3RI Technologies™

Respect • Rationale • Response • Innovative

Python Programming

Learn the way industry wants it...

Table of Contents

Chapter 1

Basic Python

Topics Covered

- **Python Introduction**
- **Python History**
- **Python Version**
- **Python Features**
- **Local Environment Setup**
- **Basic Fundamentals of python**

1.1:-Python Introduction

- Python is a general purpose, dynamic, high level and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.
- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

1.2:-Python History

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

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 copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress

1.3:-Python Version.

Python programming language is being updated regularly with new features and supports. There are lots of updations in python versions, started from 1994 to current release.

A list of python versions with its released date is given below Table.

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1996
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 16, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.6	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 26, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.6.4	December 19, 2016

1.4:-Python Features

- Python provides lots of features that are listed below.
- **Easy to Learn and Use**

  Python is easy to learnand use.It is developer-friendly and high level programming language.

- **Expressive Language**

  Python language is more expressive means that it is more understandable and readable.
- **Interpreted Language**

  Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.
- **Cross-platform Language**

  Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So,we can say that Python is a portable language.
- **Free and Open Source**

  Python language is freely available at offical web address.The source-code is also available. Therefore it is open source.
- **Object-Oriented Language**

  Python supports object oriented language and concepts of classes and objects come into xistence.
- **Extensible**

  It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.
- **Large Standard Library**

  Python has a large and broad library and prvides rich set of module and functions for rapid application development.
- **GUI Programming Support**

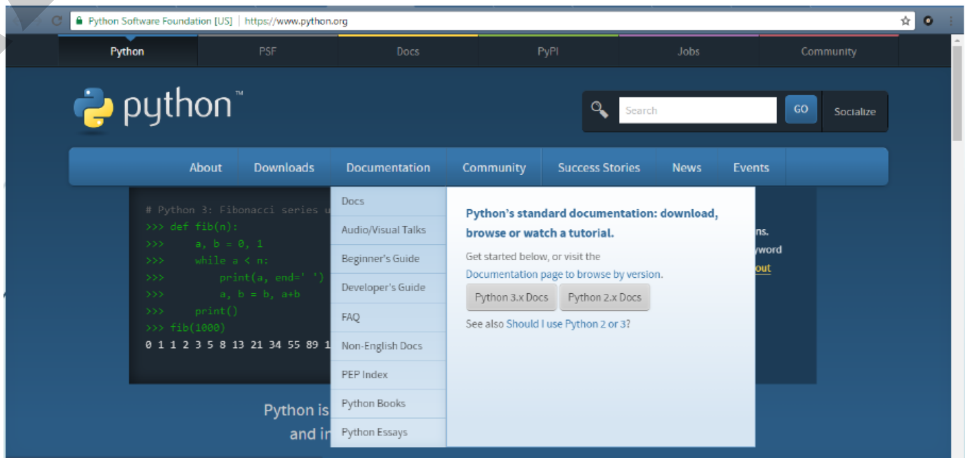
  Graphical user interfaces can be developed using Python.
- **Integrated**

  It can be easily integrated with languages like C, C++, JAVA etc.
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

1.5:-Local Environment Setup

- To start with Python, first make sure that the Python is installed on local computer.
- To install Python, visit the official site and download Python from the download section.
- To install Python on Ubuntu operating system, visit our installation section where we have provided detailed installation process.
- For Windows operating system, the installation process is given below.

1. To install Python, firstly download the Python distribution from www.python.org/download.



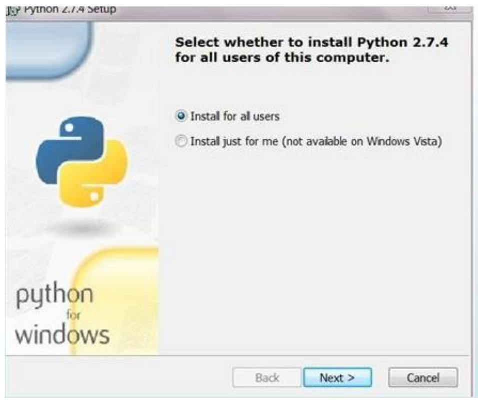
```
# Python 3: Fibonacci series u
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 1
```

Python's standard documentation: download, browse or watch a tutorial.

Get started below, or visit the Documentation page to browse by version.

See also Should I use Python 2 or 3?

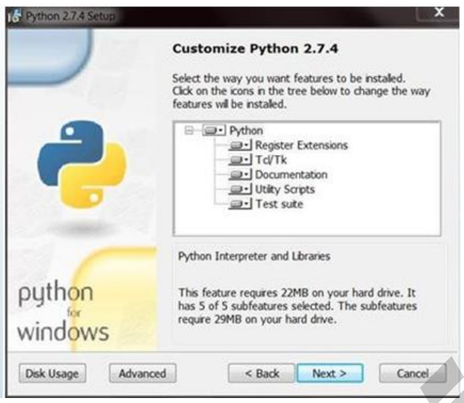
2. After downloading the Python distribution, double click on the downloaded software to execute it. Follow the following installation steps


Python 2.7.4 Setup

Select whether to install Python 2.7.4 for all users of this computer.

- Install for all users
- Install just for me (not available on Windows Vista)

figure: 1


Python 2.7.4 Setup

Customize Python 2.7.4

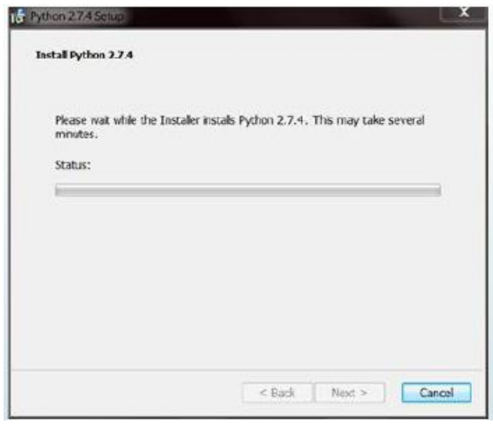
Select the way you want features to be installed.
Click on the icons in the tree below to change the way features will be installed.

- Python
  - Register Extensions
  - Tcl/Tk
  - Documentation
  - Utility Scripts
  - Test suite

Python Interpreter and Libraries

This feature requires 22MB on your hard drive. It has 5 of 5 subfeatures selected. The subfeatures require 29MB on your hard drive.

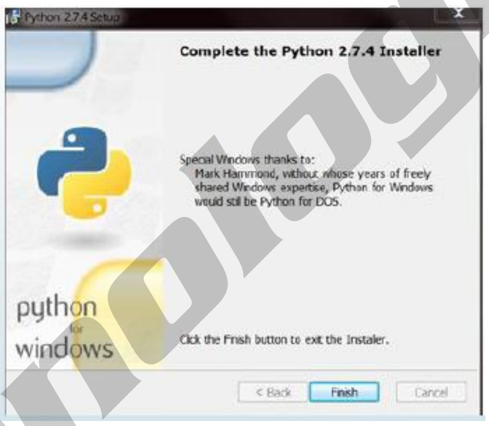
figure: 2


Python 2.7.4 Setup

Install Python 2.7.4

Please wait while the Installer installs Python 2.7.4. This may take several minutes.

Status:


Python 2.7.4 Setup

Complete the Python 2.7.4 Installer

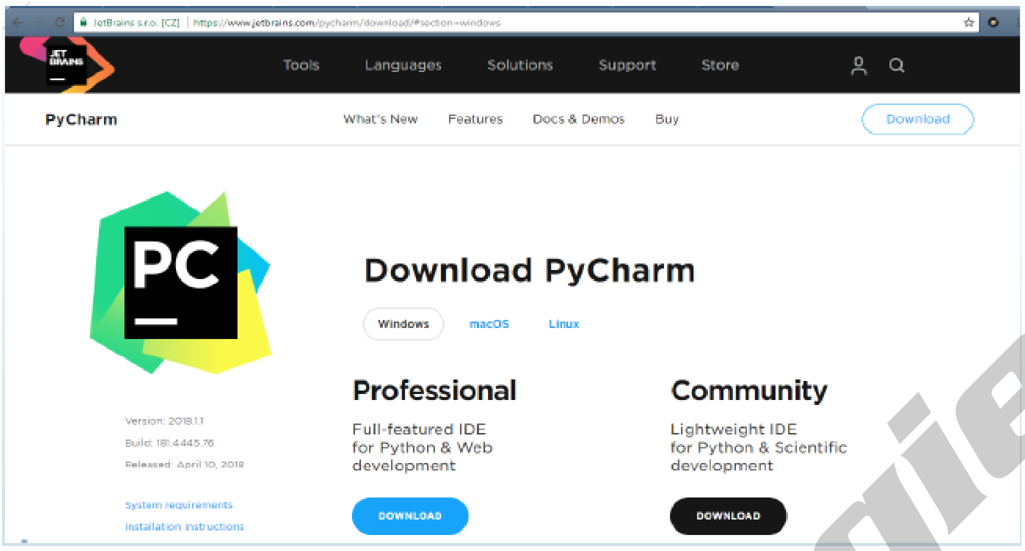
Special Windows thanks to:
Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

Click the Finish button to exit the Installer.

figure: 3

figure: 4

1.5.1:-IDE



If you want use ide for easiest so you can download from official site of PyCharm.

Python is very good ide for python .

1.5.2:-Basic Syntax

simple hello world example

```
print("hello world by python!")
```

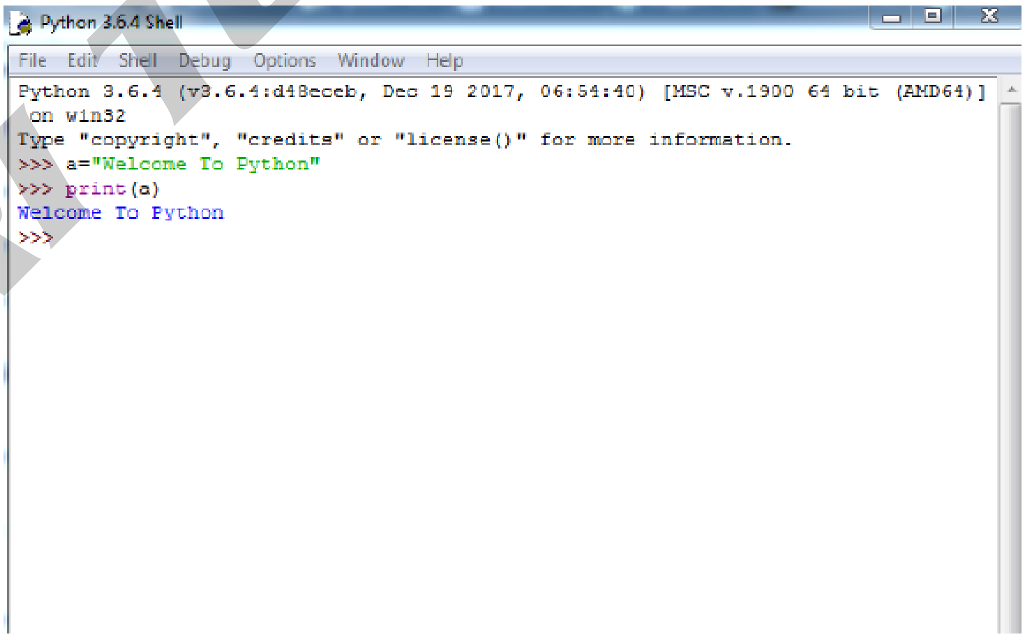
After executing, it produces the following output to the screen.

Output

```
hello world by python!
```

1.5.3:-Python Example using Interactive Shell

```
a="Welcome To Python"
print (a)
Welcome To Python
```

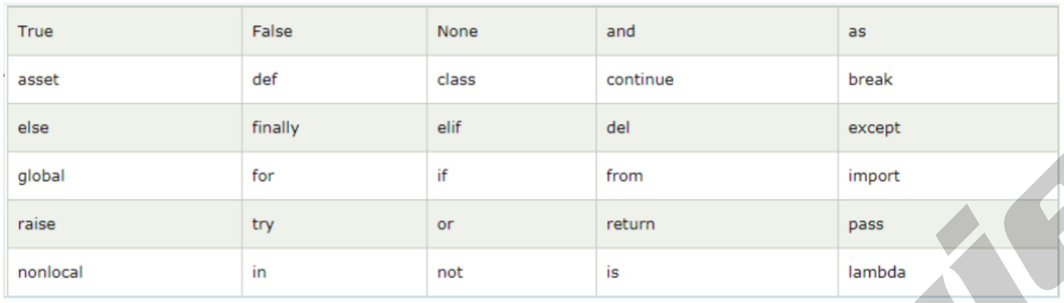


```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> a="Welcome To Python"
>>> print(a)
Welcome To Python
>>>
```

1.6:-Basic Fundamentals of python

1.6.1:Python Keywords

- Python Keywords are special reserved words which convey a special meaning to the compiler/interpreter. Each keyword have a special meaning and a specific operation. These keywords can't be used as variable. Following is the List of Python Keywords.



True	False	None	and	as
asset	def	class	continue	break
else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass
nonlocal	in	not	is	lambda

1.6.2:Identifiers

- Identifiers are the names given to the fundamental building blocks in a program.
- These can be variables ,class ,object ,functions , lists , dictionaries etc.
- There are certain rules defined for naming i.e., Identifiers.

