Introduction to Python for Scientists and Engineers

IEM Petaling Jaya - June 4, 2014

P.C. Boey
Principal Consultant, Pytech Resources

pcboey@pytech.com.my

A C program is like a fast dance on a newly waxed dance floor by people carrying razors.

- Waldi Raven

C++: Hard to learn and built to stay that way.

- Anonymous

Java is, in many ways, C++—.

- Michael Feldman

And now for something completely different …

- Monty Python's Flying Circus

Goals of this presentation

- Introduce the core Python programming language and standard libraries
- Introduce the scientific libraries
  - NumPy
  - SciPy
  - Matplotlib
  - Pandas

Goals of this presentation (2)

- Introduce the IPython enhanced interactive shell and the Ipython Notebook
- IPython + Text Editor → interactive computing and development envoironment
- Brief demos

What is Python?

- object-oriented, high-level interpreted language
- developed by Guido van Rossum in early '90s
- readable, simple and elegant
- provides high-level built-in data structures
- open source, freely distributable
- cross-platform

Together with NumPy, SciPy, Matplotlib and Pandas,

Python provides a great environment for scientific computing

Different Ways to Get Python

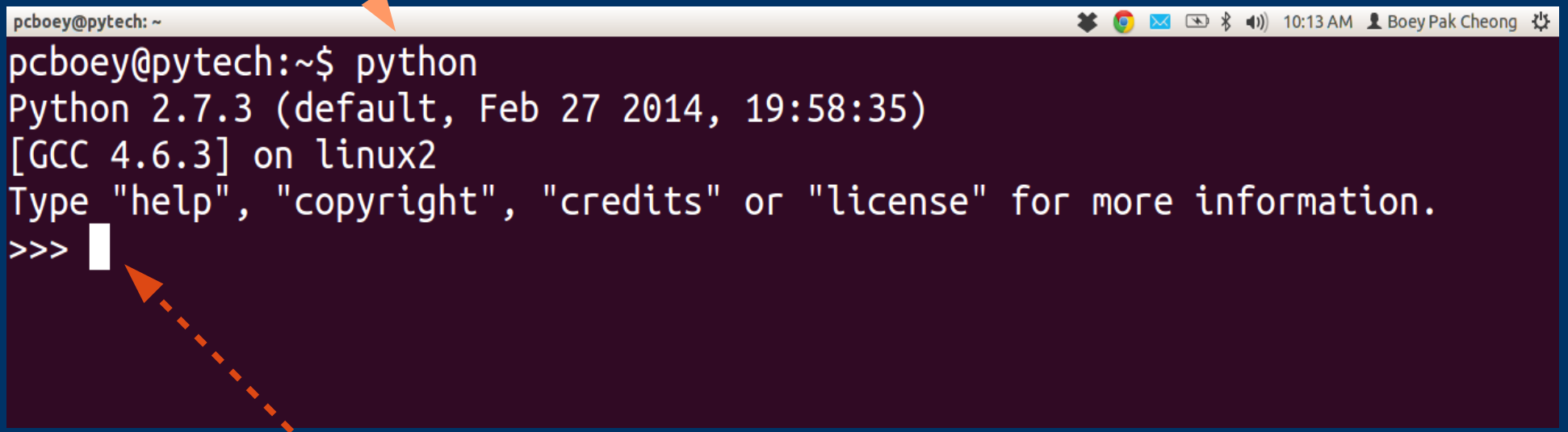
- Official Website - http://www.python.org
- Python for scientific environments
  - Enthought Python Distribution
    - https://www.enthought.com/products/epd/
  - Python(x,y)
    - https://code.google.com/p/pythonxy

How to Run Python Programs

- Ways to start python
  1) Interactive interpreter from command-line
  2) Python Integrated Development Environment (IDLE)
  3) Run script from command-line
  4) Integrated development environment (IDE)

