

An Introduction to Programming with Python

*Meifeng Lin
Computational Science Center*

*BNL Mini-Semester Workshop
January 6, 2016*



Agenda

- 09:00-09:45, Software Installation and Warm-up
- 09:45-10:30, Introduction to Python
- 10:30-10:45, Break
- 10:45-12:00, Hands-On Exercises

What is Programming?

- Here programming is short for “computer programming”, aka “coding”.
- Coding is writing a set of instructions for the computers.
 - For most of us, this means writing a human-readable program which is then transformed to machine codes that the computer can understand and execute.

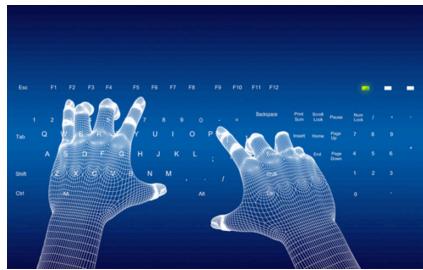


Image source: <http://aquinashub.co.uk/>



Image Source: <http://ipoint-tech.com/>

- But programming goes beyond simply coding, as it also involves analyzing the problem, coming up with a suitable algorithm and designing the implementation strategies etc.

What is a programming language?

- Just as we have different human languages, there are a lot of different computer programming languages, such as FORTRAN, C, C++, Python, Java, Perl, etc.
- A programming language defines a set of rules to write a computer program.
- Just like a human language, a programming language has a certain “vocabulary” and “grammar”.

Why do we need to know about programming?

- Programming is embedded in every aspect of modern life.
- Computers.
- Playstation, XBOX, ...
- Smart phones/TVs/watches/refrigerators/...
 - How are they smart? Computer programs!
 - Smart houses? (Eureka, anyone?)

In the Context of Science and Engineering...

- Numerical simulations and big data analytics play a significant role in the advancement of science and engineering.
- Programming is essential...
 - Aircraft simulation
 - Genome sequencing
 - Drug discovery
 - Supernova simulation
 - The origin of mass and the evolution of the universe!
 - And much more...
- Want to take a YouTube break?
Empty Space is Not Empty

Introduction to Python

What is Python?



- Python is a popular computer programming language with many applications.
- It is a high-level language that is meant to be very readable, flexible and portable across different platforms.
- It is an interpreted language. The program statements are checked statement by statement.
- Needs the python interpreter.
- No need to compile and link the program, unlike C/C++.

Lesson Plan

- Python Basics with Demonstration [45 minutes]
 - We'll use Canopy for demonstration in class.
 - More experienced users feel free to use your own installation of python.
- Hands-On Exercises [75 minutes]
 - Generate random numbers
 - Calculate average and standard deviation
 - Monte Carlo methods

My Goals for this course

- Introduce you to the world of programming
- Give you a taste of flavors of Python
- Hopefully can motivate you to learn more about programming and python.

Starting up Canopy

- If you have Canopy installed on your own laptop, double click the launcher.

