High Level Scripting
Part I

Gino Tosti
University & INFN Perugia

Scripting Languages

- What is a script?
  - It is a small program able to automate a repetitive and boring job;
  - It is a list of commands that can be executed without user interaction
- Scripting languages are often used as "glue" languages. That is they are used to glue together tools and programs which may not, themselves, be written in the scripting language.

Scripting Languages

- Dedicated to simple tasks:
  - awk
  - Sed
- Dedicated to drive specific applications:
  - Visual Basic (MS Office)
  - Javascript (Web)
  - Emacs Lisp (Emacs)
- General purpouse :
  - Python
  - Perl
  - Ruby

What is Python

- It is a simple programming language
  - "interpreted" (rather than compiled into machine code)
  - High level
  - Simple to learn and to use
  - Powerful and productive
  - Similar to pseudo-code
- Moreover
  - open source (www.python.org)
  - multiplatform
  - Easy to be integrated with C/C++ and Java

- http://wiki.python.org/moin/LanguageComparisons

Basic libraries

- Numpy (http://numpy.scipy.org/)
- Pyfits (http://www.stsci.edu/resources/software_hardware/pyfits)
- Matplotlib (http://matplotlib.sourceforge.net/)
- Scipy (http://www.scipy.org/; http://www.scipy.org/Download)

Overview

- Running Python and Output
- Data Types
- Input and File I/O
- Control Flow
- Functions
- Classes

More here: http://docs.python.org/tutorial/

Hello World

- Open a terminal window and type "python"
- Or open a Python IDE like IDLE
- At the prompt type:

```
>>> print 'hello world!'
'hello world!'
```

Python Overview

- Programs are composed of modules
- Modules contain statements
- Statements contain expressions
- Expressions create and process objects

The Python Interpreter

- Python is an interpreted language
- The interpreter provides an interactive environment to play with the language
- Results of expressions are printed on the screen

```
>>> 3 + 7
10
>>> 3 < 15
True
>>> 'print'
'print'
>>> print 'print '
Print
>>>
```

Output: The print Statement

- Elements separated by commas print with a space between them

- A comma at the end of the statement (print 'hello',) will not print a newline character

```
>>> print 'hello'
hello
>>> print 'hello', 'Ciao'
hello Ciao
```

Variables

- Are not declared, just assigned
- The variable is created the first time you assign it a value
- Are references to objects
- Type information is with the object, not the reference
- Everything in Python is an object

Naming Rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.

  `bob  Bob  bob  2 bob  bob 2  BoB`

- There are some reserved words:

```
and, assert, break, class, continue,
def, del, elif, else, except, exec,
finally, for, from, global, if,
import, in, is, lambda, not, or,
pass, print, raise, return, try,
while
```

Numbers: Integers

- Integer – the equivalent of a C long
- Long Integer – an unbounded integer value. Newer versions of Python will automatically convert to a long integer if you overflow a normal one

```
>>> 132224
132224
>>> 132323 ** 2
17509376329L
>>>
```

Numbers: Floating Point

- int(x) converts x to an integer
- float(x) converts x to a floating point
- The interpreter shows a lot of digits, including the variance in floating point
- To avoid this use "print"

```
>>> 1.23232
1.2323200000000001
>>> print 1.23232
1.23232
>>> 1.3E7
13000000.0
>>> int(2.0)
2
>>> float(2)
2.0
```

Numbers: Complex

- Built into Python
- Same operations are supported as integer and float

```
>>> x = 3 + 2j
>>> y = -1j
>>> x + y
(3+1j)
>>> x * y
(2-3j)
```

Math Module

- Many math functions in the math module

```
$ python
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
    'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
    'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor', 'fmod',
    'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log', 'log10',
    'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
    'tanh', 'trunc']
```

String Literals

- Strings are immutable
- There is no char type like in C++ or Java
- + is overloaded to do concatenation

```
>>> x = 'hello'
>>> x = x + ' there'
>>> x
'hello there'
```

Strings

- Can use single or double quotes, and three double quotes for a multi-line string

```
>>> 'this is a string'
'this is a string'
>>> "this is a string"
'this is a string'
>>> """this is a
... long string"""
'this is a \nlong string'
```

Substrings and Methods

```
>>> s = '012345'
>>> s[3]
'3'
>>> s[1:4]
'123'
>>> s[2:]
'2345'
>>> s[:4]
'0123'
>>> s[-2]
'4'
```

- len(String) – returns the number of characters in the String

- str(Object) – returns a String representation of the Object

```
>>> len(x)
6
>>> str(10.3)
'10.3'
```

String Formatting

- Similar to C's printf
- <formatted string> % <elements to insert>
- Can usually just use %s for everything, it will convert the object to its String representation.

```
>>> "One, %d, three" % 2
'One, 2, three'
>>> "%d, two, %s" % (1,3)
'1, two, 3'
>>> "%s two %s" % (1, 'three')
'1 two three'
>>>
```

