What is Python?

- Python is a high-level, interpreted, interactive and object-oriented scripting language. Python was designed to be highly readable which uses English keywords frequently where as other languages use punctuation and it has fewer syntactical constructions than other languages.

Python is Interpreted, Interactive and Object Oriented:

- **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 with the interpreter 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 derived from:

- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and UNIX shell and other scripting languages.

Python Features:

- **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 much more clearly defined and visible to the eyes.
- **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.
- **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.
- **Databases:** 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.
- Support for functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be **compiled to byte-code** for building large applications.
- Very high-level dynamic data types and supports **dynamic type checking**.
- Supports **automatic garbage collection**.
- It can be easily **integrated with C, C++, COM, ActiveX, CORBA and Java**.

Installing Python

Unix & Linux Installation:

Here are the simple steps to install Python on Unix/Linux machine.

- Download source code form http://www.python.org/download/
- Extract files.
- Editing the *Modules/Setup* file if you want to customize some options.
- run ./configure script
- make
- make install

This will install python in a standard location */usr/local/bin* and its libraries are installed in */usr/local/lib/pythonXX* where XX is the version of Python that you are using.

- **Setting Path In the csh shell:**

  type setenv PATH "$PATH:/usr/local/bin/python" and press Enter.

- **Setting Path In the bash shell (Linux):**

  type export PATH="$PATH:/usr/local/bin/python" and press Enter.

- **Setting Path In the sh or ksh shell:**

  type PATH="$PATH:/usr/local/bin/python" and press Enter.

Windows Installation:

Here are the steps to install Python on Windows machine.

- Download source code form http://www.python.org/download/
- Run the downloaded file by double-clicking it in Windows Explorer. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you're ready to roll!
- Go to Environment Variable tab in My Computer's Property and edit the value of PATH variable (value must be path of the python directory).

Write code in python:

Write following code in command prompt so that you will be getting python prompt:

```
$ python
Python 2.7.9 (#1, Dec 10 2014, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Now type

```
print "TYPE";
```

```
Output: TYPE
```

Creating Scipt file for python:

- Open noepad and write the above print statement it that. Save the file as "first.py".
- Open command prompt and move upto the location of your python file (i.e. d:\python\myprog). To execute that script file, write either ***python first.py*** or ***first.py***

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, **Manpower** and **manpower** are two different identifiers in Python.

Here are rules of identifier naming convention for Python:

- Class names start with an uppercase letter and all other identifiers with a lowercase letter.
- Starting an identifier with a single leading underscore indicates by convention that the identifier is meant to be private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

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. All the Python keywords 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

Indenting is mandatory:

- Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.
- Use 4 spaces which will be considered as indenting.

Example:

```
var  = 10
if (var == 10) :
    print "Values is 10"
else :
    print "Values is Not 10"

Output:
Values is 10
```

Multi-Line Statements:

In python program, a single instruction can be written in multi line as describe below.

```
a = 10
b = 20
c = 30
total = a + \
        b + \
        c
print total

Output:
60
```

Comments in Python:

Method-1: Single line comment

(Hash) is used for giving comment in python.

```
#This is first program of python

Print “BAPS”;      #This is for printing

Output:
BAPS
```

Method- 2: Multi line comment

```
"""This is also
another type of comment"""
```

Taking Input from User:

raw_input() function is used to take input in python. But the input entered by user is bydefault considered as string data.

```
a = int(raw_input("Enter A : "));  #int is used to convert
                                   string into integer
b = raw_input("Enter B : ");
c = raw_input("Enter C : ");

total = a + int(b) + int(c)        #int is used to convert
                                   string into integer
print total                        #print total
```

input() function is used to take data from user. This data is considered as integer data.

```
b = input("Enter B : ");
c = input("Enter C : ");
total = c + b
print total
```

Commandline :

```
import sys                    #Add package system

a = int(sys.argv[1])          #first argument is argv[1]
b = int(sys.argv[2])          #argv[0] refers program name

print a + b


void main(int argc, char *argv[])
```

Variables in Python:

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.

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
```

Assigning values to variables:

```
a = b = c = 1          # Adding same values to all va
a, b, c = 1, 2, "john"
```

String – immutable type

It can be represented either by single quote or by double quote, either single time or by three times as shown below.

