Unit - IV

4.1 Basic Features of Python

Python is simple, easy to understand, high level programming language. This was developed by Guido Van Rossum during the period 1980 to 1990 at National Research Institute for Mathematics and Computer Science, Netherrlands. Python is a scripting language which does not create intermediate object code file while executing a program.

Features

- Python is an Interpreted Language. It doesnot generate intermediate code.
- Python is an Object-Oriented Language. It implements the object oriented concepts such as class, objects and inheritance.
- Python is an easy to understand and easy to write programming language..
- Python supports dynamic type checking. This allows the interpreter to assign the datatype of a variable at runtime based on the value that is assigned to that variable.
- Python is a portable language. This works in all operating system withiout need to make change in its code.
- Python is aScalable lanugage. This allows us to develop large programs and maintain them easily.

Installing - Windows

IDLE is Python's built-in Integrated Development Environment. This can be downloaded from the website www.python.org.

Steps to Install

- Download Python 2.7.14 MSI installer for windows.
- Start Installation by double click MSI Installer file.
- In Setup Wizard select installation folder for Python.
- To start Python select IDLE(Python GUI) from start menu.

Installing - Linux

To install Python in Linux OS the command's that are to be executed in the command line are given below.

```
$ sudo apt-get install idle
```

To start the Python application once the installation is completed the command needed to be typed in command line is idle..

```
$ idle
```

IDLE - Features

- Cross - Platform: IDLE can be executed on multiple OS.

- Interactive Interpreter: IDLE has the syntax highlighting and code completion functions as inbuilt options.
- Multi-Window: IDLE allows us to open multiple windows to type and execute multiple scripts at the same time.
- Basic text Operations: IDLE supports basic text operations such as cut, copy, paste and find.
- Debugger: IDLE supports break point option and execution of the program line by line. This allows us to examine the program for errors.
- Interactive Mode: In interactive mode the python script is typed directly in the interpreter window and it is executed.
- Script Mode: In script mode the script is typed and saved with .py extension and executed.

4.2 Variables and Strings

In Python variables are used to store the values that change during program execution. Python does not require its variables be defined before its use. Python assigns the datatype of the variable when a value is assigned to that variable.

Nameing Rules

- Starting letter of a variable must be A-Z or a-z or an underscore(_).
- The remaining letters can be character, underscore or nunmber (0-9).
- Keywords cannot be used as variable name.

Example

```
x = 10
y = 20
_sum = x + y
print "Result is ",_sum," by adding ",x,y
```

Strings

String is a collection of characters. In Python string value are given within single quotes(‘) or double quotes(“). When more than one line is needed to be stored in a string variable triple quotes(“““) is used.

Example:

```
name = ‘Mani’
course = “Computer”
college = “““Tamilnadu Polytechnic College
        Madurai 11”””
```

Data Types

Data Type specify the type of value that is to be sored in a variable. Python has siz different datatypes they are

1. Numeric
2. List
3. Tuple
4. String
5. Set
6. Dictionary

Numeric

In Python numeric data type is used to store numberic values. Python has four numeric types which can be stored in a variable.

Integer - It is used to store whole numbers.

Example

```
#integer
a = 10
b = 20
res = a + b
print "Result",res
```

Long Integer - In this we can store large numeric values.

Example

```
#long integer
a = 1234567890
b = 1987654321
res = a + b
print "Long Int Result", res
```

Float - is used to store decimal values.

Example

```
#float
a = 1.23
b = 3.45
res = a + b
print "Float Result", res
```

Complex - It is used to store complex number (2+3i). In python to denote complex number we use complex() or j is used.

Example

```
#complex numbers
a = complex(2,3)
b = 1+4j
res = a + b
print res
```

List

List is used to store a collection of values in a single name, it is similar to that of an array but not like an array. It is created using square bracket ([]). It can store different data types in a single name.