Python Interactive Shell

Type python in your OS shell

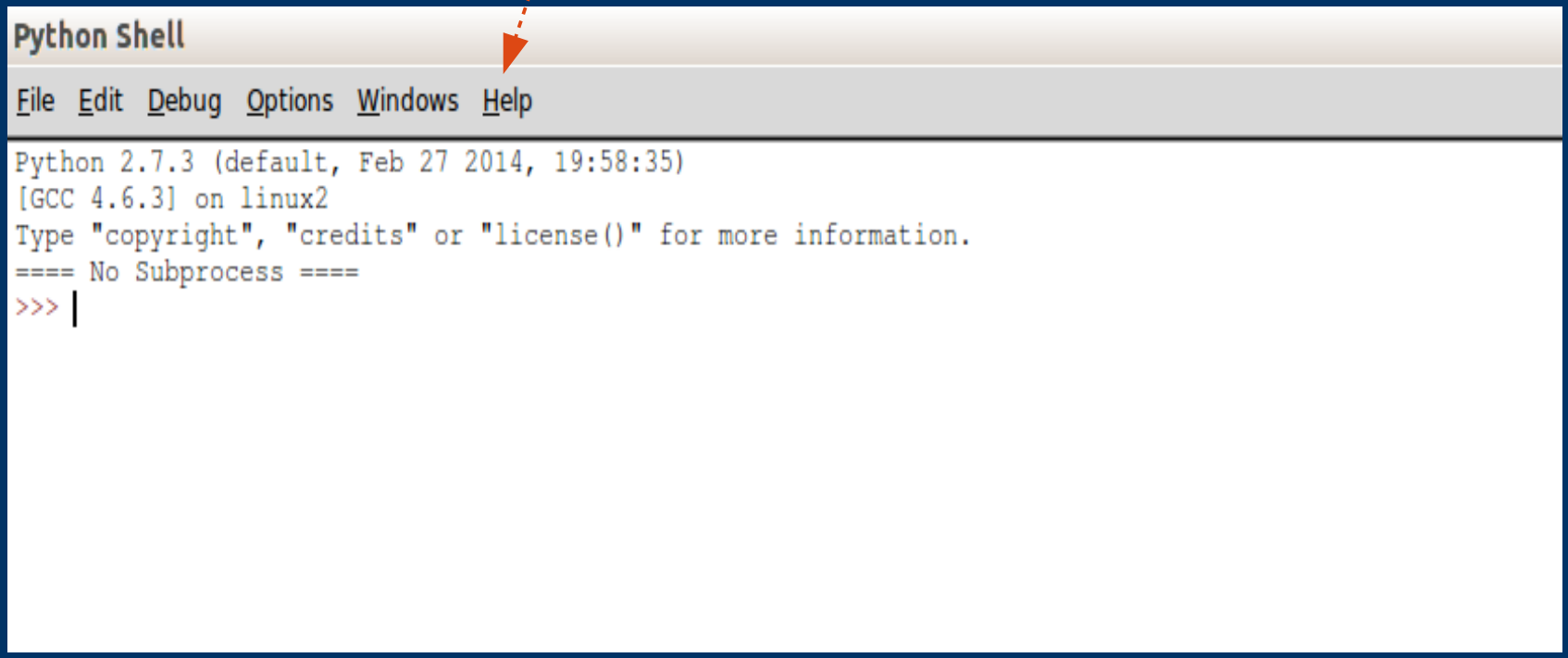


Try entering `print "Hello World!"` followed by the Enter key

IDLE - Python GUI

- Start Menu → Python → IDLE

Type F1 or Help → Python Docs to get full Python Docs
Most useful is “Library Reference”



Data and the Python Object Model

- All data takes the form of objects
- Built-in types for data
  - Number
  - List
  - Tuple
  - String
  - Dictionary
  - Set

Numbers

- Dynamic typing
- Created by simple assignment

```
a = 38                  # plain integer
b = 89765999999L        # long integer
c = 3.2567              # float
d = 2.3 + 5.7j          # complex number
e = 0.2345e10           # scientific notation
```

Integers

- *Plain* integers (same as C longs)

```
>>> import sys; sys.maxint
9223372036854775807 (on 64 bit system)
```