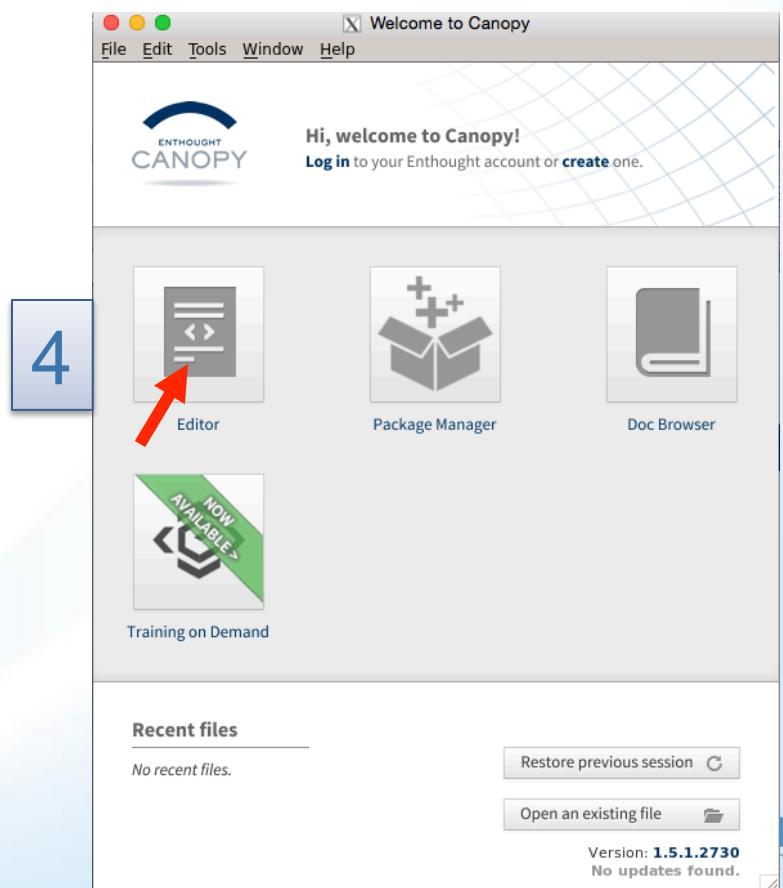
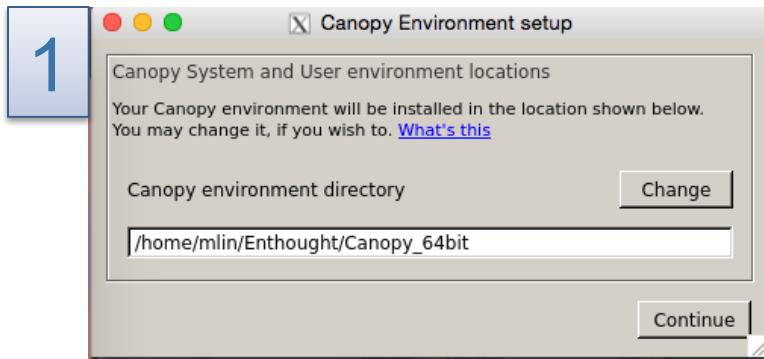
Something like this

