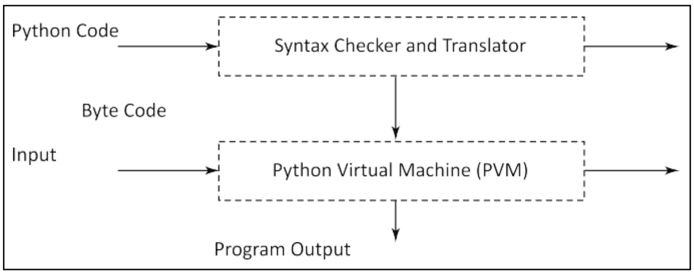
Python Assignment No. 1

Q2.} Explain internal working of python with diagram.

Ans. When a programmer tries to run a Python code as a script or instructions in an interactive mannerin a Python shell, then Python performs various operations internally.



The Python interpreter performs the following steps to execute a Python program or run a set of instructions in interactive mode.

STEP 1: The interpreter reads a Python code or instruction. Then it verifies that the instruction is well formatted, i.e. it checks the syntax of each line. If it encounters any error, it immediately halts the translation and shows an error message.

STEP 2: If there is no error, i.e. if the Python instruction or code is well formatted then the interpreter translates it into its equivalent form in low level language called “Byte Code”. Thus, after successful execution of Python script or code, it is completely translated into byte code.

STEP 3: Byte code is sent to the Python Virtual Machine (PVM). Here again the byte code is executed on PVM. If an error occurs during this execution then the execution is haltedwith an error message.

Q.3} Explain Python character set.

Ans. Any program written in Python contains words or statements which follow a sequence of characters. When these characters are submitted to the Python interpreter, they are interpreted or uniquely identified in various contexts, such as characters, identifiers, names or constants. Pythonuses the following character set:

Letters: Upper case and lower case letters

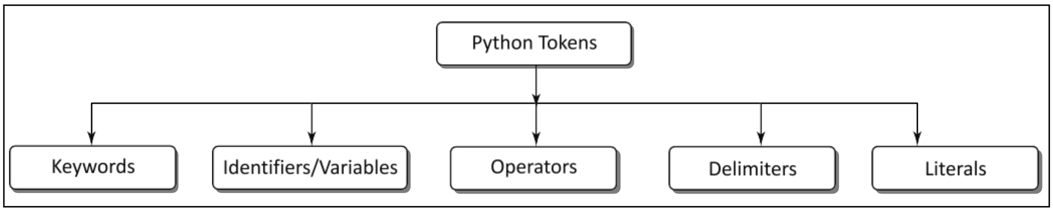
Digits: 0,1,2,3,4,5,6,7,8,9

Special Symbols: Underscore (_), (,), [,], {,}, +, -, *, &, ^, %, $, #, !, Single quote(‘), Double quotes(“), Back slash(\), Colon(:), and Semi Colon (;)

White Spaces: (‘\t\n\x0b\x0c\r’), Space, Tab.

Q.4} Explain Python Tokens in details.

Ans. A program in Python contains a sequence of instructions. Python breaks each statement into a sequence of lexical components known as tokens. Each token corresponds to a substring of a statement. Python contains various types of tokens. Figure 2.1 shows the list of tokens supported by Python.



1. Literal

Literals are numbers or strings or characters that appear directly in a program. A list of some literals in Python is as follows:

Example

78 #Integer Literal

2l.98 #Floating Point Literal

'Q' #Character Literal

"Hello" #String Literal

Python also contains other literals, such as lists, tuple and dictionary.

2. Value and Type on Literals

Programming languages contain data in terms of input and output and any kind of data canbe presented in terms of value. Here value can be of any form like literals containing numbers, characters and strings.

To know the exact type of any value, Python offersan in-built method called type.

The syntax to know the type of any value is type (value)

Example

```
>>> type('Hello World')
<class 'str'>
>>> type(123)
<class 'int'>
```

3. Keywords (any 5 to 6 keywords as example mhnun dee).

Keywords are reserved words with fixed meanings assigned to them. Keywords cannot be used asidentifiers or variables.

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

4. Operator

Python contains various operators, viz. arithmetic, relational, logical and bitwise operators.

Operator Type	*Operators*
+ - * / // % **	Arithmetic Operator
== != <> <= >=	Relational Operator
and not or	Logical Operator
& | ~ ^ << >>	Bitwise Operator

5. Delimiter

Delimiters are symbols that perform a special role in Python like grouping, punctuation andassignment. Python uses the following symbols and symbol combinations as delimiters.

( ) [ ] { }

, : . ‘ = ;

+= -= *= /= //= %= &= |= ^= >>= <<= **=

6. Identifier/Variable

Identifier is the name used to find a variable, function, class or other objects. All identifiers mustobey the following rules.

An identifier:

- Is a sequence of characters that consists of letters, digits and underscore.
- Can be of any length
- Starts with a letter which can be either lower or upper case

- Can start with an underscore ‘_’
- Cannot start with a digit
- Cannot be a keyword.

Some examples of valid identifiers are Name, Roll_NO, A1, _Address etc.
Python gives a syntax error if a programmer writes an invalid identifier. Some examples ofinvalid identifiers are First Name, 12Name, for, Salary@

Q.5} Define Print() function. Explain it with suitable example.

Ans. In Python, a function is a group of statements that are put together to perform a specific task. Thetask of printfunction is to display the contents on the screen. The syntax of print function is:

Syntax of print() function:

print(argument)

The argument of the print function can be a value of any type int, str, floatetc. It can also bea value stored in a variable.

>>> print(‘Hello Welcome to Python Programming’)Hello Welcome to Python Programming

Q.6} Write down a simple python program with step wise instructions along with the algorithm to calculate area of circle.

Ans.

STEP 1: Design an algorithm for the given problem.

An algorithm describes how a problem is to be solved by listing all the actions that need to be taken. An algorithm helps a programmer to plan for the program before actually writing it in a programming language. Algorithms are written in simple English language along with some programming code.

STEP 2: Translate an algorithm to programming instructions or code.Let us now write an algorithm to calculate the area of a rectangle.

- Algorithm to Calculate the Area of a Circle.

a) Get the radius of the circle from the user.
b) Use the relevant formula to calculate the area
   Area = pi*radius*radius
c) Finally display the area of the circle.

```
radius = int(input('Please Enter the Radius of Circle: '))
print(' Radius = ', radius)              #Print Radius
PI = 3.1428                              #Initialize value of PI
Area = PI * radius * radius              #Calculate Area
print(' Area of Circle is: ',Area)       #Print Area
```

Output

```
Please Enter the Radius of Circle: 4
Radius = 4
Area of Circle is: 50.2848
```

Q.7} Explain input() function in details.