- Numbers can be expressed as decimal, octal or hex

```
65535
0177777    # octal
0o177777   # octal in Python 3
0xffff     # hex

# Conversions
oct(65535) → 0177777
hex(65535) → 0xffff
int(0xffff) → 65535
```

Long Integers

- Arbitrary precision integers
  - Size limited only by available memory
- Append L or l to number (but stick to L)

```
4200388L -0xff89L -234567881
```

- Integers auto promoted to longs if maximum size exceeded

```
>>> sys.maxint+1
92233720368547758081
```

Floating Point

- Represented as IEEE 754 64-bit double precision (implemented as C doubles)
- decimal or scientific notation
- Examples

```
55.678      -8.236   7.65e-13
float(13)   -80.      8.9E-10
```

Complex Numbers

- Complex number comprised of real and imaginary parts
- Syntax `real + imagj`
- Examples

```
3.14+1j  -3.23-456J   -.05678+0j  1.35e-10+1.5j
```

Boolean

- Two values : `True` and `False`
- `True` mapped to 1
- `False` mapped to 0

The None Type

- `None` type denotes a null object
- Returned by functions that don't explicitly return a value
- Often used as default value of optional arguments in a function
- `None` has no attributes
- `None` evaluates to `False` in Boolean expressions

Operations Supported By All Numbers

x + y	Addition
x – y	Subtraction
x * y	Multiplication
x / y	Division
x ** y	Power (x^y)
x % y	Modulo
-x	Unary minus
+x	Unary plus

Operations on Numbers (2)

Shifting and Bitwise Operators

x << y	Left shift
x >> y	Right shift
x & y	Bitwise and
x | y	Bitwise or
x ^ y	Bitwise exclusive or
~x	Bitwise negation

Operations on Numbers (3)

- Built-in maths functions

abs(x)	Returns absolute value
round(x [,n])	Round to n digits of precision
pow(x,y)	Returns (x ** y)
divmod(x,y)	Returns (int(x/y), x % y)

Example
Working with a formula

Vertical motion of ball

$$y(t) = v_0 t - \tfrac{1}{2} g t^2$$

Solving for $y = 0 \rightarrow t = 0$ or $t = 2v_0/g$

Python Shell – a Powerful Calculator

```
>>> V0 = 5          # m/s, integer
>>> g = 9.81        # m/s^2, float
>>> t = 2*v0/g      # muliply operator *, division /
>>> print t
1.019
>>> t = 0.7
>>> y = v0*t - 0.5 * g * t**2
>>> print y
1.09655
>>> y = 5 * 0.7 - 0.5 * 9.81 * 0.7**2
```

Script File

```
# ball.py
# Run as python ball.py from OS shell
V0 = 5      # m/s, integer
g = 9.81    # m/s^2, float
t = 0.7
y = v0*t - 0.5 * g * t**2   # 1.09655
print 'Height of ball at time %s = %s.' % (t, y)
```

Strings

- Creating strings
  - Single, double and triple quotes

```
>>> s1 = 'Hello World!'
>>> print “Hello\n my friend”
Hello
 My friend
>>> s2 = 'I\'m back.'   # escape embedded quote
>>> s3 = “I'm back.”    # better, no escape needed
```

Strings – Triple quoted

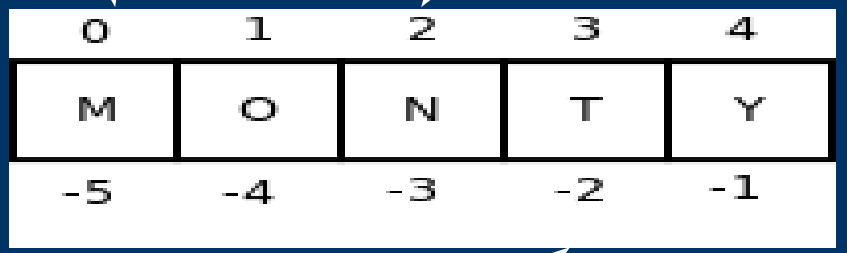
- Triple quotes preserves white space

```
>>> zen = """The Zen of Python, by Tim Peters
...
... Beautiful is better than ugly.
... Explicit is better than implicit.
...
... Type import this to see all"""
```