Data Types

- Integers: 2323, 3234L
- Floating Point: 32.3, 3.1E2
- Complex: 3 + 2j, 1j
- Lists: l = [1,2,3]
- Tuples: t = (1,2,3)
- Dictionaries: d = {'hello' : 'there', 2 : 15}

Lists

- Ordered collection of data
- Data can be different types
- Lists are mutable
- Same subset operations as Strings

```
>>> x = [1,'hello', (3 + 2j)]
>>> x
[1, 'hello', (3+2j)]
>>> x[2]
(3+2j)
>>> x[0:2]
[1, 'hello']
```

Lists: Modifying Content

- x[i] = a reassigns the ith element to the value a
- Since x and y point to the same list object, both are changed
- Append also modifies the list

```
>>> x = [1,2,3]
>>> y = x
>>> x[1] = 15
>>> x
[1, 15, 3]
>>> y
[1, 15, 3]
>>> x.append(12)
>>> y
[1, 15, 3, 12]
```

Lists: not all are in-place changes

- Append modifies the list and returns None (the python version of null)
- List addition returns a new list

```
>>> x = [1,2,3]
>>> y = x
>>> z = x.append(12)
>>> z == None
True
>>> y
[1, 2, 3, 12]
>>> x = x + [9,10]
>>> x
[1, 2, 3, 12, 9, 10]
>>> y
[1, 2, 3, 12]
>>>
```

Tuples

- Tuples are immutable versions of lists

```
>>> x = (1,2,3)
>>> x[1:]
(2, 3)
>>> y = (2)
>>> y
(2)
>>>
```

Dictionaries

- A set of key-value pairs
- Dictionaries are mutable

```
>>> d = {1 : 'hello', 'two' : 42, 'b' : [1,2,3]}
>>> d
{1: 'hello', 'two': 42, 'b': [1, 2, 3]}
>>> d['b']
[1, 2, 3]
```

Dictionaries: Add/Modify

- Entries can be changed by assigning to that entry
- Assigning to a key that does not exist adds an entry

```
>>> d
{1: 'hello', 'two': 42, 'b': [1, 2, 3]}
>>> d['two'] = 99
>>> d
{1: 'hello', 'two': 99, 'b': [1, 2, 3]}
>>> d[7] = 'new entry'
>>> d
{1: 'hello', 7: 'new entry', 'two': 99, 'b': [1, 2, 3]}
```

Dictionaries: Deleting Elements

- The del method deletes an element from a dictionary

```
>>> d
{1: 'hello', 2: 'there', 10: 'world'}
>>> del(d[2])
>>> d
{1: 'hello', 10: 'world'}
```

Copying Dictionaries and Lists

- The built-in list function will copy a list
- The dictionary has a method called copy

```
>>> l1 = [1]
>>> l2 = list(l1)
>>> l1[0] = 22
>>> l1
[22]
>>> l2
[1]
```

```
>>> d = {1 : 10}
>>> d2 = d.copy()
>>> d[1] = 22
>>> d
{1: 22}
>>> d2
{1: 10}
```

Input

- The raw_input(string) method returns a line of user input as a string
- The parameter is used as a prompt
- The string can be converted by using the conversion methods int(string), float(string), etc.

Input: Example

```
print "What's your name?"
name = raw_input(">")

print "What year were you born?"
birthyear = int(raw_input(">"))

print "Hi %s! You are %d years old!" % (name, 2010 - birthyear)
```

```
python input.py
What's your name?
Albert
What year were you born?
>1990
Hi Michael! You are 20 years old!
```

Files: Input

input = open(‘data’, ‘r’)	Open the file for input
S = input.read()	Read whole file into one String
S = input.read(N)	Reads N bytes (N >= 1)
L = input.readlines()	Returns a list of line strings

Files: Output

output = open(‘data’, ‘w’)	Open the file for writing
output.write(S)	Writes the string S to file
output.writelines(L)	Writes each of the strings in list L to file
output.close()	Manual close

Boolean Expressions

- Compound boolean expressions short circuit
- and and or return one of the elements in the expression
- Note that when None is returned the interpreter does not print anything

```
>>> True and False
False
>>> False or True
True
>>> 7 and 14
14
>>> None and 2
>>> None or 2
2
```

If Statements

```
x = 22

if x < 15 :
    print 'first clause'
elif x < 25 :
    print 'second clause'
else :
    print 'the else'
```

No Braces

- Python uses indentation instead of braces to determine the scope of expressions
- All lines must be indented the same amount to be part of the scope (or indented more if part of an inner scope)
- This **forces** the programmer to use proper indentation since the indenting is part of the program!

