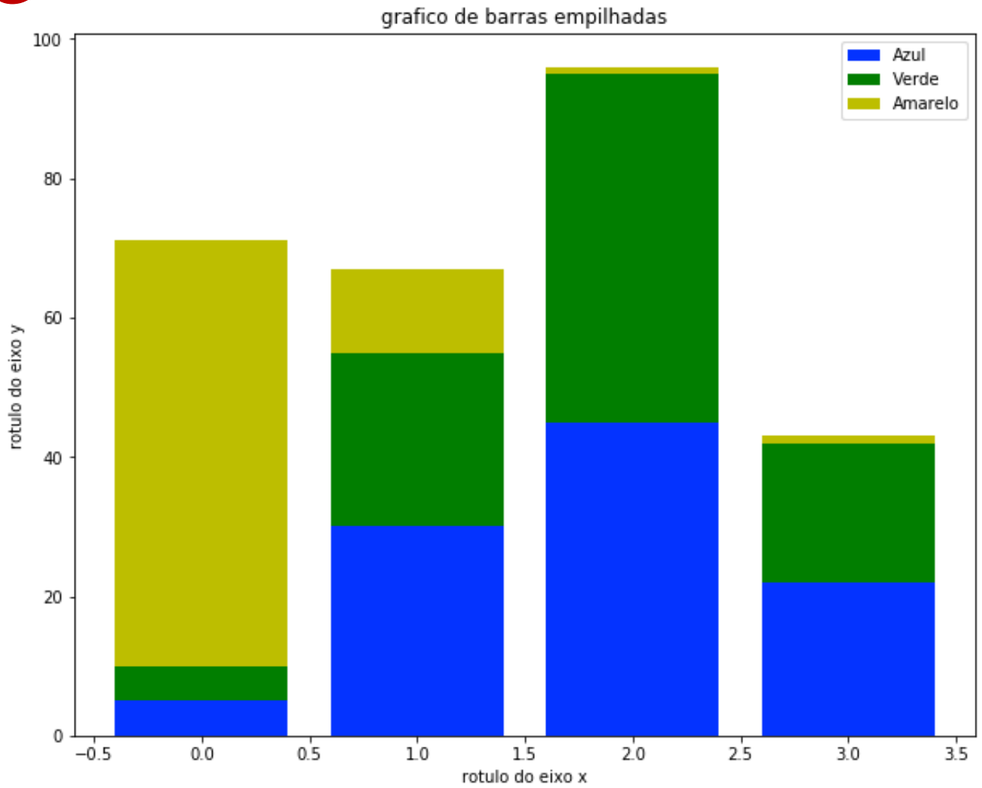
- Barras empilhadas




```
>>> A = [5., 30., 45., 17.]
>>> B = [5., 25., 50., 55.]
>>> X = range(4)

>>> plt.bar(X, A, color = 'b')
>>> plt.bar(X, B, color = 'y', bottom = A)
```

Matplotlib

- Barras empilhadas (com loop)




```
>>> data = np.array([[5., 30., 45., 22.],
                     [5., 25., 50., 20.],
                     [61.,  12.,  1.,  1.]])
>>> color_list = ['b', 'g', 'y']
>>> X = np.arange(data.shape[1])
>>> for i in range(data.shape[0]):
>>>     plt.bar(X, data[i],
                bottom = np.sum(data[:i], axis = 0),
                color = color_list[i % len(color_list)])
```

