Introdução à Linguagem Python

Prof. Cláudio Fleury

Set-22

Importação de Módulos

Importação de Módulos

- Permite a inclusão de classes, funções, constantes etc. ao conteúdo padrão do Python, ampliando as funcionalidades da linguagem
- Sintaxe: `import nome_do_módulo`

NumPy

- Carrega uma coleção de ferramentas de cálculo numérico usando Python: "**Numerical Python**"
- Sintaxe: `import numpy`

- Para procurar por uma função, raiz quadrada, por exemplo, no pacote importado use o comando:

```
>>> numpy.lookfor('sqrt')
>>> numpy.sqrt(2)   →   1.4142
```

Depois de ter importado um módulo, você pode chamar suas funções, citando o nome do módulo, um ponto, e, em seguida, o nome da função desejada, como no exemplo da raiz quadrada.

NumPy

- Pode-se usar um apelido para os módulo importados, afim de economizar digitação...
- Sintaxe: `import numpy as np`
- Uso da raiz quadrada, por exemplo:

```
>>> np.sqrt(2)    →    1.4142
```

```
from numpy import sqrt
```

```
>>> sqrt(2)       →    1.4142
```

NumPy

- Arranjo numérico (vetor, matriz) - uma grade de números, todos do mesmo tipo de dado
- NumPy faz cálculos eficientes com todo o arranjo numérico
- Criando um arranjo unidimensional (vetor) com np.arange()

```
>>> np.arange(1,6) → array((1.,2.,3.,4.,5.))
```

NumPy

- Arranjo numérico (vetor, matriz) - uma grade de números, todos do mesmo tipo de dado
- NumPy faz cálculos eficientes com todo o arranjo numérico
- Criando um arranjo unidimensional (vetor) com 4 elementos:

```
>>> np.zeros(4)    →    array((0.,0.,0.,0.))
```

- Criando um arranjo bidimensional (matriz) 3 x 5:

```
>>> np.zeros((3,5)) → array([[0.,0.,0.,0.,0.],
                             [0.,0.,0.,0.,0.],
                             [0.,0.,0.,0.,0.]])
```

A variável ***a***, refere-se a um objeto **numpy.ndarray,** uma matriz NumPy '*n*-dimensional'

NumPy

- Usando outras funções: np.**ones**(), np.**eye**() e np.random.**random**()

IDLE Shell 3.9.5

File Edit Shell Debug Options Window Help

```
>>> b = np.ones((2,3))
>>> b
array([[1., 1., 1.],
       [1., 1., 1.]])
>>>
```

```
>>> c = np.eye(3)
>>> c
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>>
```

```
>>> d = np.random.random((2,3))
>>> d
array([[0.12552358, 0.22130579, 0.22682279],
       [0.04649682, 0.961169  , 0.9935184 ]])
>>>
```

NumPy

- Usando outras funções: np.**size**(), np.**shape**(), np.**sum**() e np.**mean**()

```
>>> print("Qtde de elementos na matriz 'a':", np.size(a))
Qtde de elementos na matriz 'a': 15
>>> print("Formato da matriz 'a':", np.shape(a))
Formato da matriz 'a': (3, 5)
>>> print("Soma dos elementos da linha 1 =", np.sum(a[0]))
Soma dos elementos da linha 1 = 0.0
>>> print("Soma dos elementos da linha 1 de 'b'=", np.sum(b[0]))
Soma dos elementos da linha 1 de 'b'= 3.0
>>> print("Média dos elementos da linha 2 de 'd' =", np.mean(d[1]))
Média dos elementos da linha 2 de 'd' = 0.667061409273162 4
>>>
```

NumPy

- Diferenças dos **arranjos** e **listas**
  - Os elementos de:
    - uma Lista podem ser qualquer tipo de dado
    - um arranjo devem ser todos do mesmo tipo de dado

```
import numpy as np

nomes = ['joão','pedro bó','ana maria',True,-19,['R.10',23, 'Goiânia']]
idades = np.array([10,20,33,3.1415])
notas = np.array([[3,6,4,3,5,2,1,8,5],[5,4,3,6,5,7,5,8,2]],dtype=float)

print(nomes[1].capitalize(), nomes[2].title(), idades[1])
print(type(idades), type(nomes))
print(idades)
print("Notas Bim.1:", end=' ')
for col in range(0,9):
    print("%3.1f" % notas[0,col], end=' ')
print("\nNotas Bim.2:", end=' ')
for col in range(0,9):
    print("%3.1f" % notas[1,col], end=' ')
```

```
Pedro bó Ana Maria 20.0
<class 'numpy.ndarray'> <class 'list'>
[10.     20.     33.      3.1415]
Notas Bim.1: 3.0 6.0 4.0 3.0 5.0 2.0 1.0 8.0 5.0
Notas Bim.2: 5.0 4.0 3.0 6.0 5.0 7.0 5.0 8.0 2.0
```