Ans.

The input()function is used to accept an input from a user. A programmer can ask a user to inputa value by making use of input().

input() function is used to assign a value to a variable.

Syntax:

Variable_Name = input()

OR

Variable_Name = input('String')

```
Str1 = input('Enter String1:')
Str2 = input('Enter String2: ')
print(' String1 = ',Str1)
print(' String2 = ',Str2)
```

Output

```
Enter String1:Hello
Enter String2: Welcome to Python Programming
     String1 = Hello
     String2 = Welcome to Python Programming
```

The input() function is used to read the string from the user. The string values entered from the user are stored in two separate variables, viz. Str1 and Str2. Finally all the valuesare printed by making use of print()function.

Q.8} How to translate mathematical formula into equivalent Python expression. Explain with example.

Ans.

Consider the following quadratic equation written in normal arithmetic manner.

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The steps required to convert this quadratic equation into its equivalent Python expression aregiven as follows:

- STEP 1: The numerator and denominator are computed first to find the roots of the quadraticequation. Division between the numerator and denominator is performed as the last step. Hence, we can write the above expression as:

  Numerator/Denominator

- STEP 2: The denominator is just 2a, so we can rewrite the formula as:

  Numerator/((2 *a))

- STEP 3: Now we can split the numerator into two parts, i.e.

  left and right as follows:(Left+Right)/((2 *a))

- STEP 4: Substitute –b for left. There is no need to put parenthesis for –b because unary operator has higher precedence than binary addition. Hence, the above equation becomes:

  (-b+Right)/((2 *a))

- STEP 5: The right contains the expression inside the square root. Therefore, the above equationcan be rewritten as:

  (-b+sqrt(expression)/((2 *a))

- STEP 6: But the expression inside the square root contains two parts left and right. Hence, theabove equation is further rewritten as

  (-b+sqrt(left-right)/((2 *a))

- STEP 7: Now the left part contains the expression b**2 and the right part contains the expression 4*a*c. There is no need to put parenthesis for b**2 because the exponent operator has higher precedence than the * operator since the expression 4*a*c is present on the rightside. The above equation can be rewritten as

  (-b+sqrt(b**2-4*a*c)/((2 *a))

Thus, we have converted the mathematical expression into a Python expression. While converting an equation into a Python expression, one needs to only remember the rules of operatorprecedence and associativity.

```
P = 4
Q = 2
Z =( 2 + 8 * P) / 2 - ((P-Q)*(P+Q))/2 + 4 * ((P+Q)/2)
print('( 2 + 8 * P) / 2 - ((P-Q)*(P+Q))/2 + 4 * ((P+Q)/2)')
print(' Where P = ',P,' and Q = ', Q)
print(' Answer of above expression = ',Z)
```

Output

```
(2 + 8 * P) / 2 - ((P-Q)*(P+Q))/2 + 4 * ((P+Q)/2)
 Where P =  4  and Q =  2
 Answer of above expression =  23.0
```

Python Assignment No. 2

Q1.} Explain in details.

a)Boolean Not operator.

Ans.

The not operator is a unary operator. It is applied to just one value. The not operator takes a single operand and negates or inverts its Boolean value. If we apply the not operator on an expression having false value then it returns it as true. Similarly, if we apply the not operator on an expressionhaving true value then it returns it as false.

Example

Use of the not operator on a simple Boolean expression in Python, i.e. true and false.

```
>>> True
True
>>> not True
False
>>> False
False
>>> not False
True
```

b) Boolean And operator.

Ans.

The and is a binary operator. The and operator takes two operands and performs left to right evaluation to determine whether both the operands are true. Thus, and of Boolean operand is trueif and only if both operands are true.

X	*Y*	*X and Y*
True	True	True
True	False	False
False	True	False
False	False	False

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

c) Boolean Or operator.

Ans.

The or of two Boolean operands is true if at least one of the operands is true.

X	*Y*	*X or Y*
True	True	True
True	False	True
False	True	True
False	False	False

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

Q2.} Define using numbers with Boolean Operators.

Ans.

A programmer can use numbers with Boolean operators in Python.

```
>>> not 1
False
```

```
>>> 5
5
>>> not 5
False
>>> not 0
True
>>> not 0.0
True
```

Here, Python uses the Boolean operator not on the numbers and treats all numbers as True. Therefore, by writing not 1, Python substitutes 1 as True and evaluates not True, which returns False. Similarly, not is used before 5 and Python substitute True in place of 5 and it again evaluates the expression not True, which returns False. But in case of the numbers 0 and 0.0, Python treats them as False. Therefore, while evaluating not 0, it substitutes False in place of 0 andagain evaluates the expression not False, which returns True.

Q3.} Define using strings with Boolean Operators.

Ans.

Like numbers, a programmer can use strings with Boolean operators in Python.

```
>>> not 'hello'
False
>>> not ''
True
```

Here, Python uses the Boolean operator not on string. The expression not hello returns True since Python treats all strings as True. Therefore, it substitutes True in place of 'hello'and again reevaluates the expression not True, which returns False. However, if it is an empty string, Python will treat it as False. Therefore, it substitutes False in place of an empty string ''andreevaluates the expression not False, which in turn returns True.

Q4.} Explain various decision making statement:

a)if statement.

Ans.

The if statement executes a statement if a condition is true.

```
if condition:
    statement(s)
```

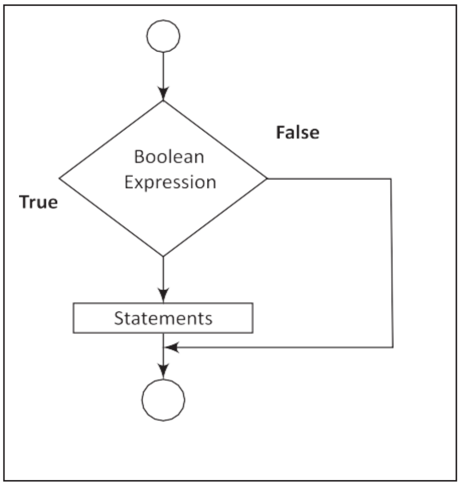
OR

```
if condition:
    Block
```

Figure 4.1 Syntax for if statement

Details of the if Statement

The keyword if begins the if statement. The condition is a Boolean expression which determines whether or not the body of if block will be executed. A colon (:) must always be followed by the condition. The block may contain one or more statements. The statement or statements are executed if and only if the condition within the if statement is true.



```
num1=eval(input("Enter First Number: "))
num2=eval(input("Enter Second Number:  "))
if  num1-num2==0:
    print("Both the numbers entered are Equal")
```

Output

```
Enter First Number: 12
Enter Second Number:  12
Both the numbers entered are Equal
```

b)if-else statement.