While Loops

```
x = 1

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

Loop Control Statements

break	Jumps out of the closest enclosing loop
continue	Jumps to the top of the closest enclosing loop
pass	Does nothing, empty statement placeholder

The Loop Else Clause

- The optional else clause runs only if the loop exits normally (not by break)

```
x = 1

while x < 3 :
    print x
    x = x + 1
else:
    print 'hello'
```

For Loops

- Similar to perl/basic for loops, iterating through a list of values

```
for x in [1,7,13,2] :
    print x
```

Function Basics

```
def max(x,y) :
    if x < y :
        return x
    else :
        return y
```

Function Not-So-Basics

- Functions are **first class objects**
  - Can be assigned to a variable
  - Can be passed as a parameter
  - Can be returned from a function
- Functions are treated like any other variable in python, the def statement simply assigns a function to a variable

Functions as Variables

- Functions are objects
- The same reference rules hold for them as for other objects

```
>>> x = 10
>>> x
10
>>> def x () :
...     print 'hello'
>>> x
<function x at 0x619f0>
>>> x()
hello
>>> x = 'b'
>>> x
'b'
```

Functions: As Parameters

```
def foo(f, a) :
    return f(a)

def bar(x) :
    retun x * x
```

```
>>> from funcasparam import *
>>> foo(bar, 3)
9
```

- The function foo takes two parameters and applies the first as a function with the second as its parameter

Functions: In Functions

- Since they are like any other object, you can have functions inside functions

```
def foo (x,y) :
    def bar (z) :
        return z * 2

    return bar(x) + y
```

Functions Returning Functions

```
def foo (x) :
    def bar(y) :
        return x + y

    return bar

# main
f = foo(3)
print f
print f(2)
```

Parameters: Defaults

- Can assign default values to parameters
- They are overridden if a parameter is given for them
- The type of the default doesn't limit the type of a parameter

```
>>> def foo(x = 3) :
...     print x
...
>>> foo()
3
>>> foo(10)
10
>>> foo('hello')
hello
```

Parameters: Named

- Can specify the name of the parameter to set
- Any positional arguments must come before named ones in a call

```
>>> def foo (a,b,c) :
...     print a, b, c
...
>>> foo(c = 10, a = 2, b = 14)
2 14 10
>>> foo(3, c = 2, b = 19)
3 19 2
```

Anonymous Functions

- The lambda returns a function
- The body can only be a simple expression, not complex statements

```
>>> f = lambda x,y : x + y
>>> f(2,3)
5
>>> l = ['one', lambda x : x * x, 3]
>>> l[1](4)
16
```

Classes

- A collection of data and methods that act on that data
- But in python functions are basically just data!

Class Syntax

- Every method in a class takes a self-pointer as the first parameter (self as convention) like this in java or C++
- __init__ is a builtin function that you override as the constructor

```
class myclass :

    def __init__(self, val) :
        self.x = val

    def print1(self) :
        print self.x
```

```
>>> from classbasic import *
>>> x = myclass(3)
>>> x.print1()
3
```

Class Method Calls

- Self is automatically added as the first parameter when a method is called on an instance of a class
- Internally self must be explicitly passed

Classes as Objects

- Classes exist as objects and contain all of there own variables
- Name resolution starts looking in the instance of the class for a variable, then walks up the inheritance tree
- Variables don't exist in the instance until they are assigned there

Classes

```
>>> class myclass :
...     x = 10
...
>>> a = myclass()
>>> b = myclass()
>>> a.x, b.x, myclass.x
(10, 10, 10)
>>> a.x = 15
>>> a.x, b.x, myclass.x
(15, 10, 10)
```

```
>>> myclass.x = 20
>>> a.x, b.x, myclass.x
(15, 20, 20)
>>> a.x = myclass.x
>>> a.x, b.x, myclass.x
(20, 20, 20)
>>> myclass.x = 99
>>> a.x, b.x, myclass.x
(20, 99, 99)
```

Modules: Imports

import mymodule	Brings all elements of mymodule in, but must refer to as mymodule.<elem>
from mymodule import x	Imports x from mymodule right into this namespace
from mymodule import *	Imports all elements of mymodule into this namespace

Standard modules
http://docs.python.org/modindex.html

Error Capture

- Check for type assignment errors, items not in a list, etc.

- Try & Except

```
try:
    a block of code that might have an error
except:
    code to execute if an error occurs in "try"
```

PyROOT

- **export** LD_LIBRARY_PATH=$ROOTSYS/lib:$PYTHONDIR/lib:$LD_LIBRARY_PATH

- **export** PYTHONPATH=$ROOTSYS/lib:$PYTHONPATH

PyROOT

```
from ROOT import gROOT, TCanvas, TF1


gROOT.Reset()
c1 = TCanvas( 'c1', 'Example with Formula', 200, 10, 700,
    500 )
fun1 = TF1( 'fun1', 'abs(sin(x)/x)', 0, 10 )
c1.SetGridx()
c1.SetGridy()
fun1.Draw()
c1.Update()
raw_input()
```