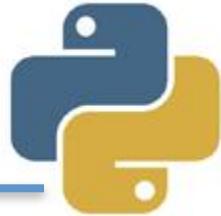


Análise de dados com Numpy e Matplotlib



Márcio Palheta, M.Sc.
marcio.palheta@gmail.com

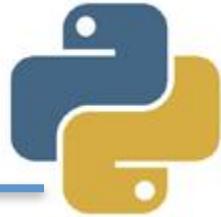


Apresentação

- Programador desde 2000
- Professor de programação desde 2009
- Mestre pelo ICOMP/UFAM – 2013
- Fundador da Buritech – 2014
- Doutorando pelo ICOMP/UFAM
- Pesquisador das áreas: Banco de Dados, Recuperação da Informação, Big Data, Mineração de dados e Aprendizado de Máquina

<https://sites.google.com/site/marciopalheta>

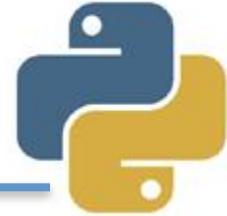




Agenda

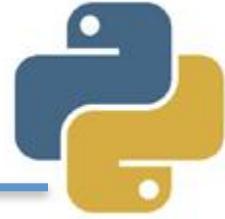
- Relembrando o trabalho com listas
 - De volta às matrizes
 - Instalação e configuração do numpy
 - Representação de dados com numpy
 - Hora dos gráficos
 - Entendendo o matplotlib
-

Revendo o trabalho com listas

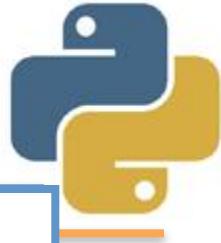


- Listas são **estruturas** muito simples para **armazenamento** de dados
 - Permite **acesso sequencial** ao dados
 - Fácil acesso e **manipulação** de **subconjuntos**
 - Um grande número de **funções PRONTAS**
 - Agora, vamos ao **jupyter notebook**
-

Exercícios de revisão



- Crie funções que recebam uma lista e:
 - devolva o maior elemento
 - devolva o menor elemento
 - devolva o seu somatório
 - Um escalar X e devolva a frequência de X na lista
 - devolva a lista invertida
-

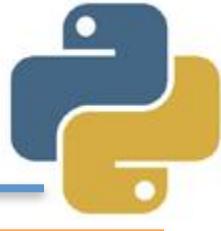


Exercício 1

Criando listas

```
# -*- coding: UTF-8 -*-
# Definição e inicialização
lista = [10, 20, 30, 40, 50, 30, 70, 80]
print type(lista), lista
lista2 = range(10)
print 'De 0 a 10', lista2
lista3 = range(3,10)
print 'De 3 a 10', lista3
lista4 = range(3,10, 2)
print 'De 3 a 10, passo=2', lista4
lista5 = [x*2 for x in range(8)]
print 'list comprehensions', lista5

<type 'list'> [10, 20, 30, 40, 50, 30, 70, 80]
De 0 a 10 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
De 3 a 10 [3, 4, 5, 6, 7, 8, 9]
De 3 a 10, passo=2 [3, 5, 7, 9]
list comprehensions [0, 2, 4, 6, 8, 10, 12, 14]
```



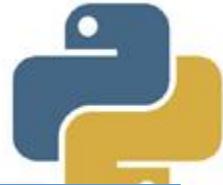
Exercício 2

Percorrendo listas

```
lista = [50, 70, 80, 40, 10, 30, 20, 30]
# Percorrer lista obtendo índice e valor
for indice, valor in enumerate(lista[:3]):
    print "lista[%d]=%d" % (indice, valor),
print '\n',lista
print 'lista invertida com passo do slicing = -1'
print [x for x in lista[::-1]]
print 'lista invertida com reversed()'
print [x for x in reversed(lista)]
print 'lista ordenada\n', [x for x in sorted(lista)]
```

```
lista[0]=50 lista[1]=70 lista[2]=80
[50, 70, 80, 40, 10, 30, 20, 30]
lista invertida com passo do slicing = -1
[30, 20, 30, 10, 40, 80, 70, 50]
lista invertida com reversed()
[30, 20, 30, 10, 40, 80, 70, 50]
lista ordenada
[10, 20, 30, 30, 40, 50, 70, 80]
```

Exercício 3



Trabalhando com referências

```
lista = [10, 20, 30, 40, 50, 60, 70, 80]

# Enviando cópia da referência
lista_ref = lista

print 'Antes da alteração\n', lista, '\n', lista_ref
lista_ref[0] = 300
print 'Depois da alteração\n', lista, '\n', lista_ref
```

Antes da alteração

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

Depois da alteração

```
[300, 20, 30, 40, 50, 60, 70, 80]
```

```
[300, 20, 30, 40, 50, 60, 70, 80]
```

Exercício 4



Trabalhando com valores

```
lista = [10, 20, 30, 40, 50, 60, 70, 80]

# Enviando cópia dos dados
lista_copia = lista[:]

print 'Antes da alteração\n', lista, '\n', lista_copia
lista_copia[0] = 300
print 'Depois da alteração\n', lista, '\n', lista_copia
```

Antes da alteração

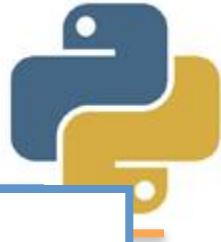
```
[10, 20, 30, 40, 50, 60, 70, 80]
```

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

Depois da alteração

```
[10, 20, 30, 40, 50, 60, 70, 80]
```

```
[300, 20, 30, 40, 50, 60, 70, 80]
```

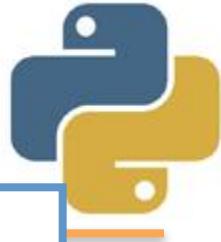


Exercício 5

Listas aleatórias

```
import random
lista = [50, 70, 80, 40, 10, 30, 20, 30]
print('lista original\n', lista)
random.shuffle(lista)
print('lista embaralhada\n', lista)
print('Pegando um número aleatório da lista:', random.choice(lista))
print('Amostra aleatória de 4 elementos')
print(random.sample(lista, 4))
print('Lista gerada com 10 números aleatórios(entre 0 e 100)')
print(random.sample(range(0, 100), 10))
```

```
lista original
[50, 70, 80, 40, 10, 30, 20, 30]
lista embaralhada
[40, 70, 10, 30, 80, 20, 50, 30]
Pegando um número aleatório da lista: 20
Amostra aleatória de 4 elementos
[30, 80, 40, 20]
Lista gerada com 10 números aleatórios(entre 0 e 100)
[66, 68, 37, 21, 17, 85, 11, 95, 25, 4]
```



Exercício 6

Slicing em listas

```
lista = [50, 70, 80, 40, 10, 30, 20, 30]
print 'Coordenada 1:', lista[1]
print 'Elementos dos índices de 1 a 4:', lista[1:5]
print 'Elementos até o índice 4:', lista[:5]
print 'Elementos a partir do índice 5:', lista[5:]
print '5 últimos elementos', lista[-5:]
print 'Começando no índice 1, de 2 em 2:', lista[1::2]
print 'Elementos de índice par', lista[::2]
print 'Elementos de índice ímpar', lista[1::2]
```

Coordenada 1: 70

Elementos dos índices de 1 a 4: [70, 80, 40, 10]

Elementos até o índice 4: [50, 70, 80, 40, 10]

Elementos a partir do índice 5: [30, 20, 30]

5 últimos elementos [40, 10, 30, 20, 30]

Começando no índice 1, de 2 em 2: [70, 40, 30, 30]

Elementos de índice par [50, 80, 10, 20]

Elementos de índice ímpar [70, 40, 30, 30]



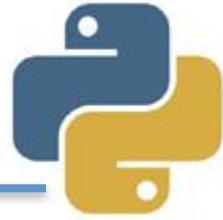
Exercício 7

Listas e funções

```
lista = [50, 70, 80, 40, 10, 30, 20, 30]
print 'somatório:', sum(lista)
print 'frequência de 30:', lista.count(30)
print 'Maior elemento:', max(lista)
print 'Menor elemento:', min(lista)
print 'Juntando duas listas e formando pares de elementos:'
lista = zip(range(0, 5), range(5, 10))
print lista
print 'Separando os elementos de uma lista de forma intercalada:'
lista = range(0, 10)
intercaladas = lista[::2], lista[1::2]
print lista, '\n', intercaladas

print 'Transformando uma lista de strings em uma string CSV:'
lista = ["Curso", "de", "data", "sciense", "em", 'python']
csv_values = ';'.join(lista)
print csv_values
```

Resultado no notebook



```
print 'Transformando uma lista de strings em uma string CSV:'  
lista = ["Curso", "de", "data", "sciense", "em", 'python']  
csv_values = ';' .join(lista)  
print csv_values
```

somatório: 330

frequência de 30: 2

Maior elemento: 80

Menor elemento: 10

Juntando duas listas e formando pares de elementos:

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

Separando os elementos de uma lista de forma intercalada:

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

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

Transformando uma lista de strings em uma string CSV:

```
Curso;de;data;sciense;em;python
```



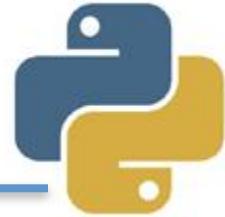
Exercício 8

Aplicando funções a elementos da lista

```
def negativo(valor):
    return valor * -1

lista = range(11)
print 'lista:', lista
print 'dupli:', lista * 2
print 'Lista de nros pares ao quadrado',[x*2 for x in lista if x % 2 == 0]
print 'Gerando uma lista de LISTAS'
print '\t',[[s.capitalize(), s.upper(), len(s)] for s in ['um', 'dois', 'tres']]
print 'Gerando uma lista de TUPLAS'
print '\t',[((s.capitalize(), s.upper(), len(s))) for s in ['um', 'dois', 'tres']]
print 'função:', map(negativo, lista)
print 'lambda:', map(lambda x: x*3, lista)
print 'Filtrando os elementos de uma lista, de acordo com um critério:'
def criterio(x):
    return x >= 0
lista = range(-5, 5)
print 'lista sem filtro, ', lista
print 'lista COM filtro, ', filter(criterio, lista)
```

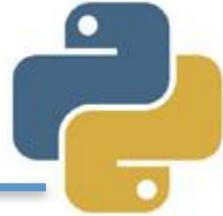
Resultado no console



```
lista = range(-5, 5)
print 'lista sem filtro, ', lista
print 'lista COM filtro, ', filter(criterio, lista)
```

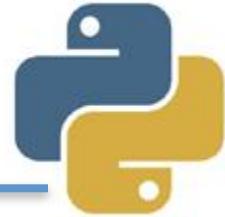
```
lista: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
dupli: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Lista de nros pares ao quadrado [0, 4, 8, 12, 16, 20]
Gerando uma lista de listas
[[['Um', 'UM', 2], ['Dois', 'DOIS', 4], ['Tres', 'TRES', 4]]]
Gerando uma lista de TUPLAS
[('Um', 'UM', 2), ('Dois', 'DOIS', 4), ('Tres', 'TRES', 4)]
função: [0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
lambda: [0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
Filtrando os elementos de uma lista, de acordo com um critério:
lista sem filtro, [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
lista COM filtro, [0, 1, 2, 3, 4]
```

Exercício 9



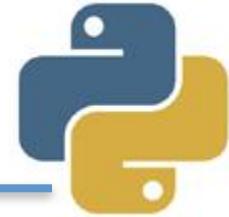
- Crie funções que recebam uma matriz e:
 - devolva o maior elemento
 - devolva o menor elemento
 - devolva o seu somatório
 - Um escalar X e devolva a frequência de X na matriz
-

Exercício 10



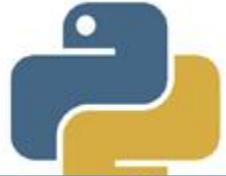
- Crie funções que recebam duas matrizes A e B e um escalar X:
 - Devolva a matriz resultante de $A + B$
 - Devolva a matriz resultante de $A * X$
 - Devolva a transposta de A
 - Devolva uma lista com a 1^a coluna de A
-

Numpy



- Biblioteca para **data science** em python
 - Tem como principal **objetivo**, otimizar o uso de **arrays** multidimensionais
 - Processamento **mais rápido** do que as listas tradicionais
 - Muitas **funções PRONTAS** para uso
-

