Curso básico de Python para estudantes de Física

Germán A. Racca

Universidade do Estado do Rio Grande do Norte
Faculdade de Ciências Exatas e Naturais
Departamento de Física
Mossoró - RN

01 de Junho de 2016

Tópicos

1. Funções
   - Argumentos
   - Parâmetros
   - Comando `return`
2. Numpy
   - `arange`
   - `linspace`
   - `array`
   - `shape`
   - Indexação
   - Slicing
   - Operações numéricas
   - `dot`
3. Material do curso

Tópicos

Funções

Definição

Blocos de código identificados por um nome, possuem parâmetros, argumentos e retornam um valor.

```
>>> def oi():
...     print "Oi gente..."
>>> oi()
Oi gente...

>>> def busca_letra():
...     contador = 0
...     for letra in palavra:
...         if letra == "a":
...             contador += 1
...     print "Na palavra", palavra, "exitem", contador, "ocorrencias da letra a"

>>> palavra = "codigo"
>>> busca_letra()
Na palavra codigo exitem 0 ocorrencias da letra a
>>> palavra = "reutilizacao"
>>> busca_letra()
Na palavra reutilizacao exitem 2 ocorrencias da letra a
```

Tópicos

Argumentos

Definição

Variáveis passadas para uma função

```
>>> def oi(nome):
...     print "Oi", nome
>>> oi("Aldinez")
Oi Aldinez
```

```
>>> def conta_letra(letra_esperada, frase):
...     contador = 0
...     for letra in frase:
...         if letra == letra_esperada:
...             contador += 1
...     print "Foram encontradas", contador, "ocorrencias da letra", letra_esperada
>>> conta_letra("i", "primeiro teste")
Foram encontradas 2 ocorrencias da letra i
```

```
>>> def mostrar_numeros(num1, num2):
...     print "Primeiro numero:", num1
...     print "Segundo numero:", num2
>>> mostrar_numeros(num2=5, num1=7)
Primeiro numero: 7
Segundo numero: 5
```

Tópicos

Parâmetros

Definição

Valores padrões que fazem um argumento se tornar opcional

```
>>> def incrementa(num1, num2=1):
...     return num1 + num2
```

```
>>> incrementa(3)
4
>>> incrementa(7, 10)
17
```

```
>>> def contar_caracteres(frase, letra=None):
...     if letra is None:
...         cont = 0
...         for l in frase:
...             cont = cont + 1
...     else:
...         cont = 0
...         for l in frase:
...             if l == letra:
...                 cont = cont + 1
...     return cont
>>> contar_caracteres("uma frase de teste")
18
>>> contar_caracteres("uma frase de teste", "e")
4
```

Tópicos

Comando return

Definição

Valor a ser devolvido como o resultado da função

```
>>> def quad_cubo(x):
...     q = x**2
...     c = x**3
...     return q, c
>>> quad_cubo(4)
(16, 64)
>>> x1, x2 = quad_cubo(5)
>>> print x1
25
>>> print x2
125

>>> # exemplo do uso de 'docstrings'
>>> def quad_cubo(x):
...     """Retorna o quadrado e o cubo de x"""
...     return x**2, x**3
>>> quad_cubo(3)
(9, 27)
>>> print quad_cubo.__doc__
Retorna o quadrado e o cubo de x
```

Tópicos

NumPy

"Numeric Python" ou "Numerical Python"

- Módulo open source para Python
- Poderosa estrutura de dados para o cómputo eficiente de *arrays* multi-dimensionais
- Biblioteca extensa de funções matemáticas de alto nível que operam sobre esses *arrays*

Exemplo simples de NumPy:

```
# temperaturas em Celsius
>>> import numpy as np
>>> c = [25.3, 24.8, 26.9, 23.9]
>>> C = np.array(c)
>>> print C
[ 25.3  24.8  26.9  23.9]

# transformamos a Farenheit
>>> F = C * 9/5 + 32
>>> print F
[ 77.54  76.64  80.42  75.02]
```

Tópicos

arange

Definição

Retorna um *array* de valores igualmente espaçados dentro de um dado intervalo
`arange([start,] stop[, step,], dtype=None)`

Exemplo de arange:

```
>>> import numpy as np
>>> a = np.arange(1, 10)
>>> print(a)
[1 2 3 4 5 6 7 8 9]
>>> x = np.arange(10.4)
>>> print x
[  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
>>> x = np.arange(0.5, 10.4, 0.8)
>>> print x
[  0.5   1.3   2.1   2.9   3.7   4.5   5.3   6.1   6.9   7.7   8.5   9.3
  10.1]
>>> x = np.arange(0.5, 10.4, 0.8, int)
>>> print x
[ 0  1  2  3  4  5  6  7  8  9 10 11 12]
```

Tópicos

linspace

Definição

Retorna um *array* de valores igualmente espaçados dentro de um dado intervalo
`linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)`

