Scientific Computing in

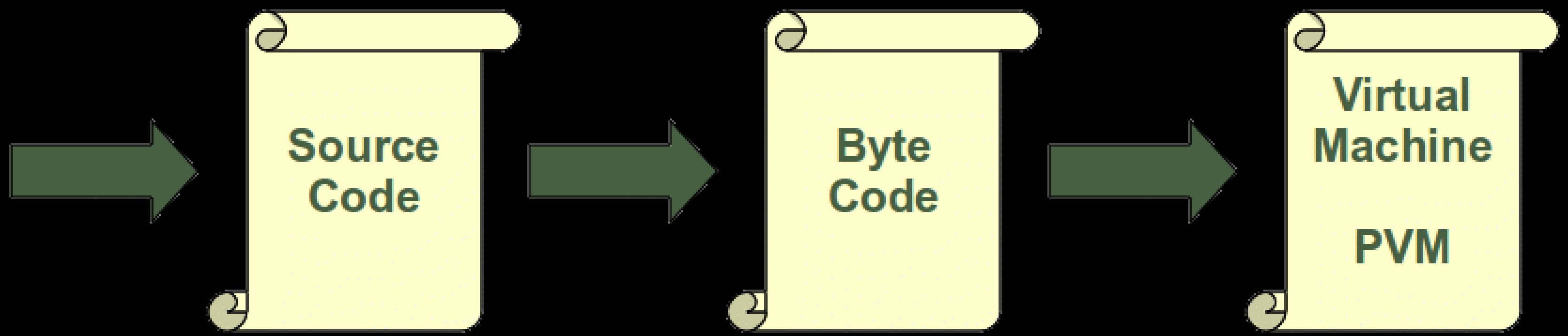
PYTHON

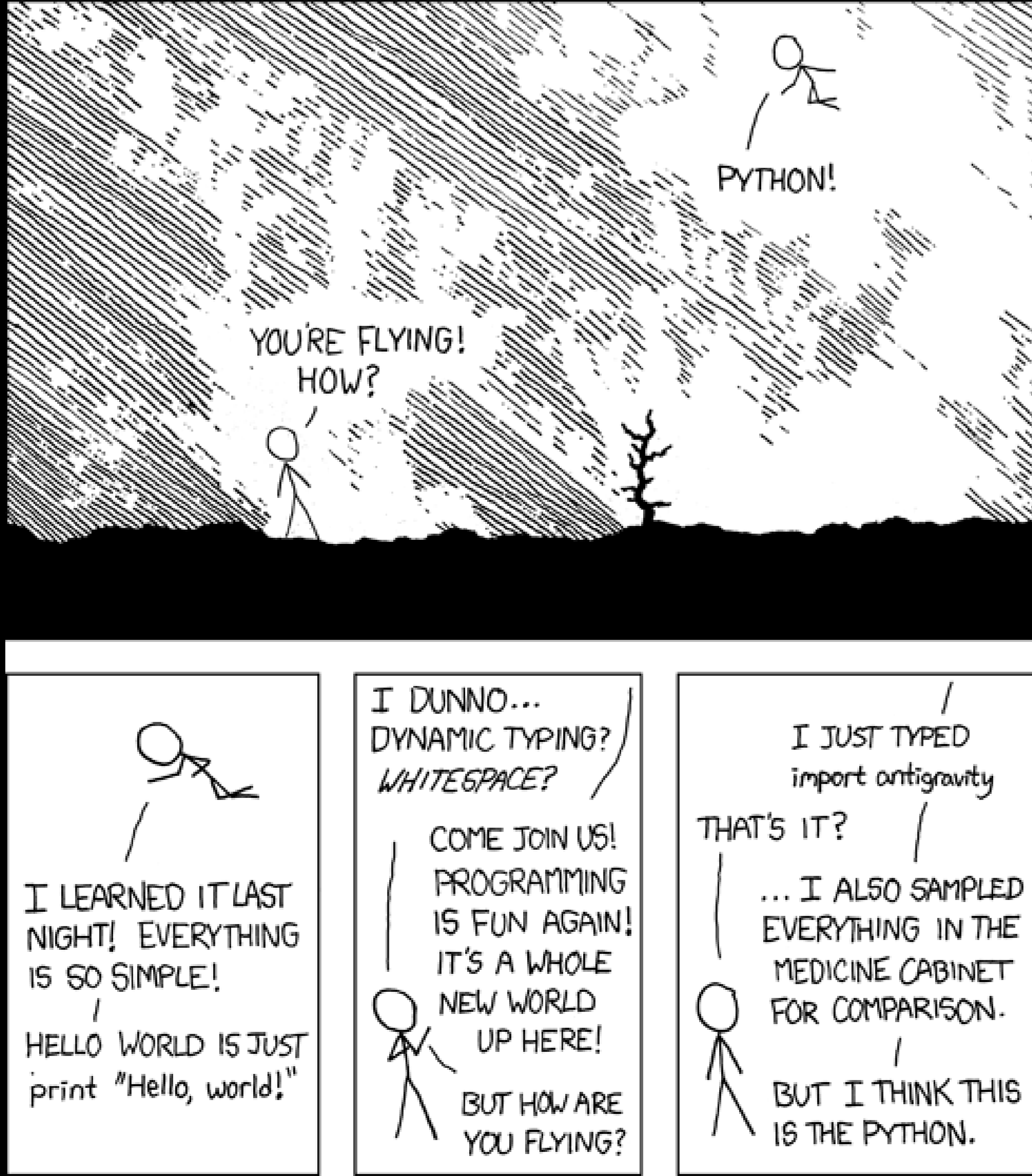
So, what is Python?

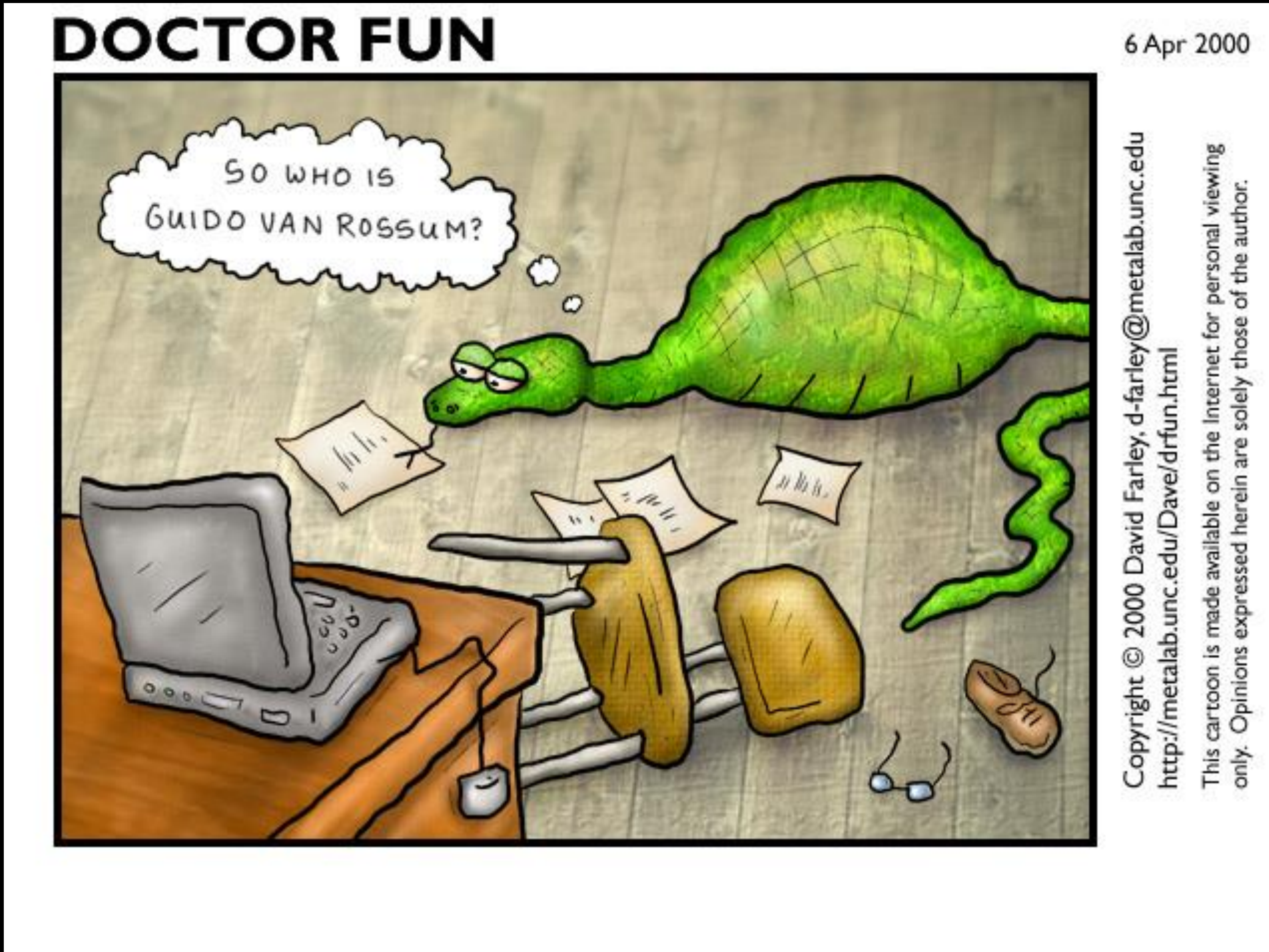
Why Python?

- Powerful programming language, easy to learn
- Efficient high dimensional Data Structure
- Clean Syntax, Code readability
- Object Oriented, imperative, functional computation
- Programs are portable
- Open Source
- Great Community Support
- Interpreter based language

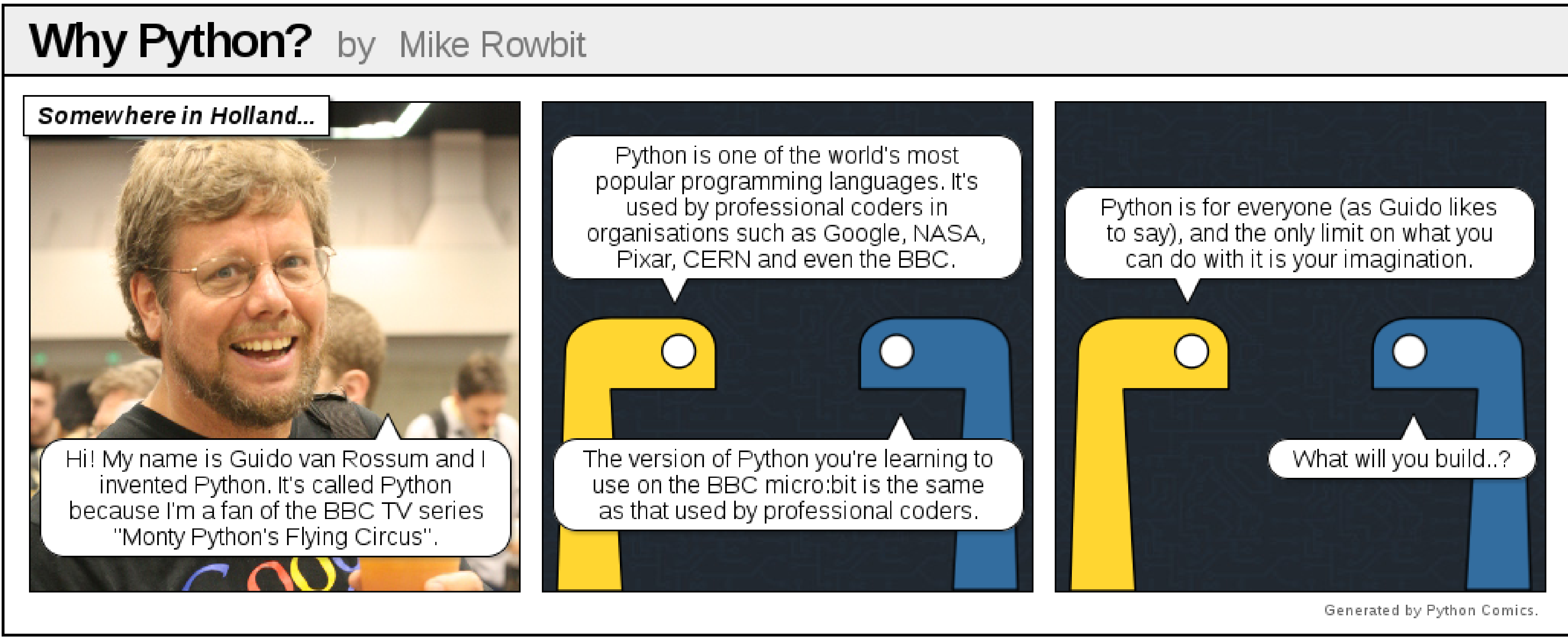
How it works




YOU'RE FLYING! HOW?
PYTHON!
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

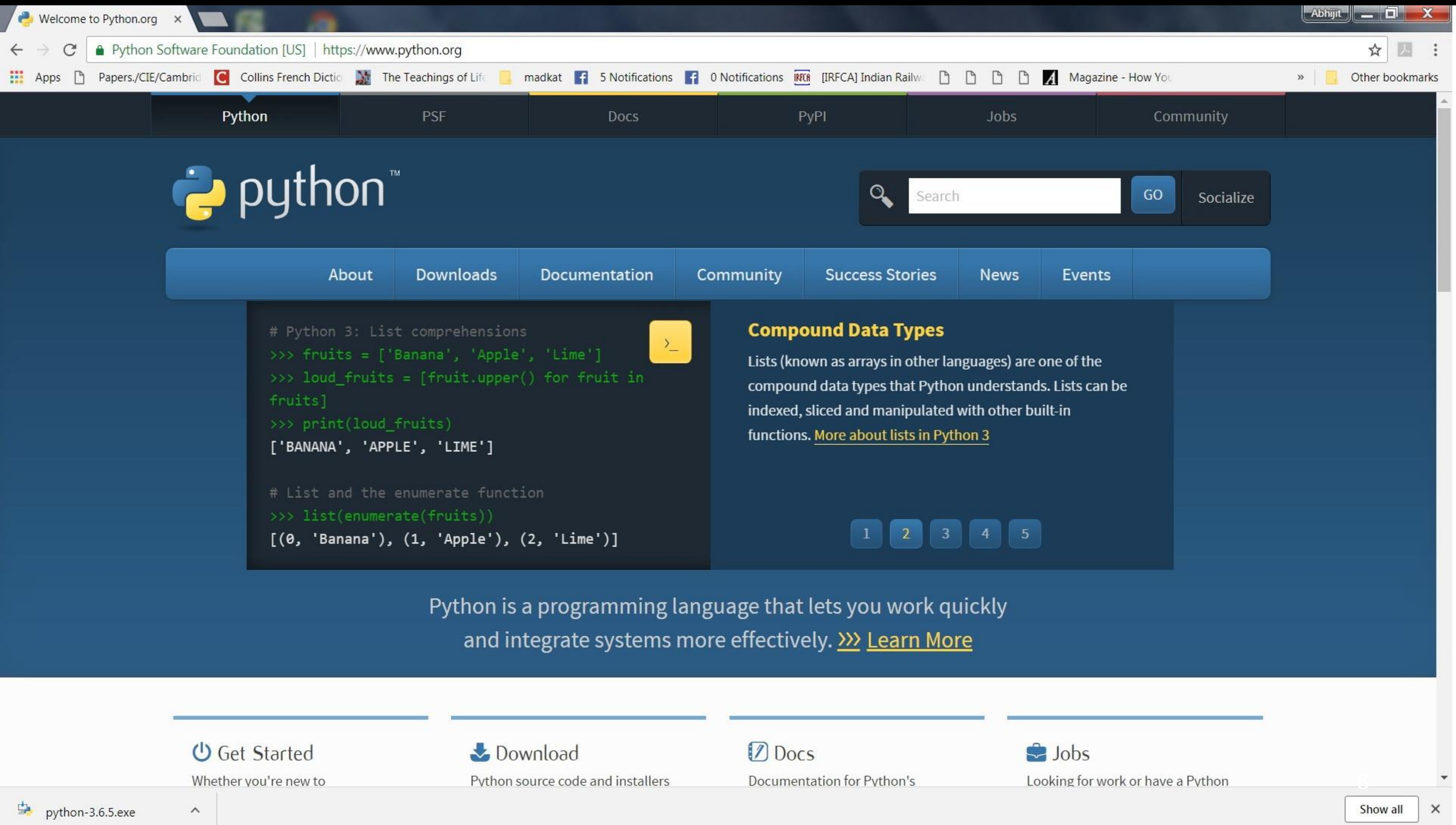

DOCTOR FUN
6 Apr 2000
SO WHO IS GUIDO VAN ROSSUM?
Copyright © 2000 David Farley, d-farley@metalab.unc.edu
http://metalab.unc.edu/Dave/drfun.html
This cartoon is made available on the Internet for personal viewing only. Opinions expressed herein are solely those of the author.

