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

Part1: Summary of Python basics

- Today we will look at:
  - History
  - Python features
  - Getting Python and help
  - Modes of running Python
  - Basics of Python scripting
  - Variables and Data types
  - Operators
  - Conditional constructs and loops

History

- Python was created by Guido van Rossum in the late 1980's at the National Research Institute for Mathematics and Computer Science in the Netherlands. Like Perl, Python source code is available under the GNU General Public License (GPL).
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Python is a general purpose interpreted, interactive, object-oriented, high-level programming language.
- Current release version is version 2.6.? available for UNIX, PC and Mac. Version 3 is under development.

What is Python

- Python is a high-level, interpreted, interactive and object oriented-scripting language, designed to be highly readable, commonly uses English keywords.
  - Python is Interpreted: This means that it is processed at runtime by the interpreter and you do not need to compile your program before executing it. This is similar to PERL and PHP.
  - Python is Interactive: This means that you can actually sit at a Python prompt and interact it directly to write your programs.
  - Python is Object-Oriented: This means that Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
  - Python is Beginner's Language: Python is a great language for the beginner programmers and supports the development of a wide range of applications.

Python Features

- Feature highlights include:
  - Easy-to-learn: Python has relatively few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language in a relatively short period of time.
  - Easy-to-read: Python code is clearly defined and if well written visually simple to read and understand.
  - Easy-to-maintain: Python's success is that its source code is fairly easy-to-maintain.
  - A broad standard library: One of Python's greatest strengths is the bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
  - Interactive Mode: Support for an interactive mode in which you can enter results from a terminal right to the language, allowing interactive testing and debugging of snippets of code.

Python Features

- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Database Aware: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable: Python provides a better structure and support for large programs than shell scripting.

Important Features

- Apart from the above mentioned features, Python has a big list of important structural features that make it an efficient programming tool, few are listed below:

  - Built-in high level data types: strings, lists, dictionaries, etc.
  - The usual control structures if, if-else, if-elif-else, while plus a powerful (for) iterator.
  - It can be used as a scripting language or can be compiled to byte-code for building large applications.
  - Supports automatic garbage collection.
  - It can be easily integrated with Fortran, C, C++, CORBA, and Java, etc……

Getting Python & Help

- Getting Python:
  - The most up-to-date and current source code, binaries, documentation, news, etc. is available at the official website of Python:Python Official Website : http://www.python.org/
- Documentation
  - You can download the Python documentation from the following site. The documentation is available in HTML, PDF, and PostScript formats: http://docs.python.org/index.html
- Tutorial
  - You should definitely check out the tutorial on the Internet at: http://docs.python.org/tutorial/.

Running Python

- There are three different ways to start Python:
  - (1) Interactive Interpreter: You can enter python and start coding right away in the interactive interpreter by starting it from the command line. You can do this from Unix, DOS, or any other system which provides you a command-line interpreter or shell window.

    >>>
  - 2) Script from the Command-line: A Python script can be executed at command line by invoking the interpreter on your application, as in the following:

```
python  script.py       # Unix/Linux
```

Running Python

- (3) Integrated Development Environment (IDE): You can run Python from a graphical user interface (GUI) environment. All you need is a GUI application on your system that supports Python.

http://wiki.python.org/moin/IntegratedDevelopmentEnvironments

First Program

- Interactive mode:
  - Invoking the interpreter without passing a script file as a parameter brings up the following prompt:

```
computer# python
Python 2.6.2 (r262:71600, Apr 16 2009, 09:17:39)
[GCC 4.0.1 (Apple Computer, Inc. build 5250)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

  - Type the following text to the right of the Python prompt and press the Enter key:

```
>>> print "Hello, Python!";
```

  - This will produce following output:

```
Hello, Python!
```

First Program

- Script Mode:
  - Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active. All python files will have extension .py.
  - Example test.py file contains.

#!/usr/bin/python

print "Hello, Python!";

  - Here I assumed that you have Python interpreter available in the /usr/bin directory. Now try to run this program as follows:

computer# chmod +x test.py # This is to make file executable

computer# python test.py

  - This will produce following output:

Hello, Python!

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).Python does not allow punctuation characters such as @, $, and % within identifiers.

- Python is a case sensitive programming language. Thus Variable and variable are two different identifiers in Python.

Python Reserved Words

- The following list shows the reserved words in Python. These reserved words may not be used as constant or variable or any other identifier names. Reserved words contain lowercase letters only.

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

Lines and Indentation

- One of the most prominent features of Python is the fact that there are no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. Example:

```
if True:
        print "True”
else:
        print "False"
```

Lines

- Multi-Line Statements: Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example:

```
total = item_one + \
        item_two + \
        item_three
```

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example:

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday' ]
```

- line containing only whitespace, possibly with a comment, is known as a blank line, and Python totally ignores it.

Lines

- Quotations in Python: Python accepts single ('), double (") and triple ('" or """) quotes to denote string literals, as long as the same type of quote starts and ends the string. The triple quotes can be used to span the string across multiple lines. For example, all the following are legal:

word = 'word’

sentence = “This is a sentence.”

paragraph = ‘’’ This is a paragraph. It is made up of multiple lines and sentences.’’’

- Comments in Python: A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the physical line end are part of the comment, and the Python interpreter ignores them.

```
# First comment
print "Hello, Python!";  # second comment
```

Lines

- Multiple Statements on a Single Line: The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block. Here is an example:

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

- Multiple Statement Groups called Suites: Groups of individual statements making up a single code block are called suites. Compound or complex statements, such as “if”, “while”, “def”, and “class”, are those which require a header line and a suite. Header lines begin the statement and terminate with a colon (:) are followed by one or more lines form the suite. Example:

```
if expression :
   suite
elif expression :
   suite
else :
   suite
```

Example Python code

- Following is the example having various statement blocks: Don't try to understand logic or different functions used. Just make sure you see the various indented blocks.

```
#!/usr/bin/python
import sys
file_finish = 'file_finish'
file_text = ''
try:  # open file stream
   file = open(file_name, "w")
except IOError:  # come here on IO Error
   print "There was an error writing to", file_name
   sys.exit()
print "Enter '", file_finish,
print "' When finished"
while file_text != file_finish:
   file_text = raw_input("Enter text: ")
   if file_text == file_finish:  # close the file
                file.close
                break
   file.write(file_text)
   file.write("\n")
file.close()
```

The Python Standard Library

- The "Python standard library" contains several different kinds of components. It contains data types that would normally be considered part of the "core" of a language, such as numbers, strings, lists etc….

- The library also contains built-in functions and objects that can be used by all Python code without the need of an import statement. http://docs.python.org/library/stdtypes.html

- The bulk of the library (the good stuff) consists of a collection of modules which are accessed using the import statement. http://docs.python.org/library/index.html

- The standard library is huge – this is the power of Python……

Variables and Data Types

- Python variables do not have to be explicitly declared to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

```
#!/usr/bin/python
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
```

Assignments

- **Multiple Assignment:**
- **You can also assign a single value to several variables simultaneously. For example:**

a = b = c = 1

- Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location.
- You can also assign multiple objects to multiple variables. For example:

a, b, c = 1, 2, "john"

- Here two integer objects with values 1 and 2 are assigned to variables a and b, and one string object with the value "john" is assigned to the variable c.