Exemplo de `linspace`:

```
>>> import numpy as np
>>> print np.linspace(1, 10)
[  1.           1.18367347   1.36734694   1.55102041   1.73469388
   1.91836735   2.10204082   2.28571429   2.46938776   2.65306122
   2.83673469   3.02040816   3.20408163   3.3877551    3.57142857
   3.75510204   3.93877551   4.12244898   4.30612245   4.48979592
   4.67346939   4.85714286   5.04081633   5.2244898    5.40816327
   5.59183673   5.7755102    5.95918367   6.14285714   6.32653061
   6.51020408   6.69387755   6.87755102   7.06122449   7.24489796
   7.42857143   7.6122449    7.79591837   7.97959184   8.16326531
   8.34693878   8.53061224   8.71428571   8.89795918   9.08163265
   9.26530612   9.44897959   9.63265306   9.81632653  10.        ]
>>> print np.linspace(1, 10, 7)
[  1.    2.5   4.    5.5   7.    8.5  10. ]
>>> print np.linspace(1, 10, 7, endpoint=False)
[ 1.          2.28571429  3.57142857  4.85714286  6.14285714  7.42857143  8.71428571]
```

Tópicos

array

Definição

Contenedores de elementos ou ítens do mesmo tipo

`array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)`

Exemplo de array zero-dimensional:

```
>>> x = np.array(42)
>>> print type(x)
<type 'numpy.ndarray'>
>>> print x.ndim
0
```

Exemplo de array uni-dimensional:

```
>>> F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
>>> print F.dtype
int64
>>> print F.ndim
1
>>> V = np.array([3.4, 6.9, 99.8, 12.8])
>>> print V.dtype
float64
>>> print V.ndim
1
```

array

Exemplo de array bi-dimensional e multi-dimensional:

```
>>> A = np.array([ [3.4, 8.7, 9.9],
...                [1.1, -7.8, -0.7],
...                [4.1, 12.3, 4.8]])
>>> print A
[[  3.4   8.7   9.9]
 [  1.1  -7.8  -0.7]
 [  4.1  12.3   4.8]]
>>> print A.ndim
2
>>> B = np.array([ [[111, 112], [121, 122]],
...                [[211, 212], [221, 222]],
...                [[311, 312], [321, 322]] ])
>>> print B
[[[111 112]
  [121 122]]

 [[211 212]
  [221 222]]

 [[311 312]
  [321 322]]]
>>> print B.ndim
3
```

Tópicos

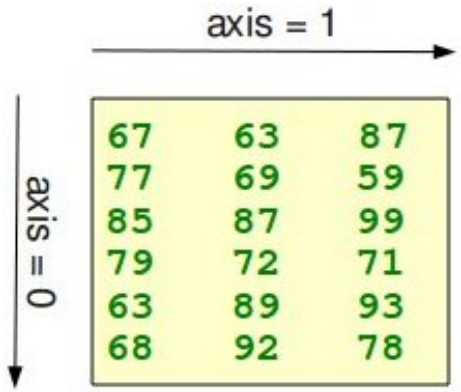
shape

Definição

Retorna a forma de um *array* como uma tupla de inteiros, os quais denotam o tamanho de cada dimensão do *array*. Em outras palavras: a forma de um *array* é uma tupla com o número de elementos por eixo (dimensão)

Exemplo de shape:

```
>>> A = np.array([ [67, 63, 87],
...                [77, 69, 59],
...                [85, 87, 99],
...                [79, 72, 71],
...                [63, 89, 93],
...                [68, 92, 78]])
>>> print A.shape
(6, 3)
>>> A.shape = (3, 6)
>>> print A
[[67 63 87 77 69 59]
 [85 87 99 79 72 71]
 [63 89 93 68 92 78]]
```




```
>>> B = np.array([ [[111, 112], [121, 122]],
...                [[211, 212], [221, 222]],
...                [[311, 312], [321, 322]] ])
>>> print B.shape
(3, 2, 2)
```

Tópicos

Indexação

Exemplo de indexação:

```
>>> F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
# primeiro elemento de F
>>> print F[0]
1
# ultimo elemento de F
>>> print F[-1]
21

>>> A = np.array([ [3.4, 8.7, 9.9],
...                [1.1, -7.8, -0.7],
...                [4.1, 12.3, 4.8]])
>>> print A[1][0]
1.1
>>> print A[1, 0]
1.1

>>> B = np.array([ [[111, 112], [121, 122]],
...                [[211, 212], [221, 222]],
...                [[311, 312], [321, 322]] ])
>>> print B[0][1][0]
121
```