Why Python?

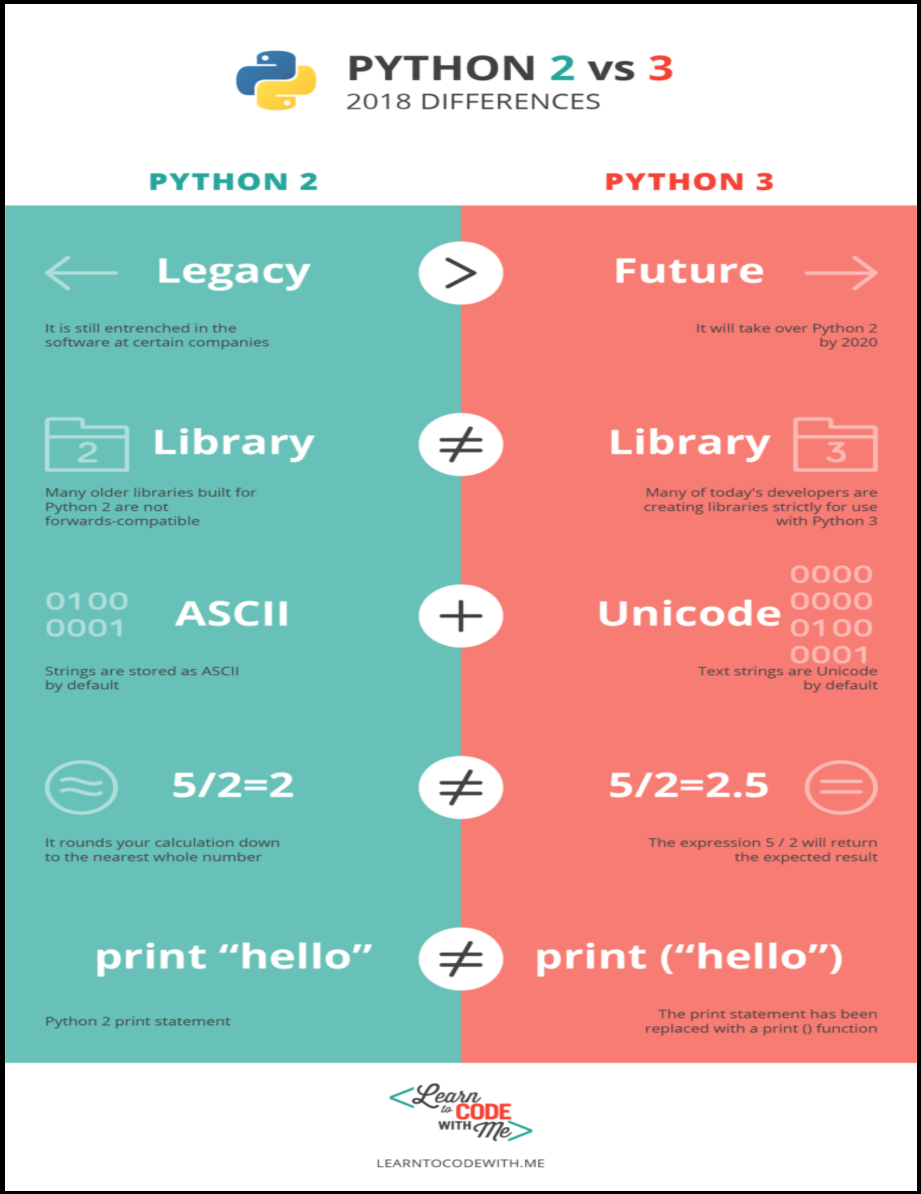


Why Python? by Mike Rowbit
Somewhere in Holland...
Hi! My name is Guido van Rossum and I invented Python. It's called Python because I'm a fan of the BBC TV series "Monty Python's Flying Circus".
Python is one of the world's most popular programming languages. It's used by professional coders in organisations such as Google, NASA, Pixar, CERN and even the BBC.
The version of Python you're learning to use on the BBC micro:bit is the same as that used by professional coders.
Python is for everyone (as Guido likes to say), and the only limit on what you can do with it is your imagination.
What will you build..?
Generated by Python Comics.

www.python.org

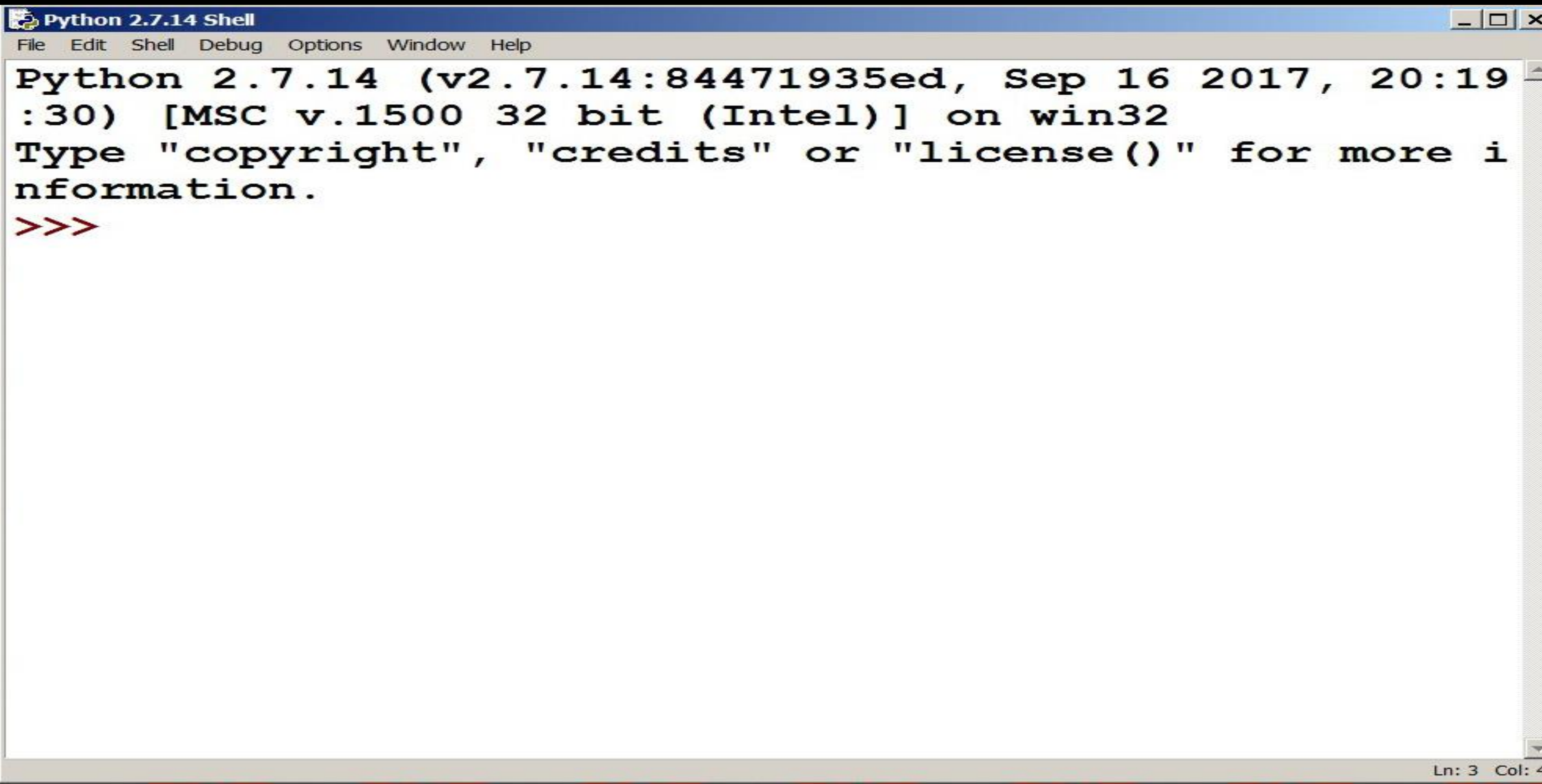


Python 2 or Python 3?



Install Python 2.7.14 exe file, on Windows, get Python 2.7.14 Shell

PYTHON IDLE



On Linux

- Linux
  PYTHON is often preinstalled!

  You can type on command line.
  For a Python Script (code), you can choose any of the existing editors like ‘*vim*’, ‘*gedit*’ or anything.

  To run:
  Write on Linux command line:

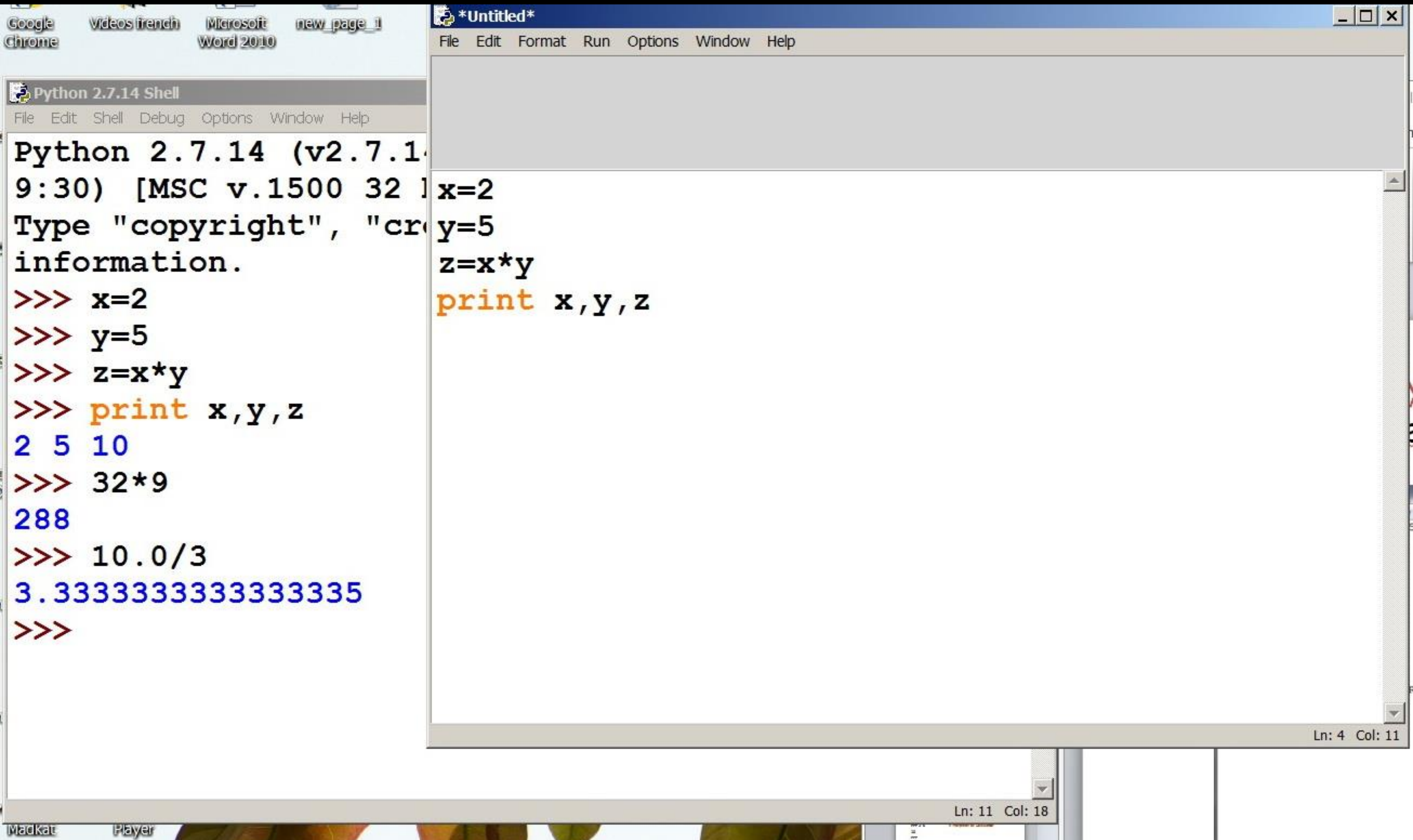
  $python filename.py

On Python Interpreter

As you enter, new '>>>' comes!

```
>>> x = 2
>>> y = 5
>>> z = x*y
>>> print x, y, z
2  5 10

>>> 2*5                    # Interpreter as Calculator
10
>>>
```

For Python Script, open a file
File Name: Something.py



PYTHON anytime, anywhere

On Android Phones

Install any suitable App.

For example, Qpython, Pydroid…

For Documents/ Tutorials:

- www.python.org
- www.scipy.org
- www.tutorials.com/python
- w3schools.com/python
- realpython.com

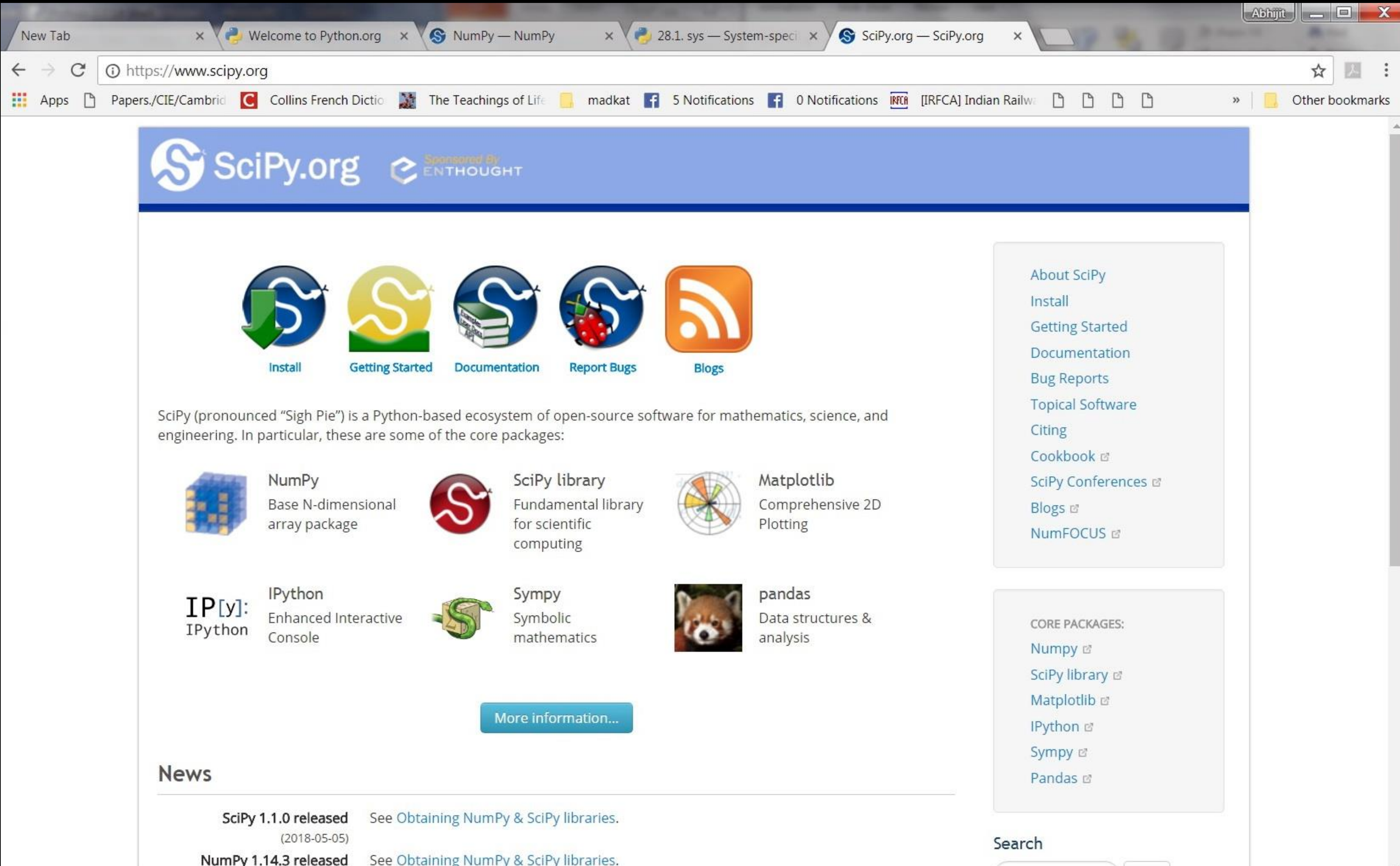
* Video and other tutorials also come with Apps on Mobile ph.