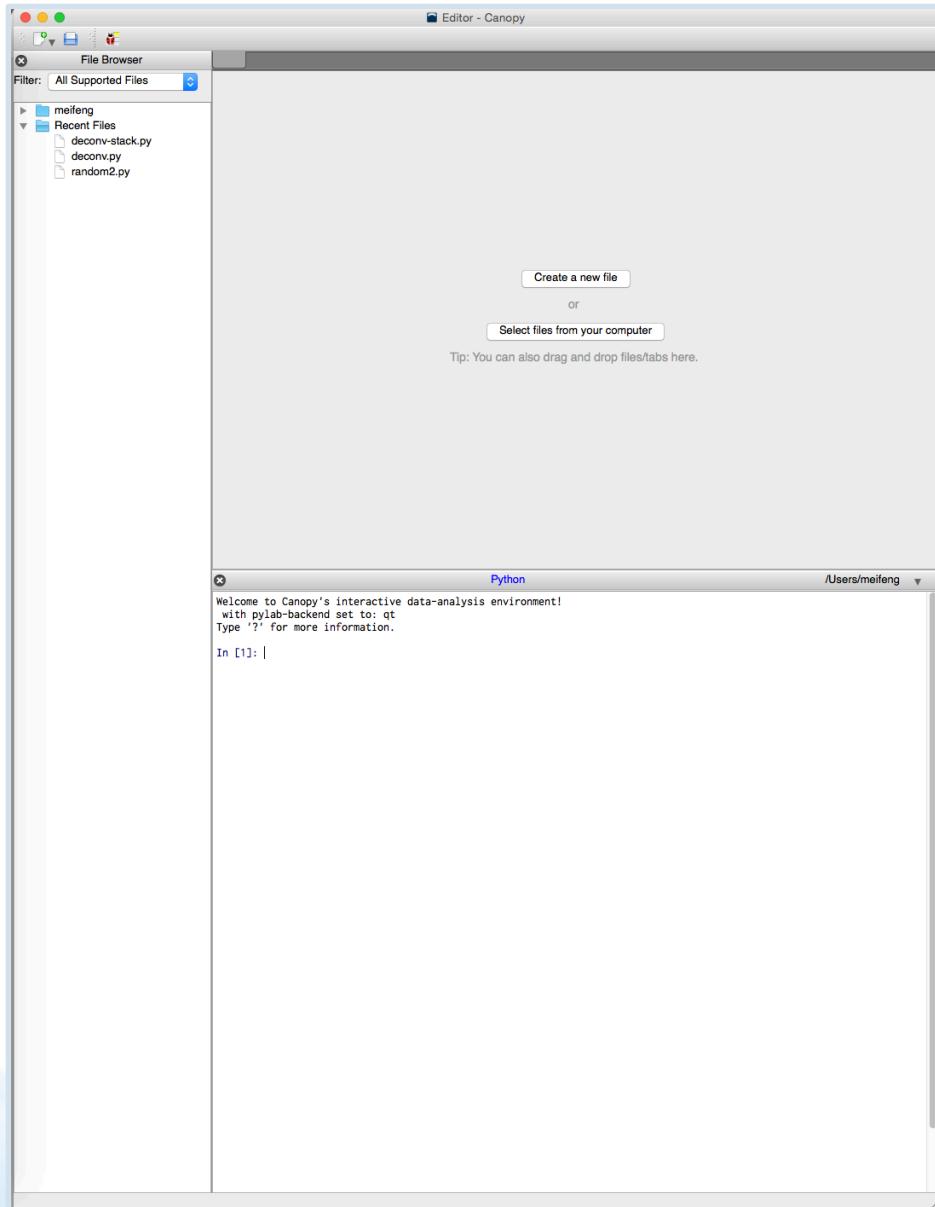


Canopy

You will be asked to do some setup the first time you run Canopy. For most of you, choosing the default should suffice.

Setting up Canopy





text editor

the IPython shell

An enhanced interactive
python shell.

Standard Python Shell vs. IPython

- The standard python shell
 - Barebones interpreter
 - This is what you get with the standard Python distribution.
- IPython -- An advanced python shell
 - Tab completion of variable names
 - Supports some Unix/Windows commands
 - Inline documentation
 - Syntax highlighting
 - Good for beginners

Two Ways to Use Python

```
Python /Users/meifeng
Welcome to Canopy's interactive data-analysis environment!
with pylab-backend set to: qt
Type '?' for more information.

In [1]: print 'hello world!'
hello world!

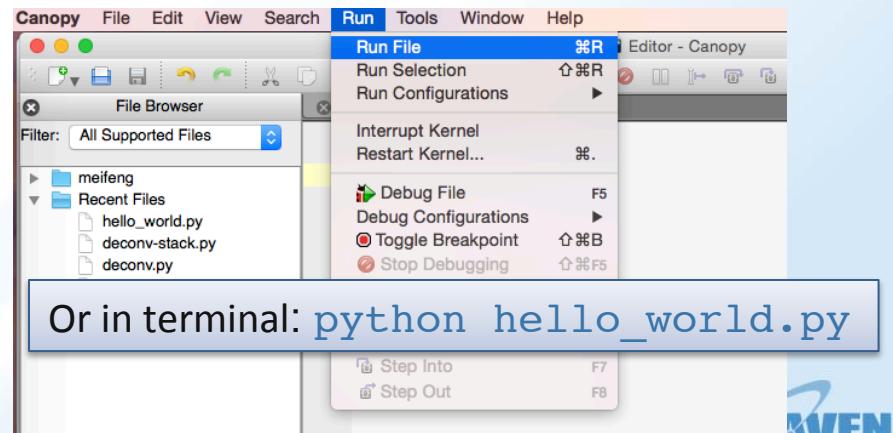
In [2]: 
```