I. An identifier is a long sequence of characters and numbers.

II. No special character except underscore (_) can be used as an identifier.

III. Keyword should not be used as an identifier name.

IV. Python is case sensitive. So using case is significant.

V. First character of an identifier can be character, underscore (_) but not digit.

1.6.3:Python Comments

Python supports two types of comments:

1. **Single lined comment:**

   If In case user wants to specify a single line comment, then comment must start with ?#?

Exapmle

```
# This is single line comment.
```

3. **Multi lined Comment:**

If In case user wants to Multi line comment, then comment must start with? ‘’’ and end with ‘’’

Example

```
""" This
  Is
  Multipline comment'''
```

Example:

```
#single line comment
print ("Hello Python")
"""This is
multiline comment'''
```

Chapter2

Python Overview

Topics Covered

2.1:-Python Variable

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.
- Variable names can be a group of both letters and digits, but they have to begin with a letter or an underscore.

Declaring Assigning Values to Variables

- Python does not bound us to declare variable before using in the application. It allows us to create variable at required time.
- We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.
- The equal (=) operator is used to assign value to a variable.

Example

```
name="raj" #  A string
school="DAV" # A string
roll_no=1 # integer assignment
waight=50.00 # floating point
print(name)
print(school)
print(roll_no)
print(waight)
```

Output

```
raj
DAV
1
50.0
```

2.1.1:-Multiple Assignment

- Python allows us to assign a value to multiple variables in a single statement which is also known as multiple assignment.
- We can apply multiple assignments in two ways either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Lets see given examples.

1. **Assigning single value to multiple variables**

```
x=y=z=50
print (x)
print (y)
print (z)
50
50
50
```

2. **Assigning multiple values to multiple variables:**

```
a,b,c=5,10,15
print(a)
5
print(b)
10
print(c)
15
print(a+b)
15
>>>
```

The values will be assigned in the order in which variables appears.

- This section contains the basic fundamentals of Python like :

a) Tokens and their types.

b) Comments a

c) Tokens:

  - Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.
  - Token is the smallest unit inside the given program.

- There are following tokens in Python:

- Keywords.
- Identifiers.
- Literals.
- Operators.

2.2:-Python Literals

- Literals can be defined as a data that is given in a variable or constant.Python support the following literals:

1. **String literals:**

   String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes for a String.

Example

```
"Aman" , '12345'
```

Types of Strings:

There are two types of Strings supported in Python:

a) Single line String- Strings that are terminated within a single line are known as Single line Strings.

Example

text1='hello'

b) Multi line String- A piece of text that is spread along multiple lines is known as Multiple line String. There are two ways to create Multiline Strings:

1. **Adding black slash at the end of each line.**

Example:

```
ext1='hello\
user'
>>> text1
'hellouser'
```

2. **Using triple quotation marks:-**

Example:

```
str2='''welcome
to
SSSIT'''
print str2
welcome
to
SSSIT
```

2.3-Standard Data Types

- The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data 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 −

- Numbers
- String
- List
- Tuple
- Dictionary

2.4:- Python Operators

- Operators are particular symbols that are used to perform operations on operands.
- Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.
- Types of Operator
- Python language supports the following types of operators.
    - Arithmetic Operators
    - Comparison (Relational) Operators
    - Assignment Operators
    - Logical Operators
    - Bitwise Operators
    - Membership Operators
    - Identity Operators

2.4.1:-Python Arithmetic Operators

Operators	Description
//	Perform Floor division(gives integer value after division)
+	To perform addition
-	To perform subtraction
*	To perform multiplication
/	To perform division
%	To return remainder after division(Modulus)
**	Perform exponent(raise to power)

Example

```
Program
a = 21
b = 10
c = 0
c = a + b
print ("Line 1 - Value of c is ", c)
c = a – b
print ("Line 2 - Value of c is ", c)
 c = a * b
print ("Line 3 - Value of c is ", c )
```

```
c = a / b
print ("Line 4 - Value of c is ", c)
c = a % b
print ("Line 5 - Value of c is ", c)
a = 2
b = 3
c = a**b
print ("Line 6 - Value of c is ", c)
a = 10
b = 5
c = a//b
print ("Line 6 - Value of c is ", c)
Result
Line 1 - Value of c is  31
Line 2 - Value of c is  11
Line 3 - Value of c is  210
Line 4 - Value of c is  2
Line 5 - Value of c is  1
Line 6 - Value of c is  8
Line 6 - Value of c is  2
```

2.4.2:-Comparison (Relational) Operators

- These operators compare the values on either sides of them and decide the relation among them.

Operators	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
<>	Not equal to(similar to !=)
Operators	Description

Example

```
Program
    a=10<20
b=10>20
c=10<=10
d=20>=15
e=5==6
```

```
f=5!=6
print("value a=10<20 is",a)
print("value b=10>20 is",b)
print("value c=10<=10",c)
print("value d=20>=15 is",d)
print("value e=5==6 is",e)
print("value f=5!=6 is ",f)
Result
value a=10<20 is True
value b=10>20 is False
value c=10<=10 True
value d=20>=15 is True
value e=5==6 is False
value f=5!=6 is  True
```

2.4.3:-Python Assignment Operators

- The following table contains the assignment operators that are used to assign values to the variables.

Operators	Description
=	To perform Assignment the value
/=	Divide and Assign
+=	Add and assign
-=	To perform multiplication
*=	Multiply and assign
%=	Modulus and assign
**=	Exponent and assign
//=	Floor division and assign

Example

```
Program
a=10
print("value of a is ",a)
a+=5
print("value of a is ",a)
a-=5
print("value of a is ",a)
a*= 2
print("value of a is ",a)
a/=2
print("value of a is ",a)
a%=3
```

```
print("value of a is ",a)
a**=2
print("value of a is ",a)
a//= 2
print("value of a is ",a)
Result
value of a is  10
value of a is  15
value of a is  10
value of a is  20
value of a is  10.0
value of a is  1.0
value of a is  1.0
value of a is  0.0
```

2.4.4:-Logical Operators

Operators	Description
and	Logical AND(When both conditions are true output will be true)
or	Logical OR (If any one condition is true output will be true)
not	Logical NOT(Compliment the condition i.e., reverse)

```
Program
a=5>4 and 3>2
print ("value of a is",a)
b=5>4 or 3<2
print ("value of b is",b)
c=not("value of c is",5>4)
print ("value of c is",c)
Result
value of a is True
value of b is True
value of c is False
```

2.4.5:-Membership Operators

Operators	Description
in	Returns true if a variable is in sequence of another variable, else false.
not in	Returns true if a variable is not in sequence of another variable, else false.

```
a=10
b=20
list=[10,20,30,40,50];
if (a in list):
   print "a is in given list"
else:
   print "a is not in given list"
if(b not in list):
   print "b is not given in list"
else:
   print "b is given in list"
output
a is in given list
b is given in
```

2.4.6:-Identity Operators

- Identity operators compare the memory locations of two objects. There are two Identity operators as explained below –

Operators	Description
is	Returns true if identity of two operands are same, else false
is not	Returns true if identity of two operands are not same, else false.

```
a=10
b=20
list=[10,20,30,40,50];
if (a in list):
   print "a is in given list"
else:
   print "a is not in given list"
if(b not in list):
   print "b is not given in list"
else:
   print "b is given in list"
20 b have same identity
20 b have different identity
```

Chapter 3

Control Statements

Topics Covered

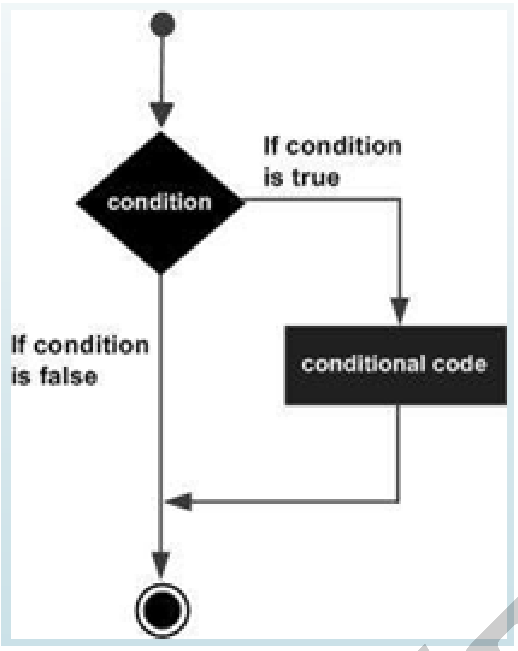
3.1:-Python If Statements

- The Python if statement is a statement which is used to test specified condition. We can use if statement to perform conditional operations in our Python application.
- The if statement executes only when specified condition is true. We can pass any valid expression into the if parentheses.
- There are various types of if statements in Python.
    - if statement
    - if-else statement
    - nested if statement

Python If Statement Syntax

```
if(condition):
  statements
```

- **Python If Statement Example**



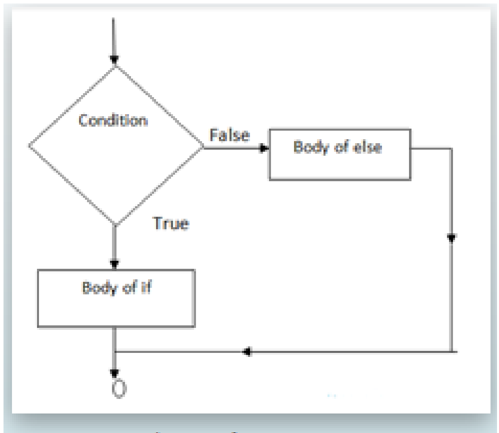
Flow Diagram

```
a=10
if a==10:
   print  ("Welcome to 3RI Technologies")
Welcome to 3RI Technologies
```

3.2 :-Python If Else Statements

- The If statement is used to test specified condition and if the condition is true, if block executes, otherwise else block executes.
- The else statement executes when the if statement is false.

```
if(condition):  False
          statements
   else:   True
           statements
```



Flow Diagram

```
Today="Sunday"
if Today==" Sunday ":
   print  ("Today is Sunday ")
else:
   print ("Today is Monday")
output
Today is Sunday
```

3.3: -Python Nested If Else Statement

- Python Nested If Else Syntax

```
If statement:
   Body
elif statement:
   Body
else:
   Body
```

- Python Nested If Else Example

```
a=10
if a>=20:
   print ("Condition is True")
else:
   if a>=15:
      print ("Checking second value")
   else:
      print ("All Conditions are false")
output
All Conditions are false
```

3.4 :-For Loop

- Python for loop is used to iterate the elements of a collection in the order that they appear. This collection can be a sequence(list or string).
- Python For Loop Syntax

```
for <variable> in <sequence>:
```

- For Loop Simple Example

```
num=1
for a in range (1,3):
   print  (num * a)
1
2
```

3.4.1 :-Experiment of for loop

- Example to Find Sum of

```
sum=0
for n in range(1,11):
   sum+=n
print (sum)
output
55
```

- Iterating by Sequence Index

```
fruits = ['Sunday', 'Monday',  'Tuesday']
for index in range(len(fruits)):
  print ('Current Day :', fruits[index])
print ("Good bye!")
output
Current Day : Sunday
Current Day : Monday
Current Day : Tuesday
Good bye!
```

3.5:-Python Nested For Loops

- Loops defined within another Loop are called Nested Loops. Nested loops are used to iterate matrix elements or to perform complex computation.
- When an outer loop contains an inner loop in its body it is called Nested Looping.

```
for  <expression>:
      for <expression>:
          Body
for i in range(0, 5):
   for j in range(0, i+1):
      print("* ",end="")
   print()
output
*
* *
* * *
* * * *
* * * * *
```

3.6 :-Python While Loop

- In Python, while loop is used to execute number of statements or body till the specified condition is true. Once the condition is false, the control will come out of the loop.
- Python While Loop Syntax

```
while <expression>:
    Body
```

- Here, loop Body will execute till the expression passed is true. The Body may be a single statement or multiple statement.

```
a=10
while a>0:
   print ("Value of a is",a)
   a = a - 2
Value of a is 10
Value of a is 8
Value of a is 6
Value of a is 4
Value of a is 2
```

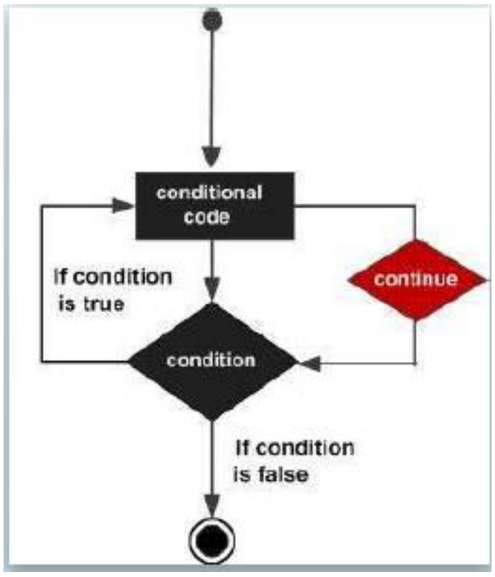
3.7 :-Python-break statement

- Break statement is a jump statement which is used to transfer execution control. It breaks the current execution and in case of inner loop, inner loop terminates immediately.
- The most common use of 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.
- **Python Break Example**

```
#!/usr/bin/python3
for letter in 'Python':  # First Example
   if letter == 'h':
      break
   print('Current Letter :', letter)
Current Letter : P
Current Letter : y
Current Letter : t
```

3.8 :-Python Continue Statement

- The continue statement in Python returns the control to the beginning of the current loop. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.
- The continue statement can be used in both while and for loops.