Exercício 11: Numpy



```
In [13]: # -*- coding: UTF-8 -*-
import numpy as np
```

```
lista = [10, 20, 30, 40, 50, 60]
np_array = np.array(lista)
type(np_array), array
```

```
Out[13]: (numpy.ndarray, array([10, 20, 30, 40, 50, 60]))
```

```
In [11]: sum(np_array)
```

```
Out[11]: 210
```

```
In [12]: np_array[2:5]
```

```
Out[12]: array([30, 40, 50])
```

Exercício 12: matrizes



```
mat = numpy.array([[1,2, 3], [4, 5, 6], [7, 8, 9]])  
print type(mat), '\n', mat
```

```
<type 'numpy.ndarray'>  
[[1 2 3]  
 [4 5 6]  
 [7 8 9]]
```

```
# Acessando uma linha e uma célula da matriz  
print 'mat[1]:', mat[1], ' - mat[1][1]:', mat[1][1]
```

```
mat[1]: [4 5 6] - mat[1][1]: 5
```

```
# Acessando uma coluna  
print 'mat[:,0]:', mat[:, 0]
```

```
mat[:,0]: [1 4 7]
```

Exercício 13: Transposta

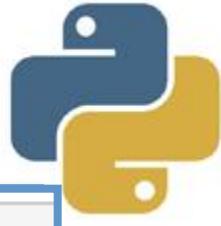


```
mat = numpy.array([[1,2, 3], [4, 5, 6], [7, 8, 9]])
print mat
# Matriz transposta - Linha vira coluna
print 'Transposta:\n', mat.transpose()
```

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

Transposta:

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



Exercício 14: operações

```
In [42]: m1 = numpy.array([[1, 2], [3, 4]])
print m1
m2 = numpy.array([[5, 6], [7, 8]])
print m2
m1 + m2
```

```
[[1 2]
 [3 4]]
[[5 6]
 [7 8]]
```

```
Out[42]: array([[ 6,  8],
 [10, 12]])
```

```
In [41]: m1 - m2
```

```
Out[41]: array([[-4, -4],
 [-4, -4]])
```

Exercício 15: funções



```
In [48]: # Soma dos valores do vetor  
m3 = numpy.array([3, 5, 8, 4])  
m3.sum()
```

Out[48]: 20

```
In [49]: # Indice do maior elemento  
m3.argmax()
```

Out[49]: 2

```
In [50]: # Indice do menor elemento  
m3.argmin()
```

Out[50]: 0



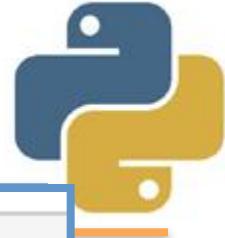
Exercício 16: mais funções

In [53]:

```
import numpy as np
matrix = np.array([[1,2, 3], [4, 5, 6], [7, 8, 9]])
print matrix
# media dos elementos
print 'matrix.mean():',matrix.mean()
# diagonal
print 'matrix.diagonal():',matrix.diagonal()
# dimensao
print 'matrix.ndim:',matrix.ndim
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
matrix.mean(): 5.0
matrix.diagonal(): [1 5 9]
matrix.ndim: 2
```

Exercício 17: Slicing



```
In [58]: import numpy as np  
  
lista = [80, 20, 40, 50, 90]  
a = np.array(lista)  
a[1:5]
```

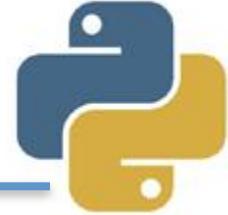
```
Out[58]: array([20, 40, 50, 90])
```

```
In [60]: a[::-2]
```

```
Out[60]: array([80, 40, 90])
```

```
In [64]: a[-3:]
```

```
Out[64]: array([40, 50, 90])
```

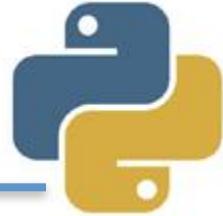


Exercício 18: por referência

```
In [72]: lista = [80, 20, 40, 50, 90]
a = np.array(lista)
b = a[:]
print 'array a\n',a
print 'array b\n',b
print '**** Alterando b[0] = 100'
b[0] = 100
print 'array a\n',a
print 'array b\n',b
```

```
array a
[80 20 40 50 90]
array b
[80 20 40 50 90]
**** Alterando b[0] = 100
array a
[100 20 40 50 90]
array b
[100 20 40 50 90]
```

Exercício 19: por valor



```
In [73]: lista = [80, 20, 40, 50, 90]
          a = np.array(lista)
          b = a.copy()
          print 'array a\n',a
          print 'array b\n',b
          print 'Alterando b[0] = 100'
          b[0] = 100
          print 'array a\n',a
          print 'array b\n',b
```

```
array a
[80 20 40 50 90]
array b
[80 20 40 50 90]
Alterando b[0] = 100
array a
[80 20 40 50 90]
array b
[100 20 40 50 90]
```



Exercício 19: dimensões

```
In [84]: matriz_base = np.arange(0,15)
print 'Arranjo inicial\n', matriz_base
mat_35 = np.reshape(matriz_base, (3, 5))
print 'Matriz 3x5\n', mat_35
mat_53 = np.reshape(matriz_base, (5, 3))
print 'Matriz 5x3\n', mat_53
```

Arranjo inicial

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 ]
```

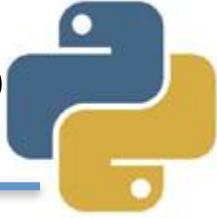
Matriz 3x5

```
[[ 0  1  2  3  4 ]
 [ 5  6  7  8  9 ]
 [10 11 12 13 14 ]]
```

Matriz 5x3

```
[[ 0  1  2 ]
 [ 3  4  5 ]
 [ 6  7  8 ]
 [ 9 10 11 ]
 [12 13 14 ]]
```

Exercício 20: numpy ou listas?



- Qual é mais rápido?

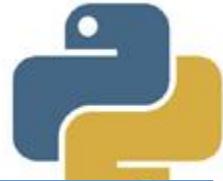
```
In [56]: # Implemente a soma para muitos elementos
soma = 0
for i in range(1, 111111119):
    soma += i
print 'soma', soma
```

```
soma 6172840327160521
```

```
In [57]: # Use o numpy para somar muitos elementos
print 'np', np.arange(1, 111111119).sum()
```

```
np 6172840327160521
```

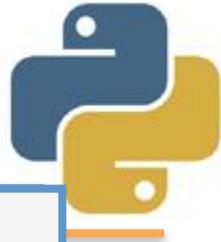
Exercício 21: pontos



```
[90]: import numpy as np
# Pontos de clientes de um programa de fidelidade
joao = [20, 30, 40, 15]
jose = [100, 50, 40, 60]
maria = [80, 90, 47, 30]
anna = [22, 40, 30, 27]

pontos = np.array([joao, jose, maria, anna])

print pontos
print 'pontos[0]:', pontos[0]
print 'pontos.item(5):', pontos.item(5)
print 'Pontuação média:', pontos.mean()
print 'Mediana:', np.median(pontos)
print 'Maior pontuação:', pontos.max()
print 'Menor pontuação:', pontos.min()
```

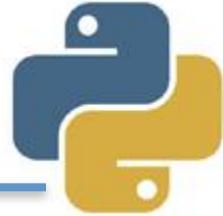


Resultado no notebook

```
print pontos
print 'pontos[0]:', pontos[0]
print 'pontos.item(5):', pontos.item(5)
print 'Pontuação média:', pontos.mean()
print 'Mediana:', np.median(pontos)
print 'Maior pontuação:', pontos.max()
print 'Menor pontuação:', pontos.min()
```

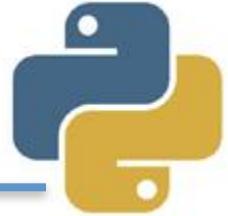
```
[[ 20  30  40  15]
 [100  50  40  60]
 [ 80  90  47  30]
 [ 22  40  30  27]]
pontos[0]: [20 30 40 15]
pontos.item(5): 50
Pontuação média: 45.0625
Mediana: 40.0
Maior pontuação: 100
Menor pontuação: 15
```

Matplotlib



- Biblioteca para **data science** em python, usada para visualização de dados
 - Pacote **muito usado** para a **visualização** de gráficos **2D**
 - **Fácil** de usar, permite que o cientista de dados gere **versões diferentes** de visualização
-

Exercício 22: 1ª Plotagem

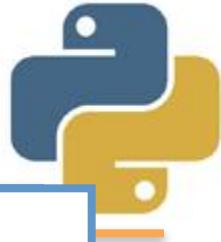


```
# -*- coding: UTF-8 -*-
```

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

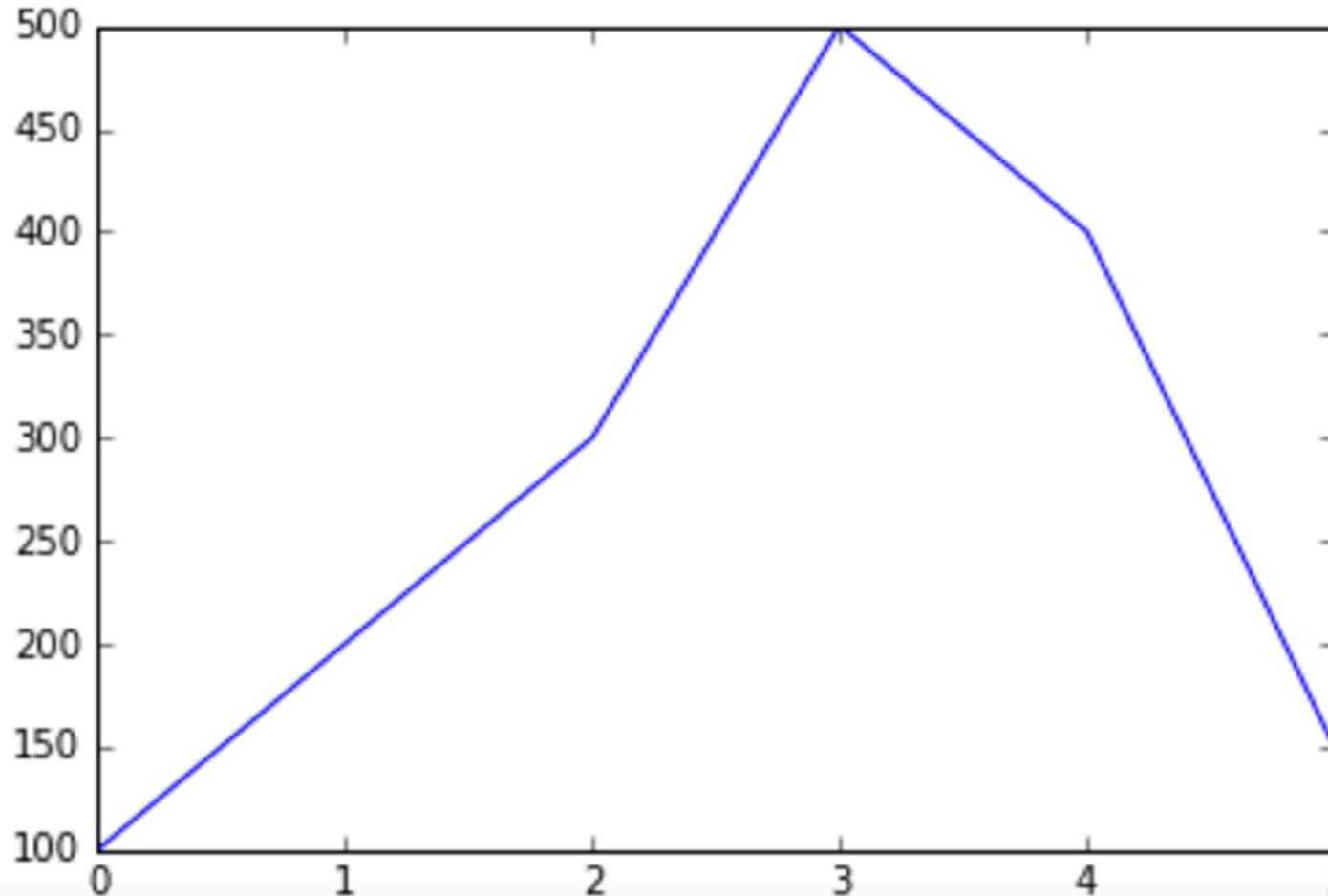
pontuacao = np.array([100, 200, 300, 500, 400, 150])
print(pontuacao)
plt.plot(pontuacao)
```

```
[100 200 300 500 400 150]
```