Example

```
#list
_mylist = [110,10.43,1+3j,'hello']
print _mylist
```

Tuple

Tuple is also used to store multiple values in a single variable name. To create a tuple the datas are enclosed within parantheses (( )). The differene between list and tuple is that tuple is read only. Values once stored in a tuple cannot be changed.

Example

```
#Tuple
_mytuple = (100,10.43,1+3j,'hello')
print _mytuple
#tuple read only
#_mytuple[1] = 12.43
```

String

String is a collection of characters. In Python a string can be accessed as an array of characters. To access the array of characters numeric index starting with 0 is used.

Example

```
#string
a = 'Welcome'
b = "to Python. "
print a
print b
#To Print 4th Character
print a[3]
```

Set

Set is an unordered collection. Unordered Collection is a collection which cannot be accessed using index value. This is created using braces ({}).

Example

```
#set
seta = {11,23,11,23,45,1}
print seta
```

Dictionary

Dictionary is a key-value pair based unordered collection. This is also created using braces ({}). In this each element is represented using Key-Value pair the general format is (Key:Value). Key is used as index to access the corresponding value.

Example

```
#Dictionary
emp = {100:'Balaji',102:'Anjay',103:'Xavier'}
print emp
print emp[102]
```

Operators

Operators are symbols that corresponds to mathematical or logical operations.

- Arithmetic Operators
- Assignment Operators
- Comparison (Relational) Operators
- Logical Operators
- Bitwise Operators
- Identity Operators
- Membership Operators

Arithmetic Operators

This operator is used to manipulate numeric values and it will return number as a result.

Symbol	Operator Name	Description
+	Addition	add two values
-	Subtraction	subtract two values
*	Multiplication	multiplies two values
/	Division	divides two values
%	Modulus	return remainder of dividing two values

Assignment Operators

Assignment Operator (=) is used to assign the value given on the right side of the operator to the variable given in the left side.

Symbol	Operator Name	Description
=	Equal	assign value from right to left side variable
+=	Add and	adds value on right to left and store in left side variable
*=	Multiply and	multiply value on right to left and store in left side variable
/=	Divide and	divide value on right to left and store in left side variable
%=	Modulus and	return remainder of value on right to left and store in left side variable

Comparison (Relational) Operators

Comparison Operators is used to compare two values.

Symbol	Operator Name	Description
==	Double Equal	check if two values are same
<>	Not equla to	check if two values are not same
>	Greater than	check if left value is greater than right value
<	less than	check if eft value is less than right value
<=	less than equal to	check if left value is less than or equal to right value
>=	greater than equal to	check if left value is greater than or equal to right value

Logical Operators

Logical Operators is used to check for more than one condition in a conditional statement.

Symbol	Operator Name	Description
or	Logical or	if any one condition is true then result is true
and	logical and	if both conditions are true then result is true
not	logical not	it reverse the result

Bitwise Operators

Bitwise Operators is used to manipulate binary value of a given number.

Symbol	Operator Name	Description
&	Binary AND	if both are 1 will return 1
|	Binary OR	if any one value is 1 will return 1
^	Binary XOR	will return 1 if both value are different and will return 0 if both value are same
<<	Binary Left Shift	moves binary value of left value by number given in right value in left

		direction
>>	Binary Right Shift	moves binary value of left value by number given in right value in right direction

Identity Operators

Identity Operators is used to compre the memory location of two variables.

Symbol	Description	Example

is	return true if both variable point to same memory	#IS
		x = 10
		y = x
		print 'x is y', (x is y)

is not	return true if both variable point to different memory	#IS NOT
		x = 10
		y = 20
		print 'x is y', (x is not y)

Membership Operators

Membership Operator is used to check if a value is present in a sequence (string, list, tuple).

