12.010 Computational Methods of Scientific Programming

Lecturers
Thomas A Herring
Chris Hill

Overview

- Part 1: Python Language Basics – getting started.
- Part 2: Python Advanced Usage – the utility of Python

Refresh

- Previous class:

- History
- Python features
- Getting Python and help
- Modes of running Python
- Basics of Python scripting
- Variables and Data types
- Operators
- Conditional constructs and loops

Part 2: Advanced Python

- Today we will look at:
  - Functions
  - Modules
  - File IO
  - Time
  - Exceptions
  - Parsing command line options/arguments
  - CGI programming
  - Database access
  - Math Modules numpy and scipy
  - Graphics with python matplotlib

Functions

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provides better modularity for your application and a high degree of code reusability.
- As you already know, Python gives you many built-in functions like print() etc. but you can also create your own functions. These functions are called *user-defined functions.*
- *Here are simple rules to define a function in Python:*
  - Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
  - Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
  - The first statement of a function can be an optional statement - the documentation string of the function or **docstring.**
  - The code block within every function starts with a colon (:) and is indented.
  - The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

A Function

- *Syntax:*

def function_name(parameters):
"function_docstring"
function_suite
return [expression]

- By default, parameters have a positional behavior, and you need to inform them in the same order that they were defined.
- The function below takes a string as an input parameter and prints it on the screen.

```
def print_me( str ):
        "This prints a passed string into this function"
        print str
        return
```

Calling a Function

- Defining a function only gives it a name, specifies the parameters that are to be included in the function, and structures the blocks of code.
- Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.
- Following is the example script to call print_me() function:

```
#!/usr/bin/python
# Function definition is here
def print_me( str ):
        "This prints a passed string into this function"
        print str
        return
# Now you can call print_me function
print_me("I'm first call to user defined function!")
print_me("Again second call to the same function")
```

This would produce following output:

```
I'm first call to user defined function!
Again second call to the same function
```

Function behavior

- Python passes all arguments using "pass by reference". However, numerical values and Strings are all immutable in place. You cannot change the value of a passed in variable and see that value change in the caller. Dictionaries and Lists on the other hand are mutable, and changes made to them by a called function will be preserved when the function returns. This behavior is confusing and can lead to common mistakes where lists are accidentally modified when they shouldn't be. However, there are many reasons for this behavior, such as saving memory when dealing with large sets.
- Concept her is similar to pass by value or by pointer in C (the array analogy in C is very close to the concept).

Function Arguments

```
#!/usr/bin/python
a, b, c = 0, 0, 0; abc = [0,0,0]
def getabc(a,b,c):
    a = "Hello ”;  b = "World ”; c = "!"
    print 'Inside: a,b,c: ',a,b,c
    return
def getlist(abc):
    seq = ['Hello','World','! ’]
    for index in range(len(abc)):
        abc.pop(0)
    abc.extend(seq)
    print 'Inside: abc: ',abc
    return
x = getabc(a,b,c)
print 'Outside: a,b,c: ',a,b,c
y = getlist(abc)
print 'Outside: abc: ',abc
```

This produces the following output:

```
Inside: a,b,c:  Hello World !
Outside: a,b,c:  0 0 0
Inside: abc:  ['Hello', 'World', '!']
Outside: abc:  ['Hello', 'World', '!']
```

Modules

- A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use.
- A module is a Python object with arbitrarily named attributes that you can bind and reference.
- Simply, a module is a file consisting of Python code. A module can define functions, classes, and variables. A module can also include runnable code.
- Example: *Here's an example of a simple module named - hello.py*

```
def print_func( par ):
   "Hello.py – prints Hello : and the passed parameter"
   print "Hello : ", par
  return
```

Modules

- You can use any Python source file as a module by executing an import statement in some other Python source file**. Import** has the following syntax:

import module1[, module2[,... moduleN]

- When the interpreter encounters an import statement, it imports the module if the module is present in the search path. The search path is a list of directories that the interpreter searches before importing a module.
- To import the module hello.py we created above, put the following command at the top of a new script – test_module.py:

```
#!/usr/bin/python
import hello # Import the module hello.py
# Now you can call the defined function that module as follows
hello.print_func("Zara")
```

When executed test_module.py would produce following output:

```
Hello : Zara
```

Modules

- A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.
- Python's *from statement lets you import specific attributes from a module into the current namespace:*

from modname import name1[, name2[, ... nameN]]

- For example, to import the function fibonacci from the module fib, use the following statement:

from fib import fibonacci

- This statement does not import the entire module fib into the current namespace; it just introduces the item fibonacci from the module fib into the global symbol table of the importing module.