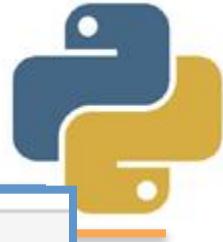


Resultado no notebook

```
[<matplotlib.lines.Line2D at 0x1167732d0>]
```

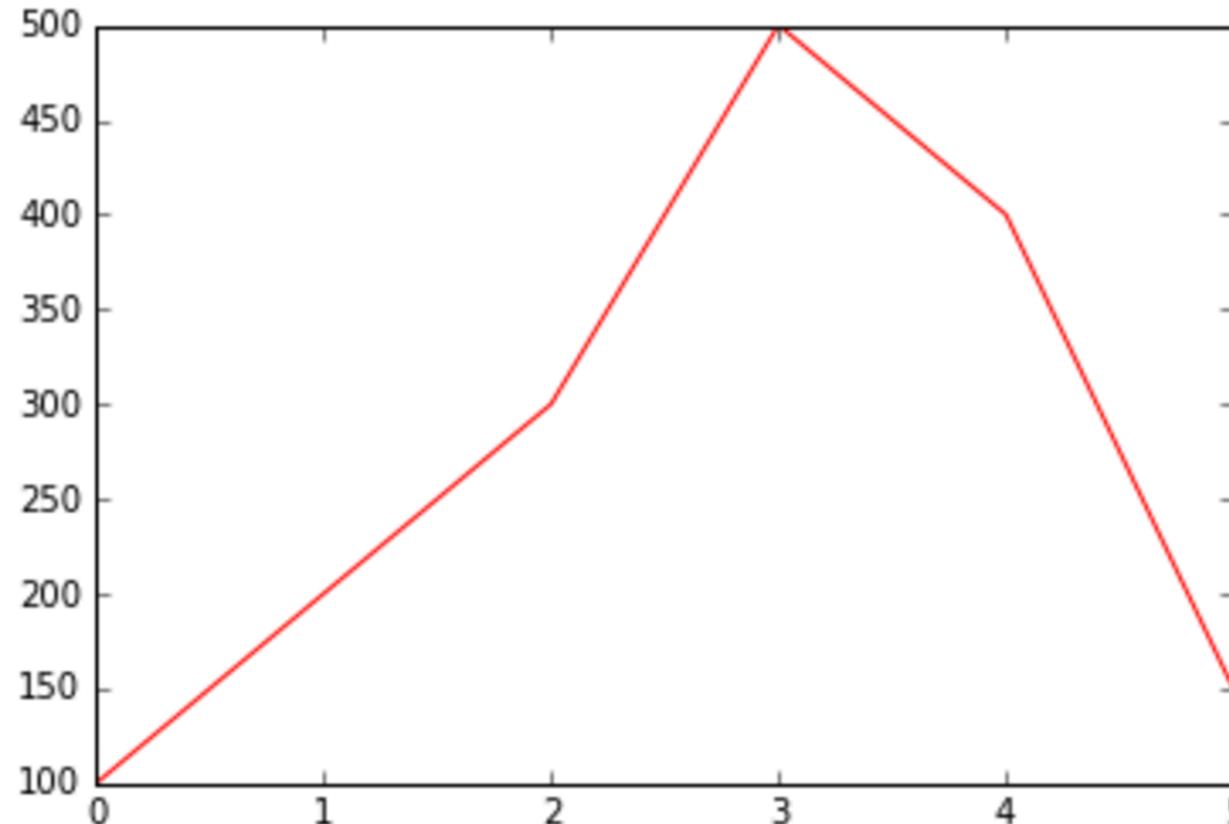


Exercício 23: Cores



```
In [47]: plt.plot(pontuacao, c='red')
```

```
Out[47]: [<matplotlib.lines.Line2D at 0x11668b6d0>]
```

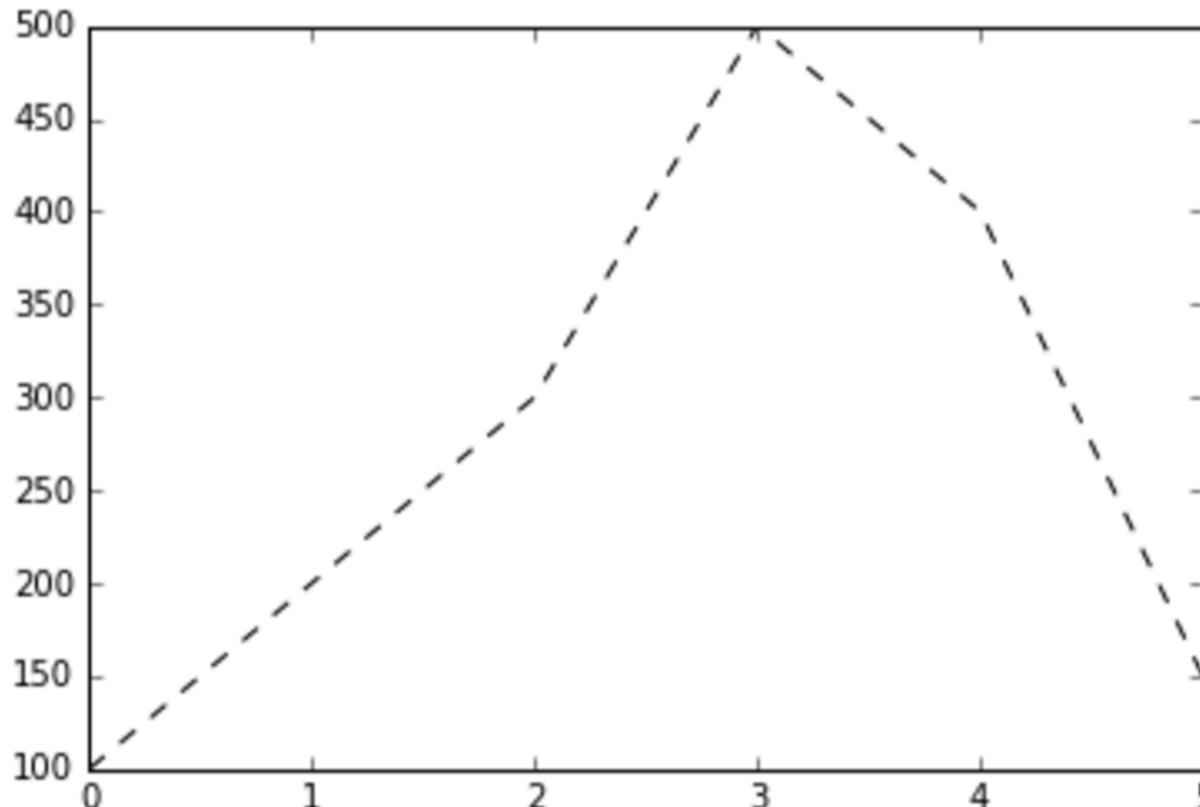


Exercício 24: estilo da linha



```
In [48]: plt.plot(pontuacao, c='black', ls='--')
```

```
Out[48]: [<matplotlib.lines.Line2D at 0x116327e50>]
```

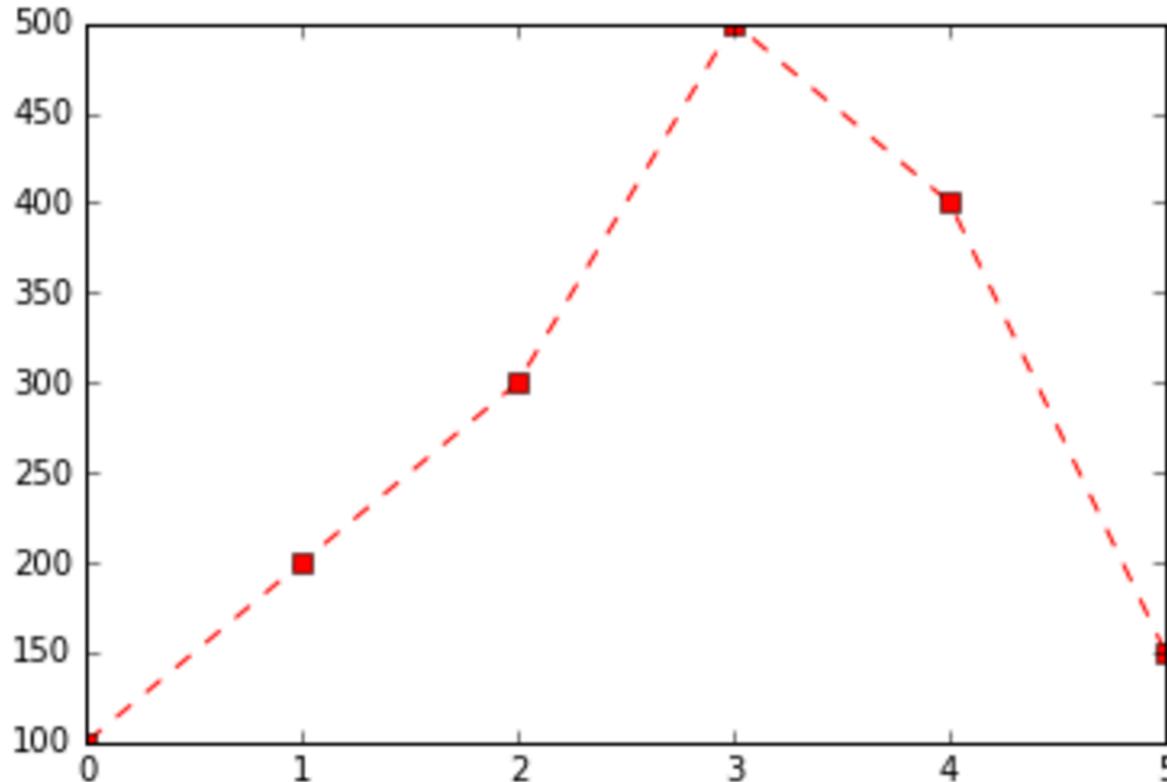




Exercício 25: Marcações

```
In [49]: plt.plot(pontuacao, c='red', ls='--', marker='s')
```

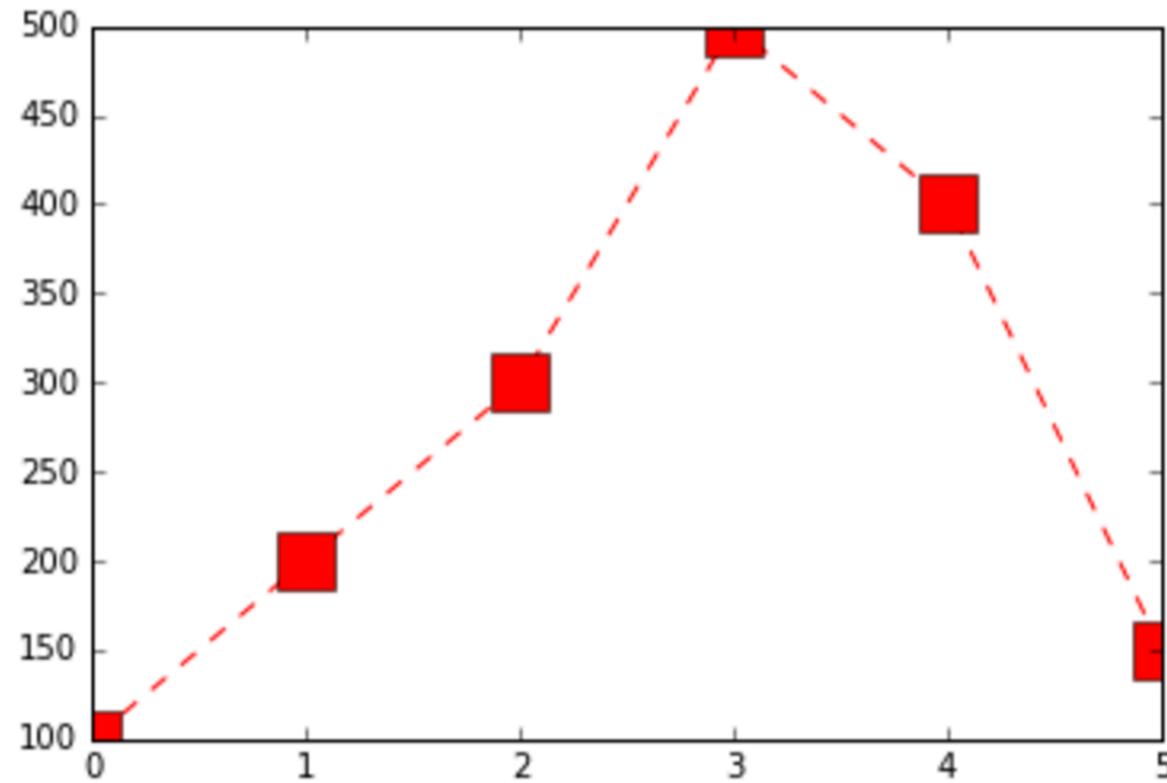
```
Out[49]: [<matplotlib.lines.Line2D at 0x115bcfe10>]
```