Symbol	Description	Example

in	return true if both the given value is found in the list	#in
		x = 'Hello world'
		print 'H in Hello world', ('H' in x)

not in	return true if both the given value is not found in the list	#not in
		x = 'Hello world'
		print 'A in Hello world', ('A' not in x)

Input/Output Functions

Input

Input funciton is used to get value from the user in Python script.

syntax

variable = input(‘Prompt Message’)

Example

```
#input from command line
a = input('Enter a:')
b = input('Enter b:')
res = a + b
print 'Result is', res
```

Output

Print function is used to display values to user in python script. Print function can be used to display output in two forms either unformatted output orformatted output. Unformatted output uses coma(,) in the print functionn to display values to the user.

Syntax

print ‘Message’,value

To display formatted output print statement uses format function. The values in the format function are accessed in the print statemnt using index values.

Syntax

```
print 'Message {0}'.format(value)
```

Example

```
#Unformatted and Formatted Output
a = 10
b = 20
res = a + b
#Unformatted
print 'sum of',a,'+',b,' =',res
#Formatted
print 'sum of {1} + {0} = {2}'.format(a,b,res)
```

Decision Control-Conditional Statements

Conditional Statements are used to check for a condition and transform the script control based on the result of the condition. Conditional statement can have either TRUE/FALSE as the result. In Python language decision control is implemented using if statement.

- Simple if statement
- If .. else statement
- if .. elif statement

Simple if Statement

In this format of the if statement the statement that has to be executed if the condition is TRUE is only given. It doesnot care about the FALSE part of the condition.

Syntax

```
if condition:
    statements
```

Example

```
#simple if statement
age = input("Enter Your Age: ")
if age >= 18:
    print "voting age"
print "You are allowed to vote"
```

if..else statement

In this format of the conditional statement there are two parts one part for the statements that are to be executed when the condition is TRUE and the second part that is the else part is for the statements that to be executed when the condition is FALSE.இந்த conditional statementல் இரண்டு பகுதிகள் இருக்கும்.

Syntax

```
if condition:
    TRUE Statements
else :
    FALSE Statements
```

Example

```
#if..else statement
age = input("Enter Your Age: ")
if age >= 18:
    print "Old enough"
    print "You are allowed to vote"
else:
    print "You are young"
    print "Not allowed to vote"
```

if .. elif statement

To check more than one condtion one by one in a single statement block if..elif statement is used.

Syntax

```
if condition1:
        statements
elif condition2:
        statements
elif condition3:
        statements
else:
        statements
```

Example

```
#if..elif statement
age = input("Enter Your Age: ")
if age >= 1 and age <= 5:
        print "Child"
        print "No Ticket Required"
elif age >= 6 and age <= 12:
        print "Child"
        print "1/2 Ticket Required"
elif age >= 12 and age <= 60:
        print "adult"
        print "Full ticket"
else:
        print "Senior Citizen"
        print "20% Concession"
```

Loops

To repeat exectuion of a code block more than once looping statements are used. Python language support two types of looping statement.

- While Loop
- For Loop

While Loop

In this loop the condition is checked first and if the condition is true then only the statements given in the loop code block is executed. The statements given in the loop code block is executed until the condition becomes false.

Syntax

```
while condition:
        statements
```

Example

```
#while loop statement
print "Sum of the series"
num = input("Enter a number: ")
x = 1
ser = 0
while x <= num:
        ser = ser + x
        x += 1
print ('sum of the series 1 to {0} is {1}'.format(num,ser))
```

For Loop

For loop statement doesnot check for the condition but it uses the elements in a sequence to repeat the loop. It executes the loop until it has elements in the sequence. Sequence can be a list, tuple, set or range() function.

Syntax

```
for element in sequence:
        statements
```

Example1:

```
#for loop statement
print "sum of element's in a Tuple"
tup = (1,2,3,12,43,10,4)
res = 0
for x in tup:
        res = res + x
print "sum of Tuple is",res
```

Example2:

```
#For loop using range
a = input("Enter a number")
for i in range(0,a):
        print '*',
```

4.3 Sequences:

Lists: Introduction

List is the most commonly used sequence in Python script. To create a list square bracket is used, the values are given as comma-separated value. To create an empty list the syntax is given below

Syntax

```
lst = []
lst = list()
```

To create a list with value the following syntax is used

Syntax

```
lst = list(object)
lst = [val1, val2, val3…]
```

To access the list values index values is used. Index value can be used to change the values in the list

Example

```
lst_a = [1,2,'welcome',4]
print "Second Element: ",lst_a[1]
lst_a[0]='hai'
print lst_a
```

List Methods

List implements the following methods in it.