Modules

- When you import a module, the Python interpreter searches for the module in the following sequences:
  - The current directory.
  - If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
  - If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python/.
- The module search path is stored in the system module **sys** as the sys.path variable. The sys.path variable contains the current directory, PYTHONPATH, and the installation-dependent defaults.
- *The PYTHONPATH is an environment variable, consisting of a list of directories. The syntax of PYTHONPATH is the same as that of the shell variable PATH.*
- *Here is a typical PYTHONPATH from a Windows system:*

set PYTHONPATH=c:\python20\lib;

- And here is a typical PYTHONPATH from a UNIX system:

set PYTHONPATH=/usr/local/lib/python

Dates and Times

- There are three common ways of manipulating dates and times in Python
  - time : A python low-level standard library module
  - datetime : Another standard library module
  - mxDateTime : A popular third-party module (not discussed but if you have many time related manipulations in your code I would recommend installing this module).
- Examples

```
import time
import datetime
print "Today is day", time.localtime()[7], "of the current year"
Today is day 310 of the current year
today = datetime.date.today()
print "Today is day", today.timetuple()[7], "of ", today.year
Today is day 310 of 2010
print "Today is day", today.strftime("%j"), "of the current year"
Today is day 310 of the current year
```

Date Time Modules

- The Standard Library
- http://docs.python.org/library/datetime.html#datetime-objects
- mxDateTime
- http://www.egenix.com/products/python/mxBase/mxDateTime

Python IO

- Printing to the Screen:
- The simplest way to produce output is using the *print statement where you can pass zero or more expressions, separated by commas. This function converts the expressions you pass it to a string and writes the result to standard output as follows:*

```
#!/usr/bin/python
print "Python is really a great language,", "isn't it?";
```

This would produce following screen output:

```
Python is really a great language, isn't it?
```

Keyboard Input

- Python provides two built-in functions to read a line of text from standard input, which by default is from the your keyboard. These functions are:
  - raw_input
  - Input
- *The raw_input([prompt]) function reads one line from standard input and returns it as a string (removing the trailing newline):*

```
#!/usr/bin/python
str = raw_input("Enter your input: ");
print "Received input is : ", str
```

- This would prompt you to enter any string and it would display same string on the screen. When I typed "Hello Python!", it output is like this:

```
Enter your input: Hello Python
Received input is :  Hello Python
```

Keyboard Input

- The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you:

```
#!/usr/bin/python
str = input("Enter your input: ");
print "Received input is : ", str
```

- This would produce following result against the entered input:

```
Enter your input: [x*5 for x in range(2,10,2)]
Recieved input is :  [10, 20, 30, 40]
```

Opening a File

- Python provides basic functions and methods necessary to manipulate files. You can do your most of the file manipulation using a file object.
- The *open Function:*
- *Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object which would be utilized to call other support methods associated with it.*
- ***Syntax**:*

file object = open(file_name [, access_mode][, buffering])

- file_name: The file_name argument is a string value that contains the name of the file that you want to access.
- access_mode: The access_mode determines the mode in which the file has to be opened ie. read, write append etc. A complete list of possible values is given below. This is optional parameter and the default file access mode is read (r)
- buffering: If the buffering value is set to 0, no buffering will take place. If the buffering value is 1, line buffering will be performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action will be performed with the indicated buffer size. This is optional parameter.