Flow Diagram

Example

```
a=0
while a<=5:
   a=a+1
   if a%2==0:
      continue
   print (a)
print ("End of Loop")
output
1
3
5
End of Loop
```

3.9 :-Python 3 pass Statement

- In Python, pass keyword is used to execute nothing; it means, when we don't want to execute code, the pass can be used to execute empty. It is same as the name refers to. It just makes the control to pass by without executing any code. If we want to bypass any code pass statement can be used.

```
Syntax
pass
for i in [1,2,3,4,5]:
   if i==3:
      pass
      print ("Pass when value is",i)
   print (i)
1
2
```

```
Pass when value is 3
3
4
5
```

Chapter 4

Python OOPs Concepts

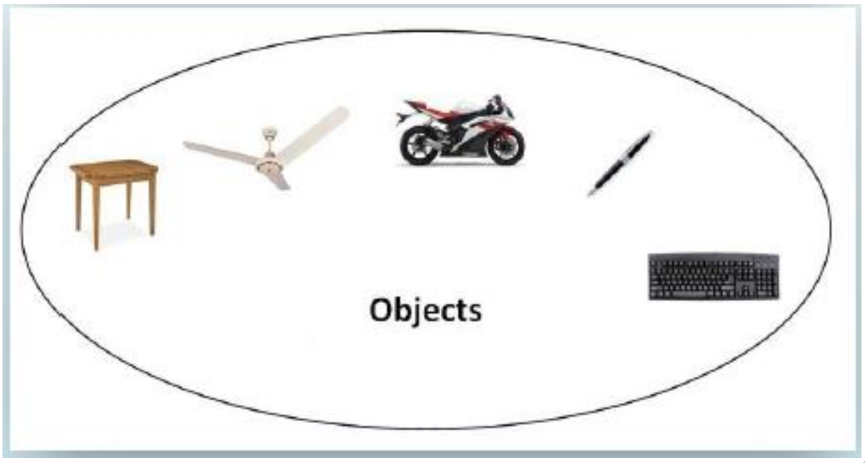
Topics Covered

4.1:-Python OOPs Concepts

- Python is an object-oriented programming language. It allows us to develop applications using Object Oriented approach. In Python, we can easily create and use classes and objects.
- Major principles of object-oriented programming system are given below
  - Object
  - Class
  - Method
  - Inheritance
  - Polymorphism
  - Data Abstraction
  - Encapsulation

4.2:-Object

- Object is an entity that has state and behavior. It may be anything. It may be physical and logical. For example: mouse, keyboard, chair, table, pen etc.
- Everything in Python is an object, and almost everything has attributes and methods. All functions have a built-in attribute doc , which returns the doc string defined in the function source code.



4.3:-Python Class

- A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation
- In Python, a class is defined by using a keyword class like a function definition begins with the keyword def.n.

```
Syntax
class ClassName:
   <statement-1>
   .
   .
   .
   <statement-N>
```

```
class Student:
   def __init__(self, name, course,branch):
      self.name = name
      self.course = course
      self.branch = branch
   def displayStudentDetails(self):
      print("Name : ", self.name, " Course: ", self.course, "Branch: ",self.branch)
             #Creating Instance Objects here
stu1 = Student("Raj", "B.Tech","CSE")
stu2 = Student("Krish", "B.E","EE")
     #Accessing Attributes
stu1.displayStudentDetails()
stu2.displayStudentDetails()
```

```
Name :  Raj  Course:  B.Tech Branch:  CSE
Name :  Krish  Course:  B.E Branch:  EE
```

4.4:-Built-In Class Attributes

Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute −

- dict__ − Dictionary containing the class's namespace.
- doc__ − Class documentation string or none, if undefined.
- name__ − Class name.
- module__ − Module name in which the class is defined. This attribute is "__main__" in interactive mode.
- bases__ − A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

4.5:-Method

Method is a function that is associated with an object. In Python, method is not unique to class instances. Any object type can have methods.

4.6:-Python Constructors

A constructor is a special type of method (function) which is used to initialize the instance members of the class. Constructor can be parameterized and non-parameterized as well. Constructor definition executes when we create object of the class. Constructors also verify that there are enough resources for the object to perform any start-up task.

4.6.1: Creating a Constructor

A constructor is a class function that begins with double underscore (_). The name of the constructor is always the same __init__().

While creating an object, a constructor can accept arguments if necessary. When we create a class without a constructor, Python automatically creates a default constructor that doesn't do anything.

In Python, Constructors can be parameterized and non-parameterized as well. The parameterized constructors are used to set custom value for instance variables that can be used further in the application.

1) Non Parameterized Constructor

Example

```
class Student:
   # Constructor - non parameterized
   def __init__(self):
      print("This is non parameterized constructor")
   def show(self, name):
      print("Hello", name)
student = Student()
student.show("Friend")
```

Output

```
This is non parameterized constructor
Hello guys.
```

2) Parameterized Constructor

Example

```
class Student:
   # Constructor - parameterized
   def __init__(self, name):
      print("This is parameterized constructor")
      self.name = name
   def show(self):
      print("Hello",self.name)
student = Student("Friend")
student.show()
```

Output

```
This is parameterized constructor
Hello Friend
```

4.7 :-Python Inheritance

Inheritance is a feature of Object Oriented Programming. It is used to specify that one class will get most or all of its features from its parent class. It is a very powerful feature which facilitates users to create a new class with a few or more modification to an existing class. The new class is called child class or derived class and the main class from which it inherits the properties is called base class or parent class.

Python supports inheritance from multiple classes, unlike other popular programming languages.

Syntax1

```
class DerivedClassName(BaseClassName):
   <statement-1>
   .
   .
   .
   <statement-N>
```

Syntax 2

```
class DerivedClassName(modulename.BaseClassName):
   <statement-1>
   .
   .
   .
   <statement-N>
```

Example

Program

```
class Student:
   def student_detail(self):
      name="Raj"
      print("Student Name: "+name)
class school(Student):
   def school_details(self):
      schoolname="DAV"
      print("School name:",schoolname)
d = school()
d.student_detail()
d.school_details()
```

Output

```
Student Name: Raj

School name: DAV
```

4.7.1 :-Python Multilevel Inheritance

Multilevel inheritance is also possible in Python like other Object Oriented programming languages. We can inherit a derived class from another derived class, this process is known as multilevel inheritance. In Python, multilevel inheritance can be done at any depth.

4.7.2:-Multilevel Inheritance Example

Program

```
class Student:
   def eat(self):
     print ('Student Name : Raj')
class school(Student):
  def bark(self):
     print ('school : DAV')
class Studentinfo(school):
   def weep(self):
      print ('D.O.B : 13/08/1999')
d=Studentinfo()
d.eat()
d.bark()
d.weep()
```

output

```
Student Name : Raj

school : DAV

D.O.B : 13/08/1999
```

4.7.3 :-Python Multiple Inheritance

Python supports multiple inheritance too. It allows us to inherit multiple parent classes. We can derive a child class from more than one base (parent) classes.

The multiderived class inherits the properties of both classes base1 and base2.

Syntax

```
class DerivedClassName(Base1, Base2, Base3):

   <statement-1>
```

Example

```
class Student(object):
  def __init__(self):
    super(Student, self).__init__()
    print("Student Name : Raj")
class school(object):
  def __init__(self):
    super(school, self).__init__()
    print("school : DAV")
class Studentinfo(school, Student):
  def __init__(self):
    super(Studentinfo, self).__init__()
    print("D.O.B : 13/08/1999")
Studentinfo()
Student Name : Raj
school : DAV
D.O.B : 13/08/1999
```

Note :-Why super () keyword

The super() method is most commonly used with __init__ function in base class. This is usually the only place where we need to do some things in a child then complete the initialization in the parent.

Example:

```
class Child(Parent):
  def __init__(self, stuff):
    self.stuff = stuff
    super(Child, self).__init__()
```

4.8 :-Polymorphism

Polymorphism is the ability to leverage the same interface for different underlying forms such as data types or classes. This permits functions to use entities of different types at different times.

For object-oriented programming in Python, this means that a particular object belonging to a articular class can be used in the same way as if it were a different object belonging to a different class.

Polymorphism allows for flexibility and loose coupling so that code can be extended and easily maintained over time.

4.8.1 : What Is Polymorphism?

is an important feature of class definition in Python that is utilized when you have commonly named methods across classes or subclasses. This allows functions to use objects of any of these polymorphic classes without needing to be aware of distinctions across the classes.

- **Polymorphism with a function:**

Example:

Create two classes: Bear and Dog, both can make a distinct sound. We then make two instances and call their action using the same method.

```
class Bear(object):
   def sound(self):
      print "Groarrr"
class Dog(object):
   def sound(self):
      print "Woof woof!"
def makeSound(animalType):
   animalType.sound()
bearObj = Bear()
dogObj = Dog()
makeSound(bearObj)
makeSound(dogObj)
```

Output

```
Groarrr
Woof woof!
```

4.9:- Data Abstraction

Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonym because data abstraction is achieved through encapsulation.

Abstraction is used to hide internal details and show only functionalities. Abstracting something means to give names to things, so that the name captures the core of what a function or a whole program does.

Method overriding

Encapsulation

Encapsulation is also the feature of object-oriented programming. It is used to restrict access to methods and variables. In encapsulation, code and data are wrapped together within a single unit from being modified by accident.

4.10:- Method Overriding,

in object-oriented programming, is a language feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes. The implementation in the subclass overrides (replaces) the implementation in the superclass by providing a method that has same name, same parameters or signature, and same return type as the method in the parent class. The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. Some languages allow a programmer to prevent a method from being overridden.

Override means having two methods with the same name but doing different tasks. It means that one of the methods overrides the other.

If there is any method in the superclass and a method with the same name in a subclass, then by executing the method, the method of the corresponding class will be executed.

```
class Rectangle():
   def __init__(self, length, breadth):
      self.length = length
      self.breadth = breadth
   def getArea(self):
      print(self.length * self.breadth, " is area of rectangle")
class Square(Rectangle):
   def __init__(self, side):
      self.side = side
      Rectangle.__init__(self, side, side)
   def getArea(self):
      print(self.side * self.side, " is area of square")
s = Square(4)
r = Rectangle(2, 4)
s.getArea()
r.getArea()
```

output

```
16 is area of square
8 is area of rectangle
```

4.11 :- Object-Oriented Vs Procedure-Oriented Programming Languages

Index	Object-oriented Programming	Procedural Programming
1.	Object-oriented programming is an problem solving approach and used where computation is done by using objects.	Procedural programming uses a list of instructions to do computation step by step.
2.	It makes development and maintenance easier.	In procedural programming, It is not easy to maintain the codes when project becomes lengthy.

3.	It simulates the real world entity. So real world problems can be easily solved through oops.	It doesn't simulate the real world. It works on step by step instructions divided in small parts called functions.
4.	It provides data hiding. so it is more secure than procedural languages. You cannot access private data from anywhere.	Procedural language doesn't provide any proper way for data binding so it is less secure.
5.	In OOP, program is divided into parts called objects.	In POP, program is divided into small parts called functions.
6.	Example of object-oriented programming languages are: C++, Java, .Net, Python, C# etc.	Example of procedural languages are: C, Fortran, Pascal, VB etc.
6.	OOP follows Bottom Up approach.	POP follows Top Down approach.
8.	OOP has access specifiers named Public, Private, Protected, etc.	POP does not have any access specifier.
9.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.	In POP, Overloading is not possible.

Chapter 5

Python Exceptions Handling

Topics Covered

- **Exceptions Handling**
- **List of Standard Exceptions**
- **Exception Handling**
- **Finally Block:**
- **Raise an Exception:**
- **Custom Exception**

5.1:- Exceptions Handling

Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.

Whenever an exception occurs the program halts the execution and thus further code is not executed. Thus exception is that error which python script is unable to tackle with.

Exception in a code can also be handled. In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

Common Exceptions

1. ZeroDivisionError: Occurs when a number is divided by zero.
2. NameError: It occurs when a name is not found. It may be local or global.
3. IndentationError: If incorrect indentation is given.
4. IOError: It occurs when Input Output operation fails.
5. EOFError: It occurs when end of the file is reached and yet operations are being performed.

The suspicious code can be handled by using the try block. Enclose the code which raises an exception inside the try block. The try block is followed except statement. It is then further followed by statements which are executed during exception and in case if exception does not occur.

Syntax:

```
try:
   malicious code
except Exception1:
   execute code
except Exception2:
   execute code
except ExceptionN:
   execute code
```

```
else:
  In case of no exception, execute the else block code.
```

Example

```
try:
  a=20/0
  print (a)
except ArithmeticError:
      print ("This statement is ArithmeticError exception ")
else:
  print ("Welcome")
```

output

```
This statement is ArithmeticError exception
```

5.2 :-List of Standard Exceptions

Sr.No.	Exception Name & Description
1	**Exception** Base class for all exceptions
2	**StopIteration** Raised when the next() method of an iterator does not point to any object.
3	**SystemExit** Raised by the sys.exit() function.
4	**StandardError** Base class for all built-in exceptions except StopIteration and SystemExit.
5	**ArithmeticError** Base class for all errors that occur for numeric calculation.
6	**OverflowError** Raised when a calculation exceeds maximum limit for a numeric type.
6	**FloatingPointError** Raised when a floating point calculation fails.
8	**ZeroDivisionError** Raised when division or modulo by zero takes place for all numeric types.