Method	Description
list.count(item)	Count number of occurrece of *item*
list.append(item)	Add *item*value at last of the list
list.insert(index,item)	Add *item*at the given *index* position
list.index(item)	Get index position of *item*
list.pop()	remove an *item* from end of the list
list.remove(item)	Remove the given *item* from the list
list.reverse()	Reverse the list content
list.sort()	Sort the list content

Fixed size lists and arrays

To create a list with fixed size like that of an array with size in any other programming language the repetition operator (*) is used in python along with None value.

Example

```
a = [None] *3
print a
a[0] = 3
a[1] = 1
a[2] = 5
print a
```

Output

```
[None, None, None]
[3,1,5]
```

This method is used to create two dimensional array in python easily.

Syntax

```
variable = [None] * rowSize
variable[row index] = [None] * columnSize
variable[row index][col index]
```

Example

```
#Two Dimensional Array 3 row, 3 column
b = [None]*3
b[0] = [1]*3
b[1] = [2]*3
b[2] = [3]*3
b[0][0], b[0][1], b[0][2] = 10,20,30
b[1][0], b[1][1], b[1][2] = 11,21,31
b[2][0], b[2][1], b[2][2] = 12,22,32
print  b
```

Output

```
>>>[[10, 20, 30], [11, 21, 31], [12, 22, 32]]
```

Lists and Loops

List can be accessed using for loop in python. When the list variable name is given in the for loop the element is the list is accessed one by one until all the elements are read.

Example

```
lsta = [1,2,3,4,5]
for x in lsta:
        print x,
```

Output

```
>>> 1 2 3 4 5
```

Assignment and Reference

Python uses assignement (=) operator to copy the reference of a value but not its value.

Example

```
a = [1,2,3]
b = a
b[1] = 7
print "Value in a: ",a
```

Output

```
>>>Value in a: [1,7,3]
```

To copy the values without copying the reference the following syntax is used.

Syntax

```
list2 = list1[:]
```

Example

```
a = [1,2,3]
b = a[:]
b[2]=5
print "Value in A: ",a
print "Value in B: ",b
```

Output

```
>>>
Value in A:  [1, 2, 3]
Value in B:  [1, 2, 5]
```

Identity and Equality

Identity is checked in python by the use of is operator and equality is check using double equal to (==) operator. The difference between Identity and equality is, identity checks if the two list corresponds to the same object whereas equality is used to check if two lists has the same number of elements and same values.

Example 1

```
#Equality
lst1 = [1,2,3]
lst2 = [1,2,3]
if lst1 == lst2:
        print "both are equal"
```

Example 2

```
#Identity
lst1 = [10,20,30]
lst2 = [10,20,30]
if lst1 is lst2:
        print "both are identical"
else:
        print "not identical"
```

Sorted List

Sorted list consists of elements that are sorted in an order. To sort a list sorted function or sort method is used.

Example

```
#sorted List
lst = [23,19,12,30,15,1,6,2]
print "Unsorted List", lst
lst = sorted(lst)
print "Sorted Function: ",lst
lst.append(2)
print "New List: ",lst
lst.sort()
print "Sorted Method: ", lst
```

Output

```
Unsorted List [23, 19, 12, 30, 15, 1, 6, 2]
Sorted Function:  [1, 2, 6, 12, 15, 19, 23, 30]
New List:  [1, 2, 6, 12, 15, 19, 23, 30, 2]
Sorted Method:  [1, 2, 2, 6, 12, 15, 19, 23, 30]
```

Tuples

Tuple is similar to that of a list it uses parenthesis (( )) for storing elements. Tuple's value cannot be changed once assigned. A tuple can be changed to list or a list can be changed to tuple.