Scientific and Plotting packages

- NUMPY [Array Manipulation]
- SCIPY [Packages for FFT, Linear Algebra etc.]
- MATPLOTLIB [Plotting from within the Python Script]
- PANDAS [Data Structure]

Go to www.scipy.org and check.

Python(x,y): Python distribution with scientific packages for Windows.


New Tab
Welcome to Python.org
NumPy — NumPy
28.1. sys — System-speci
SciPy.org — SciPy.org
https://www.scipy.org
Apps
Papers./CIE/Cambric
Collins French Dictio
The Teachings of Life
madkat
5 Notifications
0 Notifications
[IRFCA] Indian Railw
Other bookmarks
SciPy.org
ENTHOUGHT
Install
Getting Started
Documentation
Report Bugs
Blogs
SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:
NumPy
Base N-dimensional array package
SciPy library
Fundamental library for scientific computing
Matplotlib
Comprehensive 2D Plotting
IPython
Enhanced Interactive Console
Sympy
Symbolic mathematics
pandas
Data structures & analysis
More information...
News
SciPy 1.1.0 released (2018-05-05)
See Obtaining NumPy & SciPy libraries.
NumPy 1.14.3 released
See Obtaining NumPy & SciPy libraries.
About SciPy
Install
Getting Started
Documentation
Bug Reports
Topical Software
Citing
Cookbook
SciPy Conferences
Blogs
NumFOCUS
CORE PACKAGES:
Numpy
SciPy library
Matplotlib
IPython
Sympy
Pandas
Search

How to install a package?

Windows:

c:\python27\scripts> pip install numpy

Linux:

Command line> sudo apt install python-numpy python-scipy

Administrator: C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Abhijit>cd..

C:\Users>cd..

C:\>cd python27\scripts

C:\Python27\Scripts>pip install numpy.whl_
```

Python in-built Libraries/ Modules

- import math

  Math functions: `sqrt(),abs(),exp(), sin()`...
- `import cmath`
- import sys

  System specific parameters and functions
- import random

  Random numbers of various kinds

Handling Numbers

- Integer
- Real
- Complex
- Random numbers

Number Operations

Arithmetic Operators	+, - *, /, **, %, //
Comparison Operators	==, !=, >, <, >=, <=, <>
Assignment Operators	=, +=, -=, *=, /=, %=, **=, //=
Bitwise Operators	a&b, a|b, a^b, ~a, a << 2, a >> 2 (Bit by bit operations between two binary numbers.)
Logical Operators	and, or, not
Membership Operators	in, not in
Identity Operators	is, is not

For random Numbers

```
>>> import random
>>> random.random()
0.48159822008044695
```

Variables

- Variable names can be of any length.
- But they should not be the reserved words.
- Words with special key-board characters, not allowed
- Words beginning with numbers, not allowed
- Case sensitive

To create a range of integers

```
>>> x = range(10)
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> sum(x)
45
>>> x = range(2, 11, 2)
>>> x
[2, 4, 6, 8, 10]          #Becomes a List
>>> sum(x)
30
```

Input/ Output

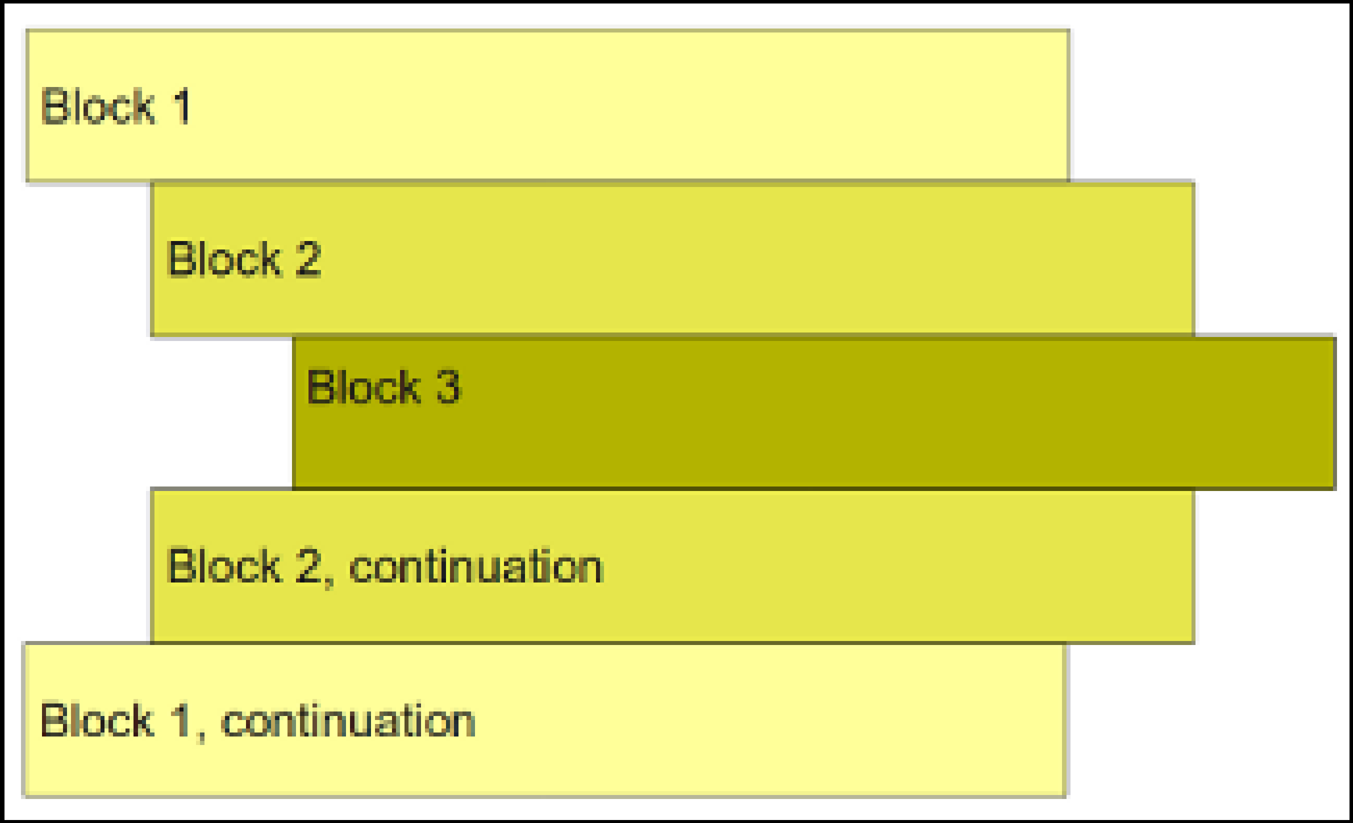
Input function:

```
input()                # The input will be interpreted.
raw_input()            #  Does not interpret the input.
```

Output function:

```
print x, y
f = open('file name', 'w')
print >> f, x, y
```

Structure and Indentation



Loop

For Loop

```
for i in range(10):
        print i
```

While Loop

```
i = 1
while i < 10:
     print i
     i = i + 1
```

Logical Structure

```
a, b, c = input('Enter a, b, c: \n')
x = b*b-4*a*c
if x < 0:
   print 'x is negative.'
elif x > 0:
   print 'x is positive.'
else:
   print 'x is zero.'
```

User defined function

```
def f(x):
    return x*x
print f(5)
```

Python Script: Example

```
import sys
n = input('Enter a number: \n')
for i in range(2,n):
    if n%i == 0:
        print n, 'is not a prime number'
        sys.exit()
print n, 'is a prime number`
```

Special Constructs
for Collection of data

- `List []`
- `String ' '`
- `Tuple (,)`
- `Set {1,2}`
- `Dictionary {1:10, 2:20}`

Class in Python

Object:

Combination of defined variables, data structures and functions (or methods).

Class:

To construct an object, we need to create a 'blueprint' which is called 'Class'. So, a class can be thought of as an object *constructor*. On the other hand, an object is an *instance* of a Class.

```
>>> class One:
            x = 10
>>> p = One()
>>> p.x
10
```

```
>>> class Two:
        'This is my Class'
        x, y = 2, 3
        z = x*y
>>> p = Two()
>>> p.x
2
>>> p.y
3
>>> p.z
6
>>> p.__doc__
'This is my Class'
```

Class with built-in function

```
>>> class Three:
        x = 10
        def __init__(self, y):
        self.y = y
>>> p1 = Three(20)
>>> p2 = Three(30)
>>> p1.x                  # x is global variable
10
>>> p2.x
10
>>> p1.y                  # y is local variable
20
>>> p2.y
30
```

```
>>> class Snake:
            species = 'reptile'
            def __init__(self, name):
                self.name = name

>>> s1 = Snake('Python')
>>> s2 = Snake('Anaconda')
>>> s1.species
'reptile'
>>> s2.species
'reptile'
>>> s1.name
'Python'
>>> s2.name
'Anaconda'
```

```
>>> class Four:
        "This is a better Class"
         def __init__(self, x, y):
             self.x = x
             self.y = y
         def func(self):
             z = self.x + self.y
             return z

>>> p = Four(5, 10)
>>> p.x
5
>>> p.y
10
>>> p.func()
15
>>> p.__doc__
'This is a better Class'
```

```
>>> class Five:
        "This is even better Class"
        def __init__(self, x, y):
            self.x = x
            self.y = y
        def add(self):
            z = self.x + self.y
            return z
        def dis(self):
            import math
            d = math.sqrt(self.x**2+self.y**2)
            return d

>>> p = Five(3,4)
>>> p.add()
7
>>> p.dis()
5.0
```

```
>>> class Six:
        "This is some modification over fifth Class"
        def __init__(self, x, y):
                self.x = x
                self.y = y
        def add(arg):
                z = arg.x + arg.y
                return z
        def dis(par):
                import math
                d = math.sqrt(par.x**2+par.y**2)
                return d

>>> p = Six(3,4)
>>> p.add()
7
>>> p.dis()
5.0
>>> p.__doc__
'This is some modification over fifth Class'
>>> p.x
3
>>> p.y
4
```

import

- Write the **class** **Six** in a file.
- File name: six.py (say,)

Now…

```
>>> import six
>>> p = six.Six(2,3)
>>> p.add()
5
>>> p.dis()
3.605551275463989
>>> p.x
2
>>> p.y
3
```

Deleting attributes and objects

```
>>> del p.x
>>> p.x = 6
>>> p.add()
10
>>> del p
>>> p

Traceback (most recent call last):
  File "<pyshell#284>", line 1, in
<module>
    p
NameError: name 'p' is not defined
```

Inheritance

- **Parent Class**
- **Child Class**

A child class is inherited from a parent class.

Inheriatnce: Example

```
>>> class Seven(Six):
        pass

>>> p = Seven(2,3)
>>> p.add()
5
>>> p.dis()
3.605551275463989
```

Note: In the above, the class 'Seven' is a child class of the parent class 'Six'.

Inheritance: Example

```
>>> class Seven(Six):
        def __init__(self, x, y):
            Six.__init__(self, x, y)
            self.z = self.x*self.y

>>> p = Seven(2,3)
>>> p.x
2
>>> p.y
3
>>> p.z
6
```

Inheritance: Example

Method added…

```
>>> class Seven(Six):
      def __init__(self, x, y, z):
            Six.__init__(self, x, y)
            self.z = z
      def f(self):
            a = self.z*self.z
              return a
              # print 'a = ', a

>>> p = Seven(2,3,4)
>>> p.x
2
>>> p.y
3
>>> p.z
4
>>> p.f()
16
```

```
>>> class Vector:
        """This is a Vector Class"""
        i = (1,0,0)
        j = (0,1,0)
        k = (0,0,1)

        def __init__(s, vx,vy,vz):
                s.x = vx
                s.y = vy
                s.z = vz

        def norm(s):
                import math
                xx, yy, zz = s.x**2, s.y**2, s.z**2
                return math.sqrt(xx+yy+zz)

       @classmethod
        def ijk(cls):
                return cls.i + cls.j + cls.k
        @staticmethod
        def xxx(r):
                return r**2
```

```
>>> v = Vector(2,3,4)
>>> v.norm()
5.385164807134504
>>> v.ijk()
(1, 0, 0, 0, 1, 0, 0, 0, 1)
>>> v.xxx(2)
4
>>> v.i
(1, 0, 0)
>>> v.j
(0, 1, 0)
>>> v.k
(0, 0, 1)
```

About Formatting

```
>>> x, y, z = 2.5, 0.34, 5
>>> x
2.5
>>> print x
2.5
>>> str(x)
'2.5'
>>> print str(x)
2.5

>>> print x, y, z
2.5 9.34 5
>>> print 'x =' + str(x) + ' ' 'y = ' + str(y) + ' ' + 'z = ' + 
str(z)
x = 2.5 y = 0.34 z = 5
```

The Pythonic Way: The String format method

```
>>> "{0}, {1}, {2}".format(x,y,z)
'2.5, 9.34, 5'
>>> print "{0}, {1}, {2}".format(x,y,z)
2.5, 9.34, 5

>>> print "{2}, {1}, {0}".format(x,y,z)
5, 9.34, 2.5

>>> print "{0:.2f}, {1:.3f}, 
{2:3d}".format(x,y,z)
2.50,  9.340,   5
```

Exponential format
[0-9].e[0-9]
	`>>> "{:e}".format(0.00015)`
	`'1.500000e-04'`
	`>>> "{:E}".format(0.00015)`
	`'1.500000E-04'`

Fixed point representation
 	
Precision
	`>>> print "{:.16f}".format(22.0/7)`
	`3.1428571428571428`
	
	`>>> print "{:.16}".format(22.0/7)`

Decimal representation	`>>> print "{:d}".format(72)` `72`

Binary representation	`>>> print "{:b}".format(72)` `1001000`
Octal representation	`>>> print "{:o}".format(72)` `110`

Hexadecimal representation	`>>> print "{:x}".format(3487)`
	`d9f`
	`>>> print "{:X}".format(3487)`
	`D9F`

Character representation (ASCII)	`>>> print "{:c}".format(65)`
	`A`
	`>>> print "{:c}".format(80)`
	`P`
	`>>> print "{:c}".format(66)`
	`B`
	`>>> print "{:c}".format(64)`
	`@`

Thousand separator	`>>> print "{:,}".format(1000000)` `1,000,000`
Percentage	`>>> print "{:%}".format(0.35)` `35.000000%`
General format	`>>> print "{:g}".format(22.0/7)` `3.14286`

Scientific Calculations, Data Fitting, Plotting by Python

Eigen values, Eigen vectors by Numpy

```
>>> import numpy as np
>>> import numpy.linalg as lin
>>> A = np.array([[1,2],[3,4]])
>>> lin.eig(A)
(array([-0.37228132,  5.37228132]), array([[-
0.82456484, -0.41597356],[ 0.56576746, -
0.90937671]]))
>>> eigen_val, eigen_vec = lin.eig(A)

>>> eigen_val
array([-0.37228132,  5.37228132])
>>> eigen_vec
array([[-0.82456484, -0.41597356],[0.56576746, -
0.90937671]])
>>> eigen_vec[:,0]
array([-0.82456484,  0.56576746])
>>> eigen_vec[:,1]
array([-0.41597356, -0.90937671])
```

Polynomial by Numpy

```
>>> from numpy import poly1d
>>> p = np.poly1d([1,2,3])
>>> print p
```

$1\ x^2\ +\ 2\ x\ +\ 3$

```
>>> p(2)
11
>>> p(-1)
2
>>> p.c                    # Coefficients
array([1, 2, 3])
>>> p.order                # Order of the polynomial
2
```

Methods on Polynomials

```
>>> from numpy import poly1d as poly
>>> p1 = poly([1,5,6])
>>> p2 = poly([1,2])
>>> p1+p2
poly1d([1, 6, 8])
>>> p1*p2
poly1d([ 1,  7, 16, 12])
>>> p1/p2
(poly1d([1., 3.]), poly1d([0.]))
>>> p2**2
poly1d([1, 4, 4])
>>> from numpy import sin
>>> sin(p2)
array([0.84147098, 0.90929743])
```

```
>>> p = np.poly1d([1, -5, 6])
>>> p.r
array([3., 2.])           # Real roots: 3, 2
>>> p.deriv(1)            # First derivative
poly1d([2, 2])
>>> p.deriv(2)            # Second derivative
poly1d([2])
>>> p.integ(1)
poly1d([0.33333333, 1. , 3. , 0. ])
>>> p.integ(2)
poly1d([0.08333333, 0.33333333,1.5,0.,0. ])
```

Plotting by Matplotlib

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> x = np.arange(0,10,0.1)
>>> y = x**2
>>> plt.plot(x,y)
[<matplotlib.lines.Line2D object at
0x000000000BE00B70>]
>>> plt.show()
```