9	**AssertionError** Raised in case of failure of the Assert statement.
10	**AttributeError** Raised in case of failure of attribute reference or assignment.
11	**EOFError** Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
12	**ImportError** Raised when an import statement fails.
13	**KeyboardInterrupt** Raised when the user interrupts program execution, usually by pressing Ctrl+c.
14	**LookupError** Base class for all lookup errors.
15	**IndexError** Raised when an index is not found in a sequence.
16	**KeyError** Raised when the specified key is not found in the dictionary.
16	**NameError** Raised when an identifier is not found in the local or global namespace.
18	**UnboundLocalError** Raised when trying to access a local variable in a function or method but no value has been assigned to it.
19	**EnvironmentError** Base class for all exceptions that occur outside the Python environment.
20	**IOError** Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
21	**IOError** Raised for operating system-related errors.

22	**SyntaxError** Raised when there is an error in Python syntax.
23	**IndentationError** Raised when indentation is not specified properly.
24	**SystemError** Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
25	**SystemExit** Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
26	**TypeError** Raised when an operation or function is attempted that is invalid for the specified data type.
26	**ValueError** Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
28	**RuntimeError** Raised when a generated error does not fall into any category.
29	**NotImplementedError** Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

Explanation:

The malicious code (code having exception) is enclosed in the try block.Try block is followed by except statement. There can be multiple except statement with a single try block.

Except statement specifies the exception which occurred. In case that exception is occurred, the corresponding statement will be executed.At the last you can provide else statement. It is executed when no exception is occurred.

5.3 :- Python Exception (Except with no Exception) Example

Except statement can also be used without specifying Exception.

Syntax:

```
try:
      code
   except:
    code to be executed in case exception occurs.
  else:
    code to be executed in case exception does not occur.
```

Example

```
try:
   a=10/0
except:
   print ("Arithmetic Exception")
else:
   print ("Successfully Done")
```

output:

```
Arithmetic Exception
```

5.3.1 :- Declaring Multiple Exception in Python

Python allows us to declare multiple exceptions using the same except statement.

Syntax:

```
try:
   a=10/0
except ArithmeticError as StandardError:
   print ("Arithmetic Exception")
else:
   print ("Successfully Done")
```

Output:-

```
Arithmetic Exception
```

5.4:- Finally Block:

In case if there is any code which the user want to be executed, whether exception occurs or not then that code can be placed inside the finally block. Finally block will always be executed irrespective of the exception.

Syntax:

```
try:
   Code
finally:
   code which is must to be executed.
```

Example

```
try:
   a=10/0;
   print "Exception occurred"
finally:
   print "Code to be executed"
```

output

```
Code to be executed
Traceback (most recent call last):
  File "C:/Python3.5/noexception.py", line 2, in <module>
    a=10/0;
ZeroDivisionError: integer division or modulo by zero
```

In the above example finally block is executed. Since exception is not handled therefore exception occurred and execution is stopped.

5.5:-Raise an Exception:

You can explicitly throw an exception in Python using ?raise? statement. raise will cause an exception to occur and thus execution control will stop in case it is not handled.

Syntax:

```
raise Exception_class,<value>
```

Example

```
try:
    a = 10
    print(a)
    raise NameError("Hello")
except NameError as e:
    print("An exception occurred")
    print(e)
```

output:-

```
10
An exception occurred
Hello
Assertions:
```

An assertion is a sanity-check that you can turn on or turn off when you are done with your testing of the program.

Explanation:

i) To raise an exception, raise statement is used. It is followed by exception class name.

ii) Exception can be provided with a value that can be given in the parenthesis. (here, Hello)

iii) To access the value "as" keyword is used. "e" is used as a reference variable which stores the value of the exception.

5.6: Custom Exception:

Python has many built-in exceptions which forces your program to output an error whensomething in it goes wrong.

However, sometimes you may need to create custom exceptions that serves your purpose.In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class. Most of the built-in exceptions are also derived form this class.

Syntax:

```
>>> class CustomError(Exception):
...     pass
>>> raise CustomError
Traceback (most recent call last):
...__main__.CustomError
>>> raise CustomError("An error occurred")
Traceback (most recent call last):
__main__.CustomError: An error occurred
```

Here, we have created a user-defined exception called CustomError which is derived from the Exception class. This new exception can be raised, like other exceptions, using the raisestatement with an optional error message.

When we are developing a large Python program, it is a good practice to place all the user-defined exceptions that our program raises in a separate file. Many standard modules do this. They define their exceptions separately as exceptions.py or errors.py (generally but not always).

User-defined exception class can implement everything a normal class can do, but we generally make them simple and concise. Most implementations declare a custom base class and derive others exception classes from this base class. This concept is made clearer in the following example.

Example:

This program will ask the user to enter a number until they guess a stored number correctly. To help them figure it out, hint is provided whether their guess is greater than or less than the stored number.

```
# define Python user-defined exceptions
class Error(Exception):
  """Base class for other exceptions"""
  Pass
small"""class ValueTooSmallError(Error):
  """Raised when the input value is too
  pass
class ValueTooLargeError(Error):
  """Raised when the input value is too large"""   pass
# our main program
# user guesses a number until he/she gets it right
# you need to guess this number
number = 10
while True:
  try:
     i_num = int(input("Enter a number: "))
     if i_num < number:
        raise ValueTooSmallError
     elif i_num > number:
        raise ValueTooLargeError
     Break
  except ValueTooSmallError:
     print("This value is too small, try again!")
     print()
  except ValueTooLargeError:
     print("This value is too large, try again!")
     print()
print("Congratulations! You guessed it correctly.")
```

Output:

```
Enter a number: 20
This value is too large, try again!
Enter a number: 8
This value is too small, try again!
Enter a number: 10
Congratulations! You guessed it correctly.
```

The easiest way to think of an assertion is to liken it to a **raise-if** statement (or to be more accurate, a raise-if-not statement). An expression is tested, and if the result comes up false, an exception is raised.Assertions are carried out by the assert statement, the newest keyword to Python, introduced in version 1.5.

Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.

Chapter 6

PYTHON STRINGS

Topics Covered

- **Accessing Python Strings**
- **How to create a string in Python?**
- **How to access characters in a string?**
- **Python Strings Operators**
- **How to change or delete a string**
- **Python String Functions and Methods**

A string is a sequence of characters.

A character is simply a symbol. For example, the English language has 26 characters.

Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encoding used.

Python string is a built-in type text sequence. It is used to handle textual data in python. Python Strings are immutable sequences of Unicode points. Creating Strings are simplest and easy to use in Python.

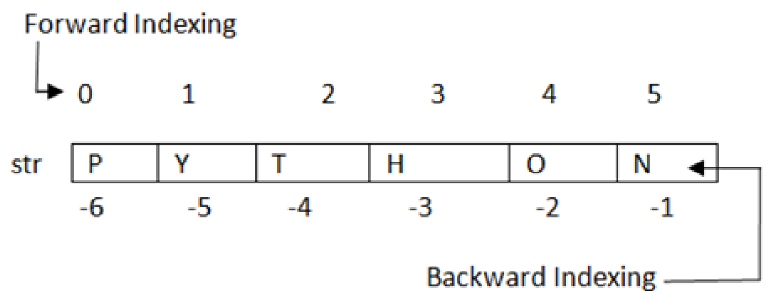
Strings are amongst the most popular types in Python. We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable

We can simply create Python String by enclosing a text in single as well as double quotes. Python treat both single and double quotes statements same.

6.1:- Accessing Python Strings

- In Python, Strings are stored as individual characters in a contiguous memory location.
- The benefit of using String is that it can be accessed from both the directions (forward and backward).
- Both forward as well as backward indexing are provided using Strings in Python.
    - Forward indexing starts with 0,1,2,3,....
    - Backward indexing starts with -1,-2,-3,-4,....

Example



```
str[0]='P'=str[-6] , str[1]='Y' = str[-5]  ,  str[2] = 'T' = str[-4]  ,  str[3] = 'H' = str[-3]
str[4] = 'O' = str[-2]  ,  str[5] = 'N' = str[-1].
```

6.2:- How to create a string in Python?

Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings

Example.

```
# all of the following are equivalent
my_string = 'Hello'
print(my_string)
my_string = "Hello"
print(my_string)
my_string = '''Hello'''
print(my_string)
# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
        the world of Python"""
print(my_string)
```

output

```
Hello
Hello
Hello
Hello, welcome to
        the world of Python
```

6.3:- How to access characters in a string?

We can access individual characters using indexing and a range of characters using slicing. Index starts from 0. Trying to access a character out of index range will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator (colon).

```
str = 'programiz'
print('str = ', str)
#first character
print('str[0] = ', str[0])
#last character
print('str[-1] = ', str[-1])
#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])
#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```

output :-

```
str =  programiz
str[0] =  p
str[-1] =  z
str[1:5] =  rogr
str[5:-2] =  am
str =  programiz
str[0] =  p
str[-1] =  z
str[1:5] =  rogr
str[5:-2] =  am
In [1]:
```

If we try to access index out of the range or use decimal number, we will get errors.

```
# index must be in range
>>> my_string[15]
IndexError: string index out of range
# index must be an integer
>>> my_string[1.5]
TypeError: string indices must be integers
```

Slicing can be best visualized by considering the index to be between the elements as shown below.

If we want to access a range, we need the index that will slice the portion from the string.

6.4:- Python Strings Operators

To perform operation on string, Python provides basically 3 types of Operators that are given below.

1. Basic Operators.

2. Membership Operators.

3. Relational Operators.

6.4.1: Python String Basic Operators

There are two types of basic operators in String "+" and "*".

1. String Concatenation Operator (+)

The concatenation operator (+) concatenates two Strings and creates a new String.

- **Python String Concatenation Example**

```
>>> "3RI" + " Technologies"
```

Output:

```
'3RI Technologies'
```

Expression	Output
'10' + '20'	'1020'
"s" + "006"	's006'
'abcd123' + 'xyz4'	'abcd123xyz4'

NOTE: Both the operands passed for concatenation must be of same type, else it will show an error.

Example:

```
>>> 'abc' + 3
```

output:

```
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    'abc' + 3
TypeError: must be str, not int
>>>
```

2. Python String Replication Operator (*)

Replication operator uses two parameters for operation, One is the integer value and the other one is the String argument.

The Replication operator is used to repeat a string number of times. The string will be repeated the number of times which is given by the integer value.

Example

```
 3*"3RI"
```

Output

```
'3RI3RI3RI'
```

Expression	Output
"3RI"*2	'3RI3RI'
3*'1'	'111'
'$'*5	'$$$$$'

We can use Replication operator in any way i.e., int * string or string * int. Both the parameters passed cannot be of same type.

6.4.2:-Python String Membership Operators

Membership Operators are already discussed in the Operators section. Let see with context of String.

- There are two types of Membership operators
  1. **in:**"in" operator returns true if a character or the entire substring is present in the specified string, otherwise false.
  2. **not in:**"not in" operator returns true if a character or entire substring does not exist in the specified string, otherwise false.

Example

```
#!/usr/bin/python
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];
if ( a in list ):
  print "Line 1 - a is available in the given list"
else:
  print "Line 1 - a is not available in the given list"
if ( b not in list ):
  print "Line 2 - b is not available in the given list"
else:
  print "Line 2 - b is available in the given list"
a = 2
if ( a in list ):
  print "Line 3 - a is available in the given list"
else:
  print "Line 3 - a is not available in the given list"
```

6.4.3:-Relational Operators

All the comparison (relational) operators i.e., (<,><=,>=,==,!=,<>) are also applicable for strings. The Strings are compared based on the ASCII value or Unicode(i.e., dictionary Order).

Example

```
>>> "RAJAT"=="RAJAT"

True

>>> "afsha">='Afsha'

True

>>> "Z"<>"z"

True
```

Explanation:

The ASCII value of a is 96, b is 98, c is 99 and so on. The ASCII value of A is 65,B is 66,C is 66 and so on. The comparison between strings are done on the basis on ASCII value.

6.5:- How to change or delete a string?

Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply reassign different strings to the same name.

```
>>> my_string = 'programiz'

>>> my_string[5] = 'a'

...TypeError: 'str' object does not support item assignment

>>> my_string = 'Python'

>>> my_string

'Python'
```

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword del.

```
>>> del my_string[1]

...TypeError: 'str' object doesn't support item deletion

>>> del my_string

>>> my_string

...NameError: name 'my_string' is not defined
```

6.6:- Python String Functions and Methods

Python provides various predefined or built-in string functions. are given below

capitalize()	It capitalizes the first character of the String.
count(string,begin,end)	It Counts number of times substring occurs in a String between begin and end index.
endswith(suffix ,begin=0,end=n)	It returns a Boolean value if the string terminates with given suffix between begin and end.

find(substring ,beginIndex, endIndex)	It returns the index value of the string where substring is found between begin index and end index.
index(subsring, beginIndex, endIndex)	It throws an exception if string is not found and works same as find() method.
isalnum()	It returns True if characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise it returns False.
isalpha()	It returns True when all the characters are alphabets and there is at least one character, otherwise False.
isdigit()	It returns True if all the characters are digit and there is at least one character, otherwise False.
islower()	It returns True if the characters of a string are in lower case, otherwise False.
isupper()	It returns False if characters of a string are in Upper case, otherwise False.
isspace()	It returns True if the characters of a string are whitespace, otherwise false.
len(string)	It returns the length of a string.
lower()	It converts all the characters of a string to Lower case.
upper()	It converts all the characters of a string to Upper Case.
startswith(str ,begin=0,end=n)	It returns a Boolean value if the string starts with given str between begin and end.
swapcase()	It inverts case of all characters in a string.
lstrip()	It removes all leading whitespace of a string and can also be used to remove particular character from leading.

rstrip()	It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.

1. String capitalize()

This method capitalizes the first character of the String.

Example

```
>>> 'abc'.capitalize()
'Abc'
```

2. String count(string) Method

Example

This method counts number of times substring occurs in a String between begin and end index.