String is a sequence

Zero-based indexing

"MONTY"[2]



"MONTY"[-2]

Negative indexes count from string end

Strings

- Slicing

```
>>> s = 'Kuala Lumpur'
>>> s[0:3]          # 'Kua'
>>> s[6:]           # 'Lumpur'
```

- Concatenation

```
>>> s1 = 'See you in '
>>> s2 = 'September'
>>> print s1 + s2 + '!'
See you in September!
```

Strings

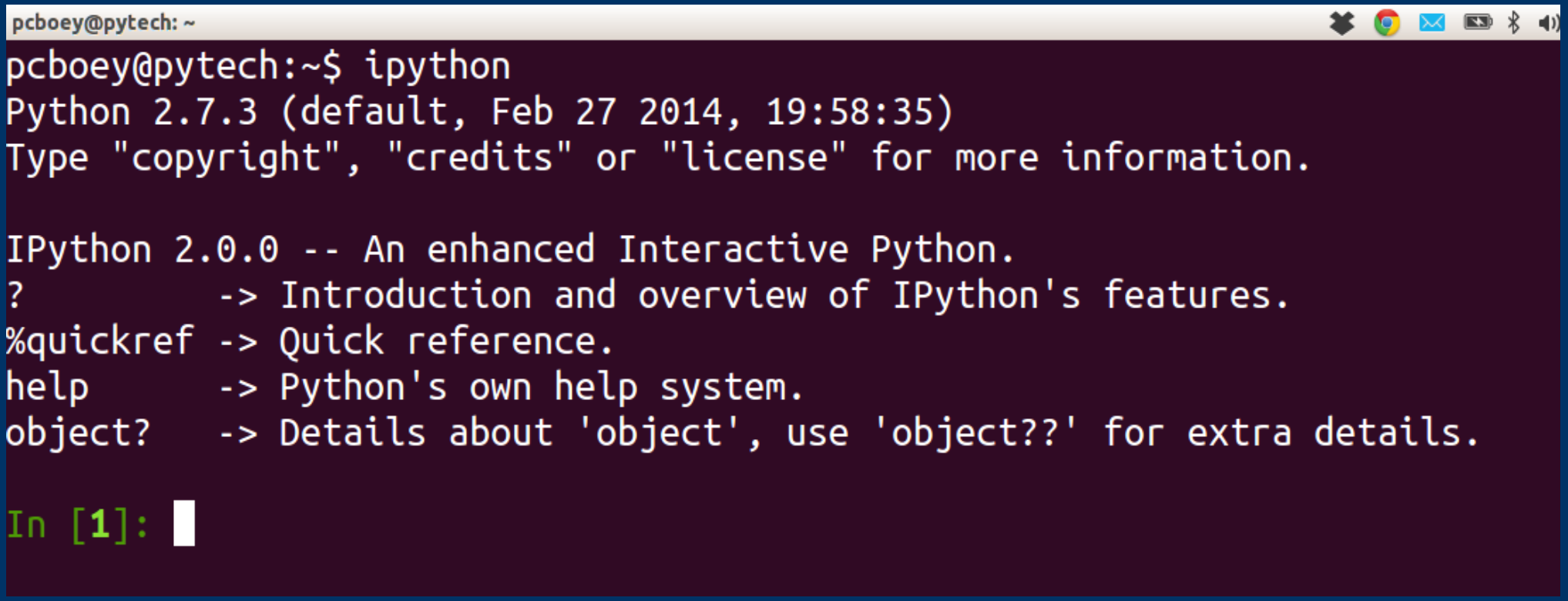
- Methods
  - Strings are immutable
  - Always return a new string object
  - Refer Python Standard Library docs for full list
  - Once familiar with the available methods, the IPython shell provides quick and easy help.