Ans.

The execution of the if statement has been explained in the previous programs. We know, the if statement executes when the condition following if is true and it does nothing when the conditionis false. The if-else statement takes care of a true as well a false condition.

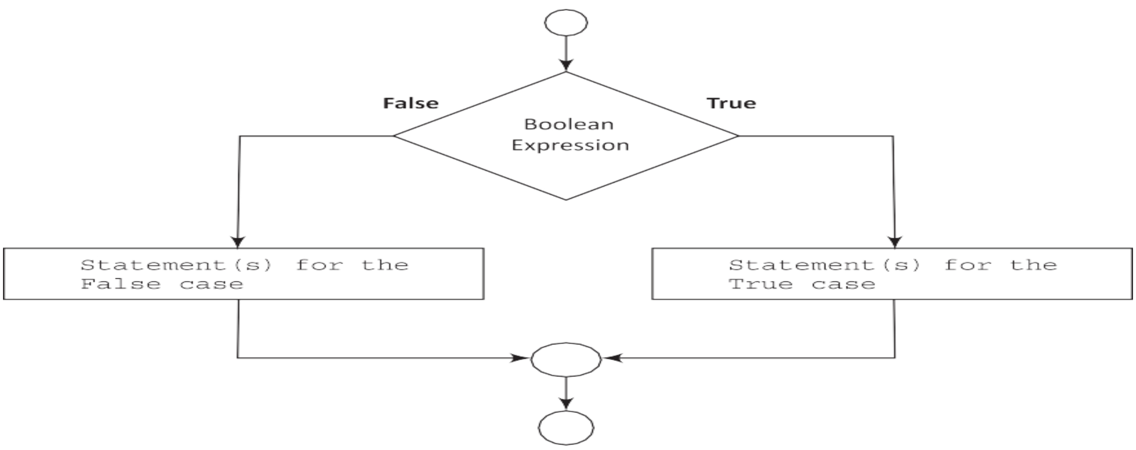
```
if condition:
    statement(s)
else:
```

OR

```
if condition:
    if_Block
else:
```

Details of if-else Statement

The if-else statement takes care of both true and false conditions. It has two blocks. One block is forif and it may contain one or more than one statements. The block is executed when the condition is true. The other block is for else. The else block may also have one or more than one statements. It is executed when the condition is false. A colon (:) must always be followed by the condition. The keyword else should also be followed by a colon (:).



```
num1=int(input("Enter the First Number:"))
num2=int(input("Enter the Second Number:"))
if num1>num2:
    print(num1,"is greater than ",num2)
else:
    print(num2,"is greater than ",num1)
```

```
Output

Enter the First Number:100
Enter the Second Number:43
100 is greater than 43
```

c)nested if statement.

Ans.

When a programmer writes one if statement inside another if statement then it is called a nestedif statement. A general syntax for nested if statements is given as follows:

```
if Boolean-expression1:
     if Boolean-expression2:
         statement1


       else:
           statement2
  else:
  statement3
```

In the above syntax, if the Boolean-expression1 and Boolean-expression2 are correct then statement1 will execute. If the Boolean-expression1 is correct and Boolean-expression2 is incorrectthen statement2 will execute. And if both Boolean-expression1 and Boolean-expression2 areincorrect then statement3 will execute.

d) multiway if-elif-else statement:

```
If Boolean-expression1:
        statement1
    elif Boolean-expression2 :


        statement2
    elif Boolean-expression3 :
        statement3
-   - - - - - - - - - - - - -
-   - - - - - - - - - - -- -
elif Boolean-expression n :
    statement N
else :
    Statement(s)
```

In this kind of statements, the number of conditions, i.e. Boolean expressions are checked from top to bottom. When a true condition is found, the statement associated with it is executed and therest of the conditional statements are skipped. If none of the conditions are found true then the last else statement is executed. If all other conditions are false and if the final else statement is not present then no action takes place.

Q.5} Explain the importance of loop control statements.

Ans.

In our day-to-day life, we perform certain tasks repeatedly. It can be tedious to perform such tasksusing pen and paper. For instance, teaching multiplication tables to multiple classes can become easier if the teacher uses a simple computer program with loop instructions instead of pen and paper.

Let us try to understand the concept of control statements in this context. Suppose a programmer wants to display the message, "I Love Python" 50 times. It would be tedious for him/her towrite the statement 50 times on a computer screen or even on paper. This task can become very easy, quick and accurate if the programmer completes it using loop instructions in a computer programming language. Almost all computer programming languages facilitate the use of control loop statements to repeatedly execute a block of code until a condition is satisfied.

Q.6} Explain range function in details.

Ans. There is a inbuilt function in Python called range(), which is used to generate a list of integers. The range function has one, two or three parameters. The last two parameters in range() are optional.
The general form of the range function is:

range(begin, end, step)

The ‘begin’ is the first beginning number in the sequence at which the list starts.

The ‘end’ is the limit, i.e. the last number in the sequence.

The ‘step’ is the difference between each number in the sequence.

```
>>> list(range(1,6))
    [1,2,3,4,5]
```

```
>>> list(range(1,20,2))
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Q.7} Explain for loop in details.

Ans. The for loops in Python are slightly different from the for loops in other programming languages. The Python for loop iterates through a sequence of objects, i.e. it iterates through each value in a sequence, where the sequence of object holds multiple items of data stored one after another.

```
for var in sequence:
     statement(s)
     ..................................
     ...............................
           ..................................
```

The for loop is a Python statement which repeats a group of statements for a specified numberof times. As described in the syntax, the keywords for and in are essential keywords to iterate the sequence of values. The variable var takes on each consecutive value in the sequence and the statements in the body of the loop are executed once for each value. A simple example of for loop is:

```
for var in range(m,n):
    print var
```

```
for i in range(1,6):
    print(i)
print(“End of The Program”)
```

Output

```
1
2
3
4
5
End of The Program
```

```
for j in range(1,6):
    square=j*j
    print("Square of ",j," is: ",square)
print("End of Program")
```

Output

```
Square of 1 is:  1
Square of 2 is:  4
Square of 3 is:  9
Square of 4 is:  16
Square of 5 is:  25
End of Program
```

```
sum=0
print("Even numbers from 0 to 10 are as follows")
for j in range(0,11,1):
    if i%2==0:
        print(j)
        sum=sum+i
print("Sum of Even numbers from 0 to 10 is = ",sum)
```

Output

```
Even numbers from 0 to 10 are as follows
0
2
4
6
8
10
Sum of Even numbers from 0 to 10 is = 30
```

Q.8} Explain nested loops and write a program to display multiplication table from 1 to 10.

Ans.

The for and while loop statements can be nested in the same manner in which the if statements are nested. Loops within the loops or when one loop is inserted completely within another loop, then it is called nested loop.