```
    msg = "welcome to sssit"
substr1 = "o"
print  (msg.count(substr1, 4, 16))
substr2 = "t"
print  (msg.count(substr2))
```

output

```
2
2
```

3. String endswith(string) Method

This method returns a Boolean value if the string terminates with given suffix between begin and end.

```
string1 = "Welcome to 3RI Technologies"
substring1 = "3RI Technologies"
substring2 = "to"
substring3 = "of"
print(string1.endswith(substring1))
print(string1.endswith(substring2, 2, 16))
print(string1.endswith(substring3, 2, 19))
print(string1.endswith(substring3))
```

Output:

```
True
False
False
False
```

4. String find(string) Method

This method returns the index value of the string where substring is found between begin index and end index.

```
str="Welcome to 3RI Technologies"
substr1="come"
substr2="to"
print (str.find(substr1))
print (str.find(substr2))
print (str.find(substr1,3,10))
print (str.find(substr2,19))

3

8

3

-1
```

5. String index() Method

This method returns the index value of the string where substring is found between begin index and end index.

```
str="Welcome to 3RI Technologies"
substr1="come"
substr2="3RI"
print (str.index(substr1))
print (str.index(substr2))
print (str.index(substr1,3,10))
print (str.index(substr2,10))
```

output

```
3

11

3

11
```

6. String isalnum() Method

This method returns True if characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise it returns False.

```
str="Welcome to 3RI"
print (str.isalnum())
str1=("Python3")
print (str1.isalnum())
```

output

```
False

True
```

7. String isalpha() Method

Example

It returns True when all the characters are alphabets and there is at least one character, otherwise False.

```
string1="HelloPython"    # Even space is not allowed
print (string1.isalpha())
string2=("This is Python3.6.4")
print (string2.isalpha())
```

output

```
True

False
```

8. String isdigit() Method

This method returns True if all the characters are digit and there is at least one character, otherwise False.

Example

```
string1="HelloPython"
print (string1.isdigit())
string2="98564638"
print (string2.isdigit())
```

output

```
False

True
```

9. String islower() Method

This method returns True if the characters of a string are in lower case, otherwise False.

Example

```
string1="Hello Python"
print (string1.islower())
string2=("welcome to ")
print (string2.islower())
```

output

```
False

True
```

10. String isupper() Method

This method returns False if characters of a string are in Upper case, otherwise False.

```
string1="Hello Python"
print (string1.isupper())
string2="WELCOME TO 3RI"
print (string2.isupper())
```

output

```
False

True
```

11. String isspace() Method

This method returns True if the characters of a string are whitespace, otherwise false.

Example

```
string1="   "
print (string1.isspace())
string2=("WELCOME TO WORLD OF PYT")
print (string2.isspace())
```

output

```
True

False
```

12. String len(string) Method

This method returns the length of a string.

```
string1="    "

print (len(string1))

string2=("WELCOME TO python")

print (len(string2))
```

output

```
4

16
```

13. String lower() Method

It converts all the characters of a string to Lower case.

```
string1="Hello Python"
print (string1.lower())
string2="WELCOME TO 3RI"
print (string2.lower())
```

output:

```
hello python

welcome to 3ri
```

14. String upper() Method

```
string1="Hello Python"
print (string1.upper())
string2="welcome to 3RI"
print (string2.upper())
```

output

```
HELLO PYTHON

WELCOME TO 3RI
```

15. String startswith(string) Method

This method returns a Boolean value if the string starts with given str between begin and end.

Example

```
string1="Hello Python"
print (string1.startswith('Hello'))
string2="welcome to SSSIT"
print (string2.startswith('come',3,6))
```

output

```
True

True
```

16. String swapcase() Method

It inverts case of all characters in a string.

Example

```
string1="Hello Python"
print (string1.swapcase())
string2="welcome to 3RI"
print (string2.swapcase())
```

output

```
hELLO pYTHON

WELCOME TO 3ri
```

17. String lstrip() Method

It removes all leading whitespace of a string and can also be used to remove particular character from leading.

Example

```
string1="    Hello Python"
print (string1.lstrip())
string2="@@@@@@@@welcome to 3RI"
print (string2.lstrip('@'))
```

output

```
Hello Python

welcome to 3RI
```

18. String rstrip() Method

It removes all trailing whitespace of a string and can also be used to remove particular character from trailing.

Example

```
string1="    Hello Python     "
print (string1.rstrip())
string2="@welcome to Technologies!!!"
print (string2.rstrip('!'))
```

output

```
    Hello Python

@welcome to Technologies
```

Chapter - 7

Python List

Topics Covered

- **How to create a list?**
- **How to access elements from a list?**
- **List Index**
- **Elements in a Lists:**
- **Python List basic Operations**
- **Indexing, Slicing and Matrixes**
- **Python List Other Operations**
- **Built-in List Functions and Methods**

A **list** is a data structure in Python that is a mutable, or changeable, ordered sequence of elements. Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ].

The most basic data structure in Python is the sequence. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.

The list is the most versatile datatype available in Python, which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that the items in a list need not be of the same type.

It works as a container that holds other objects in a given order. We can perform various operations like insertion and deletion on list.

A list can be composed by storing a sequence of different type of values separated by commas.

When thinking about Python lists and other data structures that are types of collections, it is useful to consider all the different collections you have on your computer: your assortment of files, your song playlists, your browser bookmarks, your emails, the collection of videos you can access on a streaming service, and more.

- **Python List Syntax**

```
<list_name>=[value1,value2,value3,...,valuen];
```

7.1:- How to create a list?

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

```
# empty list
my_list = []
# list of integers
my_list = [1, 2, 3]
```

```
# list with mixed datatypes
my_list = [1, "Hello", 3.4]
```

Also, a list can even have another list as an item. This is called nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

7.2:- How to access elements from a list?

Syntax to Access Python List

```
<list_name>[index]
```

Python allows us to access value from the list by various ways.

7.3:- List Index

We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

Trying to access an element other that this will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.

Python Accessing List Elements Example

```
data1=[1,2,3,4]
data2=['x','y','z']
print (data1[0])
print (data1[0:2])
print (data2[-3:-1])
print (data1[0:])
print (data2[:2])
```

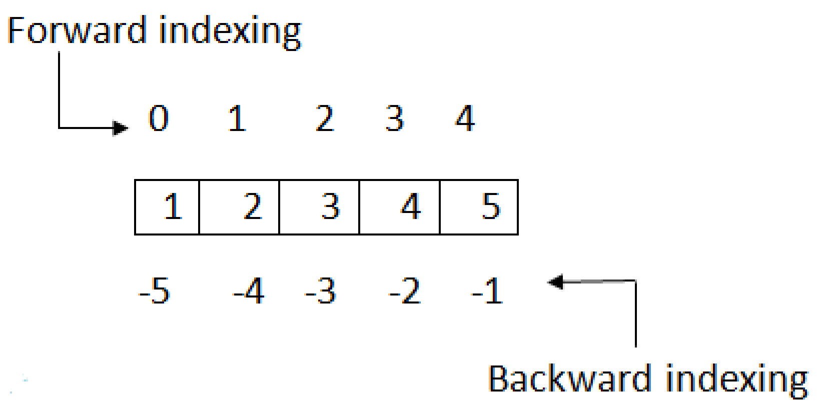
output

```
1
[1, 2]
['x', 'y']
[1, 2, 3, 4]
['x', 'y']
```

7.4:- Elements in a Lists:

Following are the pictorial representation of a list. We can see that it allows to access elements from both end (forward and backward).

```
Data=[1,2,3,4,5];
```



Data[0]=1=Data[-5] , Data[1]=2=Data[-4] , Data[2]=3=Data[-3] ,

=4=Data[-2] , Data[4]=5=Data[-1].

Note: Internal Memory Organization:
List do not store the elements directly at the index. In fact a reference is stored at each index which subsequently refers to the object stored somewhere in the memory. This is due to the fact that some objects may be large enough than other objects and hence they are stored at some other memory location.

7.5:- Python List basic Operations

Apart from creating and accessing elements from the list, Python allows us to perform various other operations on the list. Some common operations are given below

In fact, lists respond to all of the general sequence operations we used on strings in the prior chapter.

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1,2,3] : print (x,end = ' ')	1 2 3	Iteration

7.6:- Indexing, Slicing and Matrixes

A subpart of a list can be retrieved on the basis of index. This subpart is known as list slice. This feature allows us to get sub-list of specified start and end index.

Example

```
list1=[1,2,4,5,6]
print (list1[0:2])
print (list1[4])
list1[1]=9
print (list1)
```

Output:

```
[1, 2]

6

[1, 9, 4, 5, 6]
```

Note: If the index provided in the list slice is outside the list, then it raises an IndexError exception.

7.7:- Python List Other Operations

Python allows various other operations on List such as Updating, Appending and Deleting elements from a List.

7.7.1:- Updating List

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append () method.

Example

```
list = ['Krish', 'Roshan', 1996, 2000]
print ("Value available at index 2 : ", list[2])
list[2] = 2001
print ("New value available at index 2 : ", list[2])
```

output:

```
Value available at index 2 :  1996

New value available at index 2 :  2001
```

7.7.2:- Appending Python List

Python provides, append() method which is used to append i.e., add an element at the end of the existing elements.

Example

```
list1=[3,"3RI",'Technologies']
print ("Elements of List are: ")
print (list1)
list1.append(15.2)
print ("List after appending: ")
print (list1)
```

output

```
Elements of List are:

[3, '3RI', 'Technologies']

List after appending:

[3, '3RI', 'Technologies', 15.2]
```

7.7.3:- Deleting Elements

In Python, del statement can be used to delete an element from the list. It can also be used to delete all items from startIndex to endIndex.

Example

```
list1 = ['Krish', 'Roshan', 1999, 2000]
print (list1)
del list1[2]
print ("After deleting value at index 2 : ")
print (list1)
```

output

```
['Krish', 'Roshan', 1999, 2000]

After deleting value at index 2 :
```

['Krish', 'Roshan', 2000]

7.8:- Built-in List Functions and Methods

Python provides various Built-in functions and methods for Lists that we can apply on the list.

7.8.1:-Following are the common list functions.

Function	Description
min(list)	It returns the minimum value from the list given.
max(list)	It returns the largest value from the given list.
len(list)	It returns number of elements in a list.
cmp(list1,list2)	It compares the two list.
list(sequence)	It takes sequence types and converts them to lists.

1. List min() method

This method is used to get min value from the list.

Syntax :-

```
    min(list)
```

Example

```
list1, list2 = ['C++','Java', 'Python'], [456, 600, 200]

print ("min value element : ", min(list1))

print ("min value element : ", min(list2))
```

output

```
min value element :  C++

min value element :  200
```

2. List len() method

The **len()** method returns the number of elements in the list.

Syntax :-

```
len(list
```

Example

```
list1 = ['physics', 'chemistry', 'maths']
print (len(list1))
list2 = list(range(5)) #creates list of numbers between 0-4
print (len(list2))
```

output

```
3
5
```

3. List max() Method

The **max()** method returns the elements from the list with maximum value.

Syntax

```
    max(list)
```

Example

```
list1, list2 = ['C++','Java', 'Python'], [456, 600, 200]
print ("Max value element : ", max(list1))
print ("Max value element : ", max(list2))
```

4. List cmp() Method

Explanation: If elements are of the same type, perform the comparison and return the result. If elements are different types, check whether they are numbers.

- If numbers, perform comparison.
- If either element is a number, then the other element is returned.
- Otherwise, types are sorted alphabetically .

Syntax

```
cmp(list1, list2)
```

Example

```
list1, list2 = [123, 'xyz'], [456, 'abc']
print cmp(list1, list2)
print cmp(list2, list1)
list3 = list2 + [686];
print cmp(list2, list3)
```

output

```
-1
1
-1
```

5. List list() Method

- The **list()** method takes sequence types and converts them to lists. This is used to convert a given tuple into list.
- **Note** – Python Tuple are very similar to lists with only difference that element values of a tuple can not be changed and tuple elements are put between parentheses instead of square bracket. This function also converts characters in a string into a list.

Syntax

```
list( seq )
```

Example

```
#!/usr/bin/python3
aTuple = (123, 'php', 'Java', 'Python')
list1 = list(aTuple)
print ("List elements : ", list1)
str="Hello World"
list2=list(str)
print ("List elements : ", list2)
```

output

```
List elements :  [123, 'php', 'Java', 'Python']
List elements :  ['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd']
```

7.8.2:-There are following built-in methods of List

Methods	Description
index(object)	It returns the index value of the object.
count(object)	It returns the number of times an object is repeated in list.
pop()/pop(index)	It returns the last object or the specified indexed object. It removes the popped object.
insert(index,object)	It inserts an object at the given index.

extend(sequence)	It adds the sequence to existing list.
remove(object)	It removes the object from the given List.
reverse()	It reverses the position of all the elements of a list.
sort()	It is used to sort the elements of the List.

Chapter 8

Python Tuple

Topics Covered

A tuple is a sequence of immutable objects, therefore tuple cannot be changed. It can be used to collect different types of object.

The objects are enclosed within parenthesis and separated by comma.

Tuple is similar to list. Only the difference is that list is enclosed between square bracket, tuple between parenthesis and List has mutable objects whereas Tuple has immutable objects.

Example

```
data=(10,20,'ram',56.8)
data2 = "a", 10, 20.9
print( data)
print(data2
```

output:

```
(10, 20, 'ram', 56.8)

('a', 10, 20.9)
```

NOTE: If Parenthesis is not given with a sequence, it is by default treated as Tuple.

8.1:- Accessing Tuple

Accessing of tuple is prity easy, we can access tuple in the same way as List. See, the following example.