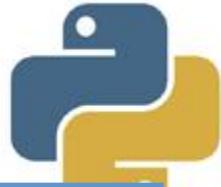




Exercício 26: Pontos maiores

```
In [50]: plt.plot(pontuacao, c='red', ls='--', marker='s', ms='18')  
plt.show()
```

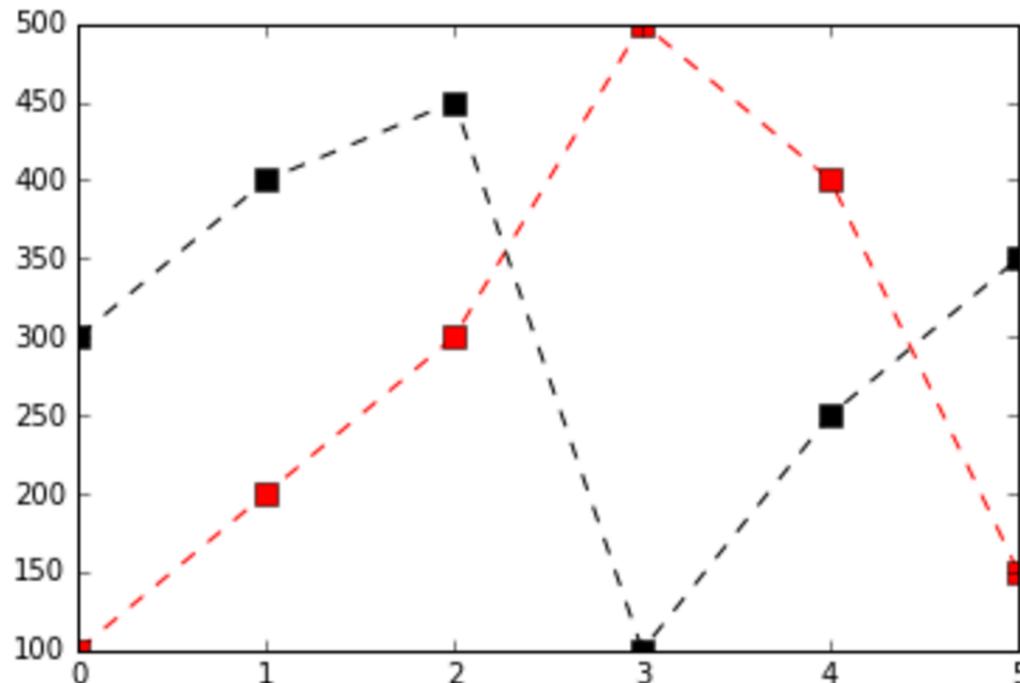




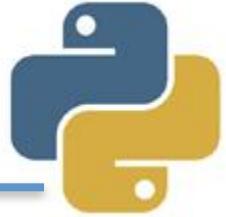
Exercício 27: várias linhas

```
In [51]: pontuacao2 = np.array([300, 400, 450, 100, 250, 350])
plt.plot(pontuacao, c='red', ls='--', marker='s', ms='8')
plt.plot(pontuacao2, c='black', ls='--', marker='s', ms='8')
```

```
Out[51]: [
```

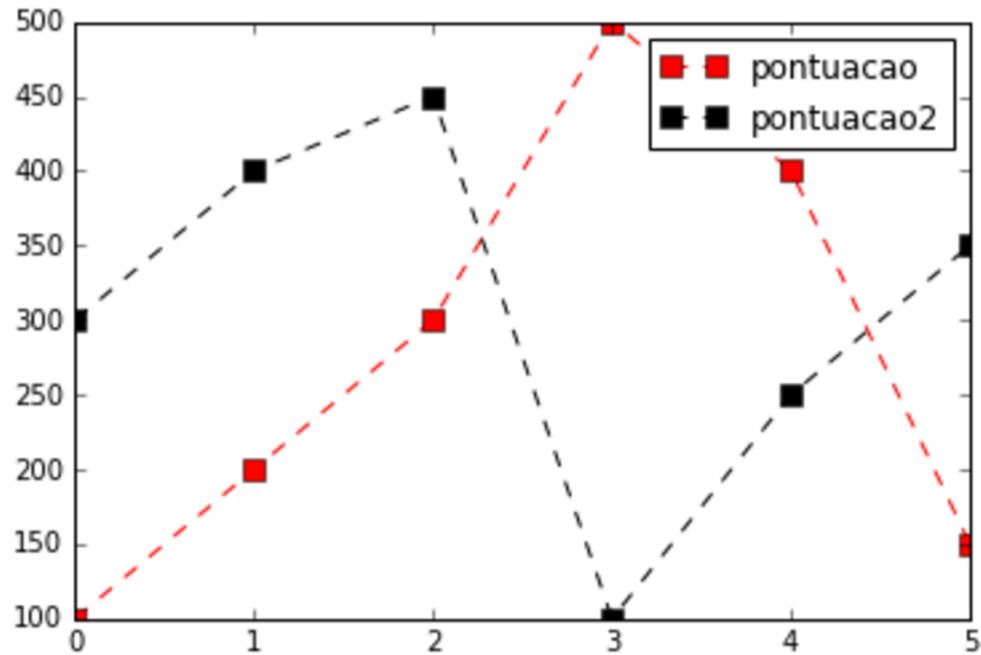


Exercício 28: legenda



Colocando LEGENDA

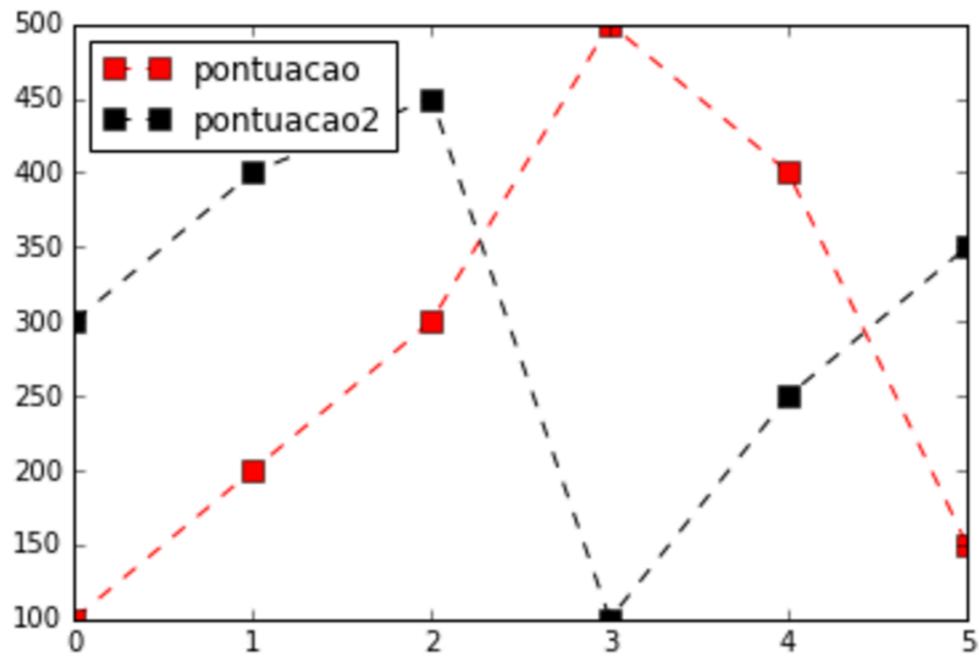
```
plt.plot(pontuacao, c='red', ls='--', marker='s', ms='8', label='pontuacao')
plt.plot(pontuacao2, c='black', ls='--', marker='s', ms='8', label='pontuacao2')
plt.legend()
plt.show()
```



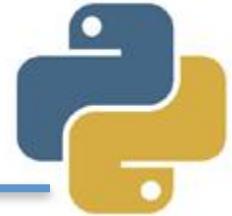
Exercício 29: posição da legenda



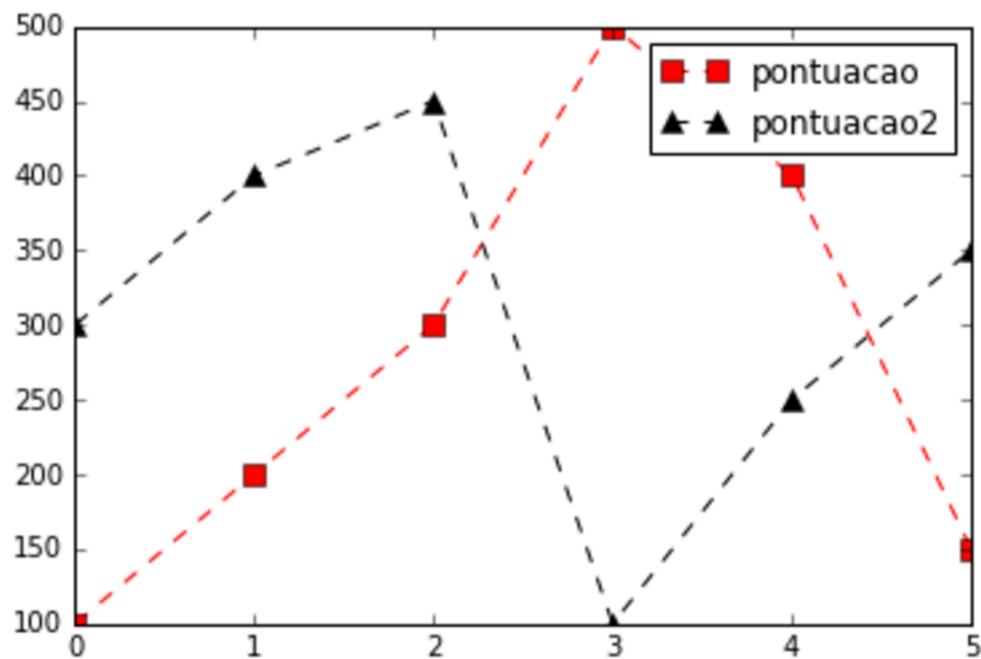
```
plt.plot(pontuacao, c='red', ls='--', marker='s', ms=8, label='pontuacao')
plt.plot(pontuacao2, c='black', ls='--', marker='s', ms=8, label='pontuacao2')
# Alterando a posição da legenda
plt.legend(loc='upper left')
plt.show()
```



Exercício 30: outros marcadores



```
plt.plot(pontuacao, c='red', ls='--', marker='s', ms='8', label='pontuacao')
plt.plot(pontuacao2, c='black', ls='--', marker='^', ms='8', label='pontuacao2')
plt.legend()
plt.show()
```

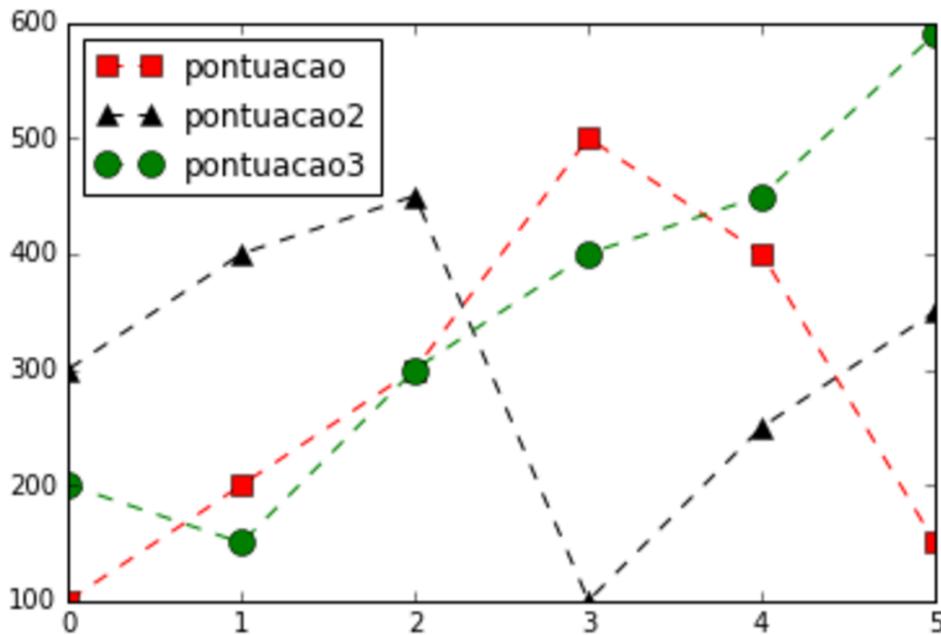




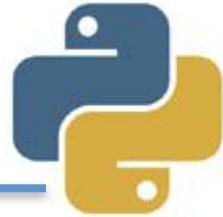
Exercício 31: mais dados

Incluindo mais uma série dados

```
pontuacao3 = np.array([200, 150, 300, 400, 450, 590])
plt.plot(pontuacao, c='red', ls='--', marker='s', ms=8, label='pontuacao')
plt.plot(pontuacao2, c='black', ls='--', marker='^', ms=8, label='pontuacao2')
plt.plot(pontuacao3, c='green', ls='--', marker='o', ms=10, label='pontuacao3')
plt.legend(loc='upper left')
plt.show()
```