Data Types

- The data stored in memory can be of many types. For example, a persons age is stored as a numeric value and the address is stored as alphanumeric characters.
- Python has some standard types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types:
  - Number
  - String
  - List (entries enclosed in [ ], list methods available)
  - Tuple (comma separated values of possible different types).
  - Dictionary (un-ordered, key:value sequences in [ ]

Numbers

- Number data types store numeric values. They are immutable data types, which means that changing the value of a number data type results in a newly allocated object. Number objects are created when you assign a value to them.
- Python supports four different numerical types:

  - int (signed integers) = C long precision
  - long (long integers [can also be represented in octal and hexadecimal]) unlimited precision
  - float (floating point real values) = C double precision
  - complex (complex numbers) = C double precision

Number Examples

- Here are some examples of numbers:

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-472188598529L	70.2-E12	4.53e-7j

- Python allows you to use a lowercase L with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floatingpoint numbers denoted by a + bj, where a is the real part and b is the imaginary part of the complex number.

Number conversions

- Python converts numbers internally in an expression containing mixed types to a common type for evaluation. But sometimes, you'll need to convert a number explicitly from one type to another to satisfy the requirements of an operator or function parameter.
  - Type int(x) to convert x to a plain integer.
  - Type long(x) to convert x to a long integer.
  - Type float(x) to convert x to a floating-point number.
  - Type complex(x) to convert x to a complex number with real part x and imaginary part zero.
  - Type complex(x, y) to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

Number Functions

- Some are some built in functions. http://docs.python.org/library/stdtypes.html#numeric-types-int-float-long-complex
- Some need to be added via the import statement. http://docs.python.org/library/math.html

Example:

```
Import math
x=[1,2,3,4,5]
math.fsum(x)
15.0
```

Strings

- Strings in Python are identified as a contiguous set of characters in between quotation marks.

- Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

- The plus (+) sign is the string concatenation operator, and the asterisk (*) is the repetition operator.

String Examples

```
#!/usr/bin/python
str = 'Hello World!'
print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 6th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

- The above code will produce following output:

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

String Formatting

- String Formatting Operator:
- One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the lack C's printf().
- Example:

```
#!/usr/bin/python

print "My name is %s and weight is %d kg!" % ('Zara', 21)
```

- This will produce following result:

```
My name is Zara and weight is 21 kg!
```

- Details on string formatting found at: http://docs.python.org/library/stdtypes.html#string-formatting-operations

Built in String Method

- Python includes following built in string method:
- Type help(str) or goto - http://docs.python.org/library/stdtypes.html#string-methods
- Example:

```
>>>'The happy cat ran home.'.upper()
'THE HAPPY CAT RAN HOME.'
>>> 'The happy cat ran home.'.find('cat')
10
>>> 'The happy cat ran home.'.find('kitten')
-1
>>> 'The happy cat ran home.'.replace('cat', 'dog')
'The happy dog ran home.'
```

String Module

- You can also use the equivalent and extra functions from the string module. http://docs.python.org/library/string.html

- Example:

```
>>> import string
>>> s1 = 'The happy cat ran home.'
>>> string.find(s1, 'happy')
4
```

Lists

- Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end-1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.
- Items may also be inserted or added to lists

List Examples

```
#!/usr/bin/python
list = [ 'abcd', 786 , 2.23, 'john', 70.200]
tinylist = [123, 'john']
print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd to 4th
print list[2:]      # Prints elements starting from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```

- The above code will produce following output:

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Built in List Functions and Methods

Python List functions

cmp(list1, list2)	*Compares elements of both lists.*
len(list)	*Gives the total length of the list.*
max(list)	*Returns item from the list with max value.*
min(list)	*Returns item from the list with min value.*
list(seq)	*Converts a tuple into list.*

Python list methods

list.append(obj)	*Appends object obj to list*
list.count(obj)	*Returns count of how many times obj occurs in list*
list.extend(seq)	*Appends the contents of seq to list*
list.index(obj)	*Returns the lowest index in list that obj appears*
list.insert(index, obj)	*Inserts object obj into list at offset index*
list.pop(obj=list[-1])	*Removes and returns last object or obj from list*
list.remove(obj)	*Removes object obj from list*
list.reverse()	*Reverses objects of list in place*
list.sort([func])	*Sorts objects of list, use compare func if given*

List Functions and Method Examples

items = [111, 222, 333]

>>> items[111, 222, 333]

•To add an item to the end of a list, use:

>>> items.append(444)

>>> items[111, 222, 333, 444]

•To insert an item into a list, use:

>>> items.insert(0, -1)

>>> items[-1, 111, 222, 333, 444]

•You can also push items onto the right end of a list and pop items off the right end of a list with append and pop.