Example

```
#tuple
tupa = (10,20,30,40)
print tupa
print tupa[3]
lsta = [12,30,15]
tupb = tuple(lsta)
print tupb
tupa[0]=20
```

Output

```
(10, 20, 30, 40)
40
(12, 30, 15)
Traceback (most recent call last):
  File "C:\Python\test.py", line 9, in <module>
        tupa[0]=20
TypeError: 'tuple' object does not support item assignment
```

In this example when we try to assign a value like tupa[0]=20 it will generate error.

Tuple and string formatting

Tuple can be used in the print statement to display values with formatting. Like C programs format specifier used in printf statement python also has certain format specifiers.

Format Specifier	Data Type
%d	Integer Value
%s	String Value
%f	Float Value

Syntax

```
print “text with string formatting” %tuple
print “text with string formatting” %(values)
```

Example

```
#String Formatting
a = 10
b = 10.23
str1 = "welcome"
tup_fmt = (a,b,str1)
print  "Integer: %d\nFloat: %f\nString: %s" %tup_fmt
print "float %.1f\n Integer:%d" %(2.37,10)
```

String Functions

Python language has the following functions to manipulate string values

Function	Description
s.capitalize()	Capitalizes first character of s
s.count(sub)	Count number of occurrences of sub in s
s.find(sub)	Find first index of sub in s, or -1 if not found
s.index(sub)	Find first index of sub in s, or raise ValueError if not found
s.rfind(sub)	Same as find, but last index
s.rindex(sub)	Same as index, but last index
s.lower()	Convert s to lower case

s.split()	Return a list of words in s
s.join(lst)	Join a list of words into a single string with s as separator
s.strip()	Strip leading/trailing white space from s
s.upper()	Convert s to upper string
s.replace(old, new)	Replace all instances of old with new in string

4.4 Dictionary - Introduction

Dictionary is created using braces ({ }). To access the values in the dictionary key value is used instead of numeric index.

Syntax

dct = {Key:Value}

Example

```
dicta = {101:'arun',102:'balu',103:'mani'}
print dicta
        print dicta[101]
```

Output

```
>>> {101: 'arun', 102: 'balu', 103: 'mani'}
    arun
```

Dictionary Methods

Dictionary implements the following methods

Method	Description
len(d)	Number of elements in d
d[k]	Item in d with key k
d[k] = v	Set item in d with key k to v
d.items()	Return a list of (key, value) pairs
d.keys()	Return a list of keys in d
d.values()	Return a list of values in d
d.clear()	Remove all items from dictionary d
d.copy()	Make a shallow copy of d

Combining two Dictionary

To combine two dictionary values update method is used. When combining two dictionary's if the two dictionary has the same key value then the key value pair in the first dictionary will be erased and the value from the second dictionary is retained.

Example

```
dicta = {101:'arun',102:'balu',103:'mani'}
dictb = {101:'anbu',104:'ramu',105:'rajesh'}
dicta.update(dictb)
print dicta
```

Output

```
>>>{101: 'anbu', 102: 'balu', 103: 'mani', 104: 'ramu', 105: 'rajesh'}
```

Making copies

In Python when assignment (=) statement is used to copy one dictionary to another dictionary only reference is copied and not the

values. So to copy the values of one dictionary to another dictionary copy() method is used.

Example

```
dicta = {101:'arun',102:'balu',103:'mani'}
dictb = dicta
dictb[101]='Anbu'
print "Dict A: ",dicta
dictc = dicta.copy()
dictc[101]='Antony'
print "Dict A: ",dicta
print "Dict C: ",dictc
```

Output

```
Dict A:  {101: 'Anbu', 102: 'balu', 103: 'mani'}
Dict A:  {101: 'Anbu', 102: 'balu', 103: 'mani'}
Dict C:  {101: 'Antony', 102: 'balu', 103: 'mani'}
```

Persistent Variables

Persistent Variable is a variable whose value can be retained over multiple different program execution. In python this is implemented using the shelve module. This enables a value to be made available in multiple programs.