NumPy

- Semelhanças dos **arranjos** e **listas**
  - Podem ter mais de uma dimensão

```
medias = np.zeros(9)       # vetor para guardar as médias dos 9 alunos

for col in range(0,9):     # cálculo das médias
    soma_col = 0           # soma das notas de cada aluno
    for lin in range(0,2):
        soma_col += notas[lin][col]
    medias[col] = soma_col/2

print("\nMédias:     ", end=' ') # apresentação das médias
for col in range(0,9):
    # print("%3.1f" % medias[col], end=' ')
    print(f"{medias[col]:3.1f}", end=' ')
```

```
Notas Bim.1: 3.0 6.0 4.0 3.0 5.0 2.0 1.0 8.0 5.0
Notas Bim.2: 5.0 4.0 3.0 6.0 5.0 7.0 5.0 8.0 2.0
Médias:      4.0 5.0 3.5 4.5 5.0 4.5 3.0 8.0 3.5
```

NumPy

- Semelhanças dos **arranjos** e **listas**
  - Cálculo das médias com o método **sum()**

```
medias = np.zeros(9)                # vetor p/ médias dos 9 alunos
'''
for col in range(0,9):              # cálculo das médias
    soma_col = 0                    # soma das notas de cada aluno
    for lin in range(0,2):
        soma_col += notas[lin][col]
    medias[col] = soma_col/2
'''
medias = notas.mean(axis=0)         # ou: medias = notas.sum(axis=0)/2

print("\nMédias:     ", end=' ') # apresentação das médias
for col in range(0,9):
    # print("%3.1f" % medias[col], end=' ')
    print(f"{medias[col]:3.1f}", end=' ')
```

```
Notas Bim.1: 3.0 6.0 4.0 3.0 5.0 2.0 1.0 8.0 5.0
Notas Bim.2: 5.0 4.0 3.0 6.0 5.0 7.0 5.0 8.0 2.0
Médias:      4.0 5.0 3.5 4.5 5.0 4.5 3.0 8.0 3.5
```

NumPy

- **Atenção**: os resultados dos comandos np.**zeros**(3), np.**zeros**((1,3)), e np.**zeros**((3,1)) são diferentes
- NumPy não considera o resultado do primeiro comando nem vetor linha, nem vetor coluna. Trata-se de uma arranjo unidimensional – um *array* com apenas um índice

NumPy

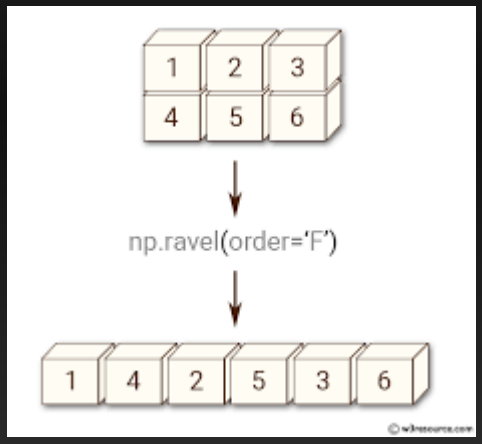
- Pode-se remodelar uma arranjo unidimensional em um vetor coluna ou vetor linha, se necessário:

```
>>> a = np.arange(12)

>>> b = np.reshape(a, (3, 4) )
>>> c = b.reshape( (2, 6) )
>>> d = c.reshape( (2, 3, 2) )
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> c
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11]])
>>> d
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],

       [[ 6,  7],
        [ 8,  9],
        [10, 11]]])
>>>
```

NumPy

- Achatamento/desfiamento de Matriz
  - Uma matriz pode ter qualquer número de dimensões que podem ser reduzidas a uma única dimensão
  - A função np.**ravel**() faz o desfiamento de uma matriz, assim como os métodos **flatten**() e **ravel**() da classe **array**

```
>>> x = np.array( [ [1, 2], [2, 1] ] )
>>> x
array([[1, 2],
       [2, 1]])
>>> x.flatten()
array([1, 2, 2, 1])
>>> np.ravel(x)
array([1, 2, 2, 1])
>>> x.ravel()
array([1, 2, 2, 1])
>>>
```

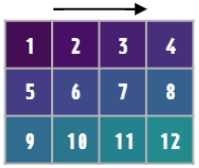


Diferença entre os métodos:
- **flatten()** retorna uma nova matriz achatada e independente;
- **ravel()** retorna um objeto array desfiado, com acesso aos mesmos dados, numa forma diferente (equivalente ao método **reshape()**).