To plot the polynomial

```
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(0,10,100)
>>> plt.plot(x,p(x))
>>> plt.show()
```

Curve fitting by Polynomial

x	0	10	20	30	40	50	60	70	80	90
y	76	92	106	123	132	151	179	203	227	249

To fit the above data by polynomial…

Step 1:

```
>>> import numpy as np
>>> x = np.array([0,10,20,30,40,50,60,70,80,90])
>>> y = np.array([76,92,106,123,132,151,179,203,227,249])
```

Step 2:

```
>>> import numpy.polynomial.polynomial as poly
>>> coeffs = poly.polyfit(x, y, 2)
>>> coeffs
array([7.81909091e+01, 1.10204545e+00, 9.12878788e-03])
```

Step 3:

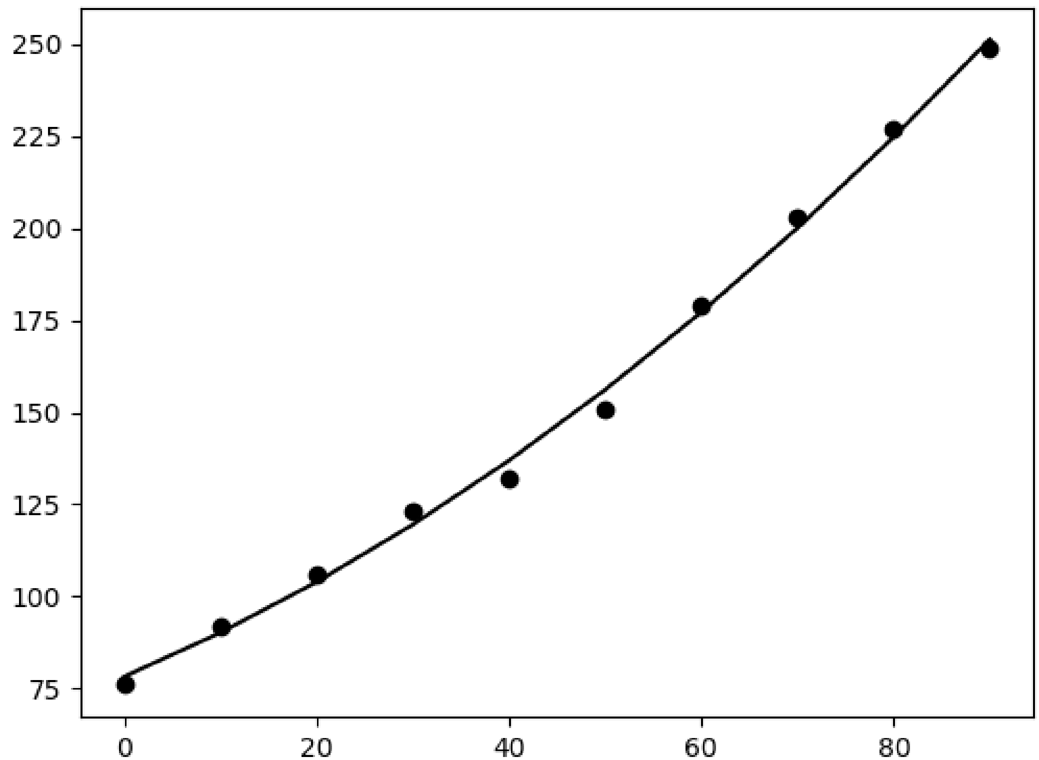
```
>>> yfit = poly.polyval(x,coeffs)
>>> yfit
array([ 78.19090909,  90.12424242, 103.88333333,
119.46818182,136.87878788, 156.11515152, 177.17727273,
200.06515152,224.77878788, 251.31818182])
```

Step 4:

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(x, y, x, yfit )
>>> plt.show()
```

Python Script for Polynomial fitting

```
# Polynomial fitting by Numpy (with plot)

import numpy as np
import numpy.polynomial.polynomial as poly
import matplotlib.pyplot as plt

x = np.array([0,10,20,30,40,50,60,70,80,90])
y = np.array([76,92,106,123,132,151,179,203,227,249])

coeffs = poly.polyfit(x,y,2)
yfit = poly.polyval(x,coeffs)

plt.plot(x, y, 'ko', x, yfit, 'k-')
plt.title('Fitting by polyfit', size = '20')
plt.show()
```


250
225
200
175
150
125
100
75
0
20
40
60
80

Fitting with user defined function

Input Data

```
>>> import numpy as np
>>> x = np.array([0,10,20,30,40,50,60,70,80,90])
>>> y = np.array([76,92,106,123,132,151,179,203,227,249])
```

Define fitting function

```
>>> def f(x,a,b,c):
        return a + b*x + c*x**2
```

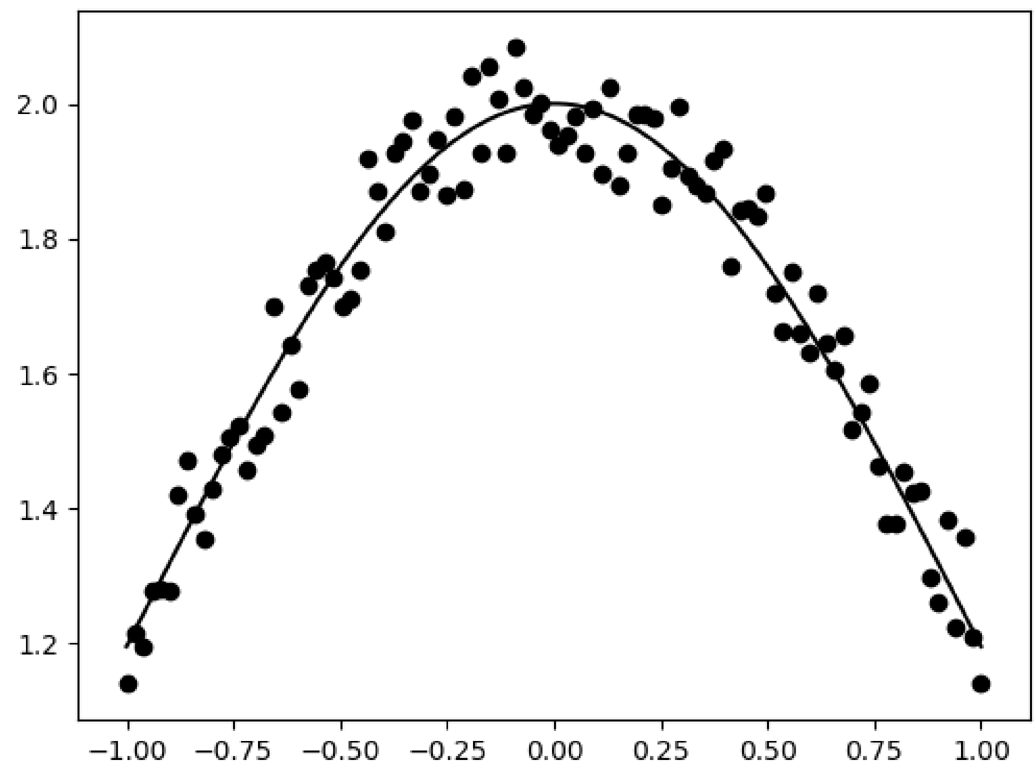
Optimize the parameters

```
>>> from scipy.optimize import curve_fit
>>> par, var = curve_fit(f, x, y)
>>> a, b, c = par
```

To plot

```
>>> plt.plot(x, f(x,a,b,c))
```

```python
# Python Script: Curve fitting by user defined function
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# Define fitting function
def f(x, a, b):
    return a*np.exp(-b*x**2)

# Input Data
x = np.linspace(-1, 1, 100)
y = f(x, 2, 0.5)
y = y + 0.1*np.random.uniform(-1,1,100)

# Optimization
par, var = curve_fit(f,x,y)

# Plot and show
plt.plot(x, f(x, par[0], par[1]))
plt.scatter(x,y)
plt.show()
```


2.0
1.8
1.6
1.4
1.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Legendre Polynomial

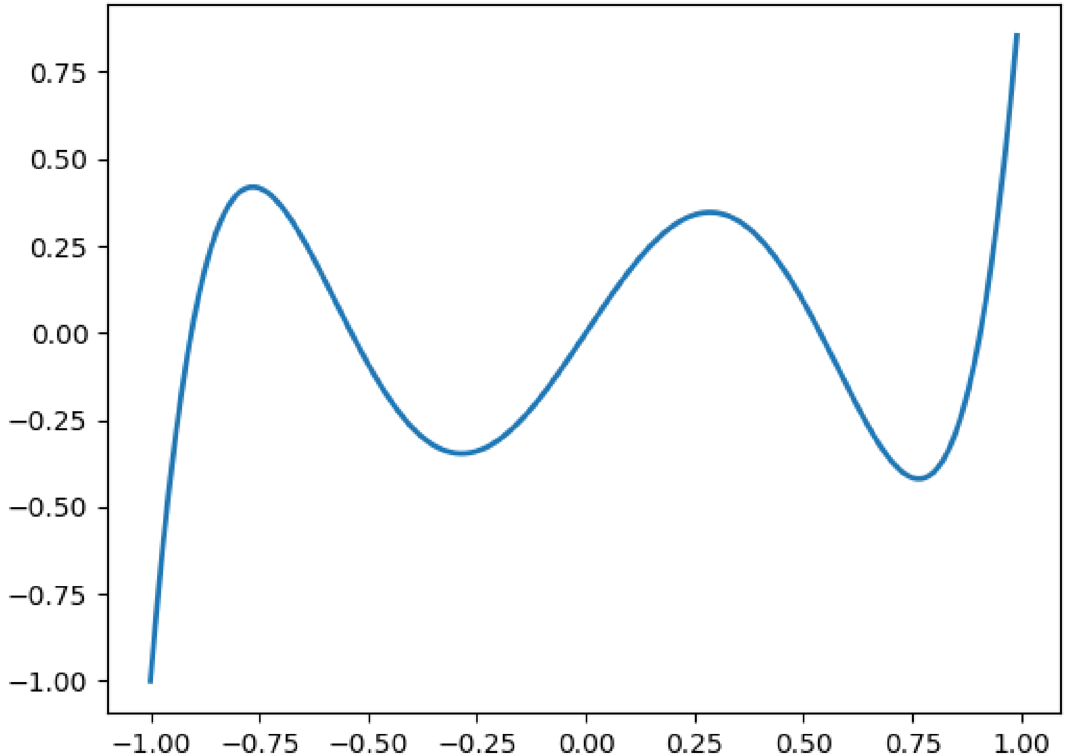
$$P_0(x) = 1 \quad ,$$

$$P_1(x) = x = 1.x + 0,$$

$$P_2(x) = \tfrac{1}{2}(3x^2 - 1) = \tfrac{3}{2}x^2 + 0.x - \tfrac{1}{2},$$

$$P_3(x) = \frac{1}{2}(5x^3 - 3x) = \frac{5}{2}x^3 + 0.x^2 - \frac{3}{2}x + 0$$

```
>>> from scipy.special import legendre
>>> legendre(1)
poly1d([1., 0.])
>>> legendre(2)
poly1d([ 1.5,  0. , -0.5])
>>> legendre(3)
poly1d([ 2.5,  0. , -1.5,  0. ])
>>> p = legendre(5)
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.arange(-1,1,0.01)
>>> plt.plot(x, p(x), lw = 2)
>>> plt.show()
```


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Integration

Composite Simpson's $1/3^{rd}$ rule:

$$I = \int_a^b f(x)dx$$

$y = f(x),$
$y_0 = f(a) \equiv f(x_0),$
$y_n = f(b) \equiv f(x_n),$
$h = \dfrac{b-a}{n}$

$$I = \frac{\boldsymbol{h}}{\mathbf{3}}[\boldsymbol{y_0} + \mathbf{4}(\boldsymbol{y_1} + \boldsymbol{y_3} + \cdots . \boldsymbol{y_{n-1}}) + \mathbf{2}(\boldsymbol{y_2} + \boldsymbol{y_4} + \cdots . \boldsymbol{y_{n-2}}) + \boldsymbol{y_n}]$$

```
# Integration by Composite Simpson's Rule

def f(x):
    return x**2

a, b, n = input('Lower limit, Upper limit, n \n')

h = float(b-a)/n
sum = f(a) + f(b)
d = 4

for k in range(1, n):
    x = a + k*h
    sum += d*f(x)
    d = 6-d

sum = h/3*sum
print 'Value of Integral = ', sum
```

Calculations with Scipy

```
>>> from scipy.integrate import simps
>>> import numpy as np
>>> x = np.linspace(0,2,1000)
>>> def f(x):
     return x*x

>>> y = f(x)
>>> simps(y,x)
2.6666666800400 8
```

Ordinary Differential Equations

Solutions by RK4:

$$\frac{dy}{dx} = f(x, y)$$

$$\boldsymbol{k_1 = h.f(x_0, y_0)}$$

$$\boldsymbol{k_2 = h.f(x_0 + \frac{h}{2}, y_0 + k_1/2)}$$

$$\boldsymbol{k_3 = h.f(x_0 + \frac{h}{2}, y_0 + k_2/2)}$$

$$\boldsymbol{k_4 = h.f(x_0 + h, y_0 + k_3)}$$

$$\boldsymbol{y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).}$$

1^{st} order ODE: $\frac{dy}{dx} = (1 + x)y + 1 - 3x + x^2 = f(x, y)$

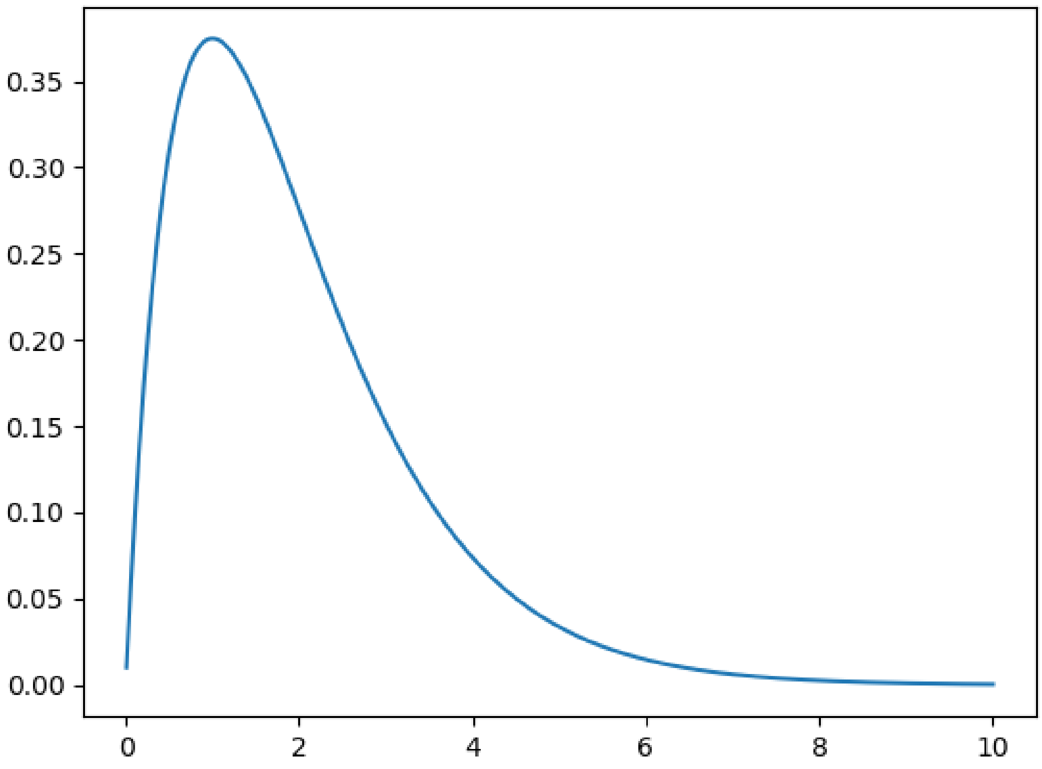
```
# Runge-Kutta 4th Order (RK4)

def f(x,y): return (1+x)*y + 1 - 3*x + x*x

x, y, h = 0.0, 0.0, 0.05

for i in range(100):
    k1 = h*f(x,y)
    k2 = h*f(x+0.5*h,y+0.5*k1)
    k3 = h*f(x+0.5*h,y+0.5*k2)
    k4 = h*f(x+h,y+k3)
    y = y + (k1+2*k2+2*k3+k4)/6.0
    x = x + h

    print x, y
```

2nd order ODE: $\frac{d^2y}{dx^2} + 2\frac{dy}{dx} + y = 0$

$$\frac{dy}{dx} = z = f(x, y, z)$$

$$\frac{dz}{dx} = -2z - y = g(x, y, z)$$