Interactive mode

Good for prototyping code segments or as a calculator.

```
hello_world.py
1 print 'hello world!'
```

Script mode



Or in terminal: `python hello_world.py`

Good to know

- There are two major releases of Python: Python 2.x and Python 3.x
- Python 3.x is not backward compatible. So make sure you know which version you are using.
 - If you are using a terminal, you can check by running:
 - `python --version`
 - In the interactive mode, you can check with
 - `print sys.version`
 - Mine returns:
 - `Python 2.7.6 -- 64-bit`
- We will be using Python 2.x.

Good to know

- Code blocks are organized by indentation
- Single-line comments start with `#`
- Multiline comments start and end with matching `'''`
(three single quotation marks) or `"""` (three double quotation marks)
- Comments are ignored by the interpreter, except when the first line begins with `#!`
 - `#!/usr/bin/python`
 - It tells the os that the program is to be processed by /usr/bin/python.
- Demo: `hello_world.py`

Python Basic Building Blocks

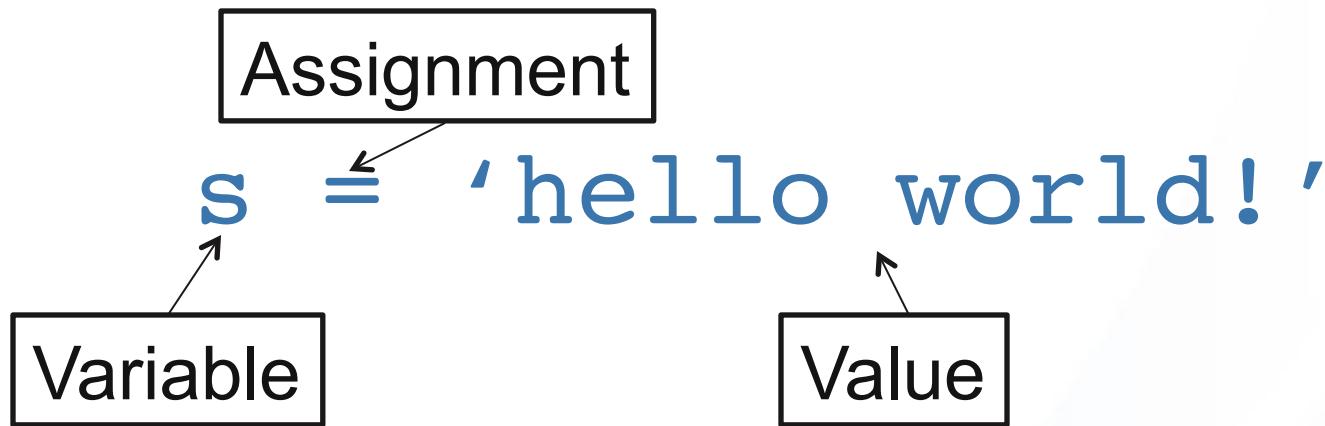
1. Data Types: `int, float, str, long, bool, ...`
 2. Variables: `variable_name = value`
 3. Collection Data Types: `tuples, lists, sets, dictionaries`
 4. Logical Operations: `==, >, <, and, or, not, in`
 5. Arithmetic Operators: `+, -, /, *, **, %, ...`
 6. Input / Output: `input/raw_input, print, open, close`
 7. Control Flow Statements: `if/elif/else, while, for`
 8. Creating and Using Functions: `def, import`
-
- Other programming languages have more or less the same components, with different syntaxes.

Data Types

- Built-in data types:
 - `int` positive or negative whole numbers, e.g. 5, 100, -2
 - `float` decimal numbers, 1.2, 3.14159, -10.0
 - `str` 'hello', "This is a string"
 - `bool` Only two values of this type: `True` or `False`
- Useful Utilities
 - `type(some_value or variable)`
 - `sys.float_info`
 - `sys.maxint`
- Type conversion:
 - `float(2) -> 2.0, int(1.1) -> 1`
 - **Be careful:** `2/3 -> 0`. Make sure one of them is a float.