A=’BAPS’ #One time single quote
print A
Output: BAPS
A=”BAPS” #One time double quote
print A
Output: BAPS

A=’’’BAPS’’’ #Three times single quotes
print A
Output: BAPS

A=”””BAPS””” #Three times double quotes
print A
Output: BAPS

A=”JAY””BAPS” #One time double quotes
print A
Output: JAYBAPS

A=”JAY”””””BAPS””” #Combination of one & three times double quotes
print A
Output: JAYBAPS

A=’’’’’’BAPS’
print A
Output: BAPS

String Operations:

Equality: Two strings are equal if and only if they have exactly the same contents, meaning that they are both the same length and each character has a one-to-one positional correspondence.

```
str1 = "BAPS"
str2 = "BAPS"

if str1 == str2:
    print str1, "and", str2, "are same"
else:
    print str1, "and", str2, "are not same"
```

Output:

```
BAPS and BAPS are same
```

```
str1 = "BAPS"

if str1 == 'BAPS':
    print str1, "and", '\'BAPS\'', "are same"
else:
    print str1, "and", '\'BAPS\'', "are not same"
```

Output:

```
BAPS and 'BAPS' are same
```

```
if "BAPS" == 'BAPS':
    print "\"BAPS\"", "and", '\'BAPS\'', "are same"
else:
    print "\"BAPS\"", "and", '\'BAPS\'', "are not same"
```

Output:

```
"BAPS" and 'BAPS' are same
```

is operator for checking equality using memory location & identity

Note: In ***is*** operator, for numbers it works perfectly only up to number 256. For string, no length constraints are specified but if string contains special character then ***is*** will not work perfectly because due to special character memory location differes.

```
if str1 is str1:
    print "Both are same"
else:
    print "Both are not same"
```

Output:

```
Both are same
```

```
if str1 is str2:
    print "Both are same"
else:
    print "Both are not same"
```

Output:

```
Both are same
```

```
if str1 is "BAPS":
    print "Both are same"
else:
    print "Both are not same"
```

Output:
Both are same

```
if 'BAPS' is str1:
    print "Both are same"
else:
    print "Both are not same"
```

Output:
Both are same

```
X=225
Y=225

>>> X is Y
Output: True

>>> x is 225
Output: True

X = 257
Y = 257

>>> X is Y
Output: False


>>> print str1 * 5 #print str1 for 5 times
Output: BAPSBAPSBAPSBAPSBAPS
```

in operator for checking existence of substring

```
#This will check for the existence of second string in
#first string
str1 = "LDRP-ITR"
str2 = "IT"

if str2 in str1:
    print str2, "exists in", str1

Output: IT exists in LDRP-ITR
```

Accessing individual Character of String

```
str12="This is LDRP"
i=0
while(i<len(str12)):
    print str12[i],   # ‘,’ used to print in the same line
    # print str12[i],; str12[i] – This will also work
    i=i+1
print "\n"

Output: T h i s   i s   L D R P
```

String Slicing:

Syntax: str[startIndex : endIndex : Increment]

Element is considered from startIndex and move up to endIndex and increment factor is same as third parameter.
If startIndex is not provided by default starts from first element (index 0)
If endIndex is not provided by default moves up to last element (length – 1)
If Icrement is not provided by default increment is of 1

```
Positive index      0  1  2  3  4  5  6  7
String Elements     L  D  R  P  -  I  T  R
Negative Index     -8 -7 -6 -5 -4 -3 -2 -1
```

If the increment (third parameter) is positive then prints from left to right

```
str1 = ‘LDRP-ITR’

print "Slice of String : ", str1[1:4:1]      Output: DRP

print "Slice of String : ", str1[0:-1:2]     Output: LR-T

print "Slice of String : ", str1[3::1]       Output: P-ITR

print "Slice of String : ", str1[:]          Output: LDRP-ITR

print "Slice of String : ", str1[4:1]        Output:
```

Note : No output in above mentioned case because start index is on right side of end index so left to right printing is not possible.

If the increment (third parameter) is negative then prints from right to left

```
str1 = 'LDRP-ITR'

print "Slice of String : ", str1[4:1:-1]     Output: -PR

print "Slice of String : ", str1[-1:0:-2]    Output: RIPD

print "Slice of String : ", str1[3::-1]      Output: PRDL

print "Slice of String : ", str1[::-3]       Output: R-D

print "Slice of String : ", str1[1:4:-1]     Output:
```

Note : No output in above mentioned case because start index is on left side of end index so right to left printing is not possible.