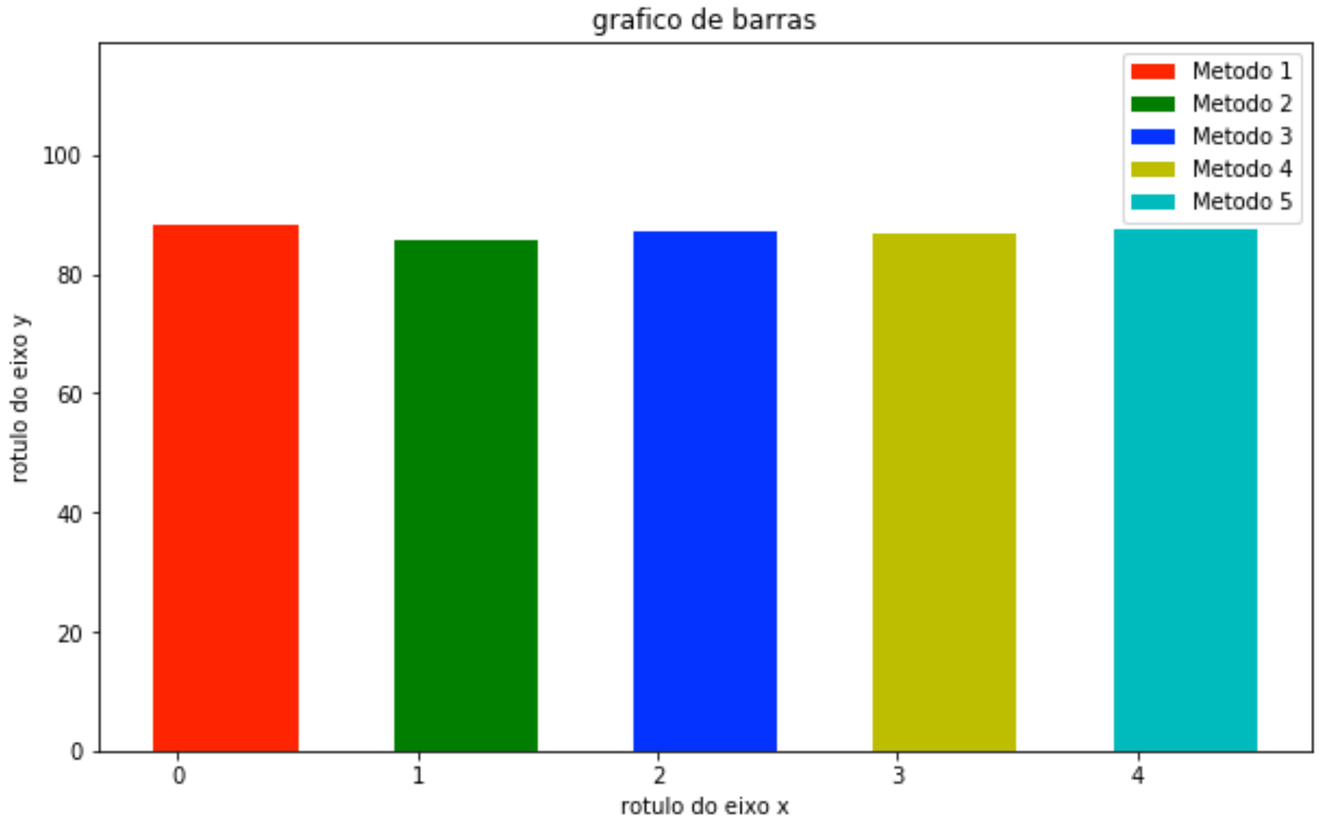
Matplotlib

- Plot de dados **lidos de arquivo**

```
>>> arr = np.loadtxt('dados.csv', delimiter=',', dtype=np.float64)
>>> X = np.arange(5)
>>> cores = ['r', 'g', 'b', 'y', 'c']
>>> for i in range(0, 5):
>>>       # plt.bar( coordenada X, coordenada Y, cor, comprimento da barra)
>>>       plt.bar(X[i] + 0.2, np.average(arr[i,:]), color=cores[i], width=0.6)
```

