Processamento numérico com Numpy

O pacote `numpy` contém a estrutura básica de operações com vetores homogêneos em Python: `ndarray`

- Trata-se de uma classe que representa arrays multidimensionais de elementos do mesmo tipo
- O conteúdo do array pode ser de qualquer tipo reconhecido pelo Python, ou mesmo de tipos definidos pelo pacote numpy

Criando um `ndarray`

In [1]:

```
import numpy as np

def print_arr(arr):
    print("{} de tamanho {} com tipo: {}\n".format(arr, arr.shape, arr.dtype))

# Criando um array unidimensional a partir de uma lista
arr1 = np.array([1, 2, 3, 4])

# Criando um array bidimensional a partir de uma lista de listas
arr2 = np.array([[1, 2.0, 3], [4, 5, 6]])

# Criando um array unidimensional com números aleatórios entre 0 e 1
arr3 = np.random.rand(6)

# Criando um array unidimensional preenchido com 5 zeros do tipo float
arr_zeros = np.zeros(5, dtype=float)

# Criando um array bidimensional 3x2 preenchido com uns do tipo int
arr_uns = np.ones([3, 2], dtype=int)

# Criando um array unidimensional tipo bool sem inicialização
arr6 = np.empty([7], dtype=bool)

# Criando uma sequencia iniciando em 5, terminando antes de 30 e com passo de increment
o 4
arr7 = np.arange(5, 30, step=4, dtype=float)

print_arr(arr1)
print_arr(arr2)
print_arr(arr3)
print_arr(arr_zeros)
print_arr(arr_uns)
print_arr(arr6)
print_arr(arr7)
```

```
[1 2 3 4] de tamanho (4,) com tipo: int32

[[1. 2. 3.]
 [4. 5. 6.]] de tamanho (2, 3) com tipo: float64

[0.84870639 0.65012368 0.15376803 0.89260111 0.80274038 0.06699044] de tam
anho (6,) com tipo: float64

[0. 0. 0. 0. 0.] de tamanho (5,) com tipo: float64

[[1 1]
 [1 1]
 [1 1]] de tamanho (3, 2) com tipo: int32

[False False False  True False  True False] de tamanho (7,) com tipo: bool

[ 5.  9. 13. 17. 21. 25. 29.] de tamanho (7,) com tipo: float64
```

Algumas propriedades do `ndarray`:

- `dtype`: indica o tipo de dados dos elementos do array
- `shape`: tupla que indica o número de elementos em cada uma das dimensões do array

Operações lógicas e aritméticas

As operações e funções matemáticvas são aplicadas aos arrays elemento a elemento. Podemos realizar:

- Soma, multiplicação, divisão, etc. de dois arrays do mesmo tamanho
- Soma, multiplicação, divisão, etc. de um array com uma constante
- Comparação de valores em dois arrays do mesmo tamanho
- Comparação de valores de array com uma constante
- Funções matemáticas aplicadas aos elementos do array

Exemplos:

In [2]:

```
import numpy as np

# Criando dois arrays unidimensionais
arr1 = np.random.rand(6)
arr2 = np.array([1, 2, 3, 4, 5, 6])

# Somando e Multiplicando os arrays elemento a elemento
arr3 = arr1 + arr2
arr4 = arr1 * arr2

# Somando 1 a arr1
arr5 = arr1 + 1.0

# Elevando arr2 ao quadrado
arr6 = arr2 ** 2

# Tirando a raiz quadrada de arr5
arr7 = np.sqrt(arr5)

# Comparando dois arrays
arr8 = np.array([-3, -2, -1, 0, 1, 2, 3]) >= np.zeros(7)

print_arr(arr1)
print_arr(arr2)
print_arr(arr3)
print_arr(arr4)
print_arr(arr5)
print_arr(arr6)
print_arr(arr7)
print_arr(arr8)
```

```
[0.55622717 0.37416744 0.16153661 0.91795287 0.69291115 0.17019441] de tam
anho (6,) com tipo: float64

[1 2 3 4 5 6] de tamanho (6,) com tipo: int32

[1.55622717 2.37416744 3.16153661 4.91795287 5.69291115 6.17019441] de tam
anho (6,) com tipo: float64

[0.55622717 0.74833487 0.48460984 3.67181149 3.46455574 1.02116643] de tam
anho (6,) com tipo: float64

[1.55622717 1.37416744 1.16153661 1.91795287 1.69291115 1.17019441] de tam
anho (6,) com tipo: float64

[ 1  4  9 16 25 36] de tamanho (6,) com tipo: int32

[1.24748835 1.17224888 1.07774608 1.38490176 1.30111919 1.08175524] de tam
anho (6,) com tipo: float64

[False False False  True  True  True  True] de tamanho (7,) com tipo: bool
```

Acessando elementos em um array

O nparray é muito flexível no acesso aos seus elementos. Através de colchetes [] podemos acessar um ou mais elementos:

In [3]:

```
import numpy as np

def print_cmd (cmd):
    print("{}:\n{}\n".format(cmd, eval(cmd)))

# Matriz:
arr = np.arange(10, 70, step=5).reshape([3, 4])
print_arr(arr)

# Acesso a um unico elemento:
print_cmd('arr[1, 2]') # Linha 2, coluna 3

# Acesso a um conjunto de valores através de slices:
print_cmd('arr[1:3, 2:]') # Omitindo o final pegamos até o último valor
print_cmd('arr[:3, :]') # Omitindo o início pegamos desde o primeiro valor

# Acesso a um subarray através de listas ou arrays de inteiros
# É necessário um array para cada dimensão
print_cmd('arr[[0, 2, 1], [0, 2, 3]]')

# Usando um array lógico para acessar os elementos
print_cmd('arr[arr > 20]')
```

```
[[10 15 20 25]
 [30 35 40 45]
 [50 55 60 65]] de tamanho (3, 4) com tipo: int32

arr[1, 2]:
40

arr[1:3, 2:]:
[[40 45]
 [60 65]]

arr[:3, :]:
[[10 15 20 25]
 [30 35 40 45]
 [50 55 60 65]]

arr[[0, 2, 1], [0, 2, 3]]:
[10 60 45]

arr[arr > 20]:
[25 30 35 40 45 50 55 60 65]
```

Plotando arrays com `matplotlib`

Vamos usar o submódulo `pyplot`para criar gráficos com uma sintaxe semelhante à do Matlab/Octave. Como primeiro exemplo, vamos mostrar a função seno

In [4]:

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

# Mudando o passo de
step = 0.1
x_vec = np.arange(0, 2 * np.pi, step)
y_vec = np.sin(x_vec)
plt.plot(x_vec, y_vec)

plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

Formatação básica do comando `plot`

O comando `plot` do Matlab possui um formato básco para controlar o estilo do gráfico. Corresponde a uma string composta por qatro partes opcionais: "<;displayname;>"

- Quando um marcador pe especificado, mas não o tipo de linha, apenas os marcadores são plotados
- Se o estilo de linha é especificado, mas não o marcador, então apenas a linha é mostrada

  Argumentos para o formatador:

  linestyle

  ```
  '-'  Usa linhas sólidas (default).
  '--' Usa linhas tracejadas.
  ':'  Usa linhas pontilhadas.
  '-.' Usa linhas traço-e-ponto.
  ```

  marker

  ```
  '+'  crosshair
  'o'  circle
  '*'  star
  '.'  point
  'x'  cross
  's'  square
  'd'  diamond
  '^'  upward-facing triangle
  'v'  downward-facing triangle
  '>'  right-facing triangle
  '<'  left-facing triangle
  'p'  pentagram
  'h'  hexagram
  ```

  color

  ```
  'k'  blacK
  'r'  Red
  'g'  Green
  'b'  Blue
  'y'  Yellow
  'm'  Magenta
  'c'  Cyan
  'w'  White
  ```

  ";displayname;"

  ```
  Aqui "displayname" é o rótulo a ser usado numa possível legenda
  ```

Um exemplo de formatação é quando queremos um gráfico de dispersão, onde não queremos a linha, apenas os pontos

In [5]:

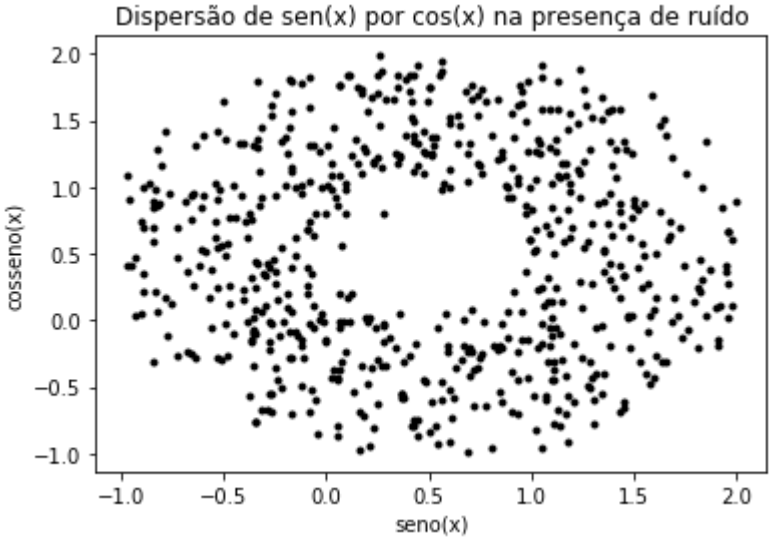
```
import numpy as np
import matplotlib.pyplot as plt

x_vec = np.arange(0, 2 * np.pi, 0.01)
seno = np.sin(x_vec) + np.random.rand(x_vec.shape[0])
cosseno = np.cos(x_vec) + np.random.rand(x_vec.shape[0])
plt.plot(seno, cosseno, 'k.')

#Altera os rótulos dos eixos
plt.xlabel("seno(x)")
plt.ylabel("cosseno(x)")

# Altera o título do gráfico
plt.title("Dispersão de sen(x) por cos(x) na presença de ruído")

plt.show()
```



- Os métodos `xlabel` e `ylabel` são usados para alterar os rótulos dos eixos x e y
- O método `title` é usado para mudar o título do gráfico

Exercícios

Criar um programa do Python que:

1. Gera dois arrays unidimensionais aleatórios com 100 valores entre -1 e 1
2. Cria um terceiro array resultante da multiplicação dos dois primeiros elemento a elemento
3. Mostra um gráfico de dispersão dos dois primeiros arrays com o marcador 'cross' e a cor verde

In [6]:

```
# Espaço para o exercício
```