```
Print("Multiplication Table from 1 to 5 ")
for i in range(1,11,1):           #Outer Loop
    for j in range(1,6,1):        #Inner Loop
```

(*Contd.*


```
        print(format(i * j,"4d"),end=" ")
    print()
print("End of Program")
```

Output

```
Multiplication Table from 1 to 5
   1   2   3   4   5
   2   4   6   8  10
   3   6   9  12  15
   4   8  12  16  20
   5  10  15  20  25
   6  12  18  24  30
   7  14  21  28  35
   8  16  24  32  40
   9  18  27  36  45
  10  20  30  40  50
End of Program
```

Python Assignment No.3

Q1.}Define Function. Describe syntax and basics of function.

Ans. It is difficult to prepare and maintain a large-scale program and the identification of the flow of data subsequently gets harder to understand. The best way to create a programming application isto divide a big program into small modules and repeatedly call these modules.

With the help of functions, an entire program can be divided into small independent modules (each small module is called a function). This improves the code's readability as well as the flow of execution as small modules can be managed easily.

A function is a self-contained block of one or more statements that performs a special task whencalled. The syntax for function is given as follows:

```
def name_of_function(Parameters):  <------ Function Header
        statement1
        statement2
        statement3
        ..........
        ..........                 <------ Function Body
        statementN
```

The syntax for the Python function contains a header and body. The function header begins with the 'def' keyword. The def keyword signifies the beginning of the function's definition. The name of the function is followed by the def keyword. The function header may contain zero or more number of parameters. These parameters are called formal parameters. If a function containsmore than one parameter then all the parameters are separated by commas. A function's body is ablock of statements. The statements within the function's body define the actions that the functionneeds to perform.

```
def Display():
    print("Welcome  to Python Programming ")
Display()             #call function
```

Output

```
Welcome to Python Programming
```

```
def print_msg():
    str1=input("Please Enter Your Name:")
    print("Dear ",str1," Welcome  to Python Programming ")
print_msg()  #call function
```

Output

```
Please Enter Your Name: Virat
Dear Virat  Welcome  to Python Programming
```

Q.2} What is the role of parameters and arguments in a function.

Ans: Parameters are used to give inputs to a function. They are specified with a pair of parenthesis in the function's definition. When a programmer calls a function, the values are also passed to the function.

While parameters are defined by names that appear in the function's definition, argumentsare values actually passed to a function when calling it. Thus, parameters define what types of arguments a function can accept.

```
def printMax(num1,num2):
        Statemen1
        Statemen2                 #Define a Function
        ........................
        ........................
        StatementN

printMax(10,20)   ←——  Call a function(Invoke)
```

In the above example, printMax(num1, num2) has two parameters, viz. num1 and num2.The parameters num1 and num2 are also called formal parameters. A function is invoked by calling the name of the function, i.e. printMax(10,20), where 10, 20 are the actual parameters. Actual parameters are also called arguments. num1 and num2 are the parameters of a function. Program 6.5 demonstrates the use of parameters and arguments in a function.

Q.3} Explain Local & Global scope of variable with example.

Ans.

Variables and parameters that are initialized within a function including parameters, are said to exist in that function's local scope. Variables that exist in local scope are called local variables. Variables that are assigned outside functions are said to exist in global scope. Therefore, variables that exist in global scope are called global variables.

```
p = 20          #global variable p
def Demo():
    q = 10      #Local variable q
    print('The value of Local variable q:',q)
    #Access global variable p within this function
    print('The value of Global Variable p:',p)
Demo()
#Access global variable p outside the function Demo()
print('The value of global variable  p:',p)
```

Output

```
The value of Local variable q: 10
The value of Global Variable p: 20
The value of global variable p: 20
```

In the above example, we have created one local variable 'q' and one global variable'p'. As global variables are created outside all functions and are accessible to all functions in their scope, in the above example as well the global variable 'p' is accessed from the function `Demo()` and it is also accessed outside the function.

Q.4} Explain Recursive function in details.

Ans. We have seen that it is legal for one function to call another function. In programming, there might be a situation where a function needs to invoke itself. Python also supports the recursive feature, which means that a function is repetitively called by itself. Thus, a function is said to be recursive if a statement within the body of the function calls itself.

Let us consider a simple example of recursion. Suppose we want to calculate the factorial valueof an integer. We know that the factorial of a number is the product of all the integers between 1 and that number, i.e. n! is defined as n * (n-1)!.

Consider the following example.

Formula to calculate the factorial of a number (n)! = n*(n-1)!

5!= 5*(4)!
= 5*4*(3)!
= 5*4*3*(2)!
= 5*4*3*2*(1)
= 120

```
def factorial(n):
    if n==0:
        return 1
    return n*factorial(n-1)
print(factorial(5))
```

Output

```
120
```

In the above program, `factorial()` is a recursive function. The number is passedto function `factorial()`. When the function factorial is executed, it is repeatedly invoked by itself. Every time a function is invoked, the value of 'n' is reduced by one and multiplication is carried out.The recursion function produces the number 5, 4, 3, 2 and 1. The multiplication of these numbers iscarried out and returned. Finally, the print statement prints the factorial of the number.

Q.5} Define strings and describe string class.

Ans. Characters are building blocks of Python. A program is composed of a sequence of characters. When a sequence of characters is grouped together, a meaningful string is created. Thus, a string is a sequence of characters treated as a single unit.

In many languages, strings are treated as arrays of characters but in Python a string is an objectof the str class. This string class has many constructors.

The str class:

Strings are objects of the str class. We can create a string using the constructor of str class as:

```
S1=str() #Creates an Empty string Object
S2=str("Hello") #Creates a String Object for Hello
```

An alternative way to create a string object is by assigning a string value to a variable.

```
S1 = ""  # Creates a Empty String
S2= "Hello" # Equivalent to S2=str("Hello")
```

All the characters of a string can be accessed at one time using the index operator.

Q.6} Describe basic inbuilt Python function for string.

Ans.

Python has several basic inbuilt functions that can be used with strings. A programmer can make use of `min()` and `max()` functions to return the largest and smallest character in a string. We canalso use `len()` function to return the number of characters in a string.

The following example illustrates the use of the basic function on strings.