Note: You can also have user-defined data types: classes, which we are not going to cover.

Variables and Assignment



- Variables have to be in one word. Can't start with a number, but otherwise can be any combinations of numbers, letters and some special characters: _ and -.
 - `my_variable, var1, var2, ...`
- Can't be any of the Python reserved words, such as `if, else, True, False, class`, etc.
- Assignments can also be
 - `a, b = 1.0, 2.2 -> a = 1.0, b = 2.2`

Collection Data Types: Represent a group of values

- List
 - `name = ['Alice', 'Bob', 'Charlie', 'David']`,
 - `age = [20, 22, 23, 21]`
 - List is mutable (append, remove, insert, ...)
 - indexing, slicing
 - `len(a), range([start,]stop[,step])`
- Dictionaries: associative key:value pairs
 - `age={'Alice':20, 'Bob':22, 'Charlie':23, 'David':21}`
 - Indexing: `age['Alice']`
 - Insertion: `age['Edward'] = 19`
 - In random order by default. Try: `print age`
- Tuples
 - `Results = (1.1, 2.0)`
 - Not mutable. Can have mixed data types, nested tuples.
- Sets: not very commonly used

Logical Operation

- Returns **True** or **False**
- Demo
 - ==
 - <, >
 - and
 - or
 - not

Arithmetic Operators

- Demo: +, -, *, /, **, %
 - **: power
 - %: modulo
- Order of executions
 - When in doubt, use parentheses

Input and Output (I/O)

- Standard I/O (Prints to screen, or takes input from keyboard)
 - `print`
 - `input` – automatically detects the type based on the input
 - `raw_input` – everything is considered a string.
 - Hint: check using `type(arg)`
- File I/O (demo in examples)
 - `open` (returns a file handle)
 - `Infile = open('mydata.dat','r')`
 - `Outfile = open('output.dat','w')`
 - `read, readlines, readline, for line in f:`
 - `write`
 - `close`

Control Flow Statements

- Conditional: `if/elif/else`

- Only execute if the statement is True
- Can have nested conditionals

- Looping: `for`, `while`

```
print "for loop:"  
for n in range(10):  
    if n%2 == 0:  
        print n, "is an even number"  
    else:  
        print n, "is an odd number"  
  
print "while loop:"  
x = 11  
while x < 20:  
    if x%2 == 0:  
        print x, "is an even number"  
    else:  
        print x, "is an odd number"  
    x=x+1
```

```
print "Type a number:"  
n = input()  
  
if type(n) == int:  
    if n%2 == 0:  
        print n, "is an even number"  
    else:  
        print n, "is an odd number"  
else:  
    print n, "is not an integer"
```

Functions

- Functions have the form `func([argument list])`
 - Examples: `sqrt(2)`, `sin(pi)`, `range(1,100,2)`, etc.
- In addition to the built-in functions, we can also define our own
 - `def my_func(arg1, arg2):
 do something
 return results`
- For example:

```
def square(x):  
    return x*x  
print square(2)
```

Classes, Modules and Libraries

- The true power of Python lies in the vast collection of open-source modules and libraries that are freely available.
- Python comes with limited built-in functions. To use the external modules and functions, we have to let python know beforehand.
- Ways to access external functions
 - `import module_name`
 - For C programmers, this is much like `#include<header.h>`
 - `from module_name import func`
 - Example:

```
import math
math.sqrt(2)
```

```
from math import sqrt
sqrt(2)
```

An Example – Random Numbers

- Task 1
 - Generate N normally/Gaussian distributed random numbers with an average of μ (mu) and standard deviation σ (sigma)
 - Write them to disk
- Task 2
 - Read them back in
 - Compute the average and standard deviation
 - Display the results on terminal
- Task 3 (Time permitting)
 - Compute and display the histogram of the numbers
 - Do a curve fit

Normal/Gaussian Distribution

From Wikipedia:

A normal distribution is:

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The parameter μ in this definition is the mean or expectation of the distribution (and also its median and mode). The parameter σ is its standard deviation; its variance is therefore σ^2 . A random variable with a Gaussian distribution is said to be normally distributed and is called a normal deviate.