String Functions (Refer program stringFunction.py)

Function	Significance	Example
isalnum()	Checks for alphanumeric values	str1.isalnum()
isalpha()	Checks for alphabets values	str1.isalpha()
isdigit()	Checks for digits	str1.isdigit()
isspace()	Checks for space	str1.isspace()
istitle()	Checks for title case	str1.istitle()
islower() / isupper()	Checks for lower / upper case	str1.islower()
title()	Converts given string to title case (words are separated by non alphabetic character)	"abc@xyz".title() str1.totle()
capitalize()	Converts given string to title case (Whole string is a single word)	"a b".capitalize()
lstrip()	Removes space or character from left side	str1.lstrip()
rstrip()	Removes space or character from right side	str1.rstrip()
strip()	Removes space or character from both sides	str1.strip('A')
ljust(SIZE)	Set width of length SIZE and set string alignment to left	str1.ljust(10)
rjust(SIZE)	Set width of length SIZE and set string alignment to right	str1.rjust(10)
center(SIZE)	Set width of length SIZE and set string	str3.center(10)

	alignment to center	
join(SEQUENCE)	Joins elements of sequence by a character specified	j = " ".join(seq)

```
str1 = "LDRP ITR"
str2 = "a A"

if str1.isalnum():  #This is false since it contains space
    print "String1 is Alpha numeric"

if str2.isalnum():
    print "String2 is Alpha numeric"


if str1.isalpha():#This is false since it contains space
    print "String1 is Alphabet"

if str2.isalpha():
    print "String2 is Alphabet"


if str1.isdigit():
    print "String1 is Digit"

if str2.isdigit():
    print "String2 is Digit"

str3="LDRP ITR"
str4=" "

if str3.isspace():
    print "String3 is space"

if str4.isspace():  #This is True
    print "String4 is space"

str3 = "23This I23S Xyz"
str4 = "this2that"

if str3.istitle():  #This   is   True   because   all   first
                    #characters are Title case
    print str3, "is Title case"
else:
    print str3, "is not Title case"

if str4.islower():
    print str4, "is lower case"
```

```
else:
    print str4, "is not lower case"


print "Title Case : ", "abc@xyz".title()
#Words are seperated by non alphabetic character
```
Output: Abc@Xyz

```
print "Title Case : ", "This is python".title()
#Words are seperated by non alphabetic character
```
Output: This Is Python

```
print "Capitalize Case : ", "This is python".capitalize()
#Whole string is a single word
```
Output: This is python

```
str3 = "Congratulations"
str4 = "on"
print "There are", str3.count(str4), "Occurances of", "\"",
str4, "\"", "in", str3
```
Output: There are 2 Occurances of " on " in Congratulations

```
str3 = "\n\tThis is LDRP\t\n"
print str3

print "LSTRIP : ", str3.lstrip()       #Removes space from
left
print "RSTRIP : ", str3.rstrip()       #Removes space from
right
print "STRIP : ", str3.strip()      #Removes space from
                                     #both sides

str3="i.ipri"
print "STRIP : ", str3.strip('i')      #Removes character
                                      #from both sides
```
Output: .ipr

```
print "Left Justified :\"", str3.ljust(10), "\""
#Create 10 blocks and print from left
```
Output: " i.ipri   "

```
print "Right Justified :\"", str3.rjust(10), "\""
#Create 10 blocks and print from right
```
Output: "   i.ipri "

```
print "Center Justified :\"", str3.center(10), "\""
```

```
#Create 10 blocks and print in center
Output: "   i.ipri   "


seq=['1a','2b',"3",'4','5']
j = " ".join(seq)
#Joins string elements, integers are not valid
print j
Output: 1a 2b 3 4 5
```

len() to know the length of string

```
str12 = 'BAPS'
print len(str12)

Output: 4

str12 = 'BA\bPS'
print len(str12)

Output: 5      #Here \b is considered as a single character
```

Note: All the back slash character sequences (\b, \t, \n, \a, etc.) are considered as a single character.

Back slash character sequences:

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F

List – Mutable type

A list in Python is an ordered group of items (or elements). It is a very general structure, and list elements don't have to be of the same type.

There are two different ways to make a list in Python. The first is through assignment ("statically"), the second is using list comprehensions ("actively").

```
l = [0]*6
print l
Output: [0, 0, 0, 0, 0, 0]

l = [[0]*3] * 3
print l
Output: [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

l = [[0]*3] * 3
l[1] = 1
print l
Output: [[0, 0, 0], 1, [0, 0, 0]]
```

```
li = [[[0]*3]*3] * 3
print li
```
Output: [[[0, 0, 0], [0, 0, 0], [0, 0, 0]], [[0, 0, 0], [0, 0, 0], [0, 0, 0]], [[0, 0, 0], [0, 0, 0], [0, 0, 0]]]

```
li = [[[0]*3]*3] * 3
li[0]=1
print li, "\n\n"
```
Output: [1, [[0, 0, 0], [0, 0, 0], [0, 0, 0]], [[0, 0, 0], [0, 0, 0], [0, 0, 0]]]

```
li = [[[0]*3]*3] * 3
li[1][2]=1
print li, "\n\n"
```
Output: [[[0, 0, 0], [0, 0, 0], 1], [[0, 0, 0], [0, 0, 0], 1], [[0, 0, 0], [0, 0, 0], 1]]

```
li = [[[0]*3]*3] * 3
li[0][2][1]=1
print li
```
Output: [[[0, 1, 0], [0, 1, 0], [0, 1, 0]], [[0, 1, 0], [0, 1, 0], [0, 1, 0]], [[0, 1, 0], [0, 1, 0], [0, 1, 0]]]

```
l = [[[0]*3]*3 for i in range(3)]  #This is not reference
l[1][2]=1
print l
```
Output: [[[0, 0, 0], [0, 0, 0], [0, 0, 0]], [[0, 0, 0], [0, 0, 0], 1], [[0, 0, 0], [0, 0, 0], [0, 0, 0]]]