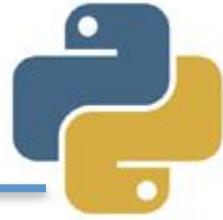
Exercício 32: notas.csv



- Carregue as **notas** de **alunos** que estão no arquivo **notas.csv**
- Exiba um **gráfico** com as **notas** e outro com as **médias**(crescente) dos alunos

notas.csv	
1	Nome;av1;av2;av3;av4;av5
2	Anna;8.5;7.6;9.0;7.8;10.0
3	Maria;7.5;4.9;8.5;8.7;9.0
4	Fernanda;10.0;9.1;8.8;9.0;8.0
5	

Exercício 32: notas.csv

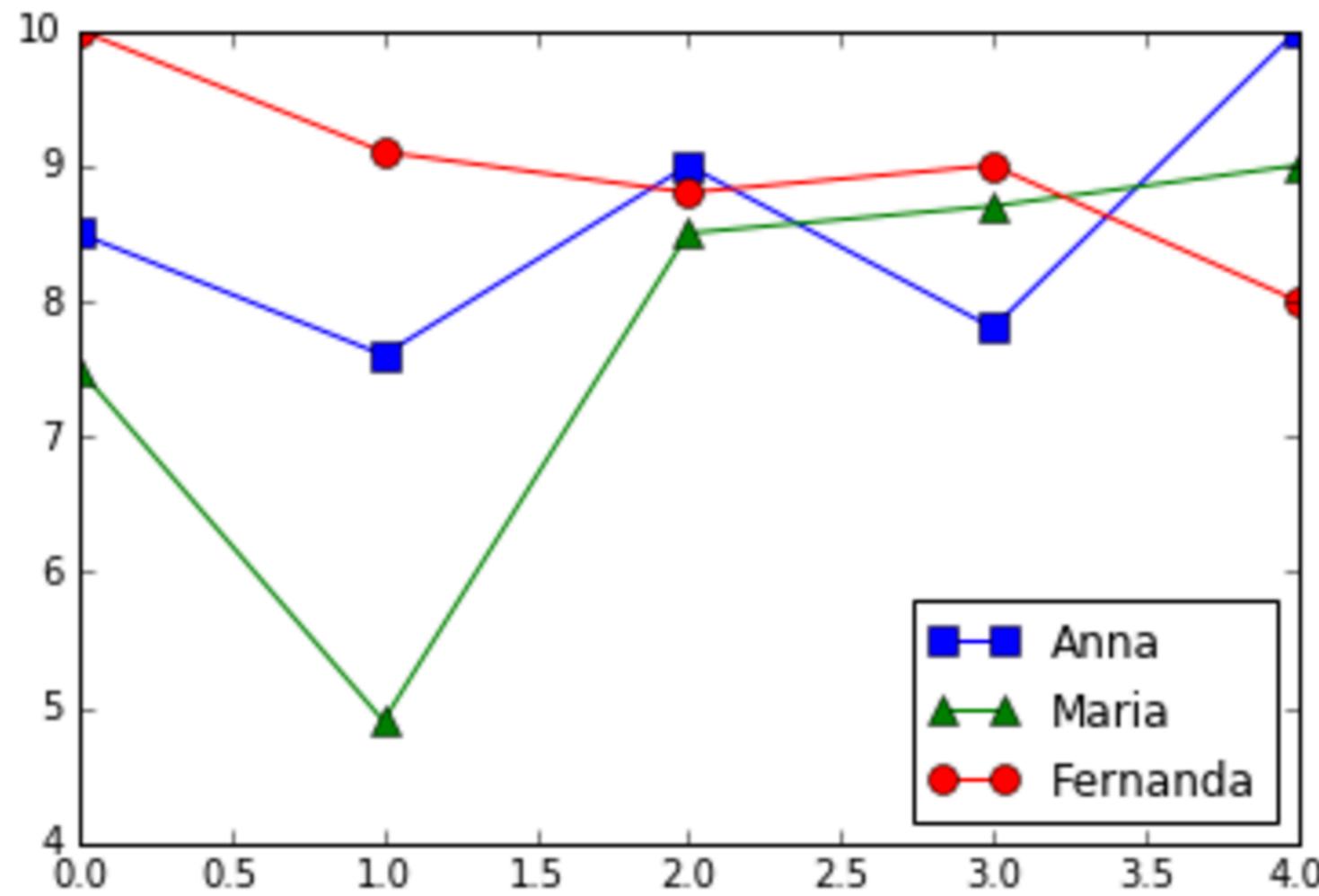
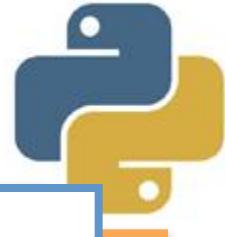


```
# -*- coding: UTF-8 -*-
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def pega_notas(nome_arquivo):
    arquivo = open(nome_arquivo, 'r')
    lista = []
    for linha in arquivo:
        lista.append(linha.strip().split(';'))
    lista.pop(0)
    notas = None
    nomes, notas = zip(*[(l[0], map(lambda item: float(item), l[1:])) for l in lista])
    return np.array(nomes), np.array(notas)

nomes, notas = pega_notas('notas.csv')
markers=['s', '^', 'o']
for indice, aluno in enumerate(nomes):
    plt.plot(notas[indice], marker=markers[indice], ms='8', label=aluno)
plt.legend(loc='lower right')
plt.show()
```

Resultado no notebook



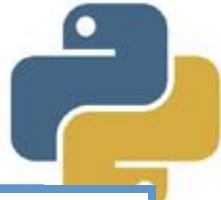
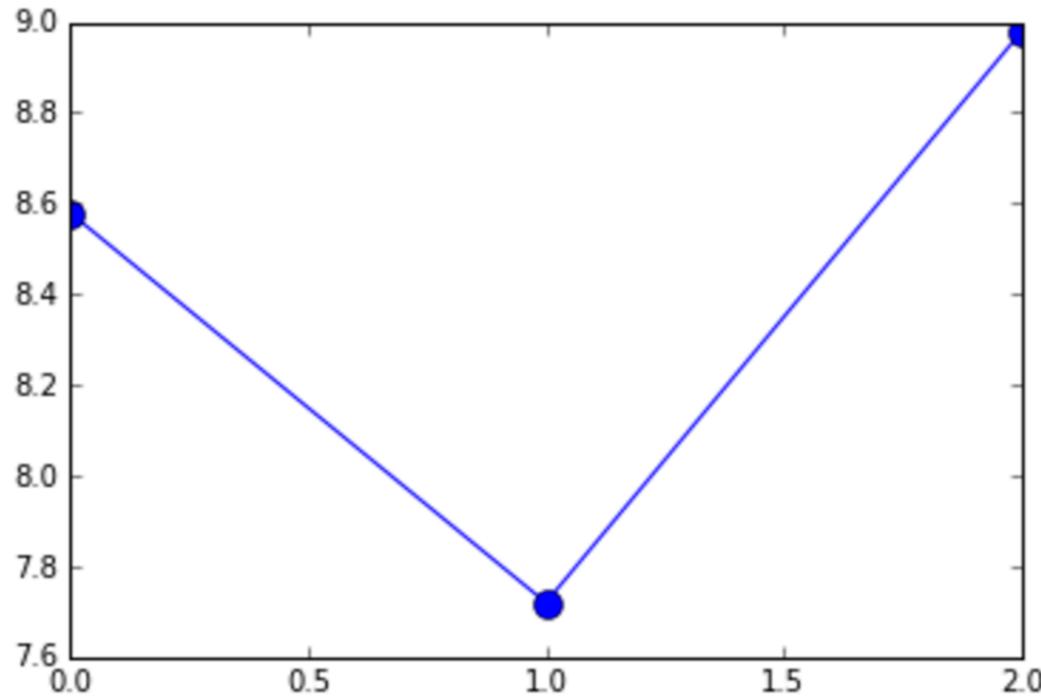
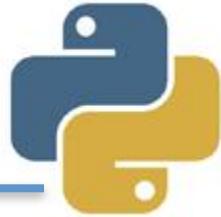


Gráfico com as médias

```
medias = map(lambda item: item.mean(), notas)
print 'médias', medias
plt.plot(medias, marker='o', ms='10', label=aluno)
plt.show()
```

médias [8.580000000000001, 7.719999999999989, 8.980000000000004]





Inserindo elementos

- Em **numpy**, podemos usar a função **insert** para **inclusão** de novos dados
 - Contudo, os **vetores** em **numpy** possuem **tamanho estático**. Com isso, a função **insert** devolve um **novo vetor**
 - Podemos **inserir** elementos em qualquer **coordenada** ou **dimensão** do vetor
-

Exercício 33: inclusão de dados



```
In [194]: import numpy as np  
  
vetor = np.array([10, 20, 50, 70, 30])  
print 'Dimensões (eixos):', vetor.ndim  
vetor
```

Dimensões (eixos): 1

Out[194]: array([10, 20, 50, 70, 30])

```
In [195]: vetor = np.insert(vetor, 1, 90)  
vetor
```

Out[195]: array([10, 90, 20, 50, 70, 30])



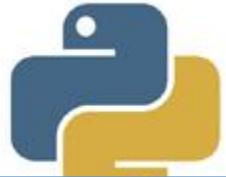
Exercício 34: inclusão em eixos

```
In [202]: # inserir o valor ZERO, na posição 1,  
# eixo 0 == nova LINHA  
matriz = np.insert(matriz, 1, 0, axis=0)  
matriz
```

```
Out[202]: array([[1, 2, 3],  
                  [0, 0, 0],  
                  [4, 5, 6],  
                  [7, 8, 9]])
```

```
In [196]: # inserir o valor -1, na posição 1,  
# eixo 1 == nova COLUNA  
matriz = np.insert(matriz, 1, -1, axis=1)  
matriz
```

```
Out[196]: array([[ 1, -1, -1,  2,  3],  
                  [ 0, -1, -1,  0,  0],  
                  [ 4, -1, -1,  5,  6],  
                  [ 7, -1, -1,  8,  9]])
```



Exercício 34: somando eixos

```
import numpy as np

matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print 'Dimensões (eixos):', matriz.ndim
print matriz
print 'Soma de colunas', matriz.sum(axis=0)
print 'Soma de linhas', matriz.sum(axis=1)
```

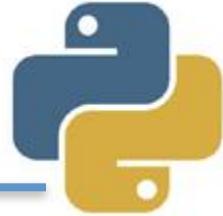
Dimensões (eixos): 2

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

Soma de colunas [12 15 18]

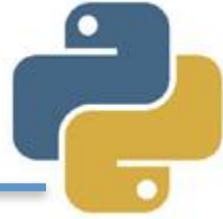
Soma de linhas [6 15 24]

A função np.append()



- Usamos a função **append** para **inclusão** de dados no **final** do arquivo
 - Retorna uma **cópia** do array, com os valores **anexados**
 - Os novos dados devem ter o mesmo **número de dimensões** do array
-

Exercício 35: append()



```
In [214]: import numpy as np
```

```
vetor = np.array([1, 2, 3])
print 'Dimensões (eixos):', vetor.ndim, vetor.__repr__()
```

```
Dimensões (eixos): 1 array([1, 2, 3])
```

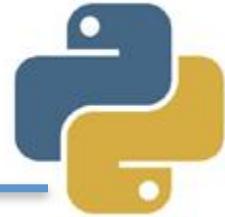
```
In [215]: vetor = np.append(vetor, 4)
print 'Dimensões (eixos):', vetor.ndim, vetor.__repr__()
```

```
Dimensões (eixos): 1 array([1, 2, 3, 4])
```

```
In [216]: vetor = np.append(vetor, [5, 6, 7])
print 'Dimensões (eixos):', vetor.ndim, vetor.__repr__()
```

```
Dimensões (eixos): 1 array([1, 2, 3, 4, 5, 6, 7])
```