```
>>> a = "PYTHON"
>>> len(a) #Return length i.e. number of characters in string a
6
>>> min(a) #Return smallest character present in a string
'H'
>>> max(a) #Return largest character present in a string
'Y'
```

Q.7} Describe Traversing string with for loop.

Ans. A programmer can use the for loop to traverse all characters in a string. For example, the followingcode displays all the characters of a string.

```
S="India"
for ch in S:
    print(ch,end="")
```

Output

```
India
```

The string 'India' is assigned to the variable S. The for loop is used to print all the characters of a string S. The statement 'for ch in S:' can read as 'for each character ch in S print ch'.

```
S="ILOVEPYTHONPROGRAMMING"
for ch  in range(0,len(S),2):#Traverse each Second character
    print(S[ch],end=" ")
```

Output

```
I O E Y H N R G A M N
```

Q.8} Explain string operations, Explain in details for following:-

a)String comparison.

Ans.

Operators such as ==,<,>,<=,>=and != are used to compare the strings. Python compares strings by comparing their corresponding characters.

Example

```
>>> S1="abcd"
>>> S2="ABCD"
>>> S1>S2
True
```

The string 'abcd' is assigned to the string S1 and the String 'ABCD' is assigned to S2. The statementS1 > S2 returns True because Python compares the numeric value of each character. In the above example, the numeric value, i.e. ASCII value of 'a' is 97 and ASCII numeric value of 'A' is 65. It means 97 > 65. Thus, it returns True. However, character by character comparison goes on till the end of the string.

Some More Examples of String Comparison

```
>>> S1="abc"
>>> S2="abc"
>>> S1==S2
True
>>> S1="ABC"
>>> S2="DEF"
>>> S1>S2|
False

>>> S1="AAA"
>>> S2="AAB"
>>> S2>S1
True

>>> S1="ABCD"
>>> S2="abcd".upper()
>>> S2
'ABCD'
>>> S1>S2
False
>>> S1>=S2
True
```

b) String.format() method.

Ans.

In Python 2 and 3, programmers can include %s inside a string and follow it with a list of values for each %.

```
>>> "My Name is %s and I am from %s"%("JHON","USA")
'My Name is JHON and I am from USA'
```

In the above example, we have seen how to format a string using % (modulus) operator. However,for more complex formatting, Python 3 has added a new string method called `format()` method. Instead of % we can use {0}, {1} and so on. The syntax for `format()` method is:

```
template.foramt(P0,P1,……… ,k0=V0,K1=V1…}
```

whereas the arguments to the .format() method are of two types. It consists of zero or more positional arguments P_i followed by zero or more keyword arguments of the form, K_i=V_i.

Example

```
>>> '{} plus {} equals {}'.format(4,5,'Nine')
'4 plus 5 equals Nine'
```

The `format()` method is called on the string literal with arguments 4,5 and 'nine'. The empty {} are replaced with the arguments in order. The first {} curly bracket is replaced with the first argument and so on. By default, the index of the first argument in format always start from zero. One can also give a position of arguments inside the curly brackets.

The following example illustrates the use ofindex as argument inside the curly bracket.

```
>>>"My Name is {0} and I am from {1}".format("Milinda","USA")
'My Name is Milinda and I am from USA'
```

The `format()` method contains various arguments. In the above example, the `format()` methodhas two arguments, viz. “Milinda” and “USA”. The index of the first argument of the `format()` method always starts from 0. Therefore, {0} replaces the 0th argument of the format. Similarly {1} replaces the first argument of the format.

c) split() method.

Ans.

The `split()` method returns a list of all the words in a string. It is used to break up a string into smaller strings.

Example

Consider the following example where names of different programming languages such as C, C++,Java and Python is assigned to a variable Str1. Applying split() method on str1 returns the list of programming languages.

```
>>>Str1="C C++ JAVA Python"#Assigns names of Programming languages to Str1
>>>Str1.split()
['C,C++,JAVA,Python']
```

```
TOP_10_Company="TCS,INFOSYS,GOOGLE,MICROSOFT,YAHOO"
Company=TOP_10_Company.split(",")
print(Company)
for c  in Company:
    print(end="")
    print(c)
```

Output

```
['TCS', 'INFOSYS', 'GOOGLE', 'MICROSOFT', 'YAHOO']
TCS
INFOSYS
GOOGLE
MICROSOFT
YAHOO
```

Q.9} Explain searching substring present in a string.

Ans.

Methods of str Class for Searching the Substring in a Given String	*Meaning*

`bool endswith(str Str1)`
Example:
`>>> S="Python Programming"`
`>>>S.endswith("Programming")`
`True`
	Returns true if the string ends with the substring Str1.

`bool startswith(str Str1)`
Example:
`>>> S="Python Programming"`
`>>>S.startswith("Python")`
`True`
	Returns true if the string starts with the substring Str1.

`int find(str Str1)`
Example:
`>>> Str1="Python Programming"`
`>>> Str1.find("Prog")`
`7#Returns the index from where the string "Prog" begins`
`>>> Str1.find("Java")`
`-1#Returns -1 if the string "Java" is not found in the string str1`
	Returns the lowest index where the string Str1 starts in this string or returns -1 if the string Str1 is not found in this string.

`int rfind(str Str1)`
Example:
`>>> Str1="Python Programming"`
`>>> Str1.rfind("o")`
`9#Returns the index of last occurrence of string "o" in Str1`
	Returns the highest index where the string Str1 starts in this string or returns -1 if the string Str1 is not found in this string.

`int count(str S1)`
Example:
`>>> Str1="Good Morning"`
`>>> Str1.count("o")`
`3`
	Returns the number of occurrences of this substring.

Q.10} Explain methods to convert string into another string.

Ans.

A string may be present in lower case or upper case. The string in lower case can be converted into upper case and vice versa using various methods of the str class.

Methods of Str Class to Convert a String from One Form to Another	*Meaning*

`str capitalize()`
Example:
`>>> Str1="hello"`
`>>> Str1.capitalize ()`
`'Hello' #Convert first alphabet of String Str1 to uppercase`
	Returns a copy of the string with only the first character capitalised.

`str lower()`
Example:
`>>> Str1="INDIA"`
`>>> Str1.lower ()`
`'india'`
	Returns a copy of the string with all the letters converted into lower case.

`str upper()`
Example:
`>>> Str1="iitbombay"`
`>>> Str1.upper()`
`'IITBOMBAY'`
	Returns a copy of the string with all the letters converted into upper case.

`str title()`
Example:
`>>> Str1="welcome to the world of programming"`
`>>> Str1.title()`
`'Welcome To The World Of Programming'`
	Returns a copy of the string with the first letter capitalised in each word of the string.

`str swapcase()`
Example:
`>>> Str1="IncreDible India"`
`>>> Str1.swapcase ()`
`'iNCREdIBLE iNDIA'`
	Returns a copy of the string which converts upper case characters into lower case characters and lower case characters into upper case characters.

`str replace (str old, str new [,count])`
Example:
`>>> S1="I have brought two chocolates, two cookies and two cakes"`
`#Replace the old string i.e "two" by new string i.e. "three".`
`>>> S2=S1.replace("two","three")`
`#Replace all occurrences of old string "two" by "three"`
`>>> S2`
`'I have brought three chocolates, three cookies and three cakes'`
	Returns a new string that replaces all the occurrences of the old string with a new string. The third parameter, i.e. the count is optional. It tells the number ofold occurrences of the string to be replaced with new occurrences of the string.

Python Assignment No. 4

Q1. Explain creating list with example.

Ans. A list is a sequence of values called items or elements. The elements can be of any type. The structure of a list is similar to the structure of a string.

CREATING LISTS

The List class define lists. A programmer can use a list's constructor to create a list. Consider the following example.

Example: Create a list using the constructor of the list class

a. Create an empty list.
```
L1 = list();
```
b. Create a list with any three integer elements, such as 10, 20 and 30.
```
L2 = list([10,20,30])
```
c. Create a list with three string elements, such as “Apple”, “Banana” and “Grapes”.
```
L3 = list([“Apple”,”Banana”,”Grapes”])
```
d. Create a list using inbuilt `range()` function.
```
L4 = list(range(0,6))    # create a list with
elements from  0 to 5
```
e. Create a list with inbuilt characters X, Y and Z.
```
L5=list(“xyz”)
```

Example: Creating a list without using the constructor of the list class

a. Create a list with any three integer elements, such as 10, 20 and 30.
```
L1=[10,20,30]
```
b. Create a list with three string elements, such as “Apple”, “Banana” and “Grapes”.
```
L2 = [“Apple”, “Banana”, “Grapes”]
```

Q2. Describe the term negative list indices.

Ans.

The negative index accesses the elements from the end of a list counting in backward direction.The index of the last element of any non-empty list is always -1, as shown in Figure

List1 =	10	20	30	40	50	60
	−6	−5	−4	−3	−2	−1

Figure 8.2 List with negative index

Accessing the elements of a list using a negative index.

Example