Joining / Modifying / Adding List

Using + operator

```
l1 = [1, "LDRP", 2]
l2 = ["Python", 3.7, 'I']

l3 = l1 + l2    #Joining two lists
print l3
```

Output: [1, 'LDRP', 2, 'Python', 3.7, 'I']

Using append() method – It takes only 1 argument

```
l1.append("XYZ")
```

```
print l1
```
Output: [1, 'LDRP', 2, ‘xyz’]

```
l1.append(l2)
print l1
```

Output: [1, 'LDRP', 2, ['Python', 3.7, 'I']]

Note: The list passed as parameter (l2) will be appended to the original list (l1) as nested list, not as separate element.

Using extend() method – It takes only 1 argument

```
l1.extend(“XYZ”)    #passed string – each character added
print l1
```
Output: [1, 'LDRP', 2, ‘x’, ’y’, ’z’]

```
l1.extend([“XYZ”])  #passed list – whole string added
print l1
```
Output: [1, 'LDRP', 2, ‘xyz’]

Indexing & Slicing in list

```
l4 = [1, "abc", 3.9]
print l4[0], l4[1][1]
```
Output: 1 b

Item Assignment / Replacement in List

```
l4 = [1, "abc", 3.9]
print "List l4 : " , l4
```
Output: List l4 : [1, 'abc', 3.9]

```
l4[1] = 10
print "List l4 : " , l4
```
Output: List l4 : [1, 10, 3.9]

```
l4[1:4] = [2, "xyz", 5.75, 10, "abc"]
print "List l4 : ", l4
```
Output: List l4 : [1, 2, 'xyz', 5.75, 10, 'abc']

```
l4[1:3] = [3.7, 4.5, "PQR", 'QPR']
```

```
print l4
```
Output: [1, 3.7, 4.5, 'PQR', 'QPR', 5.75, 10, 'abc']

Copying List without reference (separate memory location)

```
original = [1, 'element', []]
copy = original[:]
original.append("Original Append")
print "Copy : ", copy
print "Original : ", original, "\n"
```

Output:
Copy : [1, 'element', []]
Original : [1, 'element', [], 'Original Append']

Copying List with reference (same memory location)

```
original = [1, 'element', []]
copy = original
original.append("Original Append")
print "Copy : ", copy
print "Original : ", original, "\n"
```

Output:
Copy : [1, 'element', [], 'Original Append']
Original : [1, 'element', [], 'Original Append']

Appending element in nested list:

```
original = [1, 'element', []]
original[2].append(“xyz”)
print original
```

Output: [1, ’element’, [‘xyz’]]

Accessing elements of nested list / slicing

```
Copy : [1, 'element', ['1',234,456], 'Original Append']

print "ANS : ", copy[-1:-4:-2]
print “Nested : “, copy[-2][-3:-1]
```

Output:
ANS: ['Copy Append', 'element']
Nested: ['1',234,456]

Special Case:

```
print ['1'][0] * 3 #First Index is List by default and
                    second indicates index of that list
Output: 111

print ['123'][0][1] * 3
Output: 222
```

Comparing List using = = operator :

```
t1 = [1, "ab", 98.76, 'x', [10,2], [3,1]]
t2 = [1, "ab", 98.76]

if t1 == t2:        #Returns False
    print "Both list are same"

if t2 == [1, 'ab', 98.76]:    #Returns True
    print "Both list are same"
```

Sorting List

```
t1 = [1, "ab", 98.76, 'x', [10,2], [3,1], [10,4]]

t1.sort()   #sort inline, retunrs None
print "Sort : ", t1
Output: Sort :  [1, 98.76, [3, 1], [10, 2], [10, 4], 'ab',
'x']

t = sorted(t1)  #Create another object for sorted list,
                 original will be as it is


t = sorted("BCD abc".split(), key=str.lower)
print "String Sorted : ", t
Output: String Sorted :  ['abc', 'BCD']
```

Note: Second parameter is the key which will be evaluated before sorting is applied. All the elements of list converted into lower case then sorting is done.