>>> items.append(555)

>>> items[-1, 111, 222, 333, 444, 555]

>>> items.pop()

555

>>> items[-1, 111, 222, 333, 444]

Tuples

- A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]), and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only lists.**

Dictionary

- Python's dictionaries are hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs.
- Keys can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

Dictionary Examples

```
#!/usr/bin/python
dict = {}; dict['one'] = "This is one"; dict[2]     = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print dict               # Prints complete dictionary
print dict['one']      # Prints value for 'one' key
print dict[2]          # Prints value for 2 key
print tinydict         # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

- *The above code will produce following output:*

```
{'one':'This is one',2:'This is two'}
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

Data Type Conversions

- Sometimes you may need to perform conversions between the built-in types. To convert between types you simply use the type name as a function.
- There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Conversions

int(x)Converts x to an integer.
long(x) Converts x to a long integer.
float(x) Converts x to a floating-point number.
complex(real [,imag]) Creates a complex number.
str(x) Converts object x to a string representation.
repr(x) Converts object x to an expression string.
eval(str) Evaluates a string and returns an object.
tuple(s) Converts s to a tuple.
list(s) Converts s to a list.
set(s) Converts s to a set.
chr(x) Converts an integer to a character.
unichr(x) Converts an integer to a Unicode character.
ord(x) Converts a single character to its integer value.
hex(x) Converts an integer to a hexadecimal string.
oct(x) Converts an integer to an octal string.

Operators

- What is an operator?
- Simple answer can be given using expression *4 + 5 is equal to 9. Here 4 and 5 are called operands and + is called operator.*

- *Python language supports following type of operators.*
  - *Arithmetic Operators*
  - *Comparision Operators*
  - *Logical (or Relational) Operators*
  - *Assignment Operators*
  - *Conditional (or ternary) Operators*

Arithmetic Operators

Python Arithmetic Operators: Assume a = 10 and b = 20

+ Addition - Adds values on either side of the operator: a + b will give 30

- Subtraction - Subtracts right hand operand from left hand operand: a - b will give -10

* Multiplication - Multiplies values on either side of the operator: a * b will give 200

/ Division - Divides left hand operand by right hand operand: b / a will give 2

% Modulus - Divides left hand operand by right hand operand and returns remainder: b % a will give 0

** Exponent - Performs exponential (power) calculation on operators: a**b will give 10 to the power 20

// Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. 9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

Comparison Operators

Python Comparison Operators: Assume a = 10 and b = 20 then:

== Checks if the value of two operands are equal or not, if yes then condition becomes true. (a == b) is not true.

!= Checks if the value of two operands are equal or not, if values are not equal then condition becomes true. (a != b) is true.

> Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. (a > b) is not true.

< Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. (a < b) is true.

>= Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. (a >= b) is not true.

<= Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. (a <= b) is true.

Logical Operators

Python Logical Operators: There are following logical operators supported by Python language Assume variable a = 10 and b = 20 then:

and Called Logical AND operator. If both the operands are true then then condition becomes true. (a and b) is true.

or Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true. (a or b) is true.

not Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

Assignment Operators

Python Assignment Operators: Assume a = 10 and b = 20 then:

= Simple assignment operator, Assigns values from right side operands to left side operand: c = a + b will assign value of a + b into c

+= Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand: c += a is equivalent to c = c + a

-= Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand c -= a is equivalent to c = c - a

*= Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand c *= a is equivalent to c = c * a