Exercício 36: append de linha

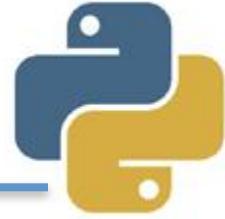


```
In [248]: print 'Inclusão de uma linha'  
np.append(matriz, [[0,0,0]], axis=0)
```

Inclusão de uma linha

```
Out[248]: array([[1, 2, 3],  
                  [4, 5, 6],  
                  [7, 8, 9],  
                  [0, 0, 0]])
```

Exercício 37: append de coluna



```
In [250]: print 'Inclusão de uma coluna'  
print matriz  
coluna = np.array([[0,0,0]])  
print 'Novos dados:', coluna, coluna.shape
```

```
Inclusão de uma coluna  
[[1 2 3]  
 [4 5 6]  
 [7 8 9]]  
Novos dados: [[0 0 0]] (1, 3)
```

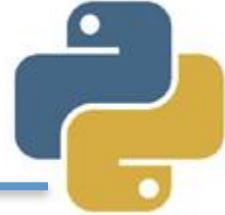
```
In [251]: coluna = np.reshape(coluna, (3, 1))  
print coluna, coluna.shape
```

```
[[0]  
 [0]  
 [0]] (3, 1)
```

```
In [252]: np.append(matriz, coluna, axis=1)
```

```
Out[252]: array([[1, 2, 3, 0],  
 [4, 5, 6, 0],  
 [7, 8, 9, 0]])
```

A função np.delete()



- A função `np.delete` é usada para eliminarmos elementos de um `np.array`
 - Devolve uma `cópia do array`, `sem` os dados excluídos
 - Podemos excluir `linhas` ou `colunas`
 - `np.delete(vetor, índice, eixo)`
 - Eixo `0 == linhas`. Eixo `1 == colunas`.
-

Exercício 38: np.delete



```
import numpy as np

vetor = np.array([10, 20, 30, 40, 50])
print 'Dimensões (eixos):', vetor.ndim, vetor.__repr__()

Dimensões (eixos): 1 array([10, 20, 30, 40, 50])
```

```
np.delete(vetor, 2)
```

```
array([10, 20, 40, 50])
```

Exercício 39: np.delete matriz

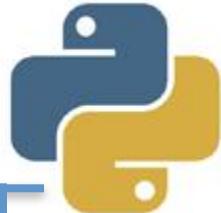


```
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print 'Dimensões (eixos):', matriz.ndim
print matriz
```

```
Dimensões (eixos): 2
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
np.delete(matriz, 1, 0)
array([[1, 2, 3],
       [7, 8, 9]])
```

```
np.delete(matriz, 1, 1)
array([[1, 3],
       [4, 6],
       [7, 9]])
```



Exercício 40: np.delete slice

```
matriz = np.arange(15)
print 'Dimensões (eixos):', matriz.ndim
print matriz
matriz = np.reshape(matriz, (5,3))
print 'Após reshape - Dimensões (eixos):', matriz.ndim
matriz
```

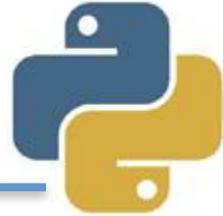
```
Dimensões (eixos): 1
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
Após reshape - Dimensões (eixos): 2
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

```
# Exclui as linhas, começando em 0
# Passo = 2
np.delete(matriz, np.s_[::2], 0)
```

```
array([[ 3,  4,  5],
       [ 9, 10, 11]])
```

A função np.repeat()



- A função `np.repeat` é usada para a criação de arrays com **elementos repetidos**
 - Podemos repetir **todos os itens** do array ou as **colunas** ou as **linhas**
 - Muito usada quando precisamos **aumentar** a **importância** de amostras
-

Exercício 41: Repetição de itens



```
In [10]: # Array com o nro 1 repetido 5x  
         np.repeat(1, 5)
```

```
Out[10]: array([1, 1, 1, 1, 1])
```

```
In [11]: matriz = np.array([[1, 2, 3], [4, 5, 6]])  
        matriz
```

```
Out[11]: array([[1, 2, 3],  
                 [4, 5, 6]])
```

```
In [12]: # repetir cada elemento 2x  
         np.repeat(matriz, 2)
```

```
Out[12]: array([1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6])
```

Exercício 42: linhas e colunas



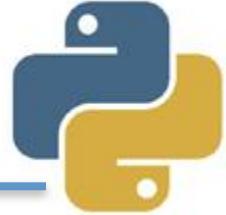
```
In [280]: # repetir cada linha 2x  
np.repeat(matriz, 2, axis=0)
```

```
Out[280]: array([[1, 2, 3],  
                  [1, 2, 3],  
                  [4, 5, 6],  
                  [4, 5, 6]])
```

```
In [282]: # repetir cada coluna 2x  
np.repeat(matriz, 2, axis=1)
```

```
Out[282]: array([[1, 1, 2, 2, 3, 3],  
                  [4, 4, 5, 5, 6, 6]])
```

A função np.tile()



- A função `np.tile()` é usada para criação de arrays com **elementos repetidos**
 - Diferente da `np.repeat()`, `np.tile()` não requer **indicação de eixo** a ser alterado
 - Informamos apenas o **array** e o **número** de repetições sobre cada eixo
-

Exercício 43: np.tile()



```
import numpy as np
```

```
vetor = np.array([1, 2, 3])  
vetor
```

```
array([1, 2, 3])
```

Repetições de elementos
np.tile(vetor, 3)

```
array([1, 2, 3, 1, 2, 3, 1, 2, 3])
```

Exercício 44: np.tile em matrizes



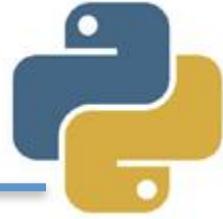
```
In [291]: matriz = np.array([[1, 2, 3],  
                           [4, 5, 6]])  
        matriz
```

```
Out[291]: array([[1, 2, 3],  
                  [4, 5, 6]])
```

```
In [292]: # Repetições de matrizes  
        np.tile(matriz, 2)
```

```
Out[292]: array([[1, 2, 3, 1, 2, 3],  
                  [4, 5, 6, 4, 5, 6]])
```

A função np.array_split()



- A função `np.array_split()` divisão do array em várias partes
 - Devemos informar o `array`, o `número de divisões` e, opcionalmente, o `eixo` para realização do corte
 - Usado quando precisamos `dividir` a base em treino e teste
-

Exercício 45: np.array_split



```
import numpy as np  
vetor = np.arange(10)  
vetor
```

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

```
np.array_split(vetor, 2)
```

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

```
np.array_split(vetor, 3)
```

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



Exercício 46: divisão em linhas

```
matriz = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])  
matriz
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6],  
       [7, 8]])
```

```
itens = np.array_split(matriz, 2, 0)  
for i, item in enumerate(itens):  
    print '%dª matriz\n' % (i+1), item
```

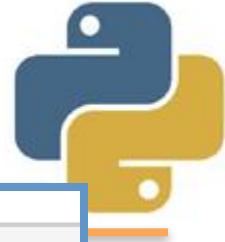
1ª matriz

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

2ª matriz

```
[[5 6]  
 [7 8]]
```

Exercício 47: divisão em colunas



```
itens = np.array_split(matriz, 2, 1)
for i, item in enumerate(itens):
    print '%da matriz\n'%(i+1), item
```

1ª matriz

[[1]

 [3]

 [5]

 [7]]

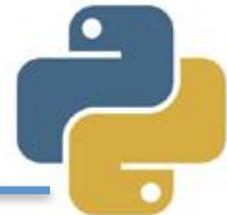
2ª matriz

[[2]

 [4]

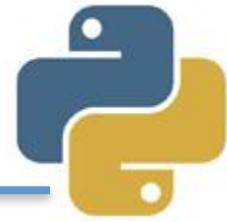
 [6]

 [8]]



Criando arrays de Zeros e Uns

- Em **data science**, vez por outra precisamos trabalhar com **matrizes** ou **vetores** que iniciem com **ZEROs** e **Uns**
 - O **numpy** oferece a função **np.zeros()**, que recebe a **quantidade** de elementos ou quantidade de **linhas** e **colunas** da matriz, conhecida com **shape**
-



Exercício 48: arrays de zeros

```
In [317]: # vetor com 3 zeros  
np.zeros(3)
```

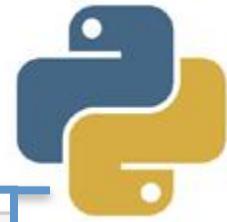
```
Out[317]: array([ 0.,  0.,  0.])
```

```
In [321]: # matriz com 2 linhas e 5 colunas  
np.zeros((2,5))
```

```
Out[321]: array([[ 0.,  0.,  0.,  0.,  0.],  
                  [ 0.,  0.,  0.,  0.,  0.]])
```

```
In [323]: # matriz com 5 linhas e 5 colunas  
np.zeros((5,5))
```

```
Out[323]: array([[ 0.,  0.,  0.,  0.,  0.],  
                  [ 0.,  0.,  0.,  0.,  0.],  
                  [ 0.,  0.,  0.,  0.,  0.],  
                  [ 0.,  0.,  0.,  0.,  0.],  
                  [ 0.,  0.,  0.,  0.,  0.]])
```



Exercício 49: arrays de uns

```
In [327]: # vetor com 5 uns  
np.ones(5)
```

```
Out[327]: array([ 1.,  1.,  1.,  1.,  1.])
```

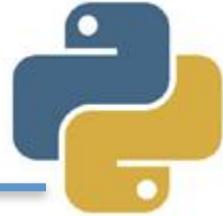
```
In [328]: # Matriz com 2 linhas e 5 colunas  
np.ones((2, 5))
```

```
Out[328]: array([[ 1.,  1.,  1.,  1.,  1.],  
                  [ 1.,  1.,  1.,  1.,  1.]])
```

```
In [326]: # matriz de ordem 5  
np.ones((5, 5))
```

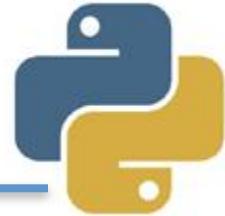
```
Out[326]: array([[ 1.,  1.,  1.,  1.,  1.],  
                  [ 1.,  1.,  1.,  1.,  1.],  
                  [ 1.,  1.,  1.,  1.,  1.],  
                  [ 1.,  1.,  1.,  1.,  1.],  
                  [ 1.,  1.,  1.,  1.,  1.]])
```

Matriz identidade



- A **matriz identidade** é uma matriz **quadrada**, ou seja, tem o **mesmo número** de linhas e colunas
 - Além disso, todos os elementos da **diagonal principal** são iguais a **1** e os **demais** são iguais a **zero**
-

Exercício 50: np.eye()

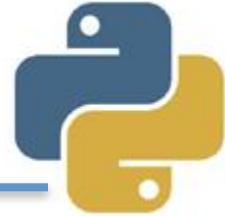


Matriz identidade

```
In [329]: import numpy as np  
# matriz identidade de ordem 5  
np.eye(5)
```

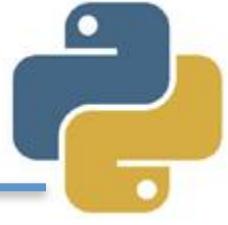
```
Out[329]: array([[ 1.,  0.,  0.,  0.,  0.],  
                  [ 0.,  1.,  0.,  0.,  0.],  
                  [ 0.,  0.,  1.,  0.,  0.],  
                  [ 0.,  0.,  0.,  1.,  0.],  
                  [ 0.,  0.,  0.,  0.,  1.]])
```

Concatenando matrizes



- O Numpy oferece ferramentas que facilitam a **concatenação** de **matrizes**
 - Podemos **concatenar** considerando:
linhas(axis=0) ou **colunas**(axis=1)
 - Quando concatenamos **por linha**, as **matrizes** precisam ter o **mesmo número** de **colunas** e vice-versa
-

Exercício 51: criação



```
In [364]: import numpy as np  
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
b = np.array([[-1, -2, -3]])
```

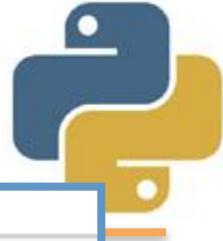
```
In [366]: a
```

```
Out[366]: array([[1, 2, 3],  
                  [4, 5, 6],  
                  [7, 8, 9]])
```