IPython – enhanced shell

- Installation
  - Enthought Python(x,y) and Python(x,y) installations already include IPython
  - For standard Python installations, use either easy_install or pip to install
    - `$ easy_install ipython`
    - `$ pip install ipython`
- Type `ipython` at command shell to start

The IPython shell

- $ ipython



Strings

- Using the IPython shell

```
In [1]: s = 'foo'
In [2]: s.
```

Press tab after the . for attribute completion

```
s.capitalize  s.encode       s.format . . .
s.center      s.endswith     s.index  . . .
```

```
In [2]: s.capitalize?
```

Use tab for completion, and ? for help on object

```
s.capitalize() -> string
Return a copy of the string S with only its first
character capitalized
```

Lists

- Sequence of arbitrary objects

```
a = [1.0, 2.0, 3.0]      # create a list
```

- Lists are mutable
- Zero-base indexing

```
x = a[1]                 # access
a[2] = 10                # modify → [1.0, 2.0, 10]
```

Lists (2)

- `range` built-in function
  - returns a list with arithmetic progression of integers

```
range(5)  → [0, 1, 2, 3, 4]
```

- Slicing works like for a string

```
range(5)[0:2] →  [0, 1]
```

Lists (3)

- Some useful built-in functions

```
n = [5, 78, 3, 1250]
len(n)    # 4
sum(n)    # 1336
min(n)    # 3
max(n)    # 1250
```

- Unpacking

```
x, y = [1.5, 3.7]    # x = 1.5, y = 3.7
```

Lists (4)

- List methods

```
s = [3*2, 'hello', 1.5, [1, 2, 3] ] # nested list
s[1:3]                  #  ['hello', 1.5]

# slice assignment
s[1:3] =  [5, 4]    # [6, 5, 4, [1,2,3] ]
s.append(2)         # [6, 5, 4, [1,2,3], 2]
s.insert(0, 7)      # [7, 6, 5 . . .]
```

- To see all the methods type `s.<TAB>` in IPython
- For help type `s.method?`, e.g. `s.remove?`

Lists (5)

- List comprehension – construct list from other lists

```
s  = ['3', '5', '7']          # list of strings
t  = [int(x) for x in s]      # t is now [3, 5, 7]
```

- Using an `if` statement

```
t2  = [int(x) for x in s if int(x) >= 5]   # [5, 7]
```

Lists (6)

- Nested lists used to represent matrices
  - For example 3 x 3 matrix

```
a = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
a[1]          # [4, 5, 6]
a[1][1]       # 5
```

- For numerical arrays, better to use `array` object from the `array` module

`array` module

- What is a module?
  - A file where functions, data, classes are defined
  - Module name is the name of the file
  - Can be loaded with the import statement

```
import module
from module import *   # import all
from module import attribute
```

```
>>> from array import array
>>> f = array('f', [0.1, 1.2, 3.14])
```

Tuples

- Similar to lists but immutable
- Creating tuples

```
 1, 2, 3     # constructor is the comma
(1, 2, 3)    # enclosed in parenthesis for clarity
(1,)         # single item tuple has comma
```

Tuples

- Why tuples?
  - Efficiency
  - Dictionary keys have to be immutable
  - Multiple function arguments require a tuple
  - Returned by some built-in functions and methods
  - String formatting require tuples
  - Ensure your data cannot be modified

Dictionaries

- Associative array or hash table
  - Containing objects indexed by keys

```
person = {"name" : "Bob",
          "age" : 36,
          "sex" : "Male"}
```

- Accessing values

```
n = person['name']
a = person['age']
```

Dictionaries (2)

Insert or modify objects

```
person['name'] = 'Charlie'
person['birthplace'] = 'Malaysia'
```

Keys can be any immutable object

- strings, tuple, numbers

Test for membership – key in dict

```
'name' in person → True or False
```

Functions

- Syntax for defining a function

```
def function_name([param1, param2, ...]):
    statements
    return return_values
```

```
>>> import math
>>> def area(radius):
...     return math.pi*radius**2
>>> area                  # function object
<function area at 0x20c46e0>
>>> print area(3.5)  #invoke with call operator ()
38.48451
```