```
>>> List1=[10,20,30,40,50,60]#Create a List
>>> List1[-1]       #Access Last element of a List
60
>>>List1[-2]        #Access the second last element of
List
50
>>> List1[-3]       #Access the Third last element of
List
40
>>>List1[-6]        #Access the first Element of the
List
10
```

Q3. Explain list slicing in details with example.

Ans.

The slicing operator returns a subset of a list called slice by specifying two indices, i.e. start and end. The syntax is:

`Name_of_Variable_of_a_List[Start_Index: End_Index]`

Example

```
>>> L1=([10,20,30,40,50])  #Create a List with 5 Different Elements
>>> L1[1:4]
  20,30,40
```

The L1[1:4] returns the subset of the list starting from index the start index 1 to one index less thanthat of the end index, i.e. 4-1 = 3.

Example

```
>>> L1=([10,20,30,40,50]) #Create a List with 5 Different Elements
>>> L1[2:5]
[30, 40, 50]
```

The above example L1 creates a list of five elements. The index operator L1[2:5] returns all theelements stored between the index 2 and one less than the end index, i.e. 5-1 = 4.

Q4. Explain list slicing with step with examples.

Ans.

LIST SLICING WITH STEP SIZE

In slicing, the first two parameters are start index and end index. Thus, we need to add a third parameter as step size to select a list with step size. To be able to do this we use the syntax:

```
List_Name[Start_Index:End_Index:Step_Size]
```

Example

```
>>>MyList1=["Hello",1,"Monkey",2,"Dog",3,"Donkey"]
>>>New_List1=MyList1[0
:6:2]
print(New_List1)
['Hello', 'Monkey', 'Dog']
```

Explanation Initially we created a list named Mylist1 with five elements. The statement MyList1[0:6:2] indicates the programmer to select the portion of a list which starts at index 0 and ends at index 6 with the step size as 2. It means we first extract a section or slice of the list which starts at the index 0 and ends at the index 6 and then selects every other second element.

Example

```
>>> List1=["Python",450,"C",300,",C++",670]
>>> List1[0:6:3]   #Start from Zero and Select every
Third Element
['Python', 300]    #Output
```

Q5. Explain Python inbuilt functions for list along with the example.

Ans.

Python has various inbuilt functions that can be used with lists. Some of these are listed inTable 8.1.

Table 8.1 Inbuilt functions that can be used with lists

Inbuilt Functions	*Meaning*
`Len()`	Returns the number of elements in a list.
`Max()`	Returns the element with the greatest value.
`Min()`	Returns the element with the lowest value.
`Sum()`	Returns the sum of all the elements.
`random.shuffle()`	Shuffles the elements randomly.

Example

```
#Creates a List to store the names of Colors and return size
of list.

>>> List1=["Red","Orange","Pink","Green"]
>>> List1
['Red', 'Orange', 'Pink', 'Green']
>>> len(List1)      #Returns the Size of List
4

#Create a List, find the Greatest and Minimum value from the
list.
>>> List2=[10,20,30,50,60]
>>> List2
[10, 20, 30, 50, 60]
>>> max(List2)     #Returns the greatest element from the
list.
60
>>> min(List2)     #Returns the minimum element from the list.
10

#Create a List, and Shuffle the elements
in random manner.#Test Case 1
>>>import random
>>> random.shuffle(List2)
>>> List2
```

```
[30, 10, 20, 50, 60]
>>> List2
[30, 10, 20, 50, 60]

#Test Case2
>>> random.shuffle(List2)
>>> List2
[20, 10, 30, 50, 60]
#Create a List, and find the sum of all the elements of a
List.
>>> List2=[10,20,30,50,60]
>>> List2
[10, 20, 30, 50, 60]
>>> sum(List2)
170
```

Q6. Explain list operators in details.

Ans.

1. **The + Operator**: The concatenation operator is used to join two lists.

Example

```
>>> a=[1,2,3]                  #Create a list with three elements
1,2, and 3
>>> a                          #Prints the list
[1, 2, 3]
>>> b=[4,5,6]                  #Create a list with three elements
4,5, and 6
>>> b                          #print the list
[4, 5, 6]
>>> a+b
                               #Concatenat
e the list a and b [1, 2, 3, 4, 5, 6]
```

2. The * Operator: The multiplication operator is used to replicate the elements of a list.

Example