If $\mu=0$ and $\sigma=1$, the distribution is called the **standard normal distribution** or the unit normal distribution

Normal/Gaussian Distribution

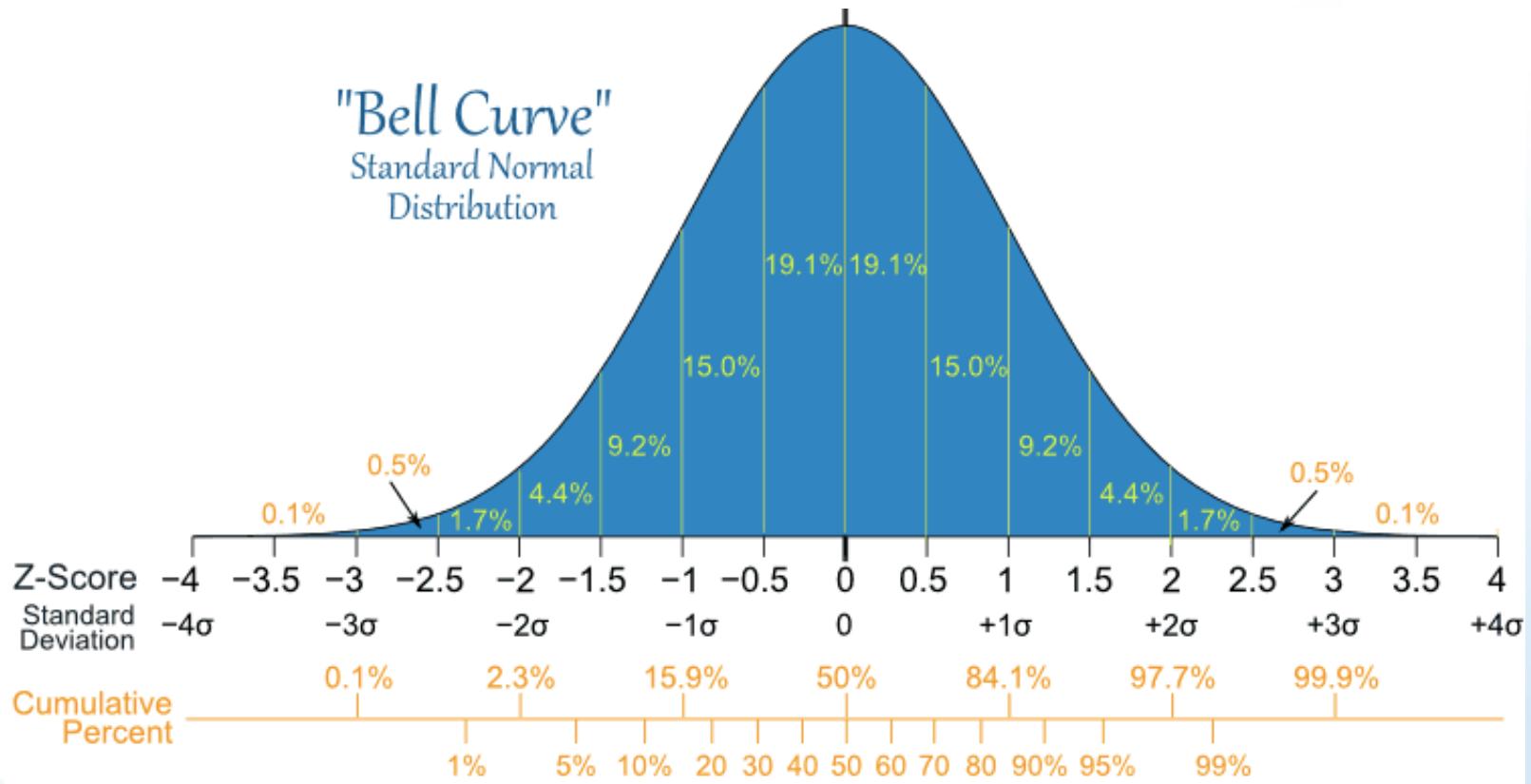


Image source: <http://www.mathsisfun.com>

Task1: Generate and save the random numbers

- Code demo: generate.py
- Features demonstrated:
 - Import external functions
 - Assignments
 - List operation
 - For loop
 - File output

```
import os
import random

# change the current directory to my workshop directory
workshop_dir = '/Users/meifeng/Dropbox/BNL/CSC/Mini-Semester-Jan2015/examples/'
os.chdir(workshop_dir)

# set the number of random numbers we want
N = 10000

# mean
mu = 0.0

# standard deviation
sigma = 1.0

# create an empty list
data = []

# append each random number to the list
for i in range(N):
    data.append(random.gauss(0,1.0))

# open the output file to write
outfile = open('ran_data.dat', 'w')

# write the data to disk
for i in range(N):
    outfile.write("%d %e\n" %(i, data[i]))

outfile.close()
```

Statistical Analysis of Data

- We can calculate the mean and standard deviation of the random numbers we just generated, and compare them to the input values: $\mu = 0$, $\sigma = 1.0$
- Mean:

$$\mu = \frac{1}{N} \sum_{i=0}^{N-1} x_i = \frac{1}{N} (x_0 + x_1 + \dots + x_{N-1})$$