Inserting Element:

```
#insert element @ specific location
ins = [1,2,3]
ins.insert(1,3) # Syntax: insert(index, value), Adding
                   element 4 @ 1st location
ins.insert(1,['a','b'])
print "Insert : ", ins
```

Output: Insert : [1, ['a', 'b'], 3, 2, 3]

Removing Element:

```
t = [1,2,3]
```

#Removing Element – **Using pop()**

```
print t.pop(1)  #Remove and returns item at specified index
```

Output: 2

```
print t.pop()   #Remove and returns item at Last index by
                 default
```

Output: 3

#Removing Element - **Using remove()**

```
#Removing specific item from list
ins.remove(3)           #Remove first item from left to
                         right if duplicate item exist
print ins
```

Check existence of element using *in* operator

```
if “xyz” in t:
    print "Element Exist"
else:
    print "Element Not Exist"
```

#To access all element of list

```
for word in t:
    print word
```

#Getting position of the specific element from the list

```
print "index is : ", ins.index(“xyz”)
```

```
#count existance of specific element in list
print "Occurance : ", ins.count(“xyz”)


#Reverse the list inline
ins.reverse()
print ins
```

Using List as Queue

To access list as queue (first in first out) import following module.

```
from collections import deque
```

Use deque() method to declare list as queue, use popleft() to pop leftmost element.

```
queue = deque([1,2,3,4,5])
queue.append(6)
queue.append(7)
val = queue.popleft();
print "POPLEFT : ", val
```

Output: POPLEFT : 1

Creating list using list Comprehension

```
l = [i**2 for i in range(1,5)]     #Creating list of square
```
Output: [1, 4, 9, 16]

```
#Creating pair of (x , y) - tuple
l=[(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
print l
```
Output: [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

```
l=[-3,-2,-1,0,1,2,3]
li = [x for x in l if x >= 0]
print li
```
Output: [0, 1, 2, 3]

```
li = [(x, x**3) for x in range(1, 8)]
print li
```
Output: [(1, 1), (2, 8), (3, 27), (4, 64), (5, 125), (6, 216), (7, 343)]

```
li = [str1.lower() for str1 in ["Abc", "xyz", "CDF", "fGh"]]
print li
```
Output: ['abc', 'xyz', 'cdf', 'fgh']

```
matrix = [[1,2,3],
          [4,5,6],
          [7,8,9]]
trans = [[row[i] for row in matrix] for i in range(3)]
print trans
```

Output: [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

Filter(), map(), reduce()

filter() :

filter(function, sequence) returns a sequence consisting of those items from the sequence for which function(item) is true. If sequence is a string or tuple, the result will be of the same type; otherwise, it is always a list.

```
def fun(x):
    if x % 3 == 0 or x % 5 == 0:
        return x

print "Filter : ", filter(fun, range(-12, 13))
```

Output: Filter : [-12, -10, -9, -6, -5, -3, 3, 5, 6, 9, 10, 12]

Note: Here filter makes a list of all x, for which function returns true. For 0, condition is true but returns 0 means false so 0 is not included in list (filter returns only non-zero values).

map()

map(function, sequence) calls function(item) for each of the sequence's items and returns a list of the return values.

```
def fun(x):
    if x % 3 == 0 or x % 5 == 0:
        return x


print "Map : ", map(fun, range(-12, 13))
```

Output: Map : [-12, None, -10, -9, None, None, -6, -5, None, -3, None, None, 0, None, None, 3, None, 5, 6, None, None, 9, 10, None, 12]

reduce()

reduce(function, sequence) returns a single value constructed by calling the binary function on the first two items of the sequence, then on the result and the next item, and so on. For example, to compute the sum of the numbers 1 through 10

```
def fun(x,y):
    #print "X : ", x, "Y : ", y
    return x+y

print "Reduce : ", reduce(fun, [1,2,3])

Output: Reduce :  6
```

Lambda:

The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name. These functions are throw-away functions, i.e. they are just needed where they have been created. Lambda functions are mainly used in combination with the functions filter(), map() and reduce().