```
In [367]: b
```

```
Out[367]: array([[-1, -2, -3]])
```

Exercício 52: por linha



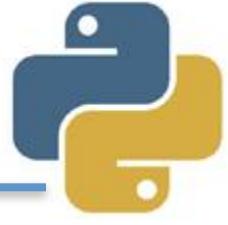
```
In [371]: np.concatenate((a, b))
```

```
Out[371]: array([[ 1,  2,  3],
                  [ 4,  5,  6],
                  [ 7,  8,  9],
                  [-1, -2, -3]])
```

```
In [374]: # Concatenando em linhas
np.concatenate((a, b), axis=0)
```

```
Out[374]: array([[ 1,  2,  3],
                  [ 4,  5,  6],
                  [ 7,  8,  9],
                  [-1, -2, -3]])
```

Exercício 53: por coluna



```
# Concatenando em colunas  
np.concatenate((a, b), axis=1)
```

```
-----  
ValueError                                     Traceback (m  
<ipython-input-382-37ae2212d833> in <module>()  
      1 # Concatenando em colunas  
----> 2 np.concatenate((a, b), axis=1)  
  
ValueError: all the input array dimensions except for
```



Exercício 54: transposta

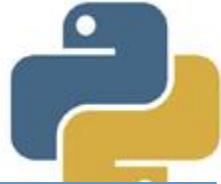
```
# Para concatenar no eixo 1, precisamos da trasposta de B
print a, '\nMatriz b\n', b, '\nTransposta\n', b.T
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Matriz b
[[-1 -2 -3]]
Transposta
[[-1]
 [-2]
 [-3]]
```

```
# Concatenando A com a Transposta de B
np.concatenate((a, b.T), axis=1)
```

```
array([[ 1,  2,  3, -1],
       [ 4,  5,  6, -2],
       [ 7,  8,  9, -3]])
```

Exercício 55: np.random.shuffle



Embaralhando dados de um array

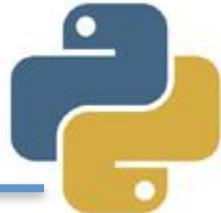
```
import numpy as np
```

```
dados = np.arange(10)  
dados
```

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

```
# embaralhando os dados  
np.random.shuffle(dados)  
dados
```

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



Exercício 56: np.linspace

Arrays uniformemente espaçados

```
# sequencia de 0 a 9  
np.arange(10)
```

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

```
# sequencia de 3 a 13  
np.arange(3, 13)
```

```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

```
# sequencia de 5 a 30, com passo 3  
np.arange(5, 30, 3)
```

```
array([ 5,  8, 11, 14, 17, 20, 23, 26, 29])
```

```
# Dados uniformemente espaçados  
np.linspace(2.0, 3.0, 5)
```

```
array([ 2. ,  2.25,  2.5 ,  2.75,  3. ])
```

Exercício 57: np.unique



Lista de elementos distintos

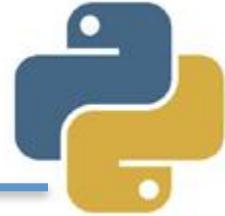
```
matriz = np.array([[1, 3], [3, 2], [1, 2], [4, 5]])  
matriz
```

```
array([[1, 3],  
       [3, 2],  
       [1, 2],  
       [4, 5]])
```

```
# lista de elementos distintos  
np.unique(matriz)
```

```
array([1, 2, 3, 4, 5])
```

Indexação booleana



- O **numpy** permite que façamos **seleção de dados** nos elementos do array
 - Para isso, usamos uma **expressão booleana** para definir o **critério da busca**
 - Além disso, o **numpy** permite que sejam visualizados os **índices da matriz** que **atendem ao critério** definido
-

Exercício 58: indexação



```
In [331]: import numpy as np  
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
matriz
```

```
Out[331]: array([[1, 2, 3],  
                  [4, 5, 6],  
                  [7, 8, 9]])
```

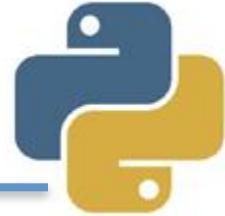
```
In [332]: # Busca por elementos maiores que 5  
matriz[matriz > 5]
```

```
Out[332]: array([6, 7, 8, 9])
```

```
In [333]: # mostrando os índices que atendem ao critério  
[matriz > 5]
```

```
Out[333]: [array([[False, False, False],  
                  [False, False, True],  
                  [ True,  True,  True]], dtype=bool)]
```

Carregando arquivos



- O **numpy** oferece **funções** para facilitar o **carregamento** de dados de arquivos
- Imagine o seguinte arquivo **dados.txt**:

A screenshot of a Mac OS X window titled "dados.txt". The window contains a table with three columns and six rows of data. The columns are labeled "FUNDEV_A", "IBOVESPA", and "IPCA".

FUNDEV_A	IBOVESPA	IPCA
0.1212	0.842	0.3672
0.104	0.5422	0.34551
0.5661	0.932	0.2344
0.762	0.847361	0.1538

Exercício 59: np.loadtxt



Leitura de arquivos com numpy

```
In [345]: import numpy as np
```

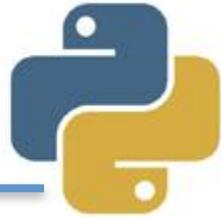
```
dados = np.loadtxt('dados.txt', skiprows=1)
dados
```

```
Out[345]: array([[ 0.1212   ,  0.842    ,  0.3672   ],
       [ 0.104    ,  0.5422   ,  0.34551  ],
       [ 0.5661   ,  0.932    ,  0.2344   ],
       [ 0.762    ,  0.847361,  0.1538   ]])
```

```
In [353]: # Carregamento por colunas
```

```
fundev, ibovespa, ipca = np.loadtxt('dados.txt', skiprows=1, unpack=True)
print 'fundev ', fundev, '\nibovespa', ibovespa, '\nipca ', ipca
```

```
fundev  [ 0.1212  0.104   0.5661  0.762 ]
ibovespa [ 0.842      0.5422     0.932      0.847361]
ipca    [ 0.3672   0.34551   0.2344   0.1538 ]
```



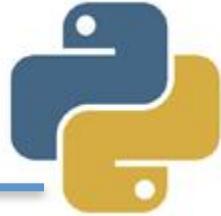
Dados faltantes

- Vez por outra, é possível que **nem todos os dados estejam presentes** em nossos **arquivos de carga**:

A screenshot of a Mac OS X TextEdit window titled "dados_faltantes.txt". The window displays the following data:

FUNDEV_A	IBOVESPA	IPCA
0.1212	0.842	0.3672
MISSING	0.5422	0.34551
0.5661	0.932	MISSING
0.762	0.847361	0.1538

Exercício 60: np.genfromtext



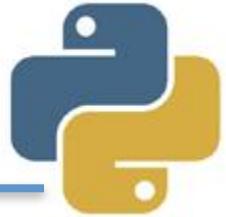
Dados faltantes

```
In [356]: dados = np.genfromtxt('dados_faltantes.txt', skip_header=1)
dados
```

```
Out[356]: array([[ 0.1212 ,  0.842   ,  0.3672  ],
                  [       nan,  0.5422 ,  0.34551 ],
                  [ 0.5661 ,  0.932   ,        nan],
                  [ 0.762   ,  0.847361,  0.1538  ]])
```

```
In [362]: # Preenchendo dados faltantes com 10.5
dados = np.genfromtxt('dados_faltantes.txt', skip_header=1, filling_values=10.5)
dados
```

```
Out[362]: array([[ 0.1212 ,  0.842   ,  0.3672  ],
                  [ 10.5    ,  0.5422 ,  0.34551 ],
                  [ 0.5661 ,  0.932   ,  10.5    ],
                  [ 0.762   ,  0.847361,  0.1538  ]])
```



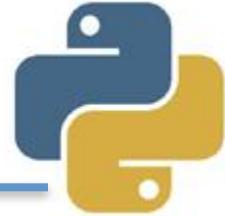
Arquivos CSV

- Imagine que precisamos **acessar** o **arquivo** abaixo, para **carregar** as notas dos alunos

A screenshot of a Mac OS X TextEdit window displaying a CSV file named "alunos_notas.csv". The file contains five rows of data, each representing a student's average grades. Row 1: 1, av1;av2;av3;av4;av5. Row 2: 2, 8.5;7.6;9.0;7.8;10.0. Row 3: 3, 7.5;4.9;8.5;8.7;9.0. Row 4: 4, 10.0;9.1;8.8;9.0;8.0. Row 5: 5, (empty).

1	av1;av2;av3;av4;av5
2	8.5;7.6;9.0;7.8;10.0
3	7.5;4.9;8.5;8.7;9.0
4	10.0;9.1;8.8;9.0;8.0
5	

Exercício 61: np.genfromtext



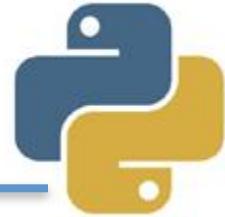
Carregando arquivos CSV

```
import numpy as np

valores = np.genfromtxt('alunos_notas.csv', delimiter=';',
                       skip_header=1)
valores

array([[ 8.5,    7.6,    9. ,    7.8,   10. ],
       [ 7.5,    4.9,    8.5,    8.7,    9. ],
       [ 10. ,   9.1,    8.8,    9. ,    8. ]])
```

Análise de dados com numpy

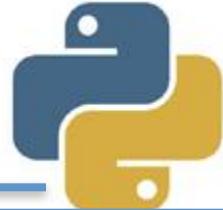


- <https://archive.ics.uci.edu/ml/datasets/Iris>

A screenshot of a web browser window displaying the Iris dataset. The address bar shows the URL: https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data. Below the address bar, there is a navigation bar with icons for back, forward, search, and other functions. A toolbar below the navigation bar includes categories like Apps, Android, C e C++, congressos, geral, ibm, java, Linux, TCCs, and UFA. The main content area of the browser displays the following data:

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
```

Exercício 62: carga de dado



Análise de dados com numpy

```
import numpy as np

#usecols - indica as colunas que devem ser carregadas
dados = np.genfromtxt('iris.data', delimiter=',', usecols=(0, 1, 2, 3))
dados[:50]

array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2],
       [ 5.4,  3.9,  1.7,  0.4],
       [ 4.6,  3.4,  1.4,  0.3],
       [ 5. ,  3.4,  1.5,  0.2],
       [ 4.4,  2.9,  1.4,  0.2],
       [ 4.9,  3.1,  1.5,  0.1],
       [ 5.4,  3.7,  1.5,  0.2],
```

Exercício 63: análise inicial



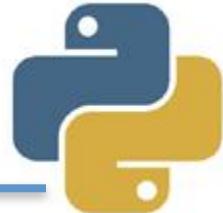
```
print 'Total de amostras', len(dados)
```

Total de amostras 150