- Standard deviation:

$$\sigma = \sqrt{\frac{\sum_{i=0}^{N-1} (x_i - \mu)^2}{N}}$$

Task2: Calculate the mean and standard deviation

- Code demo: analyze.py
- Features demonstrated:
 - File input
 - User-defined functions
 - Type conversion

```
import os
from math import sqrt

# create a function to calculate the mean and standard deviation
# of an input list
def stat(data):
    tot = 0.0
    N = len(data)
    for i in range(N):
        tot += data[i]
    avg = tot/N

    for i in range(N):
        tot += (data[i] - avg)**2
    var = tot/N
    return avg, sqrt(var)

# change the current directory to my workshop directory
workshop_dir = '/Users/meifeng/Dropbox/BNL/CSC/Mini-Semester-Jan2015/examples/'
os.chdir(workshop_dir)

data = []

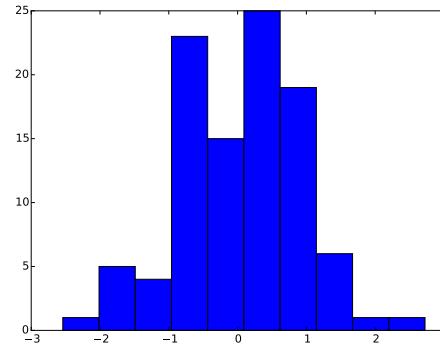
infile = open('ran_data.dat','r')
for line in infile: # line is a string
    _, val = line.split() # _ is a dummy variable
    data.append(float(val)) # needs to convert string to float

infile.close()

print 'average, standard deviation = ', stat(data)
```

Histogram

- A histogram is a graphical representation of the distribution of data.



- The histogram of the random numbers we generated should be an approximation of the normal distribution.
- The more data we generate, the closer our histogram is to the normal distribution.