Tópicos

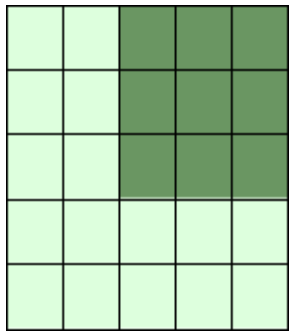
Slicing

Exemplo de *slicing* uni-dimensional:

```
>>> S = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> print S[2:5]
[2 3 4]
>>> print S[:4]
[0 1 2 3]
>>> print S[6:]
[6 7 8 9]
>>> print S[:]
[0 1 2 3 4 5 6 7 8 9]
```

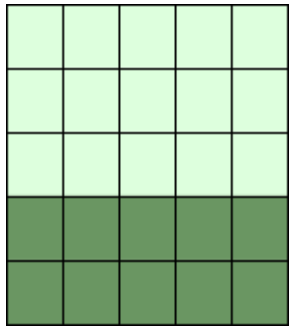
Exemplo de *slicing* multi-dimensional:

```
>>> A = np.array([
... [11,12,13,14,15],
... [21,22,23,24,25],
... [31,32,33,34,35],
... [41,42,43,44,45],
... [51,52,53,54,55]])
>>> print A[:3, 2:]
[[13 14 15]
 [23 24 25]
 [33 34 35]]
```

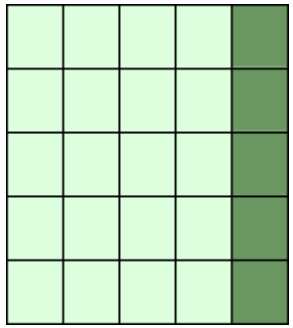


Slicing

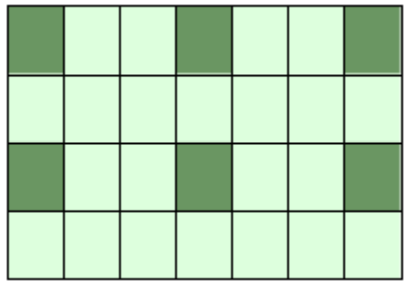
```
>>> print A[3:,:]
[[41 42 43 44 45]
 [51 52 53 54 55]]
```



```
>>> print A[:,4:]
[[15]
 [25]
 [35]
 [45]
 [55]]
```



```
>>> X = np.arange(28).reshape(4,7)
>>> print X
[[ 0  1  2  3  4  5  6]
 [ 7  8  9 10 11 12 13]
 [14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27]]
>>> print(X[::2, ::3])
[[ 0  3  6]
 [14 17 20]]
```



Tópicos

Operações numéricas

Operações aritméticas usando escalares:

```
>>> v = np.array([2, 3, 7.9, 3.3, 6.9, 0.11, 10.3, 12.9])
>>> print v + 2
[  4.     5.     9.9    5.3    8.9    2.11  12.3   14.9 ]
>>> print v * 2.2
[  4.4     6.6    17.38    7.26   15.18    0.242  22.66   28.38 ]
>>> print v**2
[  4.00000000e+00   9.00000000e+00   6.24100000e+01   1.08900000e+01
   4.76100000e+01   1.21000000e-02   1.06090000e+02   1.66410000e+02]
```

Operações aritméticas usando dois *arrays*:

```
>>> A = np.array([ [11, 12, 13], [21, 22, 23], [31, 32, 33] ])
>>> B = np.ones((3,3))
>>> print A + B
[[ 12.  13.  14.]
 [ 22.  23.  24.]
 [ 32.  33.  34.]]
>>> print A * (B + 1)
[[ 22.  24.  26.]
 [ 42.  44.  46.]
 [ 62.  64.  66.]]
```

Tópicos

dot

Definição

Para *arrays* 2D é equivalente à multiplicação matricial. Para *arrays* 1D é o mesmo que o produto interno.

`dot(a, b, out=None)`

Exemplo de `dot`:

```
>>> x = np.array([3, -2])
>>> y = np.array([-4, 1])
>>> print np.dot(x, y)
-14
>>> A = np.array([ [1, 2, 3],
...                [3, 2, 1] ])
>>> B = np.array([ [2, 3, 4, -2],
...                [1, -1, 2, 3],
...                [1, 2, 3, 0] ])
>>> print np.dot(A, B)
[[ 7  7 17  4]
 [ 9  9 19  0]]
>>> MA = np.mat(A)
>>> MB = np.mat(B)
>>> print MA * MB
[[ 7  7 17  4]
 [ 9  9 19  0]]
```

Tópicos

Material do curso

Aulas do Lázaro Camargo (INPE):

https://w2v9k6.s.cld.pt

material-python-01.tar.bz2

```
numpy_aula_00_revisao_python.pdf
numpy_aula_01_fundamentos.pdf
numpy_aula_02_funcoes_mais_comuns.pdf
numpy_aula_03_matematica_arrays.pdf
python_aula_05_funcoes.pdf
python_aula_06_arquivos.pdf
python_aula_funcao_lambda.pdf
```