```
# SHM Equation by RK4 Method
import matplotlib.pyplot as plt
def f(x,y,z): return z
def g(x,y,z): return -2*z -y

x, y, z, h = 0, 0, 1.0, 0.01
X, Y, Z = [], [], []

for i in range(1000):
    a1 = h*f(x,y,z)
    a2 = h*f(x+0.5*h,y+0.5*a1,z+0.5*a1)
    a3 = h*f(x+0.5*h,y+0.5*a2,z+0.5*a2)
    a4 = h*f(x+h,y+a3,z+a3)
    y = y +(a1+2*a2+2*a3+a4)/6.0

    b1 = h*g(x,y,z)
    b2 = h*g(x+0.5*h,y+0.5*b1,z+0.5*b1)
    b3 = h*g(x+0.5*h,y+0.5*b2,z+0.5*b2)
    b4 = h*g(x+h,y+b3,z+b3)
    z = z + (b1+2*b2+2*b3+b4)/6.0

    x = x + h
    X.append(x)
    Y.append(y)
    Z.append(z)
plt.plot(X,Y)
plt.show()
```



0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10

2nd Order ODE by Scipy

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

def shm(u, x):
    y, z = u
    f = z
    g = -2*z - y
    return [f, g]

u = [0, 1]                                  # Intial values
x = np.linspace(0,10,1000)

sol = odeint(shm, u, x)

x = sol[:,0]
y = sol[:,1]

plt.plot(x,X)
Plt.show()
```

Lorenz Curve:

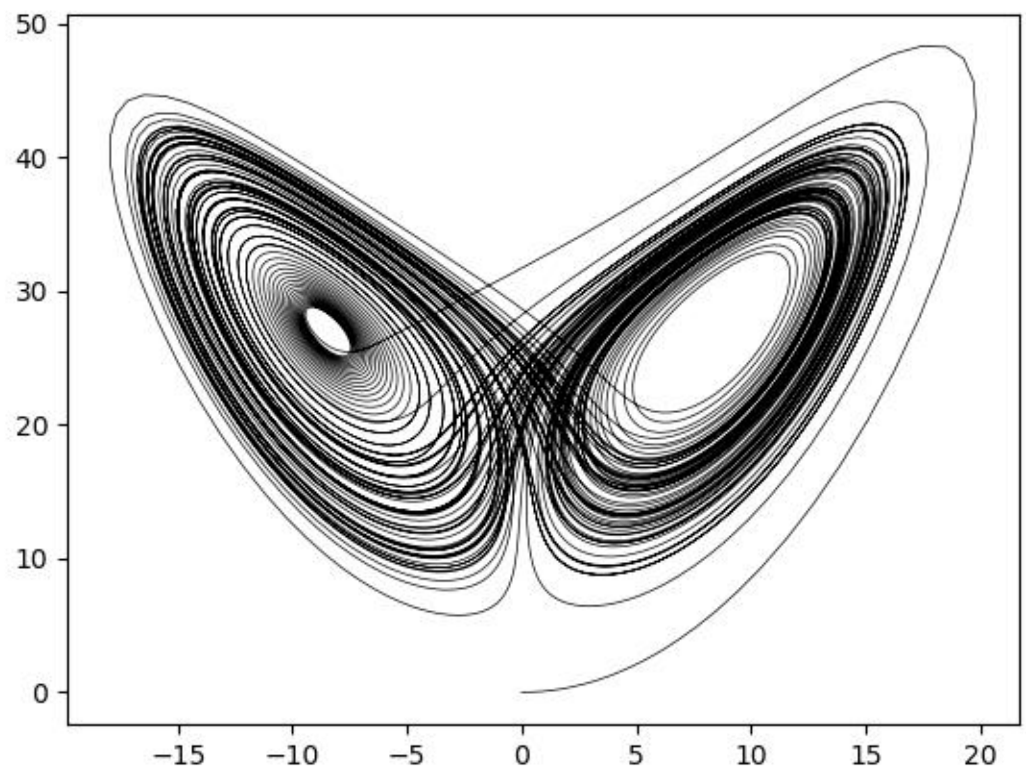
$$\frac{dx}{dt} = \sigma(\boldsymbol{y} - \boldsymbol{x}) \ \mathbf{= f(x,y,z)},$$

$$\frac{dy}{dt} = x(\rho - z) - y \ \mathbf{= g(x,y,z)},$$

$$\frac{dz}{dt} = xy - \beta z = h(x,y,z)$$

The Values of the parameters, Lorenz used: $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$

```python
# For Lorenz Curve: Solution by odeint()

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

sig, rho, beta = 10.0, 28.0, 8.0/3

def lorenz(u,t):
    x, y, z = u
    f = sig*(y-x)
    g = x*(rho-z)- y
    h = x*y - beta*z
    return [f, g, h]

u0 = [0, 1.0, 0]
t = np.linspace(0,100,10000)

sol = odeint(lorenz, u0, t)

x, y, z = sol[:,0], sol[:,1], sol[:,2]

plt.plot(x,z, 'k-', lw = 0.5)
plt.show()
```


50
40
30
20
10
0
−15
−10
−5
0
5
10
15
20

Solving predator-prey system of equations

$$\frac{dx}{dt} = x(2 - y - x)$$

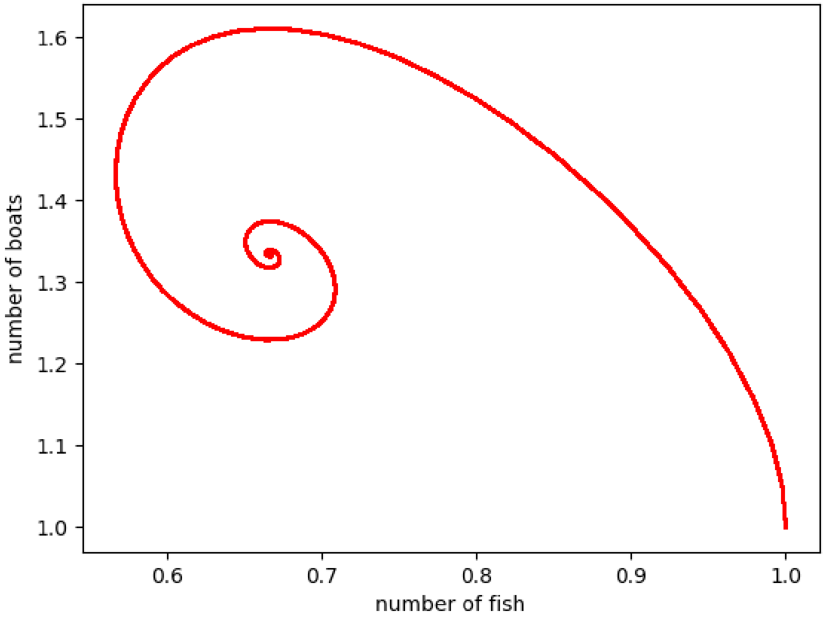
$$\frac{dy}{dt} = -y(1 - 1.5x)$$

$x =$ fish

$y =$ fishing boats

```
import matplotlib.animation as animation
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np

def BoatFishSystem(state, t):
    fish, boat = state
    d_fish = fish * (2 - boat - fish)
    d_boat = -boat * (1 - 1.5 * fish)
    return [d_fish, d_boat]

t = np.arange(0, 100, 0.1)
init_state = [1, 1]
state = odeint(BoatFishSystem, init_state, t)

fig = plt.figure()
plt.xlabel('number of fish')
plt.ylabel('number of boats')
plt.plot(state[:, 0], state[:, 1], 'b-', alpha=0.2)

def animate(i):
    plt.plot(state[0:i, 0], state[0:i, 1], 'r-')

ani = animation.FuncAnimation(fig, animate, interval=1)
plt.show()
```

Run the previous Script.. See animation.



Fast Fourier Transform (FFT)

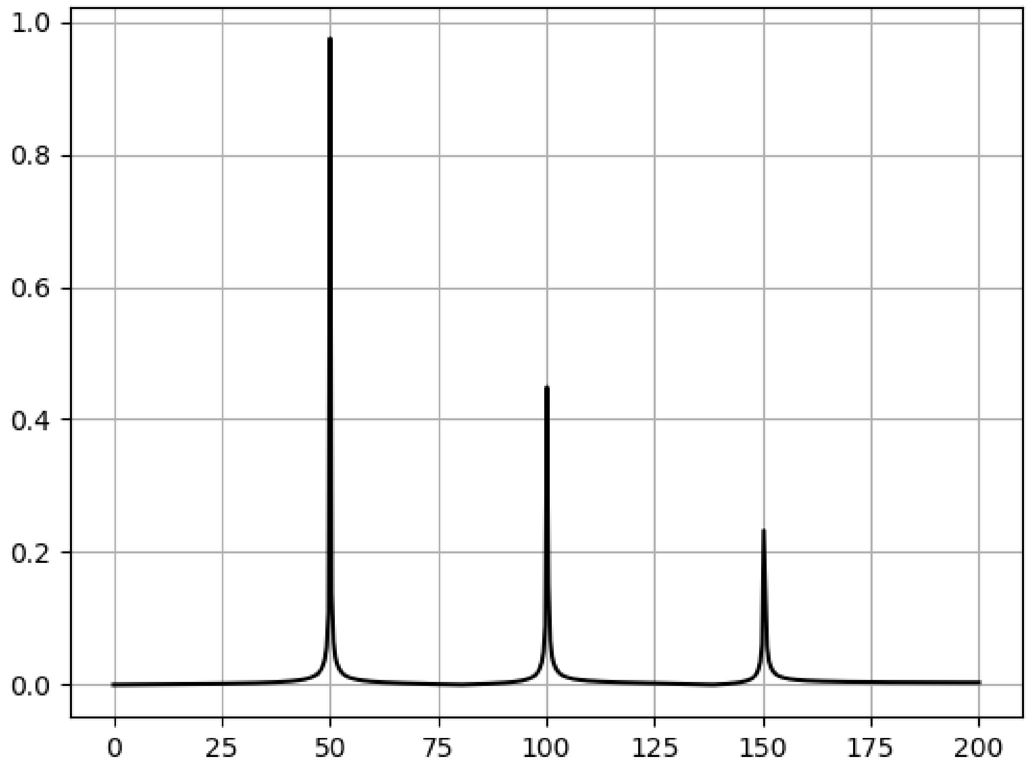
$$y[k] = \sum_{i=0}^{n-1} x[i].e^{-2\pi j \frac{k.i}{n}}$$

$$x[i] = \frac{1}{n}\sum_{k=0}^{n-1} y[k].e^{2\pi j \frac{k.i}{n}}$$

FFT

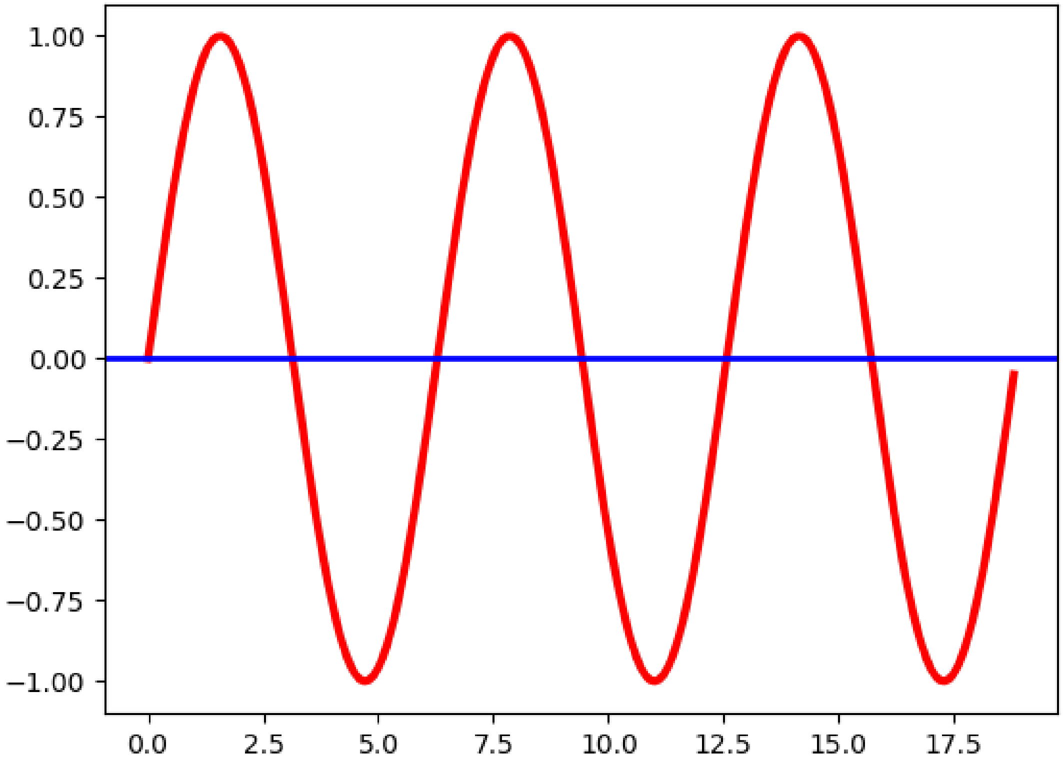
```
# FFT by fftpack module
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft

n = 1000                               #Number of sample points

t = 1.0/400.0                          #Sample spacing
x = np.linspace(0.0, n*t, n)

y = np.sin(2.0*np.pi*50*x) + 0.5*np.sin(2.0*np.pi*100*x)+
0.3*np.sin(2.0*np.pi*150*x)

yf = fft(y)
freq = np.linspace(0.0, 1.0/(2*t), n//2)

plt.plot(freq, 2.0/n * np.abs(yf[:n//2]), color = 'k')
plt.title('FFT', size ='20')
plt.grid()
plt.show()
```

FFT



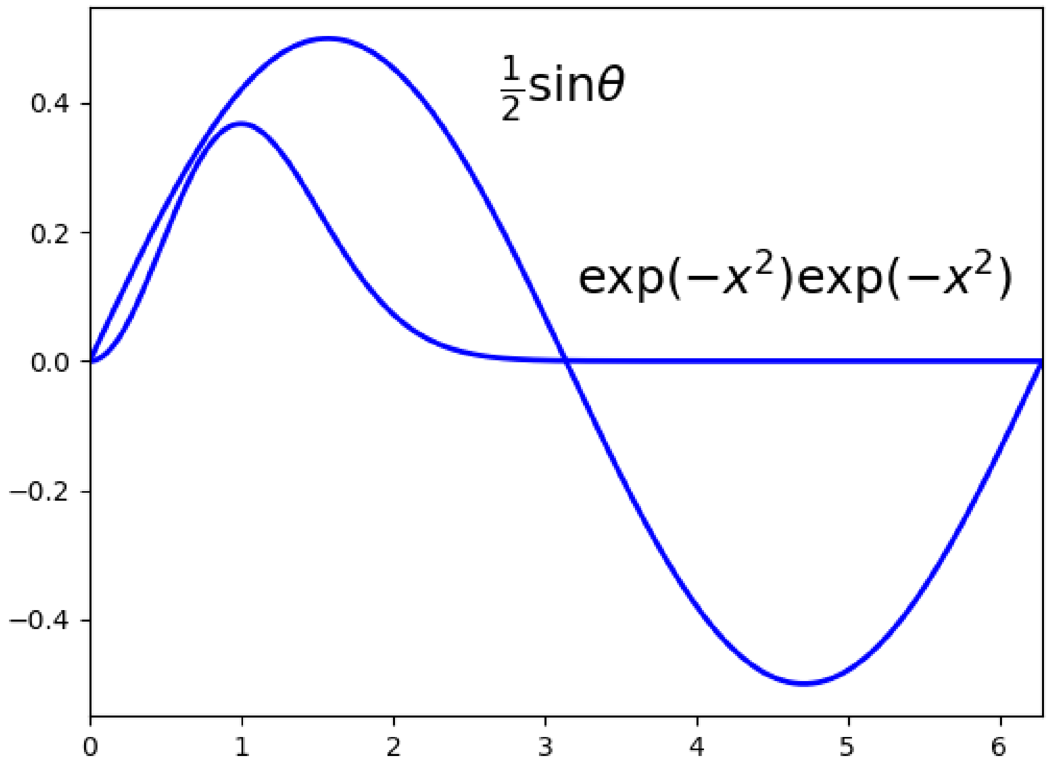
Matplotlib

```
# Plotting system function
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0,6*np.pi, 0.1)
plt.plot(x, np.sin(x), c ='r', lw = 3,)
plt.axhline(lw = 2, c = 'b')
plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Plotting multiple functions

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

x = np.arange(0,2*np.pi, 0.01)
f1 = 0.5*np.sin(x)
f2 = x**2*np.exp(-x**2)

plt.plot(x, f1, x, f2, c ='b', lw = 2)
plt.xlim(0,2*np.pi)
plt.text (2.7, 0.4, r'$\frac{1}{2}\sin\theta$', size
= 18)
plt.text (3.2, 0.1, r'$\exp(-x^2)\exp(-x^2)$', size
= 18)
plt.show()
```