File access modes

- r Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
- rb Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
- r+ Opens a file for both reading and writing. The file pointer will be at the beginning of the file.
- rb+ Opens a file for both reading and writing in binary format. The file pointer will be at the beginning of the file.
- w Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
- wb Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
- w+ Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
- wb+ Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

Access modes cont…

- a Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
- ab Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
- a+ Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
- ab+ Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for

File Attributes

- *Once a file is opened and you have one file object, you can get various information related to that file.*
- *Here is a list of all attributes related to file object:*
  - file.closed Returns true if file is closed, false otherwise.
  - file.mode Returns access mode with which file was opened.
  - file.name Returns name of the file.
- Example:

```
#!/usr/bin/python
fo = open("foo.txt", "wb") # Open a file
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
```

- This would produce following result:

```
Name of the file:  foo.txt
Closed or not :  False
Opening mode :  wb
```

Closing Files

- *The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.*
- *Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.*
- *Syntax:*

fileObject.close();

- Example:

```
#!/usr/bin/python
fo = open("foo.txt", "wb")          # Open a file
print "Name of the file: ", fo.name
fo.close()          # Close opend file
```

This would produce following result:

```
Name of the file:  foo.txt
```

Writing Files

- The *write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.*
- *The write() method does not add a newline character ('\n') to the end of the string:*
- **Syntax*:***

fileObject.write(string);

- The passed parameter "string" is to be written into the open file.
- **Example:**

```
#!/usr/bin/python
fo = open("foo.txt", "wb")    # Open a file
fo.write( "12.010 Computational Methods of Scientific Programming is great.\nYeah its great!!\n");
fo.close()              # Close opend file
```

- The above code would create *foo.txt file and would write given content in that file and finally it would close that file. The file would contain*

12.010 Computational Methods of Scientific Programming *is great.*

Yeah its great!!

Writing methods

- file.write(str) Write a string to the file. There is no return value.

- file.writelines(sequence) Write a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings.

Reading files

- The *read() method read a string from an open file. It is important to note that Python strings can have binary data and not just text.*
- **Syntax:**

fileObject.read([count])

- Here passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if *count is missing then it tries to read as much as possible, may be until the end of file.*
- *Example - Let's use file foo.txt which we have created above.*

```
#!/usr/bin/python
fo = open("foo.txt", "r+") # Open a file
str = fo.read(10);
print "Read String is : ", str
fo.close()          # Close opend file
```

- This would produce following output:

```
Read String is :  12.010 Com
```

Reading Methods

- file.next() Returns the next line from the file each time it is being called.
- file.read([size]) Read at most size bytes from the file (less if the read hits EOF before obtaining size bytes).
- file.readline([size]) Read one entire line from the file. A trailing newline character is kept in the string.
- file.readlines([sizehint]) Read until EOF using readline() and return a list containing the lines. If the optional sizehint argument is present, instead of reading up to EOF, whole lines totalling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read.

File Positions

- The *tell() method tells you the current position within the file in other words, the next read or write will occur at that many bytes from the beginning of the file:*
- *The seek(offset[, from]) method changes the current file position. The offset argument indicates the number of bytes to be moved. The from argument specifies the reference position from where the bytes are to be moved.*
  - *If **from** is set to 0, it means use the beginning of the file as the reference position.*
  - *If from is set 1 it means use the current position as the reference position.*
  - *if from is set to 2 then the end of the file would be taken as the reference position.*

File Positions

- *Take a file foo.txt which we have created before.*

```
#!/usr/bin/python
fo = open("foo.txt", "r+") # Open a file
str = fo.read(10);
print "Read String is : ", str
# Check current position
position = fo.tell();
print "Current file position : ", position
# Reposition file pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str
fo.close()  # Close opend file
```

This would produce following output:

Read String is : 12.010 Com

Current file position : 10

Again read String is : 12.010 Com

OS Module

- Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files. To use this module you need to import it first and then you can call any related methods.
- The *rename() method takes two arguments, the current filename and the new filename.*

- *Syntax:*

os.rename(current_file_name, new_file_name)

- Following is the example to rename an existing file *test1.txt:*

```
#!/usr/bin/python
import os
# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

OS Module

- Some other os file and directory methods…..
  - os.delete(file_name)
  - *os.mkdir("newdir")*
  - *os.chdir("newdir")*
  - *os.getcwd()*
  - *os.rmdir('dirname')*
- *Complete list at:*

http://docs.python.org/library/os.html#files-and-directories

Exceptions

- An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it can't cope with, it raises an exception. An exception is a Python object that represents an error.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it would terminate.

Handling an exception:

- If you have some *suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.*

Exceptions

- *Here is simple syntax of try....except...else blocks:*

Try:

Do you operations here.

except Exception_I:

If there is Exception_I, then execute this block.

except Exception_II:

If there is Exception_II, execute this block.

else:

If there is no exception then execute this block.

Exceptions

- Here are few important points above the above mentioned syntax:
  - A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
  - You can also provide a generic except clause, which handles any exception. (not recommended since you don't really know what raised the exception).
  - After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
  - The else-block is a good place for code that does not need the try: block's protection.

Handling Exceptions

- Here is simple example which opens a file and writes the content in the file and comes out gracefully because there is no problem at all:

```
#!/usr/bin/python
try:
        fh = open("testfile", "w")
        fh.write("This is my test file for exception handling!!")
except IOError:
        print "Error: can\'t find file or write data”
else:
        print "Written content in the file successfully”
        fh.close()
```

If no exception is raised this will produce following output:

Written content in the file successfully

If an exception is raised this will produce following output:

Error can’t find file or write data

Full Exception handling Docs

- Tutorial on handling exceptions

http://docs.python.org/tutorial/errors.html#handling-exceptions

- List of the python standard exceptions

http://docs.python.org/c-api/exceptions.html#standard-exceptions

Command line options

- The Python sys module provides access to any command-line arguments via the sys.argv. This serves two purpose:
  - sys.argv is the list of command-line arguments.
  - len(sys.argv) is the number of command-line arguments.
- Here sys.argv[0] is the program ie. script name.
- Consider the following script test.py:

```
#!/usr/bin/python
import sys
print 'Number of arguments:', len(sys.argv), 'arguments.'
print 'Argument List:', str(sys.argv)
```

Now run above script as follows:

```
$ python test.py arg1 arg2 arg3
```

This will produce following output:

```
Number of arguments: 4 arguments.
Argument List: ['test.py', 'arg1', 'arg2', 'arg3']
```

Parsing the Command Line

- Python provides the **getopt** module that helps you parse command-line options and arguments. This module provides two functions and an exception to enable command line argument parsing.

http://docs.python.org/library/getopt.html

- Another module (which I personally like better than the python standard library getopt) for parsing the command line is argparse

http://code.google.com/p/argparse/

CGI Programming

- The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script or program.
- To understand the concept of CGI, lets see what happens when we click a hyper link to browse a particular web page or URL.
  - Your browser contacts the HTTP web server and asks for the URL ie. filename.
  - Web Server will parse the URL and will look for the filename, if it finds that file it sends to the browser otherwise sends an error message indicating that file could not be found .
  - Web browser takes response from web server and displays either the received file or error message.
- However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your browser to display. This function is called the Common Gateway Interface or CGI.

CGI Programming

- By default, the Linux apache web server is configured to run only the scripts in the /var/www/cgi-bin directory
- Python scripts can be placed in this cgi-bin directory to serve dynamic information to web clients.

```
#!/usr/bin/python
print "Content-type:text/html\r\n\r\n”
print '<html>’
print '<head>’
print '<title>Hello Word - First CGI Program</title>’
print '</head>'print '<body>’
print '<h2>Hello Word! This is my first CGI program</h2>’
print '</body>’
print '</html>'
```

- If you click hello.py then this produces following output:

Hello Word! This is my first CGI program

http://www.tutorialspoint.com/python/python_cgi_programming.htm

Database Access

- The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.
- You can choose the right database for your application. Python Database API supports a wide range of database servers including:
  - GadFly
  - mSQL
  - MySQL
  - PostgreSQL
  - Microsoft SQL Server 2000
  - Informix
  - Interbase
  - Oracle
  - Sybase

http://www.tutorialspoint.com/python/python_database_access.htm

Numpy

- NumPy is the fundamental package needed for scientific computing with Python. It contains among other things:
  - a powerful N-dimensional array object
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities.

- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

http://www.scipy.org/Tentative_NumPy_Tutorial

scipy

- SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engineering. It is also the name of a very popular conference on scientific programming with Python.
- The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation.
- The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization.

http://www.scipy.org/

matplotlib

- matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.
- matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code.

- http://matplotlib.sourceforge.net/

Summary

- Functions
- Modules
- File IO
- Time
- Exceptions
- Parsing command line options/arguments
- CGI programming
- Database access
- Math Modules numpy and scipy
- Graphics with python matplotlib

12.010 Computational Methods of Scientific Programming
Fall 2011