Program

```
g = lambda x: x**2
print g(10)
```

Output:

100

```
def inc(n):
    #print "N :", n
    return lambda x: x + n

f=inc(1000)    #reference to inc function
g=inc(4)

print f(10), g(10), f(100)

print inc(10)(20) #first parameter is n, second is x
```

Output:
1010 14 1100
30

```
print "Filter : ", filter(lambda x: x%3==0, range(1,20))

print "Map : ", map(lambda x: x+10, range(1,20))

print "Reduce : ", reduce(lambda x,y : x+y, range(1,11))
```

Output:
Filter : [3, 6, 9, 12, 15, 18]
Map : [11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]
Reduce : 55
Nested Lambda:

```
def inc(n):
    return lambda x:  lambda y,z : x+ y + z + n

t = inc(10)
print t(5)(25,5)
```

Output:
45

Set:

- A set is an unordered collection of objects.
- Sets also cannot have duplicate members
- Any object that can be used as a dictionary key can be a set member. Integers, floating point numbers,tuples, and strings are hashable; dictionaries, lists, and other sets (except frozensets3) are not.
- Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.
- Curly braces or the set() function can be used to create sets.
- Note: to create an empty set you have to use set(), not {};

Program:

```
s = set ([123,'def'])


s.add ("SDF") #can have only one argument- added as single
               element

#s.add ('a', 200) is not possible
Print s
Output: set (['SDF', 123, 'def'])


s.update (('101', 200))   #For adding group of elements
Print s
Output: set ([200, 123, 'def', '101'])

s.update ((211,), "999","WER")
Print s
Output: set(['101', 'E', 200, 211, 'W', '9', 'R', 123, 'def'])

#Copy full list to other list using copy() constructor - different
 Address

scopy = s.copy()    #Copy full list, create separate copy
Print scopy
Output: set(['101', 'E', 200, 211, 'W', '9', 'R', 123, 'def'])

#Assign full list to other list - same address (reference)

s1 = s
print s1, "\n Address of Assigned Set : ", id(s1)
s1.add("Copy")
print "\nCopied List :", s1
print "Original List :", s

Output:
set (['101', 'E', 200, 211, 'W', '9', 'R', 123, 'def'])
Address of Assigned Set:  10666752

Copied List: set (['101', 'E', 200, 211, 'W', '9', 'R', 123, 'Copy',
'def'])
```

Original List: set (['101', 'E', 200, 211, 'W', '9', 'R', 123, 'Copy', 'def'])

Operation	Equivalent	Result
`len(s)`		cardinality of set *s*
`x in s`		test *x* for membership in *s*
`x not in s`		test *x* for non-membership in *s*
`s.issubset(t)`	`s <= t`	test whether every element in *s* is in *t*
`s.issuperset(t)`	`s >= t`	test whether every element in *t* is in *s*
`s.union(t)`	`s | t`	new set with elements from both *s* and *t*
`s.intersection(t)`	`s & t`	new set with elements common to *s* and *t*
`s.difference(t)`	`s - t`	new set with elements in *s* but not in *t*
`s.symmetric_difference(t)`	`s ^ t`	new set with elements in either *s* or *t* but not both
`s.copy()`		new set with a shallow copy of *s*

Operation	Equivalent	Result
`s.update(t)`	*s* |= *t*	return set *s* with elements added from *t*
`s.intersection_update(t)`	*s* &= *t*	return set *s* keeping only elements also found in *t*
`s.difference_update(t)`	*s* -= *t*	return set *s* after removing elements found in *t*
`s.symmetric_difference_update(t)`	*s* ^= *t*	return set *s* with elements from *s* or *t* but not both
`s.add(x)`		add element *x* to set *s*
`s.remove(x)`		remove *x* from set *s*; raises **`KeyError`** if not present
`s.discard(x)`		removes *x* from set *s* if present
`s.pop()`		remove and return an arbitrary element from *s*; raises **`KeyError`** if empty
`s.clear()`		remove all elements from set *s*

Program :

```
        engineers = set(['Rahul', 'Modi', 'soniya', 'arvind'])
        programmers = set(['soniya', 'Amit', 'Manmohan',
'arvind'])
        managers = set(['Modi', 'soniya', 'Manmohan', 'Obama'])
```

```
print "engineers",engineers

print "programmers",programmers

print "managers ",managers
```

Output :

engineers set(['Modi', 'soniya', 'arvind', 'Rahul'])

programmers set(['Amit', 'soniya', 'arvind', 'Manmohan'])

managers set(['Modi', 'Obama', 'soniya', 'Manmohan'])

```
employees = (engineers.union(programmers)).union(managers)
print employees
```

Output : set(['Manmohan', 'arvind', 'Modi', 'Amit', 'Obama', 'soniya', 'Rahul'])

```
engineering_management = engineers & managers            #
intersection
print "engineers & managers", engineering_management
```

Output : engineers & managers set(['Modi', 'soniya'])

```
fulltime_management =
(managers.difference(engineers)).difference(programmers)
print "(managers.difference(engineers)).difference(programmers) "
,fulltime_management
```