```
tup1 = ('3RI', 'Technologies', 2000, 2018)

tup2 = (1, 2, 3, 4, 5, 6, 6 )

print ("tup1[0]: ", tup1[0])

print ("tup2[1:5]: ", tup2[1:5])
```

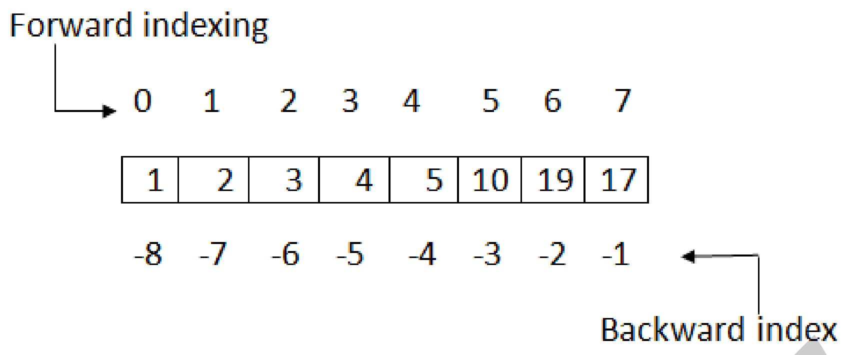
output

```
tup1[0]:  3RI

tup2[1:5]:  (2, 3, 4, 5)
```

8.2:- Elements in a Tuple

Data=(1,2,3,4,5,10,19,17)



```
Data[0]=1=Data[-8] , Data[1]=2=Data[-6] , Data[2]=3=Data[-6] ,
   Data[3]=4=Data[-5] , Data[4]=5=Data[-4] , Data[5]=10=Data[-3],
Data[6]=19=Data[-2],Data[6]=16=Data[-1]
```

8.3:- Basic Tuples Operations

Python allows us to perform various operations on the tuple. Following are the common tuple operations.

The Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1,2,3) : print (x, end = ' ')	1 2 3	Iteration

8.4:- Python Tuple Slicing

A subpart of a tuple can be retrieved on the basis of index. This subpart is known as tuple slice.

Example

```
data1=(1,2,4,5,6)
print (data1[0:2])
print (data1[4])
print (data1[:-1])
print (data1[-5:])
print (data1)
```

Output

```
(1, 2)

6

(1, 2, 4, 5)

(1, 2, 4, 5, 6)

(1, 2, 4, 5, 6)
```

8.5:- Python Tuple other Operations

8.5.1:-Updating elements in a List

Elements of the Tuple cannot be updated. This is due to the fact that Tuples are immutable. Whereas the Tuple can be used to form a new Tuple.

Example

```
data=(10,20,30)
data[0]=100
print (data)
```

8.5.2:-Creating Tuple from Existing Example

We can create a new tuple by assigning the existing tuple, see the following example.

```
>>> data1=(10,20,30)

>>> data2=(40,50,60)

>>> data3=data1+data2

>>> print(data3)
```

Output

```
>>>

(10, 20, 30, 40, 50, 60)

>>>
```

8.5.3:-Python Tuple Deleting

Deleting individual element from a tuple is not supported. However the whole of the tuple can be deleted using the del statement

```
data=(10,20,'rahul',40.6,'z')

print data

del data      #will delete the tuple data

print data  #will show an error since tuple data is already deleted
```

Output

```
>>>

(10, 20, 'rahul', 40.6, 'z')

Traceback (most recent call last):
    File "C:/Python26/t.py", line 4, in
    print data
NameError: name 'data' is not defined
>>>
```

8.6:- Built-in Functions of Tuple

There are following in-built Type Functions

Function	Description
min(tuple)	It returns the minimum value from a tuple.
max(tuple)	It returns the maximum value from the tuple.
len(tuple)	It gives the length of a tuple
cmp(tuple1,tuple2)	It compares the two Tuples.
tuple(sequence)	It converts the sequence into tuple.

1. Tuple max(tuple) Method

This method is used to get max value from the sequence of tuple.

Example

```
tuple1, tuple2 = ('maths', 'che', 'phy', 'bio'), (456, 600, 200)
print ("Max value element : ", max(tuple1))
print ("Max value element : ", max(tuple2))
```

output

```
Max value element :  phy
Max value element :  600
```

2. Tuple len(tuple) Method

This method is used to get length of the tuple.

Example

```
tuple1, tuple2 = (123, 'xyz', 'zara'), (456, 'abc')
print ("First tuple length : ", len(tuple1))
print ("Second tuple length : ", len(tuple2))
```

output:

```
First tuple length :  3
Second tuple length :  2
```

8.7:- Advantages of Tuple

1. Processing of Tuples are faster than Lists.
2. It makes the data safe as Tuples are immutable and hence cannot be changed.
3. Tuples are used for String formatting.

Chapter 9

Python Dictionary

Topics Covered

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.Dictionaries are optimized to retrieve values when the key is known.

Example

```
data={100:'Ravi' ,101:'Vijay' ,102:'Rahul'}
print (data)
```

output

```
{100: 'Ravi', 101: 'Vijay', 102: 'Rahul'}
```

Note:

Dictionary is mutable i.e., value can be updated.

Key must be unique and immutable. Value is accessed by key. Value can be updated while key cannot be changed.

Dictionary is known as Associative array since the Key works as Index and they are decided by the user.

9.1:- Accessing Dictionary Values

Since Index is not defined, a Dictionary values can be accessed by their keys only. It means, to access dictionary elements we need to pass key, associated to the value.

9.2:- Python Accessing Dictionary Element

Syntax

```
<dictionary_name>[key]
</dictionary_name>
```

Example

```
dict = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
print ("dict['Name']: ", dict['Name'])
print ("dict['Type']: ", dict['Type'])
```

output:

```
dict['Name']:  3RI
```

```
dict['Type']:  it training company
```

9.3:- Updating Dictionary Elements

The item i.e., key-value pair can be updated. Updating means new item can be added. The values can be modified.

Example

```
dict = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
dict['Type'] = 'software training company'; # update existing entry
dict['location'] = "Mumbai" # Add new entry
print ("dict['Type']: ", dict['Type'])
print ("dict['location']: ", dict['location'])
```

output:

```
dict['Type']:  software training company
dict['location']:  Mumbai
```

9.4:- Deleting Dictionary Elements

del statement is used for performing deletion operation.An item can be deleted from a dictionary using the key only.

Syntax

```
del  <dictionary_name>[key]
</dictionary_name>
```

Example

```
data={100:'Raj', 101:'Suraj', 102:'krish'}
del data[102]
print (data)
del data
print (data)   #will show an error since dictionary is deleted.
```

Output

```
{100: 'Raj', 101: 'Suraj'}
Traceback (most recent call last):
File "C:/Python3.6/dict.py", line 5, in
   print (data)   #will show an error since dictionary is deleted.
NameError: name 'data' is not defined
```

9.5:- Python Dictionary Functions and Methods

9.5.1:-Python Dictionary Functions

Functions	Description

len(dictionary)	It returns number of items in a dictionary.
cmp(dictionary1,dictionary2)	It compares the two dictionaries.
str(dictionary)	It gives the string representation of a string.

9.5.2:-Python Dictionary Methods

Methods	Description
keys()	It returns all the keys element of a dictionary.
values()	It returns all the values element of a dictionary.
items()	It returns all the items(key-value pair) of a dictionary.
update(dictionary2)	It is used to add items of dictionary2 to first dictionary.
clear()	It is used to remove all items of a dictionary. It returns an empty dictionary.
fromkeys(sequence,value1)/ fromkeys(sequence)	It is used to create a new dictionary from the sequence where sequence elements forms the key and all keys share the values ?value1?. In case value1 is not give, it set the values of keys to be none.
copy()	It returns an ordered copy of the data.
has_key(key)	It returns a boolean value. True in case if key is present in the dictionary ,else false.
get(key)	It returns the value of the given key. If key is not present it returns none.

1. Dictionary len(dictionary)

The method len() gives the total length of the dictionary. This would be equal to the number of items in the dictionary.

Example

```
dict = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
```

```
print ("dictionary Length is : %d" % len (dict))
```

output:

```
dictionary Length is : 3
```

2. Python Dictionary str(dictionary)

The method **str()** produces a printable string representation of a dictionary.

Example:

```
dict = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
print ("Equivalent String : %s" % str (dict))
```

output:

```
Equivalent String : {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
```

3. Python Dictionary keys() Method

This method returns all the keys element of a dictionary.

Example:

```
dict = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
print ("Value : %s" %  dict.keys())
```

output:

```
Value : dict_keys(['Name', 'Type', 'location'])
```

4. Dictionary values() Method

This method returns all the values element of a dictionary.

Example:

```
dict = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
print ("Values : ",  list(dict.values()))
```

output:

```
Values :  ['3RI', 'it training company', 'pune']
```

5. Dictionary items() Method

This method returns all the items(key-value pair) of a dictionary.

Example:

```
dict = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
print ("Value : %s" %  dict.items())
```

output:

```
Value : dict_items([('Name', '3RI'), ('Type', 'it training company'), ('location', 'pune')])
```

6. Dictionary update(dictionary2) Method

This method is used to add items of dictionary2 to first dictionary.

Example:

```
dict = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
dict2 = {'location': 'Mumbai' }
dict.update(dict2)
print ("updated dict : ", dict)
```

output:

```
updated dict :  {'Name': '3RI', 'Type': 'it training company', 'location': 'Mumbai'}
```

7. Python Dictionary clear() Method

The method **clear()** removes all items from the dictionary.

Example:

```
dict = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
print ("Start Len : %d" %  len(dict))
dict.clear()
print ("End Len : %d" %  len(dict))
```

output:

```
Start Len : 3
End Len : 0
```

8. Dictionary fromkeys(sequence)/ fromkeys(seq,value) Method

This method is used to create a new dictionary from the sequence where sequence elements forms the key and all keys share the values ?value1?. In case value1 is not give, it set the values of keys to be none.

Example:

```
seq = ('name', 'age', 'gender')
dict = dict.fromkeys(seq)
print ("New Dictionary : %s" %  str(dict))
dict = dict.fromkeys(seq, 10)
print ("New Dictionary : %s" %  str(dict))
```

output:

```
New Dictionary : {'name': None, 'age': None, 'gender': None}
New Dictionary : {'name': 10, 'age': 10, 'gender': 10}
```

9. Dictionary copy() Method

The method **copy()** returns a shallow copy of the dictionary.

Example:

```
dict1 = {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}

dict2 = dict1.copy()

print ("New Dictionary : ",dict2)
```

output:

```
New Dictionary :  {'Name': '3RI', 'Type': 'it training company', 'location': 'pune'}
```

10. Dictionary has_key(key) Method

The method **has_key()** returns true if a given *key* is available in the dictionary, otherwise it returns a false.

11. Dictionary get(key) Method

This method returns the value of the given key. If key is not present it returns none.

Example

```
dict = {'Name': '3RI', 'Type': 'it training company'}

print ("Value : %s" %  dict.get('Name'))

print ("Value : %s" %  dict.get('location', "NA"))
```

Output:

```
Value : 3RI

Value : NA
```

Chapter 10

Python Functions

Topics Covered

A function is a self block of code which is used to organize the functional code. Functions provide better modularity for your application and a high degree of code reusing.

Function can be called as a section of a program that is written once and can be executed whenever required in the program, thus making code reusability.

10.1:- Types of Functions:

- **There are two basic types of functions:**

I. **Built-in Functions:** Functions that are predefined and organized into a library. We have used many predefined functions in Python.

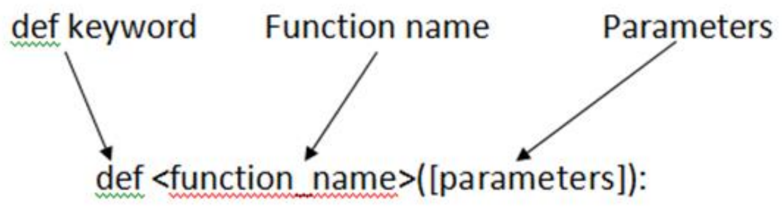
II. **User- Defined:** Functions that are created by the programmer to meet the requirements.

10.2:- Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def.** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- Parameters are passed inside the parenthesis. At the end a colon is marked.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return none.

Syntax



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

Example

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

- Python code requires indentation (space) of code to keep it associate to the declared block.
- The first statement of the function is optional. It is ?Documentation string? of function.

10.3:- Calling a Function

Defining a function gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

Example 1

