Python Basics - Numpy and Matplotlib Cheat Sheet

by Mario (mariofreitas) via cheatography.com/42813/cs/13226/

Numpy - Functions

`import numpy as np`	Numpy importation
`np.sin(0)`	Returns sine of *0* (0)
`np.cos(0)`	Returns cosine of *0* (1)
`np.sinh(0)`	Returns hiperbolic sine of *0* (0)
`np.cosh(0)`	Returns hiperbolic cosine of *0* (1)
`np.pi`	Returns the value of *pi* (3.1415...)
`np.array([1, 2, 3])`	Returns a 1-dimensional array
`np.array([[1, 2],[3, 4]])`	Returns a 2-dimensional array
`np.mat([[1, 2],[3, 4]])`	Returns a matrix
`np.linspace(0, 1, 11)`	Returns a 1-dimensional array from *0* to *1* with *11* elements
`np.logspace(1, 3, 11)`	Returns a 1-dimensional array from 10^1 to 10^3 with 11 elements
`np.arange(0, 1, 0.1)`	Returns a 1-dimensional array from *0* to *1* with step *0.1*
`np.random.random((2)`	Returns a random 1-dimensional array with 2 elements
`np.random.random((2, 2))`	Returns a random 2-dimensional array with 2 rows and 2 columns
`np.random.random((2, 2, 2))`	Returns a random 3-dimensional array with 2 elements in each dimension
`np.eye(5)`	Returns a 5x5 identity matrix
`np.zeros(5)`	Returns a null vector with 5 elements
`np.zeros((5, 2))`	Returns a null matrix with 5 rows and 2 columns
`np.ones(5)`	Returns a 5 elements vector filled with 1

Numpy - Functions (cont)

`np.ones((5, 2))`	Returns a 5x2 matrix filled with 1

[1]Numpy has all functions and constants in the library Math.
[2]Numpy arrays can be used to do all sorts of linear algebra calculations since they are treated as mathematical tensors (vectors and matrices) rather than Python lists.
[3]Numpy matrices are a special type of array that is easier to use in common linear algebra problems. Matrices are always 2D

Numpy - Linalg

`np.linalg.dot(a, b)`	Returns the dot product between arrays *a* and *b*
`np.linalg.matmul(a, b)`	Returns the matrix product between arrays *a* and *b*
`np.linalg.eigvals(a)`	Returns the eigenvalues of the square array *a*
`np.linalg.eig(a)`	Returns the eigenvalues and eigenvectors of the square array *a*
`np.linalg.norm(a)`	Returns the *a* array norm
`np.linalg.det(a)`	Returns the determinant of *a*
`np.linalg.inv(a)`	Returns the inverse of *a*
`np.linalg.pinv(a)`	Returns the pseudo-inverse of *a*
`np.linalg.solve(a, b)`	Returns the solution of *aX = b*

Numpy - Slicing

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
a[1:, 1:] = [[50, 60], [80, 90]]
print(a)
```

```
[[ 1 2 3]
[ 4 50 60]
[ 7 80 90]]
```

Matplotlib - Example

```
from matplotlib import pyplot as plt
import numpy as np
x = np.linspace(0, 10, 200)
y1 = 3*np.sin(x) + np.cos(x)
y2 = 2*np.cos(x) - np.sin(x)
```

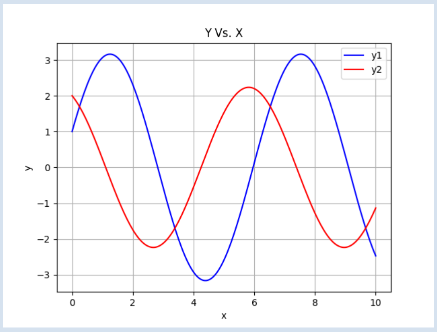
Python Basics - Numpy and Matplotlib Cheat Sheet

by Mario (mariofreitas) via cheatography.com/42813/cs/13226/

Matplotlib - Example (cont)

```
plt.plot(x, y1, '-b', label="y1")
plt.plot(x, y2, '-r', label="y2")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Y Vs. X")
plt.legend()
plt.grid()
plt.show()
```

Matplotlib - Example (Output)



Matplotlib - Pyplot Functions

`from matplotlib import pyplot as plt`	Pyplot importation
`plt.plot(x, y[,args])`	Plots *x* Vs. *y* on current figure
`plt.xlabel("x")`	Sets x-axis label to *x*
`plt.ylabel("y")`	Sets y-axis label to *y*
`plt.title("Y Vs. X")`	Sets figure title to *Y Vs. X*
`plt.legend()`	Shows legends in figure
`plt.grid()`	Shows grid in figure
`plt.show()`	Shows figures
`plt.figure()`	Starts a new figure

[1] Many args can be set on *plt.plot()*. Some are shown on the next block.

Pyplot - Lines, Markers and Colors

'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

You can use a line style, a marker and a color directly at plt.plot() like this: `plt.plot(x, y, '-+b')`