More on functions

- Functions with default arguments

```
def foo(a, b, c=9, d='Bob'):
    print a,b,c,d

foo('hi', 'five') → hi five 9 Bob

foo('hi','five', 'Bob', 'Tan') → hi five Bob Tan

foo('Hi', 'there', d='Tony')  → Hi there 9 Tony
```

More on functions(2)

- Functions with variable number of arguments
  - Prefix an asterisk * to last parameter name

```
def product(*args):
    result = 1
    for number in args:
        result = result * number
    return result

answer = product(3,5,2,10)     # 300
```

More on functions (3)

- Accept arbitrary set of keyword arguments

```
def foo(msg, **kwargs):
    print 'msg = ', msg
    # kwargs is a dict mapping arg names to values
    print 'kwargs is of type ', type(kwargs)
    print kwargs

foo('bar', color='#00ffee', border='solid',
    font_size= '10px')
```

```
msg =  bar
kwargs is of type  <type 'dict'>
{'color': '#00ffee', 'font_size': '10px',
'border': 'solid'}
```

Statements, Conditionals and Loops

- Blocks and indentation
  - code block defined by indenting a group of statements
  - Example - writing out the initial sub-sequence of the Fibonacci series

```
>>> a, b = 0, 1      #   assignment
>>> while b < 10:
...       print b,     # trailing comma stops newline
...       # switch variables, expressions on RHS
...       # evaluated first before assignment
...       a, b = b, a+b
...
1 1 2 3 5 8
```

Conditionals

- Boolean Values
  - `True`
  - `False`
- Following values are evaluated as False

```
False
None
Zero of any numeric type 0, 0L, 0.0
Any empty sequence “”, '', [ ], ()
An empty dictionary {}

All other values are considered True
```

Conditionals – if statement

- General syntax

```
if expression:
   statements
elif expression:
   statements
. . .
else:
   statements
```

Conditionals – if statement (2)

- Example

```
grade = 65
if grade >= 70 :
   print 'Distinction'
elif grade >= 60:
   print 'Good'
elif grade >= 50:
   print 'Passed'
else:
   print 'Failed'

Good
```

Loops

The `while` statement

```
while a < b:
    a = a+1     # do something
```

The `for` statement (loops over members of a sequence)

```
for i in [3,5,7,9]:
    print i

# print characters one at a time
for c in "Hello World":
    print c

# loop over a range of numbers
for i in range(0,100):
    print i
```

Files

- The open() function

```
fw = open("foo", "w")    # write mode
fr = open("bar", "r")    # read mode, default
```

- Reading and writing data

```
fw.write("Hello World")
data = fr.read()          # read all data as string
line = fr.readline()      # read a single line
lines = fr.readlines()    # read as list of lines
```

- Formatted I/O
  - Use the % operator for strings (works like C printf)

```
for i in range(0,10):
    fw.write("2 times %d = %d\n" % (i, 2*i))
```

Read lines from a file

Iterate over lines of a file :

```
f = open(“test.txt”)
for line in f:
    # process line
f.close()
```

Simpler to use file object as an iterator :

```
for line in open(“test.txt”):
    # process line
```

Write to a file

Writing program output into a file

```
# ball2.py

v0 = 5
g = 9.81
t = 0.7
y = v0*t - 0.5 * g * t**2

f = open('ball_height.txt', 'w')
f.write('Given an initial velocity of %s m/s,
 the height of ball at time %s seconds = %s
 m.\n' % (v0, t, y))
f.close()
```

Classes

- Defining a class

```
class Book(object):
    """
    Write a helpful doc string here. Not a novel.
    """
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return "%s written by %s" % (self.title, self.author)

    def update_author(self, new_author):
        self.author = new_author

    def show_title(self):
        return 'Book title : %s' % self.title
```

Classes

- Instantiate a class by calling it like a function, passing it the required arguments

```
bk = Book("War and Peace", "Tolstoy")

# bk is a class instance
```

- Accessing class attributes with dot . operator

```
bk.author  → Tolstoy
bk.show_title()  →  Book title : War and Peace
print bk  → War and Peace written by Tolstoy
```