Matplotlib

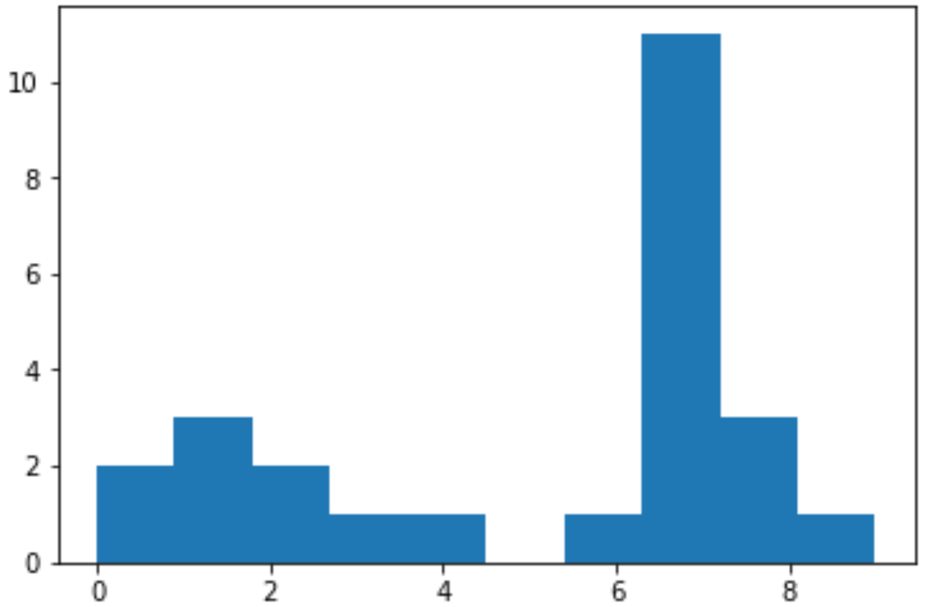
- Plot de dados **lidos de arquivo**



Matplotlib

- Plot de Histograma

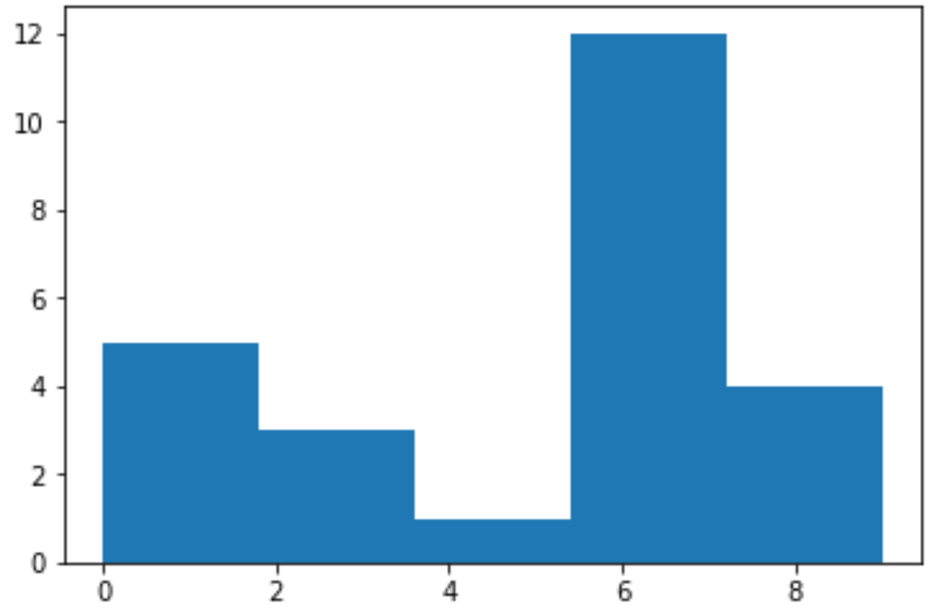
```
>>> X = np.array([0., 0., 1., 1., 1., 2., 2., 3.,
                  4., 6., 7., 7., 7., 7., 7., 7., 7., 7., 7., 7.,
                  8., 8., 8., 9.])

>>> plt.hist(X, bins = 10)
```



Matplotlib

- Plot de Histograma

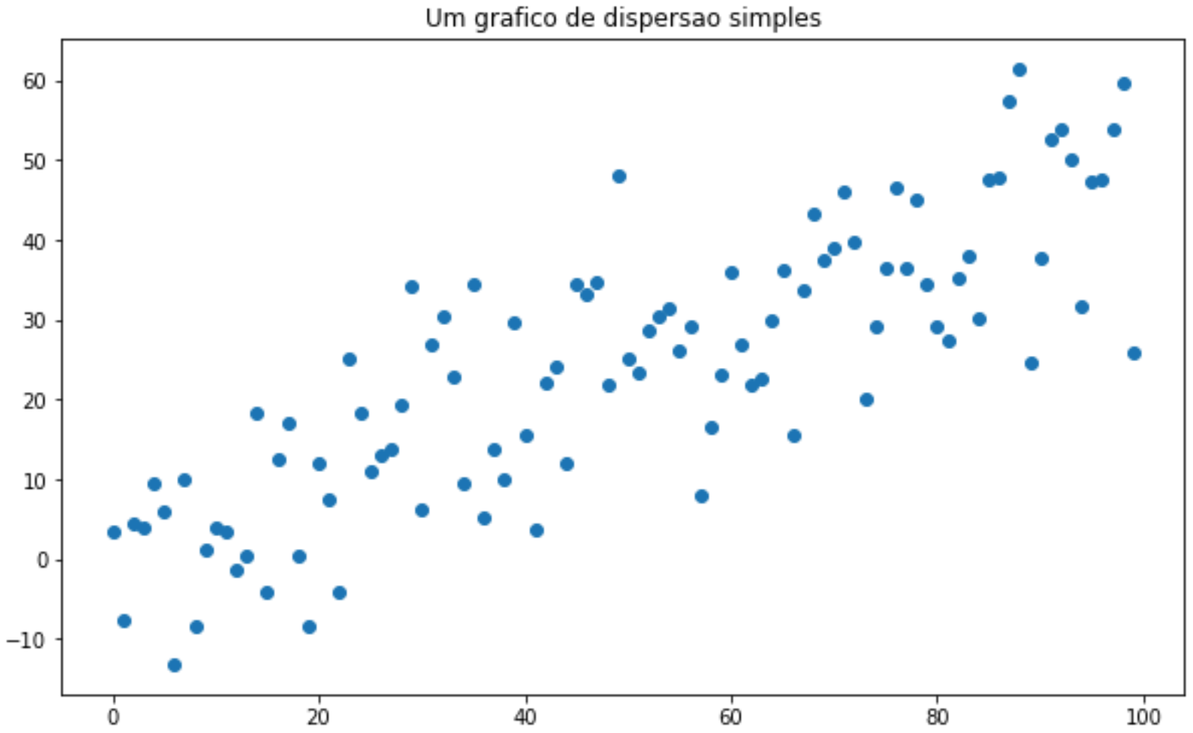
```
>>> X = np.array([0., 0., 1., 1., 1., 2., 2., 3.,
                  4., 6., 7., 7., 7., 7., 7., 7., 7., 7., 7., 7.,
                  8., 8., 8., 9.])

>>> plt.hist(X, bins = 5)
>>> # automaticamente agrega os
>>> # valores 0-1, 2-3, 4-5, 6-7, 8-9
```