```
# Function definition is here
def printme( str ):
  "This prints a passed string into this function"
  print (str)   return
# Now you can call printme function
printme("This is first call to the user defined function!")
printme("Again second call to the same function")
```

output:

```
This is first call to the user defined function!
Again second call to the same function
```

Example 2

```
# Function definition is here
```

```
def sum(a,b ):
  "This prints a passed info into this function"
  print ("Sum of a and b ",a+b)
  return
# Now you can call printinfo function
sum( a = 50, b = 5)
```

output:

```
Sum of a and b 55
```

10.4:- Function Argument and Parameter

There can be two types of data passed in the function.

1) The First type of data is the data passed in the function call. This data is called ?arguments?.
2) The second type of data is the data received in the function definition. This data is called ?parameters?.

Arguments can be literals, variables and expressions. Parameters must be variable to hold incoming values.

Alternatively, arguments can be called as actual parameters or actual arguments and parameters can be alled as formal parameters or formal arguments.

Example

```
def addition(x, y):
   print(x + y)
x = 15
addition(x, 10)
addition(x, x)
y = 20
addition(x, y)
```

Output:

```
25
30
35
```

10.5:- Passing Parameters

Python supports tree types of formal argument:

1) Positional argument (Required argument).
2) Default argument.
3) Keyword argument (Named argument)

10.6:- Positional/Required Arguments:

When the function call statement must match the number and order of arguments as defined in the function definition. It is Positional Argument matching.

Example

# Function definition is here

```
def sum(a, b):
   "Function having two parameters"
   c = (a + b)
   print(c)
sum(10, 20)
sum(20)
```

output:

```
30
Traceback (most recent call last):
  File "C:/ Python26/ function.py", line 16, in <module>
    sum(20)
TypeError: sum() missing 1 required positional argument: 'b'
```

Explanation:

1) In the first case, when sum() function is called passing two values i.e., 10 and 20 it matches with function definition parameter and hence 10 and 20 is assigned to a and b respectively. The sum is calculated and printed.
2) In the second case, when sum() function is called passing a single value i.e., 20 , it is passed to function definition. Function definition accepts two parameters whereas only one value is being passed, hence it will show an error.

10.6.1:-Function Default Arguments

Default Argument is the argument which provides the default values to the parameters passed in the function definition, in case value is not provided in the function call default value is used.

Example

```
# Function definition is here
def printinfo( name, age = 35 ):
  "This prints a passed info into this function"
  print ("Name: ", name)
  print ("Age ", age)
  return
# Now you can call printinfo function
printinfo( age = 50, name = "miki" )
printinfo( name = "miki"
```

Output:

```
Name:  miki
Age  50
Name:  miki
```

```
Age  35
```

10.6.2: Variable-length Arguments

You may need to process a function for more arguments than you specified while defining the function. These arguments are called variable-lengtharguments and are not named in the function definition, unlike required and default arguments.

Syntax:

```
def functionname([formal_args,] *var_args_tuple ):
  "function_docstring"
  function_suite
  return [expression]
```

An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function.

```
# Function definition is here
def printinfo( arg1, *vartuple ):
  "This prints a variable passed arguments"
  print ("Output is: ")
  print (arg1)
  for var in vartuple:
    print (var)
  return
# Now you can call printinfo function
printinfo( 10 )
printinfo( 60, 60, 50 )
```

10.7:- Anonymous Function

hese functions are called anonymous because they are not declared in the standard manner by using the **def** keyword. You can use the **lambda** keyword to create small anonymous functions.

- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression.
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
- Although it appears that lambdas are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is to stack allocation by passing function, during invocation for performance reasons.

Syntax

- **lambda** arg1,args2,args3,?,argsn :expression

Example

```
#Function Definiton
square=lambda x1: x1*x1
#Calling square as a function
print ("Square of number is",square(10))
```

output:

```
Square of number is 100
```

10.8:- Difference between Normal Functions and Anonymous Function:

Example:

- Normal function:

```
def square(x):
   return x * x
# Calling square function
print("Square of number is", square(10))
```

10.8.1:-Anonymous function:

```
# Function Definiton
square = lambda x1: x1 * x1
# Calling square as a function
print("Square of number is", square(10))
```

Explanation:

Anonymous is created without using def keyword.lambda keyword is used to create anonymous function.It returns the evaluated expression.

10.9:- Scope of Variables

Scope of a variable can be determined by the part in which variable is defined. Each variable cannot be accessed in each part of a program. There are two types of variables based on Scope:

1) Local Variable.
2) Global Variable.

10.9.1:-Python Local Variables

Variables declared inside a function body is known as Local Variable. These have a local access thus these variables cannot be accessed outside the function body in which they are declared.

Example

```
def msg():
   a = 10
   print("Value of a is", a)
   return
msg()
print(a)  # it will show error since variable is local
```

output:

```
Value of a is 10
Traceback (most recent call last):
  File "C:/Python26/lam.py", line 6, in <module>
   print a #it will show error since variable is local
NameError: name 'a' is not defined
```

10.9.2:-Python Global Variable

Variable defined outside the function is called Global Variable. Global variable is accessed all over program thus global variable have widest accessibility.

Example

```
def msg():
   a = 10
   print("Value of a is", a)
   print("Value of b is", b)
   return
msg()
print(b)
```

output:

```
Value of a is 10
Value of b is 20
20
```

Chapter 11

Python Input And Output

Topics Covered

- **Python "print" Statement**
- **Input from Keyboard:**
- **Opening and Closing Files**
- **Program to read and write data from a file**
- **Attributes of File:**
- **Modes of File:**
- **Methods:**

Python provides methods that can be used to read and write data. Python also provides supports of reading and writing data to Files.

11.1:- Python "print" Statement

"print" statement is used to print the output on the screen.

print statement is used to take string as input and place that string to standard output.

Whatever you want to display on output place that expression inside the inverted commas. The expression whose value is to printed place it without inverted commas.

Syntax:

```
print "expression" or print expression.
```

Example

```
print ("Python is really a great language,", "isn't it?")
```

output:

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

11.2:- Input from Keyboard:

Python 2 has two built-in functions to read data from standard input, which by default comes from the keyboard. These functions are **input()** and **raw_input()**

In Python 3, raw_input() function is deprecated. Moreover, input() functions read data from keyboard as string, irrespective of whether it is enclosed with quotes ('' or "") or not.

- **The input Function**

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

Example

```
n=input("Enter your expression ")
print ("The evaluated expression is ", n)
```

output:

```
Enter your expression 10

The evaluated expression is  10
```

11.3:- Opening and Closing Files

Python provides the facility of working on Files. A File is an external storage on hard disk from where data can be stored and retrieved.

Operations on Files:

1) **Opening a File:** Before working with Files you have to open the File. To open a File, Python built in function open() is used. It returns an object of File which is used with other functions. Having opened the file now you can perform read, write, etc. operations on the File.

Syntax:

```
obj=open(filename , mode , buffer)
```

here,

filename: It is the name of the file which you want to access.

mode: It specifies the mode in which File is to be opened.There are many types of mode. Mode depends the operation to be performed on File. Default access mode is read.

2) **Closing a File:**Once you are finished with the operations on File at the end you need to close the file. It is done by the close() method. close() method is used to close a File.

Syntax:

```
fileobject.close
```

3) **Writing to a File:**write() method is used to write a string into a file.The write() method does not add a newline character ('\n') to the end of the string −

Syntax:

```
fileObject.write(string);
```

Here, passed parameter is the content to be written into the opened file.

Example

```
# Open a file

fo = open("foo.txt", "w")

fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opend file

fo.close()
```

4) **Reading from a File:**read() method is used to read data from the File.

Syntax:

```
fileobject.read(value)
```

11.4:- Program to read and write data from a file.

```
obj=open("abcd.txt","w")
obj.write("Welcome to the world of Python")
obj.close()
```

```
obj1=open("abcd.txt","r")
s=obj1.read()
print (s)
obj1.close()
obj2=open("abcd.txt","r")
s1=obj2.read(20)
print (s1)
obj2.close()
```

Output:

```
Welcome to the world of Python

Welcome to the world
```

11.5:- Attributes of File:

There are following File attributes.

Attribute	Description
Name	Returns the name of the file.
Mode	Returns the mode in which file is being opened.
Closed	Returns Boolean value. True, in case if file is closed else false.

Example

```
obj = open("data.txt", "w")
print(obj.name)
print(obj.mode)
print(obj.closed)
```

output:

```
data.txt

w

False
```

11.6:- Modes of File:

There are different modes of file in which it can be opened.

1) Text Mode.
2) Binary Mode.

Mode	Description
R	It opens in Reading mode. It is default mode of File. Pointer is at beginning of the file.

rb	It opens in Reading mode for binary format. It is the default mode. Pointer is at beginning of file.
r+	Opens file for reading and writing. Pointer is at beginning of file.
rb+	Opens file for reading and writing in binary format. Pointer is at beginning of file.
W	Opens file in Writing mode. If file already exists, then overwrite the file else create a new file.
wb	Opens file in Writing mode in binary format. If file already exists, then overwrite the file else create a new file.
w+	Opens file for reading and writing. If file already exists, then overwrite the file else create a new file.
wb+	Opens file for reading and writing in binary format. If file already exists, then overwrite the file else create a new file.
a	Opens file in Appending mode. If file already exists, then append the data at the end of existing file, else create a new file.
ab	Opens file in Appending mode in binary format. If file already exists, then append the data at the end of existing file, else create a new file.
a+	Opens file in reading and appending mode. If file already exists, then append the data at the end of existing file, else create a new file.
ab+	Opens file in reading and appending mode in binary format. If file already exists, then append the data at the end of existing file, else create a new file.

11.7:- Methods:

here are many methods related to File Handling.

Note: There is a module "os" defined in Python that provides various functions which are used to perform various operations on Files. To use these functions 'os' needs to be imported.

Method	Description

rename()	It is used to rename a file. It takes two arguments, existing_file_name and new_file_name.
remove()	It is used to delete a file. It takes one argument. Pass the name of the file which is to be deleted as the argument of method.
mkdir()	It is used to create a directory. A directory contains the files. It takes one argument which is the name of the directory.
chdir()	It is used to change the current working directory. It takes one argument which is the name of the directory.
getcwd()	It gives the current working directory.
rmdir()	It is used to delete a directory. It takes one argument which is the name of the directory.
tell()	It is used to get the exact position in the file.

1. rename():

Syntax:

```
os.rename(existing_file_name, new_file_name)
```

Example:

```
#!/usr/bin/python3
import os
# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

2. remove():

Syntax:

```
os.remove(file_name)
```

Example:

```
#!/usr/bin/python3
import os
# Delete file test2.txt
os.remove("text2.txt")
```

3. mkdir()

Syntax:

```
os.mkdir("newdir"
```

Example

```
#!/usr/bin/python3
import os
# Create a directory "test"
os.mkdir("test")
```

4. chdir()

Syntax:

```
os.chdir("newdir")
```

Example

```
#!/usr/bin/python3
import os
# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```

5. getcwd()

Syntax:

```
os.getcwd()
```

Example

```
#!/usr/bin/python3
import os
# This would give location of the current directory
os.getcwd()
```

6. rmdir()

Syntax:

```
os.rmdir('dirname')
```

Example

```
#!/usr/bin/python3
import os
# This would  remove "/tmp/test"  directory.
os.rmdir( "/tmp/test"  )
```

NOTE : In order to delete a directory, it should be empty. In case directory is not empty first delete the files.

Chapter 12

Python Module

Topics Covered

12.1:- Python Module

Modules are used to categorize Pyhton code into smaller parts. A module is simply a Python file, where classes, functions and variables are defined. Grouping similar code into a single file makes it easy to access. Have a look at below example.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

If the content of a book is not indexed or categorized into individual chapters, the book might have turned boring and hectic. Hence, dividing book into chapters made it easy to understand.

In the same sense python modules are the files which have similar code. Thus module is simplify a python code where classes, variables and functions are defined.

Example

The Python code for a module named aname normally resides in a file namedaname.py.

```
def print_func( par ):
  print "Hello : ", par
  return
```

12.2:- Python Module Advantage

Python provides the following advantages for using module:

1) **Reusability:** Module can be used in some other python code. Hence it provides the facility of code reusability.

2) **Categorization:** Similar type of attributes can be placed in one module.

12.3:- Importing a Module:

There are different ways by which you we can import a module. These are as follows:

Using import statement:

"import" statement can be used to import a module.

Syntax:

```
import <file_name1, file_name2,...file_name(n)="">
</file_name1,>
```

Example

```
def add(a,b):
   c=a+b
   print c
   return
```

Save the file by the name addition.py. To import this file "import" statement is used.