$\frac{1}{2}\sin\theta$
$\exp(-x^2)\exp(-x^2)$
0.4
0.2
0.0
−0.2
−0.4
0
1
2
3
4
5
6

Generating two figures

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

x = np.arange(0, 2*np.pi, 0.01)

plt.figure(1)
plt.plot(x, 0.5*np.sin(x))

plt.figure(2)
plt.plot(x, np.exp(-x)*np.cos(2*np.pi*x))
plt.show()
```

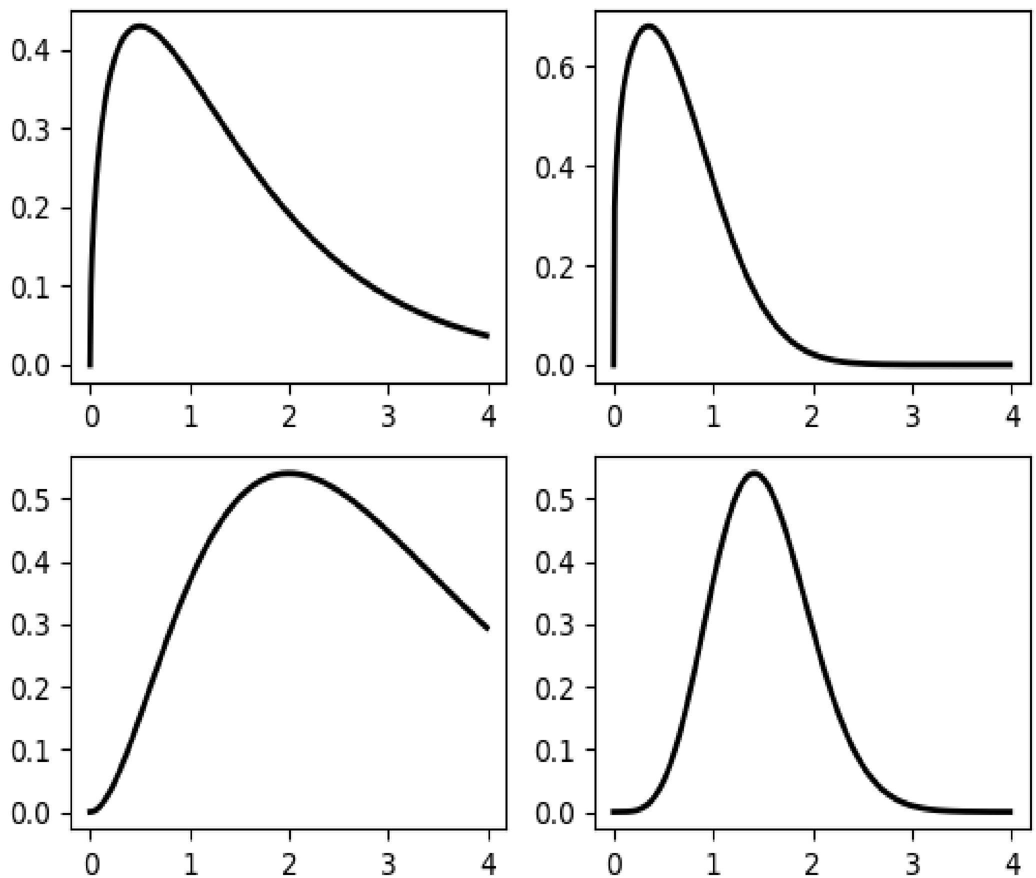
Subplot

subplot(nrows, ncolumns, plot_number)

subplot(2,2,1) or in short **subplot(221)**, this means we have divided the space into $\mathbf{2}\times\mathbf{2}$ plots and this graph is on **plot no. 1**.

1	2
3	4

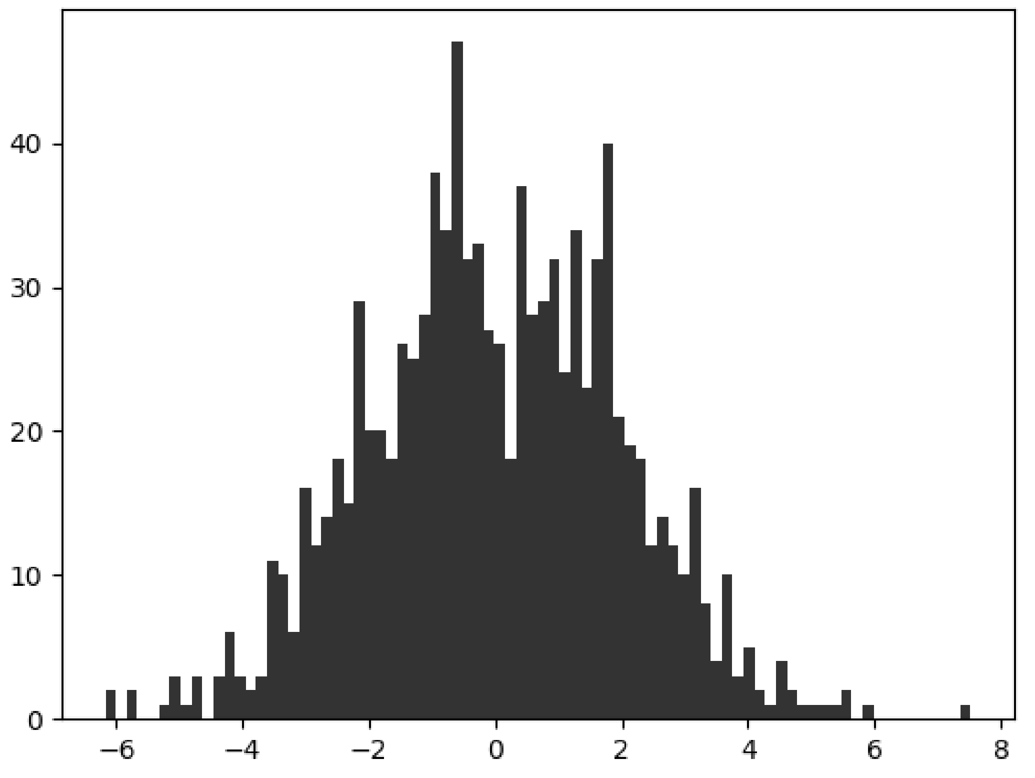
```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0, 4, 0.01)

plt.subplot(221)
plt.plot(x, x**0.5*np.exp(-x), 'k', lw = 2)

plt.subplot(222)
plt.plot(x, x**0.25*np.exp(-x**2), 'k', lw = 2)

plt.subplot(223)
plt.plot(x, x**2*np.exp(-x), 'k', lw = 2)

plt.subplot(224)
plt.plot(x, x**4*np.exp(-x**2), 'k', lw = 2)

plt.show()
```



0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
0.6
0.4
0.2
0.0
0
1
2
3
4
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4

Making Histogram

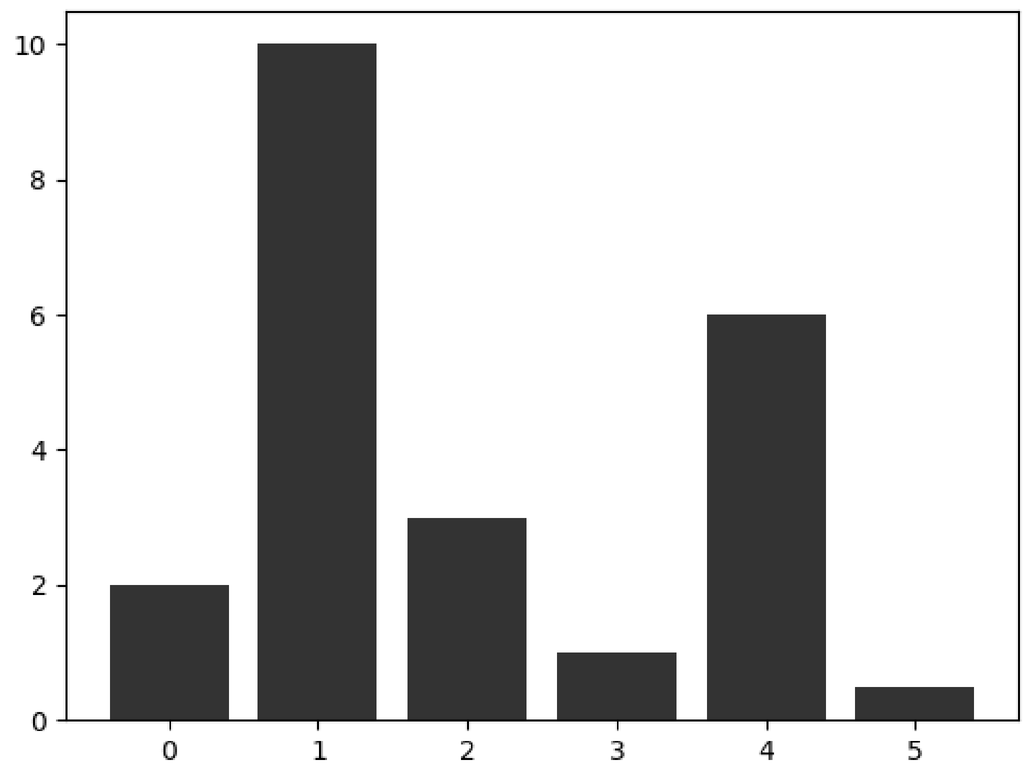
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(0, 2, 1000)

plt.hist(x, 100, color = ‘0.2’)
plt.show()
```



40
30
20
10
0
−6
−4
−2
0
2
4
6
8

Bar Diagram

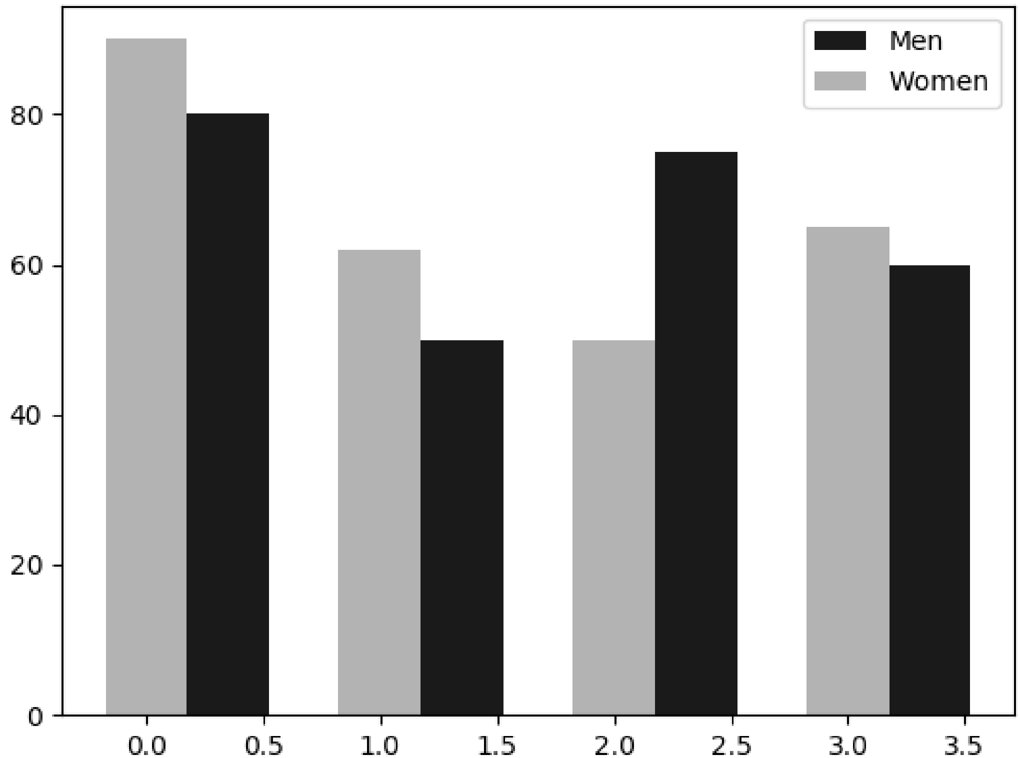
```
import matplotlib.pyplot as plt
import numpy as np

y = [2, 10, 3, 1, 6, 0.5]
x = np.arange(len(y))

plt.bar(x, y, color = '0.2')
plt.show()
```



10
8
6
4
2
0
0
1
2
3
4
5

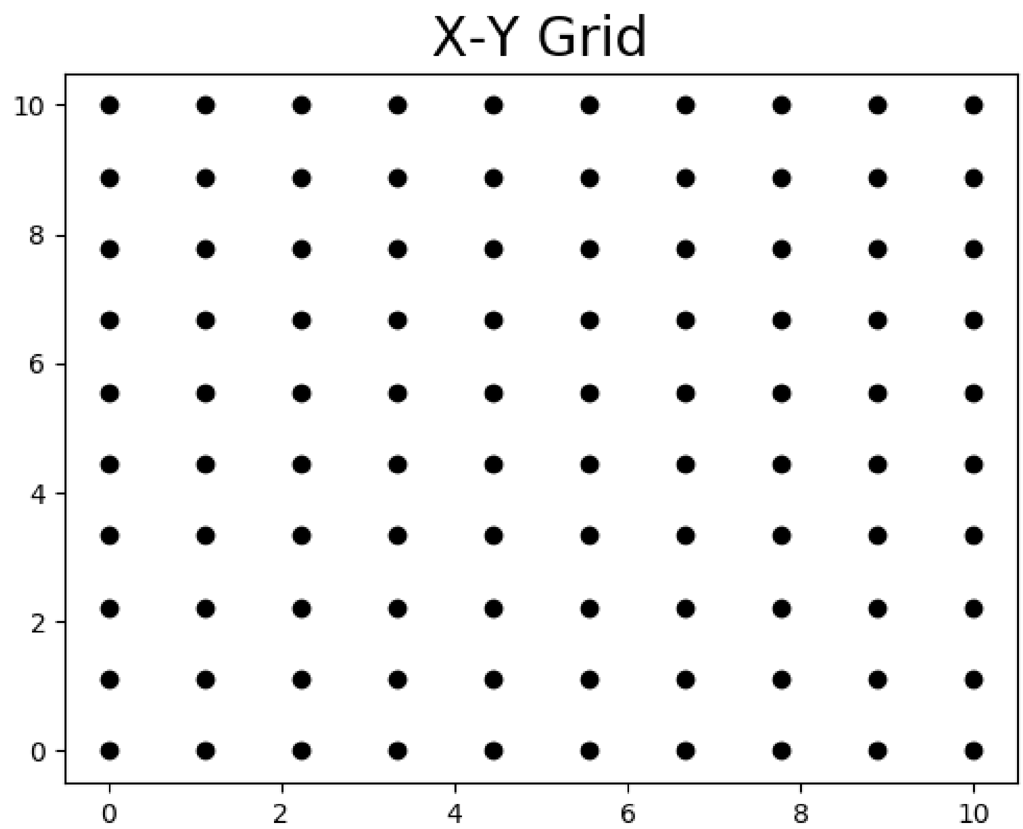
Bar Chart: Two sets of data

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

# Data to plot
men = (80, 50, 75, 60)
women = (90, 62, 50, 65)

# For x-axis
x = np.arange(len(men))
bar_width = 0.35

plt.bar(x + bar_width, men, bar_width, label = 'Men', c = '0.1')
plt.bar(x, women, bar_width, label = 'Women', color = '0.7')

plt.legend()
plt.show()
```

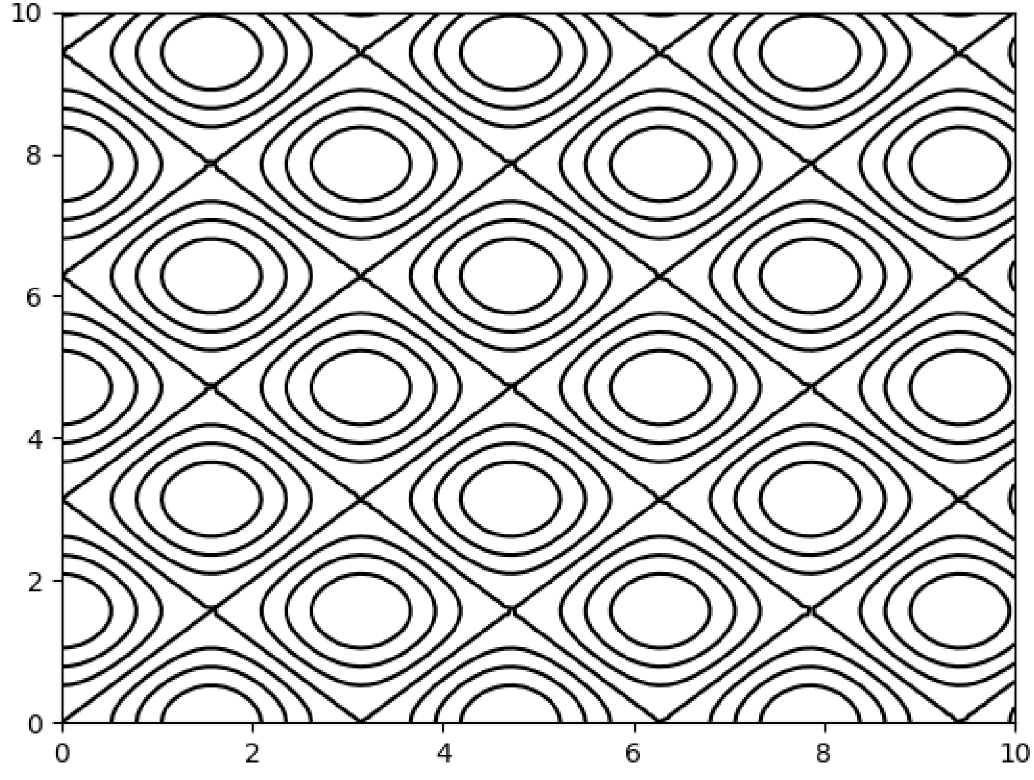

Men
Women
0
20
40
60
80
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

For 3D plot

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

x = np.linspace(0, 10, 10)
y = np.linspace(0, 10, 10)

X, Y = np.meshgrid(x, y)

plt.plot(X,Y, '.', ms = 12, c = 'k')
plt.title('X-Y Grid', size = 20)
plt.show()
```