Example store.py

```
#Persistent Storage
import shelve
data = shelve.open("info")
data["perval"] = 20
print "Persistent Value: ",data["perval"]
data.close()
```

Output

```
Persistent Value:  20
```

Example get.py

```
#Persistent Storage
import shelve
data = shelve.open("info")
print "Accessing Persistent Value across Files"
print "Persistent Value: ",data["perval"]
```

Output

```
Accessing Persistent Value across Files
Persistent Value:  20
```

In the first program store.py a value is stored using the shelve module. in the second program get.py the stored persistent value is obtained using the shelve module.

Internal Dictionaries

In Python when a program is executed the variables and values used in the program are stored using the dictionary. To access this dictionary the locals() function is used.

Example

```
#locals
a = 10
b = 20
print locals()
c = a+b
print locals()
```

Output

```
{'a': 10, 'b': 20}
{'a': 10, 'c': 30, 'b': 20}
```

4.5 Functions and Files

Functions

Function is a named code block which is used to perform a specific task. Python uses def keyword to create user defined functions. In python the function block is not defined using braces ({}) like other programing languages, it uses return statement to denote the end of a function block.

Syntax

```
def function_name(parameter)
        statements
return
```

Example

```
def add():
  a = input("Enter A Value: ")
  b = input("Enter B Value: ")
  res = a + b
  print "Sum is ",res
  return

#main Script
add()
```

Output

```
Enter A Value: 10
Enter B Value: 20
Sum is  30
```

Python function is classified in to four types based on its arguments

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Required arguments

In Required arguments format of function call in python both the function call and function definition must have the same number of parameters. If the parameter list is not same the script will generate error.

Example

```
def add(a,b):
  res = a + b
  print "Sum is ",res
  return
#main Script
add(10,20)
add (10)
```

Output

```
Sum is  30
Traceback (most recent call last):
  File "C:\Python\24-fun-req.py", line 8, in <module>
add (10)
TypeError: add() takes exactly 2 arguments (1 given)
```

Keyword Arguments

In Keyword Arguments format of python function, the function is called using parameter name and its corresponding value instead of only value. This allows the use of parameter without any order.

Example

```
def add(a,b):
res = a + b
  print "a(%d)+b(%d)=%d " %(a,b,res)
  return
#main Script
add(a=10,b=20)
add(b=15,a=4)
```

Output

```
a(10)+b(20)=30
a(4)+b(15)=19
```

Default arguments

In Default arguments format of function, we provide default value for the parameters. if the function call does not provide value for a parameter then the default value given in function definition will be used.

Example

```
def add(a,b=10):
  res = a + b
  print "a(%d)+b(%d)=%d " %(a,b,res)
  return

#main Script
add(5,23)
add(4)
```

Output

```
a(5)+b(23)=28
a(4)+b(10)=14
```

In this the first function call has both the parameters whereas in the second function call value for the first parameter is only given. So python will automatically assign the default value of 10 to the second parameter..

Variable-length arguments

In Variable-length arguments format of function, instead of providing the number of parameters in the list it is automatically assigned so that any number of values can be given as parameter list.

Example

```
def add(arg,*varlen):
   res = 0
   res = res + arg
   for x in varlen:
      res = res + x
   print "Sum of %d numbers is %d " %(len(varlen)+1,res)


#main Script
add(5,23)
add(4,1,2,3,4)
```

Output

```
Sum of 2 numbers is 28
Sum of 5 numbers is 14
```

File Handling

In Python language to perform file operations first the open() function is used to create file object using this file object the remaining file operations are performed Python has four operations that can be done on a file

- Opening a File

- Closing a File
- Writing a File
- Reading a File

Opening a File

To open a file open() function is used. It has two parameter they are filename and file opening mode.