Exceptions

- The try statement

```
try:
    f = open('nosuchfile')
except IOError, e:
    print "Unable to open 'nosuchfile' : ", e
```

- Raising exceptions

```
raise KeyError('Oops No Key!')   # trigger exception manually

def func(x,y):
  assert (x > 0 and y > 0), 'No negative numbers allowed'
  statements
```

Modules

Organize large programs into modules

```
# file : div.py
def divide(a,b):
    q = a/b
    r = a - q*b
    return (q,r)
```

Use import statement

```
import div
a, b = div.divide(2305, 29)
```

Import using different name

```
import div as foo
a, b = foo.divide(2305, 29)
```

Python Standard Library

math (cmath)	Mathematical functions and constants (complex number version)
os	OS routines for the system we are on
os.path	Manipulation of pathnames in a platform-independent manner
sys	Provide variables and functions that are closely linked to the Python interpreter
time	Functions to manipulate time values
datetime	provides classes and functions for date and time manipulation (including arithmetic) and output formatting
random	functions for generating pseudo-random numbers
re	Regular expressions
Tkinter	Python interface to the Tk GUI toolki
pdb	Interactive source code debugger

Python Third Party Libraries

- Large amount of third party libraries
- Of interest to us
  - NumPy
  - matplotlib
  - SciPy
  - Pandas
- All these are accessed using the `import` statement

```
import numpy as np


# 2-dimensional array,size 2x3, 4-byte integer elements
x = np.array([[1,2,3],[4,5,6]], np.int32)
```

NumPy

- Central package for scientific computing
- Fast, efficient N-dimensional array object, `ndarray`
- Functions that operate element-wise on arrays
- Math operations between arrays
- Basic linear algebra, Fourier transform
- Sophisticated random number functions
- Read/write array-based data sets to disk
- Tools for integrating C/C++ and Fortran code

NumPy compared to Matlab

- NumPy array operations are element-wise
- Special matrix type for linear algebra
- NumPy uses zero-based indexing
- Python provides more programming power
- Matlab function definitions quite restrictive
- NumPy/SciPy is free and widely used
- Matlab has extensive set of domain-specific add-ons ("toolboxes")

NumPy compared to Matlab (2)

- NumPy has many good plotting libraries
  - Matplotlib - matured 2D plotting library
  - PyQwt – Python bindings for the Qwt C++ class library
  - PyQwt3D – 3D plotting
  - mlab - 3D plotting of NumPy arrays

Some Matlab-NumPy equivalents

Matlab	NumPy
a = [1 2 3; 4 5 6]	a = array([[1. , 2. , 3.], [4. , 5. , 6.]])
a(2, 3) → 6	a[1, 2] → 6.0
a(end)	a[-1] → array([4., 5., 6.])
a.'	a.transpose() or a.T
a * b (matrix multiply)	dot(a, b)
a .* b (element-wise multiply)	a * b
a ./ b (element-wise divide)	a / b
a .^ 3 (element-wise exponentiation)	a ^ 3
zeros(3,4) (3x4 rank-2 array full of 64-bit floating point zeros)	zeros((3,4))
rand(3,4)	random.rand(3,4)

SciPy

- Python-based ecosystem of open-source software for maths, science and engineering
- Core packages

NumPy	Base N-dimensional array package
SciPy library	Collection of various high level science and engineering modules
Matplotlib	Mature 2D plotting library
IPython	Enhanced Python Shell
SymPy	Symbolic mathematics
Pandas	Data structures and analysis

SciPy Package Organization

Subpackage	Functionality
cluster	Clustering algorithms useful in information theory, target detection, communications, compression.
constants	Various useful constants and conversion formulae
fftpack	Discrete Fourier transform algorithms
integrate	Integration routines
interpolate	Interpolation and smoothing splines
io	Data input and output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing tools

sparse	Sparse matrices and functions
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions
weave	C/C++ integration

- Recommended import conventions

```
import numpy as np
import scipy as sp
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
```

- SciPy subpackages need to be imported directly

```
from scipy import signal, linalg
```