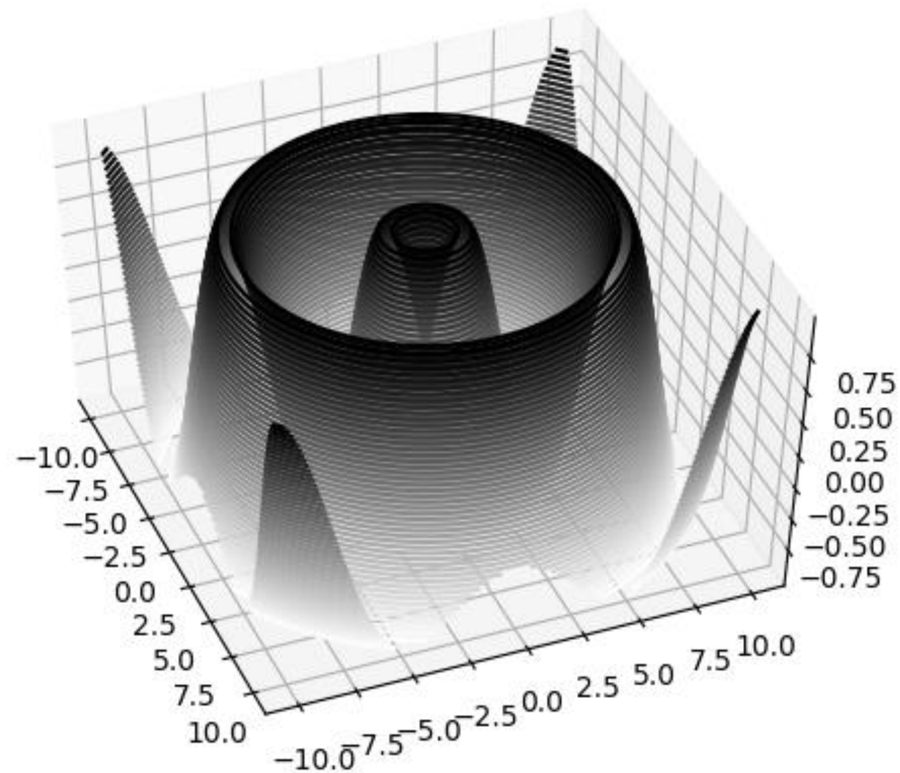

X-Y Grid
10
8
6
4
2
0
0
2
4
6
8
10

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

x = np.linspace(0, 10, 100)
y = np.linspace(0, 10, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(X)**2 + np.cos(Y)**2
plt.contour(X, Y, Z, colors = 'black')
```



10
8
6
4
2
0
0
2
4
6
8
10

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)

X, Y = np.meshgrid(x,y)
Z = np.sin(np.sqrt(X**2 + Y**2))

fig = plt.figure()
ax = plt.axes(projection = '3d')
ax.contour3D(X,Y,Z, 50, cmap = 'binary')
plt.show()
```



−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

Plotting a file…

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

f = np.loadtxt('test.dat')

plt.plot(f[:,0], f[:,1])
plt.show()
```

Arrays and Structures

- **Numpy**
- **Pandas**

ND Arrays by Numpy

```
>>> import numpy as np
>>> x = np.array([10,20,30])
>>> 10 in x
True
>>> 11 in x
False
```

Attributes: Type, Size, Dimension

```
>>> type(x)
<type 'numpy.ndarray'>
>>> x.size
3

>>> x.dtype
dtype('int32')

>>> x.ndim
1

>>> y = np.array([[1,2,3],[4,5,6]])
>>> y.ndim
2
```

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

>>> M
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
```

```
>>> x = np.array([0, -1, 2, 5, 10, 3, -2, 4])
>>> x.put(3,100)
>>> x
array([  0,  -1,   2, 100,  10,   3,  -2,   4])
```

Data Type

- **int8** (1 byte = 8-bit: Integer -128 to 127), **int16** (-32768 to 32767), **int32**, **int64**
- **uint8** (unsigned integer: 0 to 255), **uint16**, **uint32**, **uint64**
- **float16** (half precision float: sign bit, 5 bits exponent, 10 bits mantissa), **float32** (single precision: sign bit, 8 bits exponent, 10 bits mantissa), **float64** (double precision: sign bit, 11 bit exponent, 52 bits mantissa)
- **complex64** (complex number, represented by two 32-bit floats: real and imaginary components), **complex128** (complex number, represented by two 64-bit floats: real and imaginary components)

Acronyms:

i1 = int8, i2 = int16, i3 = int32, i4 = int64

f2 = float16, f4 = float32, f8 = float64

Default Data type

```
>>> x = np.array([10,23,36,467])
>>> x.dtype
dtype('int32')

>>> y = np.array([10.5,23,36,467])
>>> y.dtype
dtype('float64')

>>> a = np.array(['ab','bc', 'ca', 100])
>>> a
array(['ab', 'bc', 'ca', '100'], dtype='|S3')

*S3 = String of length 3.
```

```
>>> x = np.array([10,20,30], dtype = 'f')
>>> x
array([10., 20., 30.], dtype=float32)

>>> x = np.array([10.5,23,36,467], dtype = 'f4')
>>> x
array([ 10.5,  23. ,  36. , 467. ], dtype=float32)

>>> x = np.array([10.5,23,36,467], dtype = 'complex')
>>> x
array([ 10.5+0.j,  23. +0.j,  36. +0.j, 467. +0.j])
>>> x.dtype
dtype('complex128')

>>> x = np.array([10.5,23,36,467], dtype = 'complex64')
>>> x.dtype
dtype('complex64')
```

```
>>> A = np.array(['ab','bc', 'ca', 100], dtype = 'S10')
>>> A
array(['ab', 'bc', 'ca', '100'], dtype='|S10')

>>> A = np.array(['ab','bc', 'ca', 'abracadabra',
100], dtype = 'S6')
>>> A
array(['ab', 'bc', 'ca', 'abraca', '100'], dtype=
'|S6')

>>> A.itemsize              # Size of each item
6
```

```
>>> x = np.array([2+3j, 5+2j, 3-1j])
>>> x.real
array([2., 5., 3.])

>>> x.imag
array([ 3.,  2., -1.])

>>> x.conj()
array([2.-3.j, 5.-2.j, 3.+1.j])
```

Shape/Reshape

```
>>> M = np.array([[1,2,3],[4,5,6], [7,8,9], [10,11,12]])
>>> M
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
>>> np.shape(M)
(4, 3)
>>> M.shape
(4, 3)
>>> np.reshape(M, (3,4))
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
>>> M.reshape(3,4)
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Statistics

```
>>> x = np.array([[10,20,30], [40,50,60], [70,80,90]])

>>> x.sum()
450
>>> x.sum(0)
array([120, 150, 180])
>>> x.sum(1)
>>> np.sum(x,0)                                    # Row axis
array([120, 150, 180])
>>> np.sum(x,1)                                    # Column axis
array([ 60, 150, 240])

>>> np.sum(x)
450
>>> np.sum(x, 0)
array([120, 150, 180])
>>> np.sum(x, 1)
array([ 60, 150, 240])
```

```
>>> np.trace(x)
150

>>> np.mean(x)
50.0
>>> np.mean(x,1)
array([20., 50., 80.])
>>> np.mean(x,0)
array([40., 50., 60.])
>>> np.median(x)
50.0
>>> np.median(x,0)
array([40., 50., 60.])
>>> np.median(x,1)
array([20., 50., 80.])
```

```
>>> x = np.array([0, -1, 2, 5, 10, 3, -2, 4])
>>> np.ptp(x)                           # peak to peak
12
>>> np.var(x)                           # variance
12.984375

>>> np.std(x)                           # standard dev
3.6033838263498934
```

```
>>> x = np.array([0, -1, 2, 5, 10, 3, -
2, 4]).reshape(2,4)
>>> x
array([[ 0, -1,  2,  5],
       [10,  3, -2,  4]])
>>> np.ptp(x, 0)
array([10,  4,  4,  1])

>>> np.ptp(x, 1)
array([ 6, 12])
```

Sorting

```
>>> y = np.array([[1,2,2,3,2], [2,1,3,3,2], [1,1,0,3,2]])
>>> y
array([[1, 2, 2, 3, 2],
       [2, 1, 3, 3, 2],
       [1, 1, 0, 3, 2]])
>>> np.sort(y)
array([[1, 2, 2, 2, 3],
       [1, 2, 2, 3, 3],
       [0, 1, 1, 2, 3]])
>>> np.sort(y,0)
array([[1, 1, 0, 3, 2],
       [1, 1, 2, 3, 2],
       [2, 2, 3, 3, 2]])
>>> np.sort(y,1)
array([[1, 2, 2, 2, 3],
       [1, 2, 2, 3, 3],
       [0, 1, 1, 2, 3]])
```

Concatenation

```
>>> x = np.array([10,20,30])
>>> y = np.array([40,50,60])
>>> np.concatenate((x,y))
array([10, 20, 30, 40, 50, 60])
```

```
>>> x = np.array([[1,2,3],[4,5,6]])
>>> y = np.array([[1,2,3],[4,5,6]])
>>> np.concatenate((x,y))
array([[1, 2, 3],
      [4, 5, 6],
       [1, 2, 3],
       [4, 5, 6]])

>>> np.concatenate((x,y),axis=1)
array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

Append

```
>>> x = np.array([1,2,3,4])
>>> np.append(x,7)
array([1, 2, 3, 4, 7])

>>> x = np.array([10,20,30,40])
>>> y = np.array([100,200,300,400])
>>> np.append(x,y)
array([ 10,  20,  30,  40, 100, 200,
300, 400])
```

Iterator

```
>>> import numpy as np
>>> x = np.array([10,20,30,40])
>>> for i in x:
    print i
10
20
30
40
```

```
>>> a = np.arange(0,60,5)
>>> a = a.reshape(3,4)
>>> for i in a:
        print i

[ 0  5 10 15]
[20 25 30 35]
[40 45 50 55]
```

```
>>> for i in np.nditer(a, order = 'F'):
      print i

0
20
40
5
25
45
10
30
50
15
35
55
```

```
>>> for i in np.nditer(a, order = 'C'):
        print i

0
5
10
15
20
25
30
35
40
45
50
55
```

Methods for creation of arrays

```
np.arange(start, stop, step)
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([0., 1., 2.])
>>> np.arange(3,15,2, dtype ='float')
array([ 3.,  5.,  7.,  9., 11., 13.])

>>> np.arange(0.5, 1.0,0.1)
array([0.5, 0.6, 0.7, 0.8, 0.9])
```

```
np.linspace(start, end, num)

>>> np.linspace(10,20,5)
array([10. , 12.5, 15. , 17.5, 20. ])

>>> np.linspace(10,20,5, endpoint = True)
array([10. , 12.5, 15. , 17.5, 20. ])

>>> np.linspace(10,20,5, endpoint = False)
array([10., 12., 14., 16., 18.])

>>> np.linspace(10,20,5, retstep = True)
(array([10. , 12.5, 15. , 17.5, 20. ]),
2.5)
# returns step value
```

```
# Evenly spaced in logscale
>>> np.logspace(0,1,10)
array([ 1., 1.29154967, 1.66810054, 2.15443469,
2.7825594,3.59381366, 4.64158883, 5.9948425,
7.74263683, 10.])
# 10 vales, default base = 10

>>> x = np.logspace(0,1,10)
>>> np.log10(x)
array([0., 0.11111111, 0.22222222, 0.33333333,
0.44444444, 0.55555556, 0.66666667, 0.77777778,
0.88888889, 1.])

>>> np.logspace(0,1,10, base = 2)
array([1., 1.08005974, 1.16652904, 1.25992105,
1.36079,1.46973449, 1.58740105, 1.71448797,
1.85174942, 2.])
```

Special arrays

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

>>> np.full((3,3),5)
array([[5, 5, 5],
       [5, 5, 5],
       [5, 5, 5]])
```

Indexing, Slicing

```
>>> x = np.arange(2,15,3)
>>> x
array([ 2,  5,  8, 11, 14])
>>> x[0], x[3]
(2, 11)
>>> s = slice(1,5,2)
>>> s
slice(1, 5, 2)
>>> x[s]
array([ 5, 11])

>>> x[1:5:2]
array([1, 3])
```

```
>>> x[3:]
array([11, 14])
>>> x[:4]
array([ 2,  5,  8, 11])
>>> x[::2]
array([ 2,  8, 14])

>>> x[::-1]
array([14, 11,  8,  5,  2])
>>> x[::-3]
>>> x[::-3]
array([14,  5])
>>> x[2::]
array([ 8, 11, 14])
```

```
>>> x = np.arange(12).reshape(4,3)
>>> x
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
>>> x[2:]
array([[ 6,  7,  8],
       [ 9, 10, 11]])
>>> x[:3]
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> x[::2]
array([[0, 1, 2],
       [6, 7, 8]])
```

```
>>> x[x>3]
array([ 4,  5,  6,  7,  8,  9, 10,
11])
# returns the elements in 1D array

>>> x.flatten()
array([ 0,  1,  2,  3,  4,  5,  6,  7,
8,  9, 10, 11])
>>> x.flatten(0)
array([ 0,  1,  2,  3,  4,  5,  6,  7,
8,  9, 10, 11])
>>> x.flatten(1)
array([ 0,  3,  6,  9,  1,  4,  7, 10,
2,  5,  8, 11])
```

Arithmetic Operations

```
>>> x = np.array([1,2,3,4])
>>> y = np.array([5,6,7,8])
>>> x*y
array([ 5, 12, 21, 32])

>>> x+y
array([ 6,  8, 10, 12])

>>> x-y
array([-4, -4, -4, -4])

>>> x/y
array([0, 0, 0, 0])
>>> x**y
array([    1,    64,  2187, 65536])
```

```
>>> x = np.arange(1,7).reshape(2,3)
>>> x
array([[1, 2, 3],
       [4, 5, 6]])
>>> y = np.array([10,11,12])
>>> x+y
array([[11, 13, 15],
       [14, 16, 18]])
>>> x+2
array([[3, 4, 5],
       [6, 7, 8]])

>>> x + [2,3,4]
array([[ 3,  5,  7],
       [ 6,  8, 10]])
```

Array manipulation