Matplotlib

- Gráfico de Dispersão

```
>>> plt.scatter(x, y)
```



Matplotlib

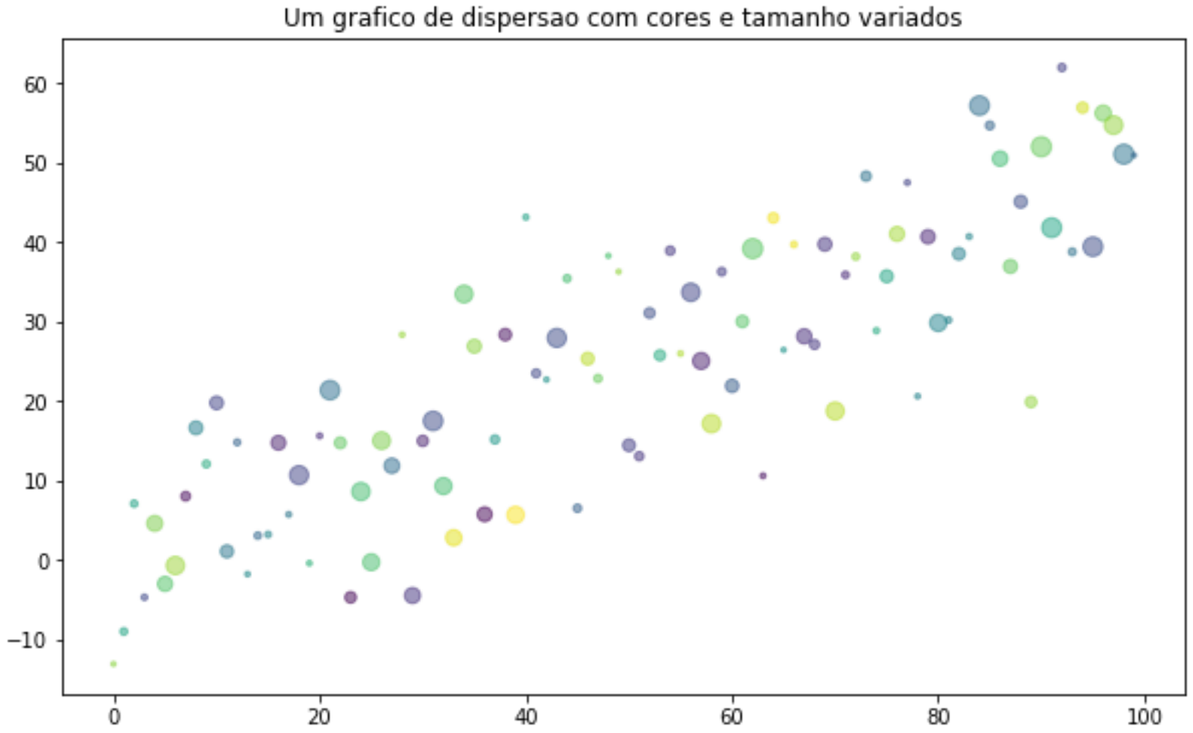
- Gráfico de Dispersão

```
>>> # dados gerados aleatoriamente, para exemplificar
>>> num_points = 100
>>> gradient = 0.5
>>> x = np.array(range(num_points))
>>> y = np.random.randn(num_points) * 10 + x*gradient
>>> plt.scatter(x, y)
```

Matplotlib

- Gráfico de Dispersão (Customizado)

```
>>> plt.scatter(x, y)
```



Matplotlib

- Gráfico de Dispersão

```
>>> # escolhe o mapa de cores
>>> plt.rcParams['image.cmap'] = 'viridis'
>>> # cores geradas aleatoriamente do mapa de cores
>>> colors = np.random.rand(num_points)
>>> # definimos o tamanho dos marcadores: (2 + pontos*8)²
>>> size = (2 + np.random.rand(num_points) * 8) ** 2

>>> plt.scatter(x, y, s=size, c=colors, alpha=0.5)
```

Numpy & Matplotlib

Camila Laranjeira, Jefersson A. dos Santos
{camilalaranjeira, jefersson}@dcc.ufmg.br