```
# 1ª coluna: comprimento da sépala  
dados[:, 0]
```

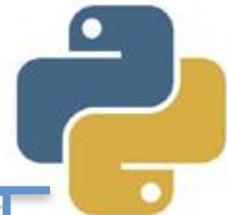
```
array([ 5.1,  4.9,  4.7,  4.6,  5. ,  5.4,  4.6,  5. ,  4.4,  4.9,  5.4,  
       4.8,  4.8,  4.3,  5.8,  5.7,  5.4,  5.1,  5.7,  5.1,  5.4,  5.1,  
       4.6,  5.1,  4.8,  5. ,  5. ,  5.2,  5.2,  4.7,  4.8,  5.4,  5.2,  
       5.5,  4.9,  5. ,  5.5,  4.9,  4.4,  5.1,  5. ,  4.5,  4.4,  5. ,  
       5.1,  4.8,  5.1,  4.6,  5.3,  5. ,  7. ,  6.4,  6.9,  5.5,  6.5,  
       5.7,  6.3,  4.9,  6.6,  5.2,  5. ,  5.9,  6. ,  6.1,  5.6,  6.7,  
       5.6,  5.8,  6.2,  5.6,  5.9,  6.1,  6.3,  6.1,  6.4,  6.6,  6.8,  
       6.7,  6. ,  5.7,  5.5,  5.5,  5.8,  6. ,  5.4,  6. ,  6.7,  6.3,  
       5.6,  5.5,  5.5,  6.1,  5.8,  5. ,  5.6,  5.7,  5.7,  6.2,  5.1,  
       5.7,  6.3,  5.8,  7.1,  6.3,  6.5,  7.6,  4.9,  7.3,  6.7,  7.2,  
       6.5,  6.4,  6.8,  5.7,  5.8,  6.4,  6.5,  7.7,  7.7,  6. ,  6.9,  
       5.6,  7.7,  6.3,  6.7,  7.2,  6.2,  6.1,  6.4,  7.2,  7.4,  7.9,  
       6.4,  6.3,  6.1,  7.7,  6.3,  6.4,  6. ,  6.9,  6.7,  6.9,  5.8,  
       6.8,  6.7,  6.7,  6.3,  6.5,  6.2,  5.9])
```

Exercício 64: Iris-setosa



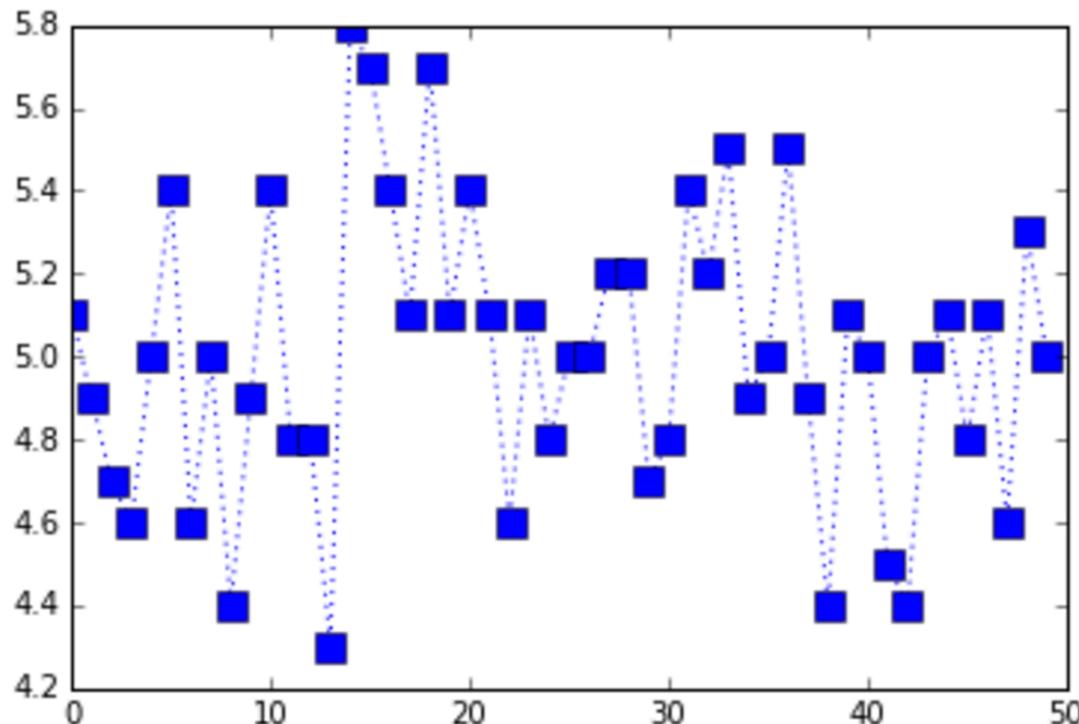
```
# comprimento da sépala da Iris-setosa  
dados[:50,0]
```

```
array([ 5.1,  4.9,  4.7,  4.6,  5. ,  5.4,  4.6,  5. ,  4.4,  4.9,  5.4,  
       4.8,  4.8,  4.3,  5.8,  5.7,  5.4,  5.1,  5.7,  5.1,  5.4,  5.1,  
       4.6,  5.1,  4.8,  5. ,  5. ,  5.2,  5.2,  4.7,  4.8,  5.4,  5.2,  
       5.5,  4.9,  5. ,  5.5,  4.9,  4.4,  5.1,  5. ,  4.5,  4.4,  5. ,  
       5.1,  4.8,  5.1,  4.6,  5.3,  5. ])
```



Exercício 65: Iris-setosa plot

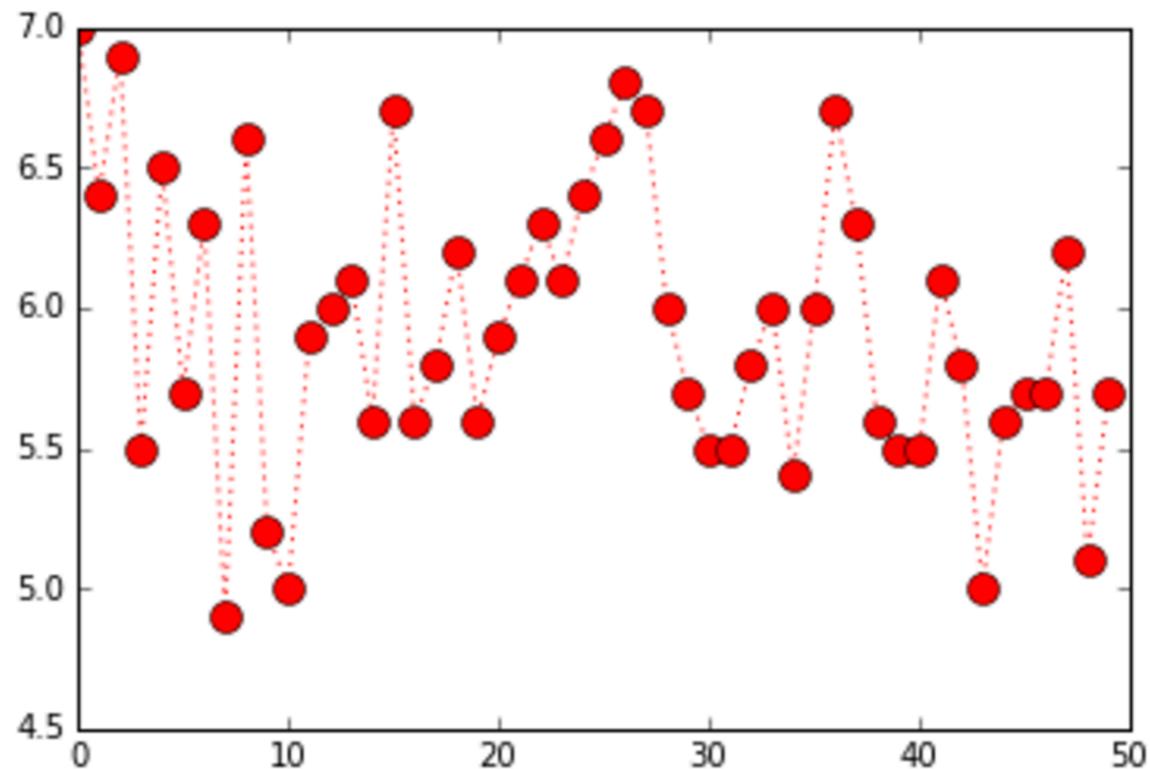
```
import matplotlib.pyplot as plt  
%matplotlib inline  
  
# 1ª coluna - comprimento da sépala da Iris-setosa  
plt.plot(dados[:50,0], ls=':', marker='s', ms=10)  
plt.show()
```



Exercício 66: Iris-versicolor plot



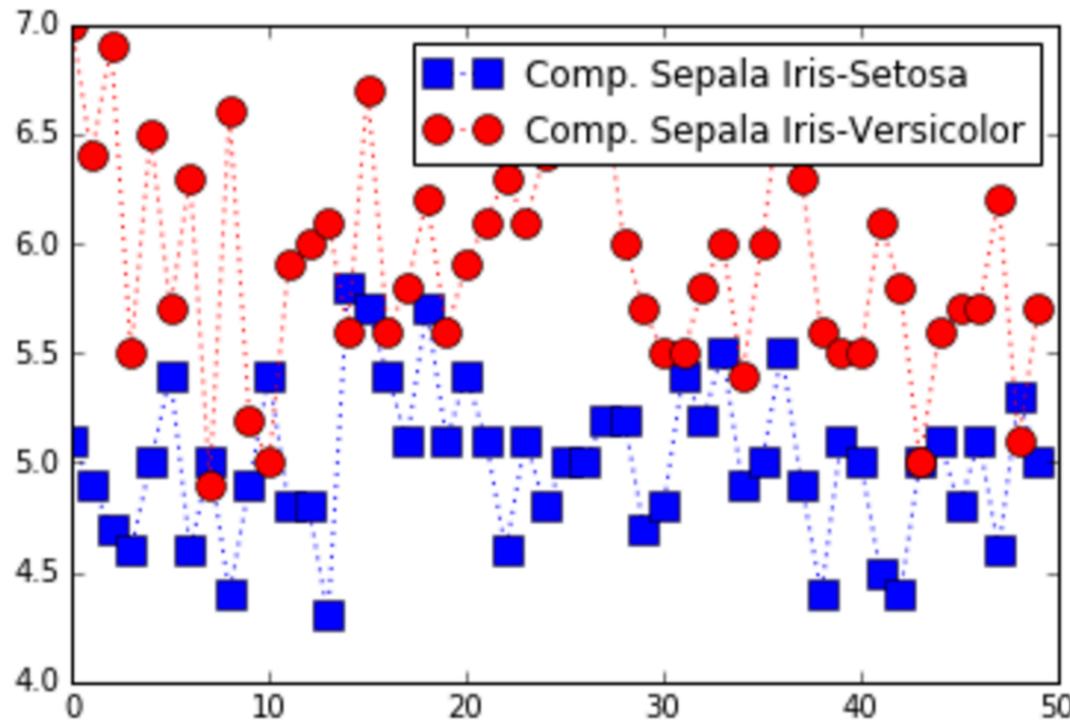
```
# 1ª coluna - comprimento da sépala da Iris-versicolor  
plt.plot(dados[50:100,0], c='red', ls=':', marker='o', ms=10)  
plt.show()
```



Exercício 68: análise



```
plt.plot(dados[:50,0], ls=':', marker='s', ms=10,  
         label='Comp. Sepala Iris-Setosa')  
plt.plot(dados[50:100,0], c='red', ls=':', marker='o', ms=10,  
         label='Comp. Sepala Iris-Versicolor')  
plt.legend()  
plt.show()
```

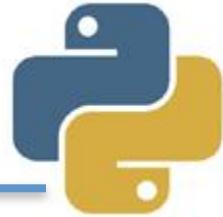




FIM

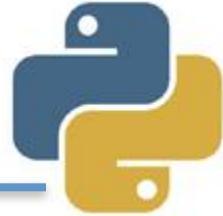


Contatos

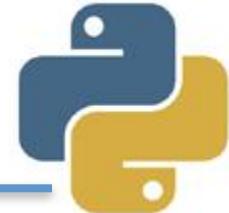


- Márcio Palheta
 - marcio.palheta@gmail.com
 - @marciopalheta
 - <https://sites.google.com/site/marciopalheta/>
-

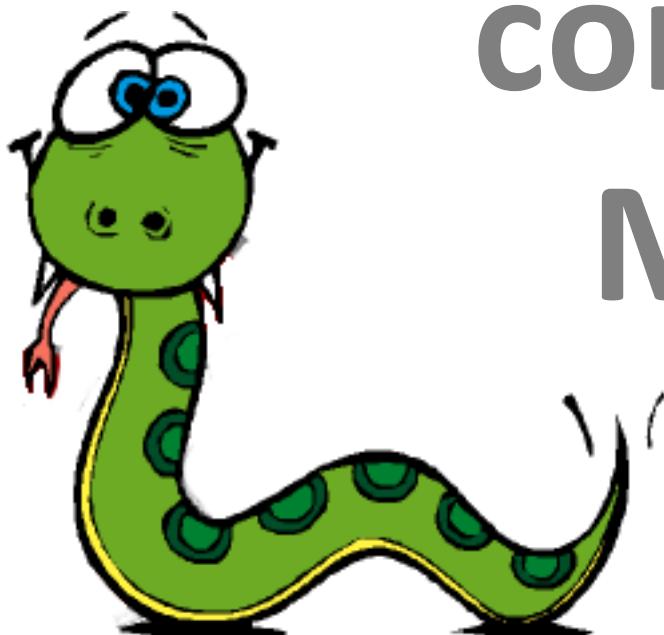
Bibliografia



- LIVRO: Apress - Beginning Python From Novice to Professional
 - LIVRO: O'Reilly - Learning Python
 - <http://www.python.org>
 - <http://www.python.org.br>
 - Mais exercícios:
 - <http://wiki.python.org.br/ListaDeExercicios>
 - Documentação do python:
 - <https://docs.python.org/2/>
-



Análise de dados com Numpy e Matplotlib



Márcio Palheta, M.Sc.
marcio.palheta@gmail.com