Output: (managers.difference(engineers)).difference(programmers) set(['Obama'])

```
sym_diff = managers.symmetric_difference (engineers)
          sym_diff_three = (managers.symmetric_difference
(engineers)).symmetric_difference (programmers)

print "managers.symmetric_difference(engineers) ->",sym_diff
         print
"(managers.symmetric_difference(engineers)).symmetric_difference(progr
ammers) ->",sym_diff_three
```

Output:

managers.symmetric_difference(engineers) -> set(['Manmohan', 'Obama', 'arvind', 'Rahul'])

(managers.symmetric_difference(engineers)).symmetric_difference(programmers) -> set(['Amit', 'Rahul', 'soniya', 'Obama'])

```
l = set(['Rahul'])
print "engineers.issuperset(l)",engineers.issuperset(l)     #
superset test
```

Output: engineers.issuperset (l) True

```
engineers.add('arvind') # will not add
print "engineers.add('arvind')",engineers
engineers.add('Arvind') # will add coz case sensitivity
print "engineers.add('Arvind')", engineers
```

Output :

engineers.add('arvind') set(['Modi', 'soniya', 'arvind', 'Rahul'])
engineers.add('Arvind') set(['arvind', 'Arvind', 'Modi', 'soniya', 'Rahul'])

```
l = set("abc")
l.update(engineers, "XYZ")
#print id(l), id(engineers) #different address
print "l is" , l , "\nengineers is -> ",engineers
print "engineers.issubset(l)",engineers.issubset(l)     # subset
test
```

Output:

l is set(['a', 'c', 'b', 'Arvind', 'arvind', 'X', 'Y', 'Modi', 'Z', 'soniya', 'Rahul'])

engineers is -> set(['arvind', 'Arvind', 'Modi', 'soniya', 'Rahul'])

engineers.issubset(l) True

```
print "engineers ->",engineers
engineers.discard('soniya')
print "engineers.discard('soniya')"
print "engineers ->",engineers
```

Output :

engineers -> set(['arvind', 'Arvind', 'Modi', 'soniya', 'Rahul'])

engineers.discard('soniya')

engineers -> set(['arvind', 'Arvind', 'Modi', 'Rahul'])

```
print "engineers ->",engineers
engineers.remove('Rahul')
print "engineers.remove('Rahul')"
print "engineers ->",engineers
```

Output:

engineers -> set(['arvind', 'Arvind', 'Modi', 'Rahul'])

engineers.remove('Rahul')

engineers -> set(['arvind', 'Arvind', 'Modi'])

```
print "engineers ->",engineers
print "engineers.pop()",engineers.pop()
print "engineers ->",engineers
```

Output :

engineers -> set(['arvind', 'Arvind', 'Modi'])

engineers.pop() arvind

engineers -> set(['Arvind', 'Modi'])

```
fs = frozenset([1,2,3,4])
print fs

s1 = set([fs, 4, 5, 6])
print s1
```

Output :

```
fs frozenset([1, 2, 3, 4])

s1 set([frozenset([1, 2, 3, 4]), 4, 5, 6])
```

```
f = set([1,5,7])
print "f <= fs",f <= fs
```

Output:

```
f <= fs False
```

```
fs.add(10)
print fs
```

```
Traceback (most recent call last):

  File "set_prac.py", line 165, in <module>

    fs.add(10)

AttributeError: 'frozenset' object has no attribute 'add'
```

Tuple:

- 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.

Program:

```
t = 1, 'abc', "XYZ", 23.45
print t

t[0]=3
print t
```

Output :

```
(1, 'abc', 'XYZ', 23.45)
Traceback (most recent call last):
  File " tuple.py", line 6, in <module>
    t[0]=3
TypeError: 'tuple' object does not support item assignment
```

```
t = t, '24','45', 67
print t[2][0]
```

Output:

4

```
t = t, (1,2)
print t
```

Output :

(((1, 'abc', 'XYZ', 23.45), '24', '45', 67), (1, 2))

```
t = ()
print t, len(t)
```

Output:

() 0

```
t = ('testing','ooo')

print t, len(t)
```

Output:

('testing', 'ooo') 2

```
t = 'testing',          #Single element of tuple
print t[0]
print "Tuple: ", t, "\nLength of Tuple : ", len(t)
```

Output:

Testing

Tuple: ('testing',)

Length of Tuple : 1

```
t = 1,'abc', 34.56  #tuple pacing
x,y,z = t           #unpacking - Same no. of operands are
there on both the sides
x,y = t[0:2:1]        #unpacking - Same no. of operands are
there on both the sides
print x, y
```

Output:

1 abc

Sequence unpacking requires the list of variables on the left to have the same number of elements as the length of the sequence. Note that multiple assignment is really just a combination of tuple packing and sequence unpacking.