```
>>> x
array([[1, 2, 3],
       [4, 5, 6]])
>>> x.T
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> x.transpose()
array([[1, 4],
       [2, 5],
       [3, 6]])
```

Stacking

```
>>> x
array([[1, 2],
       [3, 4]])
>>> y
array([[5, 6],
       [7, 8]])
>>> np.stack((x,y))                   # default axis = 0
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
>>> np.stack((x,y),axis = 1)
array([[[1, 2],
        [5, 6]],

       [[3, 4],
        [7, 8]]])
```

```
>>> np.vstack((x,y))     # stacks arrays vertically
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])

>>> np.hstack((x,y))    # stacks arrays horizontally
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])

>>> np.stack((x,y),axis = 1).ndim
3
>>> np.hstack((x,y)).ndim
2
>>> x
array([[1, 2],
       [3, 4]])
```

Inserting

```
>>> np.insert(x,1,[10])
array([ 1, 10,  2,  3,  4])
# Flattened, without axis paramter

>>> np.insert(x,1,[10], axis=0)
array([[ 1,  2],
       [10, 10],
       [ 3,  4]])

>>> np.insert(x,1,[10], axis=1)
array([[ 1, 10,  2],
       [ 3, 10,  4]])

>>> np.insert(x,1,[10,12],axis=0)
array([[ 1,  2],
       [10, 12],
       [ 3,  4]])
```

Deleting

```
>>> x = np.arange(1,13).reshape(3,4)
>>> x
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])

>>> np.delete(x,2,axis=0)
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])

>>> np.delete(x,2,axis=1)
array([[ 1,  2,  4],
       [ 5,  6,  8],
       [ 9, 10, 12]])
```

```
>>> x = np.array([1,2,1,0.5,10,2,10])
>>> np.unique(x)
array([ 0.5,  1. ,  2. , 10. ])

>>> x
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
>>> np.append(x,100)
array([  1,   2,   3,   4,   5,   6,
7,   8,   9,  10,  11,  12, 100])
# flattens and appends
```

Split

```
>>> np.split(x,3)
[array([[1, 2, 3, 4]]), array([[5, 6, 7, 8]]),
array([[ 9, 10, 11, 12]])]

>>> np.vsplit(x,3)
# same as above  (row wise split)

>>> np.hsplit(x,4)
[array([[1],
       [5],
       [9]]), array([[ 2],
       [ 6],
       [10]]), array([[ 3],
       [ 7],
       [11]]), array([[ 4],
       [ 8],
       [12]])]
```

Arrays as Vectors

```
# Inner Product
>>> u = np.array([1,2,3])
>>> v = np.array([-1,0,1])
>>> np.inner(u,v)
2
#Inner Product with a Scalar
>>> np.inner(u,2)
array([2, 4, 6])

>>> np.inner(np.eye(3),5))
array([[5., 0., 0.],
       [0., 5., 0.],
       [0., 0., 5.]])
```

```
# Multidimensional inner product
>>> A = np.array([[1,2,3],[4,5,6]])
>>> B = np.array([[1,0,1],[0,1,0]])

>>> np.inner(A,B)
array([[ 4,  2],
       [10,  5]])
```

Vector Cross Product

```
>>> x = np.array([1,2,3])
>>> y = np.array([-1,3,0])
>>> np.cross(x,y)
array([-9, -3,  5])
```

Volume of a Parallelepiped:

Three sides are given by three vectors: $\vec{A} = 2\hat{\imath} - 3\hat{\jmath}$, $\vec{B} = \hat{\imath} + \hat{\jmath} - \hat{k}$, $\vec{C} = 3\hat{\imath} - \hat{k}$

Volume $= \vec{A}.\left(\vec{B} \times \vec{C}\right) = 4$

```
>>> a = np.array([2, -3, 0])
>>> b = np.array([1, 1, -1])
>>> c = np.array([3, 0, -1])
>>> np.vdot(a, np.cross(b, c))
4
```

Matrix Operations

```
>>> A = np.array([[1,2,3],[4,5,6]])
>>> B = np.array([[1,2],[3,4],[5,6]])
>>> np.dot(A,B)
array([[22, 28],
       [49, 64]])

>>> A.dot(B)
array([[22, 28],
       [49, 64]])

>>> B.dot(A)
array([[ 9, 12, 15],
       [19, 26, 33],
       [29, 40, 51]])
```

$AB \neq BA$

```
>>> X = np.matrix(A)
>>> Y = np.matrix(B)
>>> X*Y
matrix([[22, 28],
        [49, 64]])
>>> Y*X
matrix([[ 9, 12, 15],
        [19, 26, 33],
        [29, 40, 51]])
```

Linear Algebra

$$
\begin{aligned}
x_1 \; + \; 2x_2 - \; x_3 &= 1 \\
2x_1 + \;\; x_2 + 4x_3 &= 2 \\
3x_1 + \; 3x_2 + 4x_3 &= 1
\end{aligned}
$$

$\mathrm{A} = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 1 & 4 \\ 3 & 3 & 4 \end{pmatrix}$, $\mathrm{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$, and $\mathrm{b} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$

```
>>> import numpy as np
>>> a = np.array([[1,2,-1],[2,1,4],[3,3,4]])
>>> b = np.array([1,2,1])
>>> print np.linalg.solve(a,b)
[ 7. -4. -2.]
```

Matrix Inverse

```
>>> import numpy as np
>>> A = np.array([[1,1,1],
[1,2,3], [1,4,9]])
>>> Ainv = np.linalg.inv(A)
>>> Ainv
array([[ 3. , -2.5,  0.5],
   [-3. ,  4. , -1. ],
   [ 1. , -1.5,  0.5]])
```

PANDAS

Pandas deal with the following three data structures:

- **Series**
- **DataFrame**
- **Panel**

These data structures are built over Numpy arrays.

Series

```
>>> import pandas as pd
>>> import numpy as np
>>> x = np.arange(10,50,10)
>>> pd.Series(x)
0     10
1     20
2     30
3     40
dtype: int32
```

```
>>> index = ['a', 'b', 'c', 'd']
>>> pd.Series(x, index)
a    10
b    20
c    30
d    40
dtype: int32

>>> s = pd.Series(x, index)
>>> s[0]
10
>>> s[‘a’]
10
```

Series: Methods

```
>>> s.axes
[RangeIndex(start=0, stop=4, step=1)]

>>> s.values
array([10, 20, 30, 40], dtype=int64)
>>> s.size
4
>>> s.shape
(4,)
>>> s.ndim
1
>>> s.dtype
dtype('int64')
```

```
>>> s['e'] = 50
>>> s
a     10
b     20
c     30
d     40
e     50
dtype: int64

>>> data =['a', 'b', 'c', 'd']
>>> pd.Series(data)
0     a
1     b
2     c
3     d
dtype: object
```

```
# Data as scalar

>>> index = ['a', 'b', 'c', 'd']
>>> pd.Series(10, index, int)
a     10
b     10
c     10
d     10
dtype: int32
```

Series from Dictionary

```
>>> data = {'a':10, 'b':20, 'c':30, 'd':40}
>>> pd.Series(data)
a    10
b    20
c    30
d    40
dtype: int64
>>> index = ['a', 'b', 'c', 'd', 'e', 'f']
>>> pd.Series(data, index)
a    10.0
b    20.0
c    30.0
d    40.0
e     NaN
f     NaN
dtype: float64
```

Arithmetic operations on Series

```
>>> s
a    10
b    20
c    30
d    40
e    50

>>> s*2
a     20
b     40
c     60
d     80
e    100

>>> np.sqrt(s)
a    3.162278
b    4.472136
c    5.477226
d    6.324555
e    7.071068
dtype: float64
```

```
>>> sum(s)
150L
>>> min(s)
10L
>>> max(s)
50L
>>> s[1:4]
b    20
c    30
d    40
dtype: int64
>>> s.sum()
100
>>> s.mean()
25.0
>>> s.std()
12.909944487358056
```

DataFrame

```
>>> x = [10,20,30,40]
>>> pd.DataFrame(x)
    0
0  10
1  20
2  30
3  40

>>> x = [[10,20,30,40], [50,60,70,80]]
>>> pd.DataFrame(x)
    0   1   2   3
0  10  20  30  40
1  50  60  70  80
```

```
>>> index = ['a','b']
>>> pd.DataFrame(x, index)
    0   1   2   3
a  10  20  30  40
b  50  60  70  80

>>> d = pd.DataFrame(x,index,columns =
['A', 'B', 'C', 'D'])
    A   B   C   D
a  10  20  30  40
b  50  60  70  80
```

```
>>> d['A']
a     10
b     50

>>> d['A']['a']
10
```

Methods over DataFrame

- `d.axes`
- `d.size`
- `d.ndim`
- `d.T`
- `d.empty`
- `d.values`
- `d.head(1)`
- `d.tail(1)`
- `d.sum()`
- `d.sum(1)`
- `d.mean()`
- `d.mean(1)[1]`
- `d.std()`
- `d.std(1)`
- `d.max()`
- `d.min()`
- `d.describe()          # Full Statistics`

DataFrame from the list of Dictionaries

```
>>> data = [{'x':2, 'y':10},{'x':4, 'y':20},{'x':6,
'y':30},{'x':8, 'y':40}]
>>> d = pd.DataFrame(data, index=['a','b','c','d'])
   x   y
a  2  10
b  4  20
c  6  30
d  8  40
>>> d['x']
a     2
b     4
c     6
d     8
Name: x, dtype: int64

>>> d['x']['b']
4
```

DataFrame from Dictionary of Series

```
>>> index = ['a','b','c','d']
>>> s1 = pd.Series([10,20,30,40],index)
>>> s2 = pd.Series([100,200,300,400],index)
>>> d = {'A':s1, 'B':s2}
>>> pd.DataFrame(d)
    A    B
a  10  100
b  20  200
c  30  300
d  40  400
>>> D = pd.DataFrame(d)
>>> D['A']
a     10
b     20
c     30
d     40
Name: A, dtype: int64
```

Add column to DataFrame

```
>>> D['C']= pd.DataFrame({'C':pd.Series([1000,2000,3000,4000],index)})
>>> D
    A    B     C
a  10  100  1000
b  20  200  2000
c  30  300  3000
d  40  400  4000

>>> D['C'] = pd.DataFrame(pd.Series([1000,2000,3000,4000],index))
>>> D
    A    B     C
a  10  100  1000
b  20  200  2000
c  30  300  3000
d  40  400  4000

>>> D['C'] = pd.Series([1000,2000,3000,4000],index)
>>> D
    A    B     C
a  10  100  1000
b  20  200  2000
c  30  300  3000
d  40  400  4000
```

Delete column and rows from DataFrame

```
>>> D
    A    B     C
a  10  100  1000
b  20  200  2000
c  30  300  3000
d  40  400  4000

>>> del D['A']
>>> D
     B     C
a  100  1000
b  200  2000
c  300  3000
d  400  4000
```

Slicing

```
>>> D.loc['b']
A      20
B     200
C    2000

>>> D.iloc[1]
A      20
B     200
C    2000
Name: b, dtype: int64

>>> D[1:3]
    A    B     C
b  20  200  2000
c  30  300  3000

>>> D[1:3]['A']
b    20
c    30
Name: A, dtype: int64
```

Append, Delete

```
>>> D1 = pd.DataFrame([[50,500,5000]], index =
['e'],columns=['A','B','C'])

>>> D1
     A     B      C
e   50   500   5000
>>> D.append(D1)                # Append another DataFrame
     A     B      C
a   10   100   1000
b   20   200   2000
c   30   300   3000
d   40   400   4000
e   50   500   5000

>>> D.drop('a')                  # Delete the indexed row.
     A     B      C
b   20   200   2000
c   30   300   3000
d   40   400   4000
```

Re-indexing

```
>>> index = np.arange(1,6)
>>> d = pd.DataFrame(data, index, columns = ['x', 'y'])
>>> d
     x    y
1  0.1  0.2
2  0.3  0.4
3  0.5  0.6
4  0.7  0.8
5  0.9  1.0
>>> d.reindex(np.arange(2,7), ['x','y'])
     x    y
2  0.3  0.4
3  0.5  0.6
4  0.7  0.8
5  0.9  1.0
6  NaN  NaN
```

Alignment of two DataFrames by reindexing

```
>>> data = np.random.rand(10,3)
>>> d1 = pd.DataFrame(data, index = range(1,11), columns =
['x','y','z'])
>>> d1
           x         y         z
1   0.342091  0.044060  0.773249
2   0.934012  0.038944  0.237909
3   0.670108  0.011794  0.831526
4   0.354686  0.381140  0.493882
5   0.690489  0.622695  0.409091
6   0.352255  0.205635  0.551726
7   0.371473  0.392713  0.853915
8   0.601222  0.353043  0.726287
9   0.933808  0.104148  0.718498
10  0.225576  0.812473  0.158370
```

```
>>> data = np.random.rand(8,3)
>>> d2 = pd.DataFrame(data, index = range(1,9),
columns = ['x','y','z'])
>>> d2
          x         y         z
1  0.322780  0.376841  0.957168
2  0.892635  0.248012  0.705469
3  0.006545  0.050196  0.112410
4  0.886808  0.437421  0.658757
5  0.628429  0.961192  0.190440
6  0.374883  0.450280  0.983127
7  0.257246  0.776551  0.425495
8  0.939035  0.471483  0.810289
```

```
>>> d2 = d1.reindex_like(d1)
>>> d2
           x         y         z
1   0.342091  0.044060  0.773249
2   0.934012  0.038944  0.237909
3   0.670108  0.011794  0.831526
4   0.354686  0.381140  0.493882
5   0.690489  0.622695  0.409091
6   0.352255  0.205635  0.551726
7   0.371473  0.392713  0.853915
8   0.601222  0.353043  0.726287
9   0.933808  0.104148  0.718498
10  0.225576  0.812473  0.158370
```

Panel

Panel is a 3D Container. DataFrame is a 2D container. Series is 1D.

```
>>> data = np.random.rand(2,3,4)
>>> np.random.rand(2,3,4)
array([[[0.05925325, 0.7165947 , 0.34978631, 0.68598632],
        [0.51410651, 0.50950708, 0.99801304, 0.34533087],
        [0.75854214, 0.50619351, 0.17673772, 0.4866736 ]],

       [[0.49319432, 0.03183697, 0.61576345, 0.73591557],
        [0.41456184, 0.20290885, 0.27732744, 0.63533898],
        [0.64958528, 0.42573291, 0.13674149, 0.10115889]]])
>>> p = pd.Panel(data)
>>> p
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 3 (major_axis) x 4 (minor_axis)
Items axis: 0 to 1
Major_axis axis: 0 to 2
Minor_axis axis: 0 to 3
```

```
>>> p.major_xs(0)
          0          1
0  0.483434  0.126538
1  0.061099  0.254202
2  0.754853  0.631093
3  0.298432  0.573099

>>> p.minor_xs(1)
          0          1
0  0.061099  0.254202
1  0.916231  0.034463
2  0.228343  0.853884
```

```
>>> index = ['a','b','c']
>>> data = {'A': pd.DataFrame(np.random.rand(3,4),index),
'B':pd.DataFrame(np.random.rand(3,4),index)}

>>> p = pd.Panel(data)
>>> p
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 3 (major_axis)
x 4 (minor_axis)
Items axis: A to B
Major_axis axis: a to c
Minor_axis axis: 0 to 3
```

```
>>> p.major_xs('a')
          A         B
0  0.422049  0.684155
1  0.922664  0.411938
2  0.644187  0.246746
3  0.213998  0.431654

>>> p.minor_xs(1)
          A         B
a  0.922664  0.411938
b  0.906779  0.573952
c  0.879191  0.233360
```

Methods on Panel

```
>>> p.values
array([[[0.42204928, 0.92266448, 0.64418741, 0.21399842],
        [0.42902311, 0.90677907, 0.67544671, 0.60858596],
        [0.35946858, 0.87919109, 0.16145494, 0.46737675]],

       [[0.68415499, 0.411938  , 0.24674607, 0.43165447],
        [0.15053089, 0.57395153, 0.65095238, 0.7393423 ],

>>> p.axes
[Index([u'A', u'B'], dtype='object'), Index([u'a', u'b',
u'c'], dtype='object'), RangeIndex(start=0, stop=4,
step=1)]
>>> p.size
24
>>> p.ndim
3
>>> p.shape
(2, 3, 4)
```

```
>>> p.sum(1)
          A          B
0  1.210541  1.153222
1  2.708635  1.219250
2  1.481089  1.471627
3  1.289961  1.396990

>>> p.sum(2)
          A          B
a  2.202900  1.774494
b  2.619835  2.114777
c  1.867491  1.351817
```

Thank you!