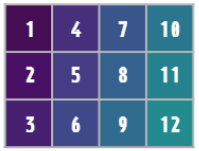
Python numpy reshape and stack cheatsheet

reshape & ravel

```
a1 = np.arange(1, 13)
```



```
a1.reshape(3, 4)
a1.reshape(-1, 4)
a1.reshape(3, -1)
  .ravel()  # back to 1D
```

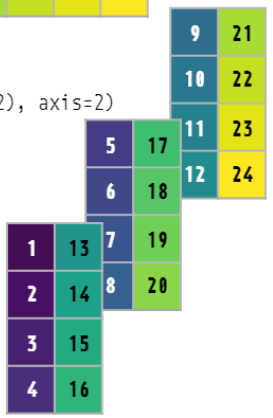


```
a1.reshape(3, -1, order='F')
  .ravel(order='F')  # back to 1D
```

3D array from 2D arrays

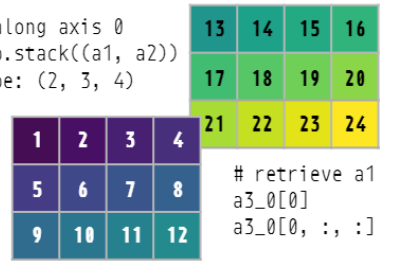
```
a1 = np.arange(1, 13).reshape(3, 4)
a2 = np.arange(13, 25).reshape(3, -1)
```

```
# stack along axis 2
a3_2 = np.stack((a1, a2), axis=2)
a3_2.shape: (3, 4, 2)
```



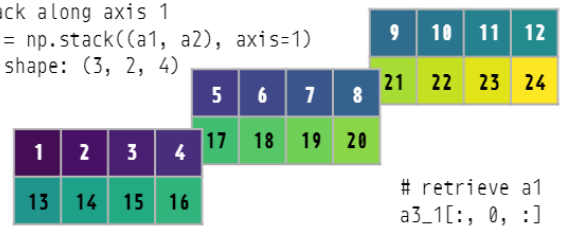
```
# retrieve a1
a3_2[:, :, 0]
```

```
# stack along axis 0
a3_0 = np.stack((a1, a2))
a3_0.shape: (2, 3, 4)
```



```
# retrieve a1
a3_0[0]
a3_0[0, :, :]
```

```
# stack along axis 1
a3_1 = np.stack((a1, a2), axis=1)
a3_1.shape: (3, 2, 4)
```

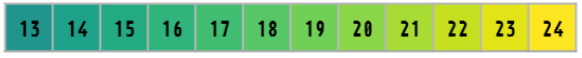


```
# retrieve a1
a3_1[:, 0, :]
```

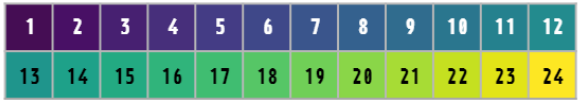
stack

```
a1 = np.arange(1, 13)
```

```
a2 = np.arange(13, 25)
```



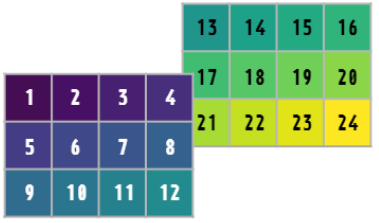
```
np.stack((a1, a2))
```



```
np.hstack((a1, a2)
```

```
np.stack((a1, a2), axis=1)
```

flatten 3D array

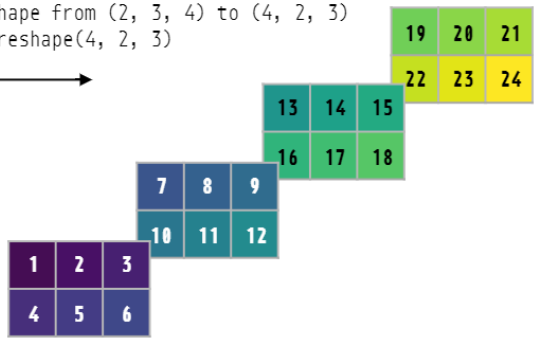


```
# flatten/ravel
a3_0.ravel()
```

```
# flatten/ravel
a3_0.ravel(order='F')
```

reshape 3D array

```
# reshape from (2, 3, 4) to (4, 2, 3)
a3_0.reshape(4, 2, 3)
```



PyPlot

- Outro pacote usual nos scripts dos cientistas, para plotar (traçar) gráficos diversos
- Sintaxe: `import matplotlib.pyplot as plt`
- **PyPlot** é um subconjunto da biblioteca **Matplotlib**
- Exemplo:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

NumPy fornece ferramentas numéricas necessárias à geração e análise de dados, e **PyPlot** fornece as ferramentas para visualizar os dados graficamente.

Fonte(s)

- Fangohr, H.; Python for Computational Science and Engineering; 2018; DOI: 10.5281/zenodo.1411868; Acessado em 10/09/2022; Disponível em: https://github.com/fangohr/introduction-to-python-for-computational-science-and-engineering
- Kinder, J.M. & Nelson, P.; A Student's Guide to Python for Physical Modeling; Princeton University Press; ISBN 978-0-691-16958-3;2015.
- Lin, H; Reshape numpy arrays in Python – a step-by-step pictorial tutorial; Acessado em 14/09/2022; Disponível em: https://towardsdatascience.com/reshaping-numpy-arrays-in-python-a-step-by-step-pictorial-tutorial-aed5f471cf0b