Syntax

```
file_obj = open(filename [,mode])
```

File Opening Modes

- r - reading
- w - writing
- a - append
- b - binary
- + - read and write

Closing a File

The gile that has been opened has to be closed using the close() function. Once the file is closed it is not possible to add data in it.

Syntax

```
file_obj.close()
```

Writing a File

To write text data in a file that is opened write() function is used.

Syntax

```
file_obj.write(content)
```

Reading a File

To read the contents of the file read() funtion or file object is used. read() function is used to read the whole content of the file. File object along with the use of for loop is used to read the content of the file line by line.

Syntax

```
file_obj.read()
```

Example Script

```
fo = open("sample.txt","w")
text = "Writing into file"
fo.write(text)
text = "\nSome Content"
fo.write(text)
fo.close()
print "File written"
fo = open("sample.txt","r")
text = fo.read()
print text
fo.close()
fo = open("sample.txt","r")
print "Reading File Using File Object"
for line in fo:
   print "Line: ",line,
fo.close()
```

Output

```
File written
Writing into file
Some Content
Reading File Using File Object
Line:  Writing into file
Line:  Some Content
```

Exception Handling

Exception is an error that occurs during runtime of the script. Exceptions are generally not identified before execution of the program.

If exception is not handled correctly it will stop the script execution abruptly. Exception handling is the mechanism of handling the runtime error that occurs during the program execution so that the program will continue execution even after runtime error. The code block that generate error is placed in try: block and the mechanism to handle that error is place in the except: block.

Syntax

```
try:
   # Code that Generate Exception
except:
   # Handle Exception
```

Pythonல் பொதுவாக வரக்கூடிய exceptionகள்

- Exception - General Error Class
- IOError - Input Output Error
- ImportError - Library Not Found Error
- ValueError - Empty Value Error
- ZeroDivisionError - Arithmetic Exception
- EOFError - File End Reached Error

Example

```
#simple Exception
try:
   fobj = open("test.txt","r")
   fobj.write("Welcome")
   fobj.close()
except IOError:
   print "Error in accessing file"
```

Output

```
Error in accessing file
```

Multiple Exceptions

In a script if there are chances of occurence of more than one error then the try: block is followed by multiple except: blocks.

Example

```
#Multiple Exception
try:
   fobj = open("test.txt","r")
   fobj.write("Welcome")
   fobj.close()
   txt = fobj.read()
except IOError:
   print "Error in accessing file"
except ValueError:
   print "Closed file cannot be read"
```

Finally Block

In a script even if error occurs or not some statements need to be executed, such statements are placed in finally block.

Example

```
#Finally Clause
try:
   fobj = open("test.txt","r")
   fobj.write("Welcome")
   fobj.close()
   print "File Closed "
except IOError:
   print "Error in accessing file"
finally:
   print "Closing the File in finally"
   fobj.close()
```

User defined Exception

User defined exceptions are used to create custom exception that are not predefined in the python language.

Syntax

```
class UserError(Exception):
   def __init__(self,data):
      self.data = data

raise UserError('Error Text')
```

Example

```
class MarkError(Exception):
def __init__(self,data):
self.data = data

print "Enter 5 subject marks"
tot = 0
i = 0
for x in range(0,5):
   try:
         mark = input("Enter mark")
         if mark < 0 or mark > 100:
            raise MarkError("Enter Marks 0-100")
         else:
            tot = tot + mark
            i = i +1
   except MarkError as e:
   print e.data
print "Total of %d subjects is %d" %(i,tot)
```

In this script mark value is obtained from the user, if the mark value is less than 0 or greater than 100 then MarkError user defined exception is arised.

Output

```
Enter 5 subject marks
Enter mark78
Enter mark90
Enter mark150
Enter Marks 0-100
Enter mark86
Enter mark67
Total of 4 subjects is 321
```

.