```
import addition
\
addition.add(20,20)
addition.add(10,40)
```

Create another python file in which you want to import the former python file. For that, import statement is used as given in the above example. The corresponding method can be used by file_name.method (). (Here, addition. add (), where addition is the python file and add () is the method defined in the file addition.py)

Output:

```
40
50
```

NOTE: You can access any function which is inside a module by module name and function name separated by dot. It is also known as period. Whole notation is known as dot notation.

12.4:- Python Importing Multiple Modules

1) **msg.py:**

```
def msg_method():
   print ("Today the weather is rainy"  )
   return
```

2) **display.py:**

```
def display_method():
   print ("The weather is Sunny")
   return
```

3) **multiimport.py:**

```
import msg,display
msg.msg_method()
display.display_method()
```

output:

```
Today the weather is rainy
The weather is Sunny
```

12.5:- Using from.. import statement:

from..import statement is used to import particular attribute from a module. In case you do not want whole of the module to be imported then you can use from ?import statement.

Syntax:

```
from  <module_name> import <attribute1,attribute2,attribute3,...attributen>
</attribute1,attribute2,attribute3,...attributen></module_name>
```

12.5.1:-from.. import

Example

```
def circle(r):
   print 3.14*r*r
   return
def square(l):
   print l*l
   return
def rectangle(l,b):
   print l*b
   return
def triangle(b,h):
   print 0.5*b*h
   return
```

- **area1.py**

```
from area import square,rectangle
square(10)
rectangle(2,5)
```

Output:

```
100
10
```

12.5.2:- The from...import * Statement

You can import whole of the module using "from? import *"

Syntax:

```
from <module_name> import *
</module_name>
```

Using the above statement all the attributes defined in the module will be imported and hence you can access each attribute.

1) area.py

Same as above example

2) area1

```
from area import *
square(10)
rectangle(2,5)
circle(5)
triangle(10,20)
```

output:

```
100
10
68.5
100.0
```

12.6:- Built in Modules in Python:

There are many built in modules in Python. Some of them are as follows:

math, random , threading , collections , os , mailbox , string , time , tkinter etc..

Each module has a number of built in functions which can be used to perform various functions.

1) math:

Using math module , you can use different built in mathematical functions.

12.7:- Functions:

Function	Description
ceil(n)	It returns the next integer number of the given number
sqrt(n)	It returns the Square root of the given number.
exp(n)	It returns the natural logarithm e raised to the given number
floor(n)	It returns the previous integer number of the given number.
log(n,baseto)	It returns the natural logarithm of the number.
pow(baseto, exp)	It returns baseto raised to the exp power.
sin(n)	It returns sine of the given radian.

cos(n)	It returns cosine of the given radian.
tan(n)	It returns tangent of the given radian.

12.7.1:-Python Math Module Example

```
import math
a=4.6
print (math.ceil(a))
print (math.floor(a))
b=9
print (math.sqrt(b))
print (math.exp(3.0))
print (math.log(2.0))
print (math.pow(2.0,3.0))
print (math.sin(0))
print (math.cos(0))
print (math.tan(45))
```

Output:

```
5.0
4.0
3.0
20.0855369232
0.69314618056
8.0
0.0
1.0
1.61966519054
```

1) **Constants:**

The math module provides two constants for mathematical Operations:

Constants	Descriptions
Pi	Returns constant ? = 3.14159...
ceil(n)	Returns constant e= 2.61828...

Example

```
import math
```

```
print math.pi
```

```
print math.e
```

Output:

```
3.14159265359
2.61828182846
```

2) random:

The random module is used to generate the random numbers. It provides the following two built in functions:

Function	Description
random()	It returns a random number between 0.0 and 1.0 where 1.0 is exclusive.
randint(x,y)	It returns a random number between x and y where both the numbers are inclusive.

Example

```
import random
print (random.random())
print (random.randint(2, 8))
```

output:

```
0.036168180802993445
6
```

12.8:- Python Package

A Package is simply a collection of similar modules, sub-packages etc..Steps to create and import Package:

1) Create a directory, say Info

2) Place different modules inside the directory. We are placing 3 modules msg1.py, msg2.py and msg3.py respectively and place corresponding codes in respective modules. Let us place msg1() in msg1.py, msg2() in msg2.py and msg3() in msg3.py.

3) Create a file __init__.py which specifies attributes in each module.

4) Import the package and use the attributes using package.

Chapter13

Python 3 - CGI Programming

Topics Covered

The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script. The CGI specs are currently maintained by the NCSA.

It is a set of standards that define a standard way of passing information or web-user request to an application program & to get data back to forward it to users. This is the exchange of information between web-server and a custom script. When the users requested the web-page, the server sends the requested web-page. The web server usually passes the information to all application programs that process data and sends back an acknowledged message; this technique of passing data back-and-forth between server and application is the Common Gateway Interface. The current version of CGI is CGI/1.1 & CGI/1.2 is under process.

13.1:- What is CGI?

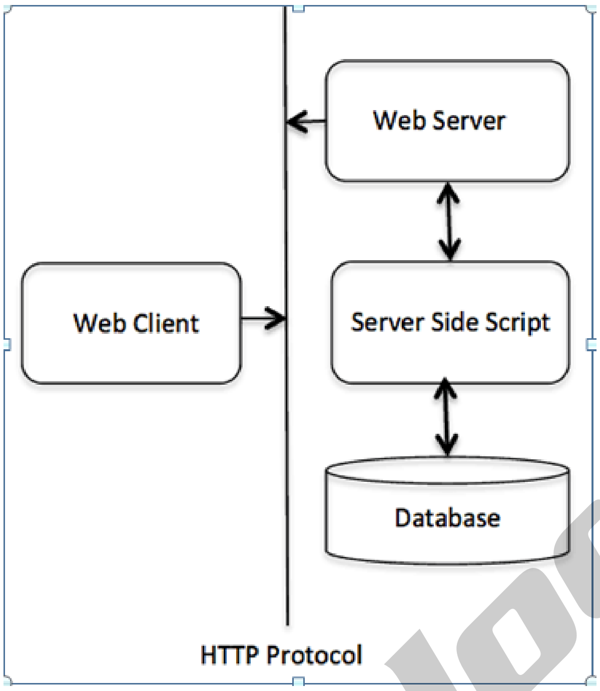
- The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.
- The current version is CGI/1.1 and CGI/1.2 is under progress.

13.2:- Browsing

- Browser contacts the HTTP web server for demanding the URL
- Parsing the URL
- Look for the filename
- If it finds that file, a request is sent back
- Web browser takes a response from the web server
- As the server response, it either shows the received file or an error message.
- It may become possible to set-up an HTTP server because when a certain directory is requested that file is not sent back; instead it is executed as a program and that program's output is displayed back to your browser.

However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your browser to display. This function is called the Common Gateway Interface or CGI and the programs are called CGI scripts. These CGI programs can be a Python Script, PERL Script, Shell Script, C or C++ program, etc.

- CGI Architecture Diagram



13.3:- Server Support and Configuration

Before you proceed with CGI Programming, make sure that your Web Server supports CGI and it is configured to handle CGI Programs. All the CGI Programs to be executed by the HTTP server are kept in a pre-configured directory. This directory is called CGI Directory and by convention it is named as /var/www/cgi-bin. By convention, CGI files have extension as. **cgi,** but you can keep your files with python extension **.py** as well.

By default, the Linux server is configured to run only the scripts in the cgi-bin directory in /var/www. If you want to specify any other directory to run your CGI scripts, comment the following lines in the httpd.conf file –

```
<Directory "/var/www/cgi-bin">
  AllowOverride None
  Options ExecCGI
  Order allow,deny
  Allow from all
</Directory>
<Directory "/var/www/cgi-bin">
Options All
</Directory>
```

Here, we assume that you have Web Server up and running successfully and you are able to run any other CGI program like Perl or Shell, etc.

CGI Program Example

Chapter 14

Database Connectivity

Topics Covered

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

You can choose the right database for your application. Python Database API supports a wide range of database servers such as:

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase
- SQLite

Note: Python has an in-built support for SQLite.

14.1:- What Database API Includes:

Using Python structure, **DB-API** provides standard and support for working with databases. The API consists of:

- Bring in the API module
- Obtain database connection
- Issue SQL statements and then store procedures

- Close the connection

14.2:- Benefits of python database programming

- Programming in Python is considerably simple and efficient with compared to other languages, so as the database programming
- Python database is portable, and the program is also portable so both can give an advantage in case of portability
- Python supports SQL cursors
- It also supports Relational Database systems
- The API of Python for the database is compatible with other databases also
- It is platform independent

14.3:- Python 3 - MySQL Database Access

MySQL is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL).

Python has an in-built support for SQLite. But if we want use MySQL database so we should Implement PyMySQL module because MySQL is not compatible with Python 3.

14.4:- What is PyMySQL ?

PyMySQL is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and contains a pure-Python MySQL client library. The goal of PyMySQL is to be a drop-in replacement for MySQLdb.

14.4.1:- Installation of Py MySQL?

The last stable release is available on PyPI and can be installed with pip

```
pip install PyMySQL
```

14.4.2:- Requirements

- Python – one of the following:
  - CPython >= 2.6 or >= 3.3
  - PyPy >= 4.0
  - IronPython 2.6
- MySQL Server – one of the following:
  - MySQL >= 4.1 (tested with only 5.5~)
  - MariaDB >= 5.1

14.5:- Database Connection

Before connecting to a MySQL database, make sure of the following points −

- You have created a database TESTDB.
- You have created a table EMPLOYEE in TESTDB.
- This table has fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- User ID "testuser" and password "test123" are set to access TESTDB.

- Python module PyMySQL is installed properly on your machine.

Example

Following is an example of connecting with MySQL database "TESTDB" –

```
#!/usr/bin/python3
import pymysql
# Open database connection
db = pymysql.connect("localhost"," username "," your password ","TESTDB")
# prepare a cursor object using cursor() method
cursor = db.cursor()
# execute SQL query using execute() method.
cursor.execute("SELECT VERSION()")
# Fetch a single row using fetchone() method.
data = cursor.fetchone()
print ("Database version : %s " % data)
# disconnect from server
db.close()
```

Output:

```
Database version : 10.1.31-MariaDB
```

If a connection is established with the data source, then a Connection Object is returned and saved into db for further use, otherwise db is set to None. Next, db object is used to create a cursor object, which in turn is used to execute SQL queries. Finally, before coming out, it ensures that the database connection is closed and resources are released.

14.6:- Creating Database Table

Once a database connection is established, we are ready to create tables or records into the database tables using execute method of the created cursor.

- Now we can create database table:

Example

```
#!/usr/bin/python3
import pymysql
# Open database connection
db = pymysql.connect("localhost","username"," your password ","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
  FIRST_NAME  CHAR(20) NOT NULL,
  LAST_NAME  CHAR(20),
  AGE INT,
  SEX CHAR(1),
  INCOME FLOAT )"""
cursor.execute(sql)
# disconnect from server
```

```
db.close()
```

14.7:- Insert data in table:

The insert operation is required when you want to create your records into a database table.

i. Example

```
#!/usr/bin/python3
import pymysql
# Open database connection
db = pymysql.connect("localhost","root","","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
  LAST_NAME, AGE, SEX, INCOME)
  VALUES ('Raj', 'kumar', 18, 'M', 10000)"""
try:
  # Execute the SQL command
  cursor.execute(sql)
  # Commit your changes in the database
  db.commit()
except:
  # Rollback in case there is any error
  db.rollback()
# disconnect from server
db.close()
```

ii. Example

```
#!/usr/bin/python3
import pymysql
# Open database connection
db = pymysql.connect("localhost","root","","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = "INSERT INTO EMPLOYEE(FIRST_NAME, \
  LAST_NAME, AGE, SEX, INCOME) \
  VALUES ('%s', '%s', '%d', '%c', '%d' )" % \
  ('Hrithik', 'Roshan', 20, 'M', 20000000)
try:
  # Execute the SQL command
  cursor.execute(sql)
  # Commit your changes in the database
  db.commit()
except:
  # Rollback in case there is any error
  db.rollback()
# disconnect from server
db.close()
```

14.8:- Read Operation

READ Operation on any database means to fetch some useful information from the database.Once the database connection is established, you are ready to make a query into this database. You can use either fetchone() method to fetch a single record or fetchall() method to fetch multiple values from a database table.

- **fetchone()** − It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **fetchall()** − It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
- **rowcount** − This is a read-only attribute and returns the number of rows that were affected by an execute() method.

Example

```
#!/usr/bin/python3
import pymysql
# Open database connection
db = pymysql.connect("localhost","root","","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = "SELECT * FROM EMPLOYEE \
      WHERE INCOME > '%d'" % (1000)
try:
  # Execute the SQL command
  cursor.execute(sql)
  # Fetch all the rows in a list of lists.
  results = cursor.fetchall()
  for row in results:
    fname = row[0]
    lname = row[1]
    age = row[2]
    sex = row[3]
    income = row[4]
    # Now print fetched result
    print ("fname = %s,lname = %s,age = %d,sex = %s,income = %d" % \
          (fname, lname, age, sex, income ))
except:
  print ("Error: unable to fetch data")
# disconnect from server
db.close()
```

Output

```
fname = Raj,lname = kumar,age = 18,sex = M,income = 10000fname = Hrithik,lname =
Roshan,age = 20,sex = M,income = 20000000
```

14.9:- Update Operation

UPDATE Operation on any database means to update one or more records, which are already available in the database.

```
import pymysql
# Open database connection
db = pymysql.connect("localhost","root","","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 2 WHERE SEX = '%c'" % ('M')
try:
  # Execute the SQL command
  cursor.execute(sql)
  # Commit your changes in the database
  db.commit()
except:
  # Rollback in case there is any error
  db.rollback()
# disconnect from server
db.close()
```

DELETE Operation

DELETE operation is use when you want to delete some records from your database.

Example

```
import pymysql
# Open database connection
db = pymysql.connect("localhost","root","","TESTDB")
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE FIRST_NAME = '%s'" % ('Raj')
try:
  # Execute the SQL command
  cursor.execute(sql)
  # Commit your changes in the database
  db.commit()
except:
  # Rollback in case there is any error
  db.rollback()
# disconnect from server
db.close()
```