Dictionaries:

- A dictionary in python is a collection of unordered values which are accessed by key.
- Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”.
- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples.

- **if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.**

- **You can’t use lists as keys**, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().
- The main operations on a dictionary are storing a value with some key and extracting the value given the key.
- If you store using a key that is already in use, the old value associated with that key is **forgotten**.

- The dict() constructor builds dictionaries directly from sequences of key-value pairs.

Program :

```
 d = {'city':'Paris', 'age':38, (102,1650,1601):    'A matrix
coordinate'}
print p
```

Output :

```
{'city': 'Paris', 'age': 38, (102, 1650, 1601): 'A matrix
coordinate'}
```

```
seq = [('city','Paris'), ('age', 38), ((102,1650,1601),'A
matrix coordinate')]
dict(seq)
print seq
```

Output:
[('city', 'Paris'), ('age', 38), ((102, 1650, 1601), 'A matrix coordinate')]

```
seq1 = ('a','b','c','d')
seq2 = [1,2,3,4]
d = dict(zip(seq1,seq2))
```

Output:

{'a': 1, 'c': 3, 'b': 2, 'd': 4}

```
d = {x: x**2 for x in (2, 4, 6)}
print d
```

Output :
{2: 4, 4: 16, 6: 36}

```
print "Keys : ", d.keys()
print "Value : ", d.values()
```

```
print "d[('a',1)] : ", d['city']
```

Output:

```
Keys :  ['city', 'age', (102, 1650, 1601)]
Value :  ['Paris', 38, 'A matrix coordinate']
d[('a',1)] :  Paris
```

```
#print 'b' in d     #Check for key existance
#print 38 in d        #Value existance can not be checked



print d
dic = {'101':0}
d.update(dic)
print d
d.update({'10':'xyz', '101' : 1, 'b':456 }) #if key exist,
modify its value
print d
d['10'] = 1000
```

Output:

```
{'city': 'Paris', 'age': 38, (102, 1650, 1601): 'A matrix
coordinate'}
{'city': 'Paris', 'age': 38, '101': 0, (102, 1650, 1601): 'A
matrix coordinate'}
{'city': 'Paris', 'b': 456, '10': 'xyz', 'age': 38, '101': 1,
(102, 1650, 1601): 'A matrix coordinate'}
```

```
print d

del d['10']
print d
del d          #delete dictionary - free memory
```

Output :

```
{'city': 'Paris', 'b': 456, '10': 1000, 'age': 38, '101': 1,
(102, 1650, 1601): 'A matrix coordinate'}
{'city': 'Paris', 'b': 456, 'age': 38, '101': 1, (102, 1650,
1601): 'A matrix coordinate'}
```

File:

Until now, you have been reading and writing to the standard input and output. Now, we will see how to play with actual data files. Python provides basic functions and methods necessary to manipulate files by default. You can do your most of the file manipulation using a file object.

Open:

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])

Parameters:

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, i.e., read, write, append, etc. 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. If negative, the buffer size is the system default(default behavior).

Program:

```
i = open("jay.txt").read()
print i
```

Output :
this is my file
so what do you want basically ?

```
str= i.read(2)
str1= i.read(4)
print str # first two characters
print str1 # it will print 4 characters but skip first two
```

Output :
th
is i

```
print i.tell() # tells the current position
i.seek(2) # change the current position
print i.tell()
```

Output:
6
2

```
for line in open("vb.txt", "r"):
    print line
```

Output :
this is my file

so what do you want basically?

```
i = open("vb.txt","r")
lines = i.readlines()# returns an array of lines
print lines
```

Output:
['this is my file\n', 'so what do you want basically?']

```
i = open("vb.txt","r")
single_line = i.readline() # return a single line
second_line = i.readline()
print single_line
```

```
print second_line
```

Output:
this is my file

so what do you want basically?

```
outputFileText = "xenesis 2015 "
open("vb.txt", "w").write(outputFileText)
i = open("vb.txt").read()
print i
```

Output :
xenesis 2015

```
appendtext = "Bau maja avse "
open("vb.txt", "w").write(appendtext)
i = open("vb.txt").read()
print i
```

Output:
xenesis 2015 Bahu maja avse

```
g = open("vb.txt").read().split()  #Create list of all the
words of file
print g
```

Output:
['xenesis', '2015', 'Bahu', 'maja', 'avse']

Exercise:
Write a program which replaces all the occurrences of specific word in file using python