Task3: Fit the data and plot the histogram

- Code demo: fit.py
- Features demonstrated:
 - Use of external libraries: numpy, scipy and matplotlib
 - Plotting
 - Curve fitting with scipy

```

import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from scipy.stats import norm

# change the current directory to my workshop directory
workshop_dir = '/Users/meifeng/Dropbox/BNL/CSC/Mini-Semester-Jan2015/examples/'
os.chdir(workshop_dir)

# input file
infile = 'ran_data.dat'

# read and unpack the data
x, y = np.loadtxt(infile, unpack=True)
nbins = 100

# fit the data to the normal distribution
# and obtain the mean and standard deviation
(mu, sigma) = norm.fit(y)

print "From Gaussian fit, mu = ", mu, "sigma = ", sigma

fig = plt.figure()

# position of the subplots. The numbers read as M x N plots, ith plot
timeseries = fig.add_subplot(211)
histogram = fig.add_subplot(223)
histogram_norm = fig.add_subplot(224)

# make the histogram plots!
histogram.hist(y, nbins, normed=False, align='mid',
                facecolor='blue', alpha=0.9)
histogram.set_xlabel('Random Number Value')
histogram.set_ylabel('Counts')
n, bins, patches = histogram_norm.hist(y, nbins, normed=True,
                                         align='mid', facecolor='green', alpha=0.9)

histogram_norm.set_xlabel('Random Number Value')
histogram_norm.set_ylabel('Fraction')

# plot the Gaussian fit
gauss = mlab.normpdf(bins, mu, sigma)
histogram_norm.plot(bins, gauss, 'r-', linewidth=2)

# time series plot
timeseries.plot(x,y,'r-')
timeseries.set_xlabel('Index')
timeseries.set_ylabel('Random Number Value')
fig.subplots_adjust(hspace=0.5,wspace=0.5)

# display the plot
fig.show()

# save a hard copy
fig.savefig('histogram.pdf')

```

Hands-On Exercises

Exercise #1

- Print numbers 1 to 100 to stdout
- Elements needed:
 - `print`
 - `range` function
 - `for` loop

Exercise #2

- Print 100 random numbers in the format of:
 - <index> <random number>
- New Elements needed:
 - `from random import random`
 - `random()` will generate one flat-distribution random number between [0,1]

Statistical Analysis of Data

- Mean:

$$\mu = \frac{1}{N} \sum_{i=0}^{N-1} x_i = \frac{1}{N} (x_0 + x_1 + \dots + x_{N-1})$$

- Standard deviation:

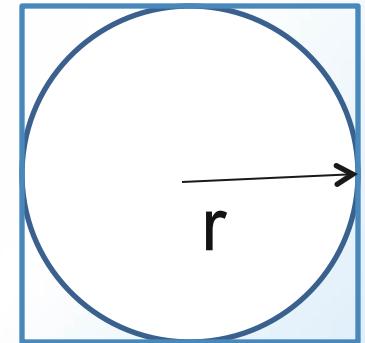
$$\sigma = \sqrt{\frac{\sum_{i=0}^{N-1} (x_i - \mu)^2}{N}}$$

Exercise #3

- Write a function to calculate the mean and standard deviation of a list
- Calculate the mean and standard deviation of the generated random numbers
- New elements needed:
 - Store the random numbers in a list
 - Self defined function: `def func()`
 - The `sqrt` function from the `math` module

Exercise #4

- Monte Carlo Methods: numerical methods that use random numbers to do repeated sampling and obtain results.
 - Estimate the value of π using the Monte Carlo method.
-
- Area of the square: $4r^2$
 - Area of the circle: πr^2
 - If we throw darts randomly at the square, the probability of them landing inside the circle is $\pi/4$
 - Monte Carlo estimation of π :
 - Generate N points with coordinates (x, y) , where x and y are random numbers between 0 and 1
 - If $\sqrt{x^2 + y^2} < 1$, $N_{in} = N_{in} + 1$
 - $N_{in} / N = \pi / 4 \rightarrow \pi = 4 N_{in} / N$



Closing Remarks

- Python is powerful and fun.
- It's a good first language to learn if you don't have any programming experience.

- You can find many useful resources on the internet.
 - Official Python website: www.python.org
 - Online video courses:
 - www.udacity.com
 - www.coursera.org
 - www.edx.org
 - Codecademy interactive coding tutorials:
 - <http://www.codecademy.com/tracks/python>

```
print "Have fun coding!"
```