Assignment operators

/= Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand c /= a is equivalent to c = c / a

%= Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand c %= a is equivalent to c = c % a

**= Exponent AND assignment operator, Performs exponential (power) calculation on operators and assign value to the left operand c **= a is equivalent to c = c ** a

//= Floor Dividion and assigns a value, Performs floor division on operators and assign value to the left operand c //= a is equivalent to c = c // a

If, Eles, Elif

- The if statement of Python is similar to that of other languages.
- The if statement contains a logical expression using which data is compared, and a decision is made based on the result of the comparison.
- The syntax of the if statement is:

if expression:

statement(s)

Note: In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

If, Else, Elif

- An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a false value.The *else statement is an optional statement and there could be at most only one else statement following an if .*
- The elif statement allows you to check multiple expressions for truth value and execute a block of code as soon as one of the conditions evaluates to true. Like the else, the elif statement is optional. However, unlike else, for which there can be at most one statement, there can be an arbitrary number of elif statements following an if.

Example

```
if expression1:
        statement(s)
elif expression2:
        statement(s)
elif expression3:
        statement(s)
else:
        statement(s)
```

While loop

- The while loop is one of the looping constructs available in Python. The while loop continues until the expression becomes false. The expression has to be a logical expression and must return either a *true or a false value*
- *The syntax of the while look is:*

while expression:

    statement(s)

Example:

```
#!/usr/bin/python
count = 0
while (count < 9):
        print 'The count is:', count
        count = count + 1
print "Good bye!"
```

Infinite loop!

- Following loop will continue till you enter CNTL+C at the command prompt:

```
#!/usr/bin/python
var = 1
while var == 1 :  # This constructs an infinite loop
        num = raw_input("Enter a number  :")
        print "You entered: ", num
print "Good bye!"
```

For Loop

- The for loop in Python has the ability to iterate over the items of any sequence, such as a list or a string.
- The syntax of the loop look is:

```
for iterating_var in sequence:
    statements(s)
```

- For a conventional loop through index system (ie., same as Fortran do I = 1, 10 or C for (i=1; I =<10;++) { use for I in range(1,n+1):
- NOTE the need to go 1 more

For Loop example

```
#!/usr/bin/python
for letter in 'Python':     # First Example
            print 'Current Letter :', letter
fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
            print 'Current fruit :', fruit
print "Good bye!”
```

This will produce following output:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

Another For loop

- Iterating by Sequence Index:
- An alternative way of iterating through each item is by index offset into the sequence itself:

```
#!/usr/bin/python
fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
        print 'Current fruit :', fruits[index]
print "Good bye!"
```

This will produce following output:

```
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

- Here we took the assistance of the len() built-in function, which provides the total number of elements in the tuple as well as the range() built-in function to give us the actual sequence to iterate over.

Break

- The *break Statement:*
- *The break statement in Python terminates the current loop and resumes execution at the next statement, just like the break in C.*
- *The most common use for break is when some external condition is triggered requiring a hasty exit from a loop.*
- *The break statement can be used in both while and for loops.*

```
#!/usr/bin/python
for letter in 'Python':     # First Example
        if letter == 'h':
                break
        print 'Current Letter :', letter
print "Good bye!”
```

This will produce following output:

```
Current Letter : P
Current Letter : y
Current Letter : t
Good bye!
```

Continue

- The continue statement in Python returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
- The continue statement can be used in both *while and for loops.*

```
#!/usr/bin/python
for letter in 'Python':     # First Example
        if letter == 'h':
                continue
        print 'Current Letter :', letter
print "Good bye!"
```

This will produce following output:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Good bye!
```

Summary

- Today we looked at:
  - History
  - Python features
  - Getting Python and help
  - Modes of running Python
  - Basics of Python scripting
  - Variables and Data types
  - Operators
  - Conditional constructs and loops

12.010 Computational Methods of Scientific Programming
Fall 2011