Matplotlib

- 2D plotting library
- Example from matplotlib website :
  - Generate 10,000 gaussian random numbers
  - plot a histogram with 100 bins

```
$ipython --pylab
In [1]: x = randn(10000)
In [2]: hist(x, 100)
```

Pandas

- Data analysis library
- Built on top of NumPy
- Rich data structures
- Functions for fast, easy and expressive working with structured data

Pandas

- Suitable for many types of data
  - Tabular data as in a SQL table or Excel spreadsheet
  - Time series data (ordered and unordered)
  - Arbitrary matrix data with row and column labels
  - Other forms of observational / statistical data sets

Pandas

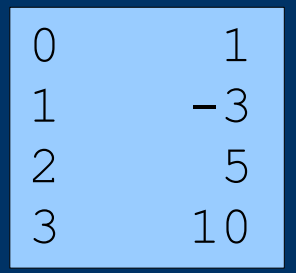
- 2 primary data structures
  - Series
    - 1-D array-like object with
      - Array of data (any NumPy data type)
      - Associated array of data labels (the index)
  - DataFrame (2-D)
    - tabular, spreadsheet-like data structure

Pandas – Series object

```
import pandas as pd

s = pd.Series( [1, -3, 5, 10] )

s.values → array([ 1, -3,  5, 10])
s.index → Int64Index([0, 1, 2, 3])
```



Often useful to specify an index for each data point

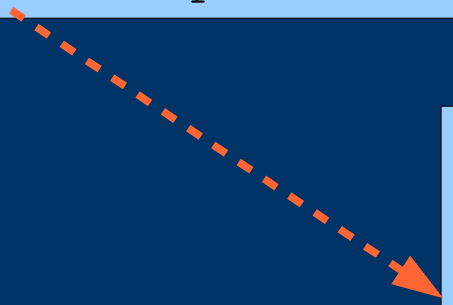
```
s2 = pd.Series([1, -3, 5, 10], index=['x', 'y', 'a', 'z'])
s2.index
Index([x, y, a, z], dtype=object)
```

Pandas – DataFrame object

- Constructing a Data Frame

```
data = {'state' : ['Johore', 'Johore',
                   'Johore', 'Kedah', 'Kedah'],
        'year' : [2008, 2009, 2010, 2008, 2009],
        'pop' : [3.252, 3.309, 3.362, 1.900, 1.925]}

frame = pd.DataFrame(data)
```



	pop	state	year
0	3.252	Johore	2008
1	3.309	Johore	2009
2	3.362	Johore	2010
3	1.900	Kedah	2008
4	1.925	Kedah	2009

Auto index

- Specify the columns sequence

```
pd.DataFrame(data, columns=['year', 'state', 'pop'])

   year   state    pop
0  2008  Johore  3.252
1  2009  Johore  3.309
2  2010  Johore  3.362
3  2008   Kedah  1.900
4  2009   Kedah  1.925
```

Python as a glue language

- What are extension modules?
- Call compiled libraries (C/C++ or Fortran)
- 2 approaches

1) Write an extension module (which is imported to Python)

2) Use the `ctypes` library to call a shared library routine (`.dll` or `.so` file)

Ways to writing extension modules

- Write directly using the Python/C API
- Cython optimising static compiler
- SWIG interface compiler
- f2py (building extension modules that interfaces to routines in Fortran 77/90/95 code)

Cython

“Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python as easy as Python itself. ”

Some Cython featues

- Combined power of Python and C :
  - Python code can call back and forth, from and to C/ C++ code natively at any point.
  - Just add static type declarations to tune Python code into C performance
  - Efficient interaction with large data sets, e.g. `ndarray`
  - Large and mature CPython ecosystem allows rapid building of your applications

SWIG

“ SWIG is an interface compiler that connects programs written in C and C++ with scripting languages such as Perl, Python, Ruby, and Tcl.”

Conclusion

- You have seen a lot of what you need to know
- Python is a relatively simple language
- Most beginners pick it up quickly and start doing things right away
- The code is readable
- When in doubt, **experiment interactively**
- IPython shell and notebook are great for doing that