```
>>> List1=[10,20]
>>> List1
[10, 20]
>>> List2=[20,30]
>>> List2
[20, 30]
>>> List3=2*List1        #Print each element of a List1 twice.
>>> List3
[10, 20, 10, 20]
```

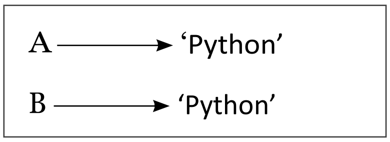
3. The in Operator: The in operator used to determine whether an element is in a list. It returns True if the element is present and False if the element is absent in the list.

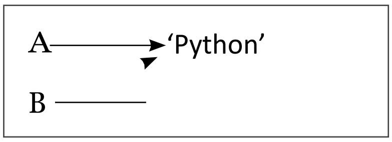
Example

```
>>> List1= [10,20]
>>> List1
[10, 20]
>>> 40 in List1   #To Check if 40 is present in List1
False
>>> 10 in List1   #To Check if 10 is present in List1
True
```

4. **The isOperator:** Let us execute the following two statements:A='Python'
 B='Python'

We know that both A and B refer to a string but we don't know whether they refer to the samestring or not. Two possible situations





are:

In the first case, A and B refer to two different objects that have the same values. In second case, they refer to the same object. To understand whether two variables refer to the same object, a programmer can use the 'is' operator.

Example

```
>>> A='Microsoft'

>>> B='Microsoft'

>>> A is B #Check if two variable refer to the same Object

True
```

From the above example, it is clear that Python created only one string object and both A and B refer to the same object. However, when we create two lists with the same elements, Python creates two different objects as well.

Example

```
>>> A=['A','B','C']
>>> B=['A','B','C']
>>> A is B  #Check if two lists refer to the same Object
False
```

Explanation: In the above example, the two lists A and B contain exactly the same number of elements. The is operator is used to check if both the variables A and B refer to the same object, but it returns False. It means that even if the two lists are the same, Python creates two different objects. State diagram for the above example is given in Figure 8.3.

A ⟶ ['A','B','C']

B ⟶ ['A','B','C']

Figure 8.3 Effect of is operator on a list

It is important to note that in the above example, we can say that the two lists are equivalent because they have the same elements. We cannot say that both the lists are identical because they don't refer to the same object.

5.**The del Operator:** The del operator stands for Delete. The del operator is used to remove the elements from a list. To delete the element of a list, the elements of the list are accessed usingtheir index position and the del operator is placed before them.

Example

```
Lst=[10,20,30,40,50,60,70]
>>> del Lst[2]     #Removes 3rd element from the List
>>> Lst
[10, 20, 40, 50, 60, 70]

Lst=[10,20,30,40,50,60,70]
>>> del Lst[-1]
>>> Lst            #Removes last element from the List
[10, 20, 30, 40, 50, 60]

>>> Lst=[10,20,30,40,50,60,70]
>>> del Lst[2:5]    #Removes element from index position 2 to 4
>>> Lst
[10, 20, 60, 70]

>>> Lst=[10,20,30,40,50,60,70]
>>> del Lst[:]    #Removes all the element from the List
>>> Lst
[]
```

Q7. Explain List comprehension. Write down a program to display even elements of the list using list comprehension where the range of elements is 1 to 30.

Ans.

List comprehension is used to create a new list from existing sequences. It is a tool for transforming a given list into another list.

Example: Without list comprehension

Create a list to store five different numbers such as 10, 20, 30, 40 and 50. Using the for loop, addnumber 5 to the existing elements of the list.

```
>>> List1= [10, 20, 30, 40, 50]

>>> List1

    [10, 20, 30, 40, 50]

>>> for i in range(0,len(List1)):

   List1[i]=List1[i]+5     #Add 5 to each element of List1

>>> List1               #print the List1 After Performing

    [15, 25, 35, 45, 55]
```

The above code is workable but not the optimal code or the best way to write a code in Python.Using list comprehension, we can replace the loop with a single expression that produces the same result.

The syntax of list comprehension is based on set builder notation in mathematics. Set builder notation is a mathematical notation for describing a set by stating the property that its members should satisfy. The syntax is

```
[<expression> for <element> in <sequence> if <conditional>]
```

The syntax is designed to read as "Compute the expression for each element in the sequence, if the conditional is true".

Example: Using list comprehension

```
>>> List1= [10, 20, 30, 40, 50]
>>> List1
     [10, 20, 30, 40, 50]

>>>for i in range(0,len(List1)):
     List1[i]=List1[i]+10
```

Without List Comprehension

```
>>> List1= [10,20,30,40,50]
>>> List1= [x+10 for x in List1]
>>> List1
[20, 30, 40, 50, 60]
```

Using List Comprehension

In the above example, the output for both without list comprehension and using list comprehension is the same. The use of list comprehension requires lesser code and also runs faster. With reference to the above example we can say that list comprehension contains:

a. An input sequence
b. A variable referencing the input sequence
c. An optional expression
d. An output expression or output variable

Example

```
List1= [20, 30, 40, 50, 60]
List1= [   x+10    for   x    in    List1]
```

(An output variable)

(An input sequence)

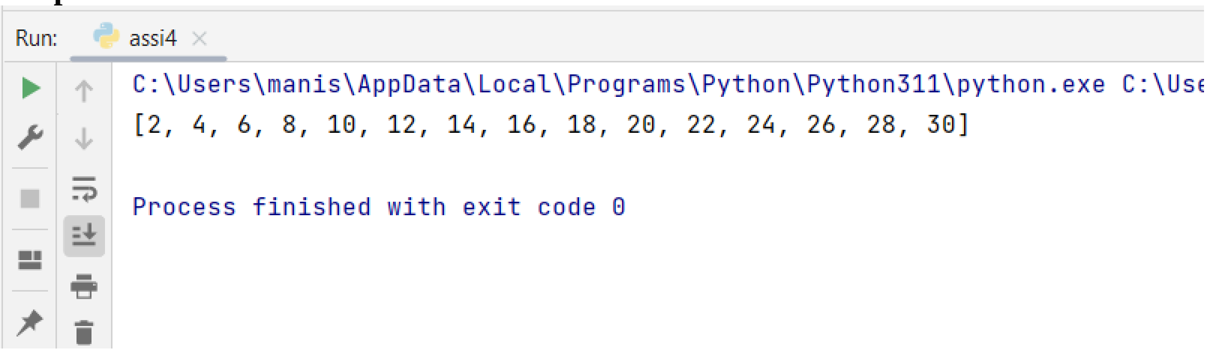
(A variable referencing an input sequence)

Output `[20, 30, 40, 50, 60]`

Program:

```
even_nums = [num for num in range(1,31) if num % 2 == 0]
print(even_nums)
```

Output:



Q8. Explain List methods in details with example.
Ans.

Once a list is created, we can use the methods of the list class to manipulate the list. Table 8.2contains the methods of the list class along with examples.

Methods of List	*Meaning*

`None append(object x)` end of
Example:
return type
`>>> List1=['X','Y','Z']`
`>>> List1`
`['X', 'Y', 'Z']`
`>>> List1.append('A') #Append element 'A' to the endof the List1`
`>>> List1`
`['X', 'Y', 'Z', 'A']`
 	
`Note: Append method is equivalent to doing: List1[len(List1):]=[Element_Name]`
Example:
`>>>List1=["Red","Blue","Pink"]`
`>>> List1`
`['Red', 'Blue', 'Pink']`
`>>> List1[len(List1):]=['Yellow']`
`>>> List1`
`['Red', 'Blue', 'Pink', 'Yellow']`
	Adds an element x to the
	the list. None is the
	of method appended.

None clear()
Example:
`>>> List1=["Red", "Blue", "Pink"]`
`>>> List1`
`['Red', 'Blue', 'Pink']`
`>>> List1.clear()  # Removes all the element of List`
`>>> List1          # Returns Empty List after removing all elements`
`[]`
	Removes all the items from the list.

int count(object x)
Example:
`>>> List1=['A','B','C','A','B','Z']`
`>>> List1`
`['A', 'B', 'C', 'A', 'B', 'Z']`
`#Count the number of times the element 'A' has appeared in the list`
`>>> List1.count('A')`
`2    # Thus, 'A' has appeared 2 times in List1`
	Returns the number of times the element x appears in the list.

List copy()
Example:
`>>> List1=["Red","Blue","Pink"]`
`>>> List1`
`['Red', 'Blue', 'Pink']`
`>>> List2=List1.copy() # Copy the contents of List1 to List2`
`>>> List2`
`['Red', 'Blue', 'Pink']`
Note: Copy() Method is equivalent to doing
`List2=List1[:] # Copies the content of List1 to List2`
Example:
`>>> List1=["Red","Blue","Pink"]`
`>>> List2=List1[:]`
`>>> List2`
`['Red', 'Blue', 'Pink']`
	This method returns a shallow copy of the list.

None extend(list L2)
Example:
`>>> List1= [1,2,3]`
`>>> List2= [4,5,6]`
`>>> List1`
`[1, 2, 3]`
	Appends all the elements of list L2 to the list.

`>>> List2`
`[4, 5, 6]`
`>>> List1.extend(List2)  #Appends all the elements of List2 to List1`
`>>> List1`
`[1, 2, 3, 4, 5, 6]`	

`int index(object x)`
Example:
`>>> List1=['A','B','C','B','D','A']`
`>>> List1`
`['A', 'B', 'C', 'B', 'D', 'A']`
`#Returns the index of first occurrence of element 'B' from the list1`
`>>> List1.index('B')`
`1     #Returns the index of element B`
	Returns the index of the first occurrence of the element *x* from the list.

`None insert(int index,Object X)`
Example:
`>>> Lis1=[10,20,30,40,60]`
`>>> Lis1`
`[10, 20, 30, 40, 60]`
`>>> Lis1.insert(4,50)  #Insert Element 50 at index 4`
`>>> Lis1`
`[10, 20, 30, 40, 50, 60]`
	Insert the element at a given index.
	Note: The index of the first element of a list is always zero.

`Object pop(i)`
Example:
`>>> Lis1=[10,20,30,40,60]`
`>>> Lis1`
`[10, 20, 30, 40, 60]`
`>>> Lis1.pop(1)   # Remove the element which is at index 1.`
`20`
`>>> Lis1    # Display List after removing the element from index 1.`
`[10, 30, 40, 60]`
`>>> Lis1.pop() # Remove the last element from the list`
`60`
`>>> Lis1`
`[10, 30, 40]   #Display the list after removing last element`
	Removes the element from the given position. Also, it returns the removed element.
	Note: The parameter i is optional. If it is not specified then it removes the last element from the list.

None remove(object x)
Example:
`>>> List1=['A','B','C','B','D','E']`
`>>> List1`
`['A', 'B', 'C', 'B', 'D', 'E']`
`>>> List1.remove('B')#Removes the first occurrence of element B`
`>>> List1`
`['A', 'C', 'B', 'D', 'E']`
	Removes the first occurrence of element x from the list.

None reverse()
Example:
`>>> List1=['A','B','C','B','D','E']`
`>>> List1`
`['A', 'B', 'C', 'B', 'D', 'E']`
`>>> List1.reverse() # Reverse all the elements of the list.`
`>>> List1`
`['E', 'D', 'B', 'C', 'B', 'A']`
	Reverses the element of the list.

None sort()
Example:
`>>> List1=['G','F','A','C','B']`
`>>> List1`
`['G', 'F', 'A', 'C', 'B']    #Unsorted List`
`>>> List1.sort()`
`>>> List1                    #Sorted List`
`['A', 'B', 'C', 'F', 'G']`
	Sort the elements of list.

Q9. Explain following in short:

a. Splitting a string in a list:
Ans.

The `list()` function breaks a stringinto individual letters. In this section, we will explore how to split a string into words.

The str class contains the split method and is used to split a string into words.

Example

```
>>> A="Wow!!! I Love Python Programming"  #A Complete String
>>> B=A.split()          # Split a String into Words
>>> B                    #Print the contents of B
['Wow!!!', 'I', 'Love', 'Python', 'Programming']
```

Explanation In the above example, we have initialised string to A as "Wow!!! I Love Python Programming". In the next line, the statement, B = A.split() is used to split "Wow!!! I Love Python Programming" into the list ['Wow!!!', 'I', 'Love', 'Python', 'Programming'].

It is fine to split a string without a delimiter. But what if the string contains the delimiter? A string containing a delimiter can be split into words by removing the delimiter. It is also possible to remove the delimiter from the string and convert the entire string into a list of words. In order to remove the delimiter, the `split()` method has a parameter called split(delimiter). The parameter delimiter specifies the character to be removed from the string. The following example illustrates the use of a delimiter inside the `split()` method.

Example

```
>>> P="My-Data-of-Birth-03-June-1991" # String with Delimiter
'-'

>>> P                                  # Print the Entire String

   'My-Data-of-Birth-03-June-1991'

>>> P.split('-')               #Remove the delimiter '-' using
split method.

   ['My', 'Data', 'of', 'Birth', '03', 'June', '1991']
```

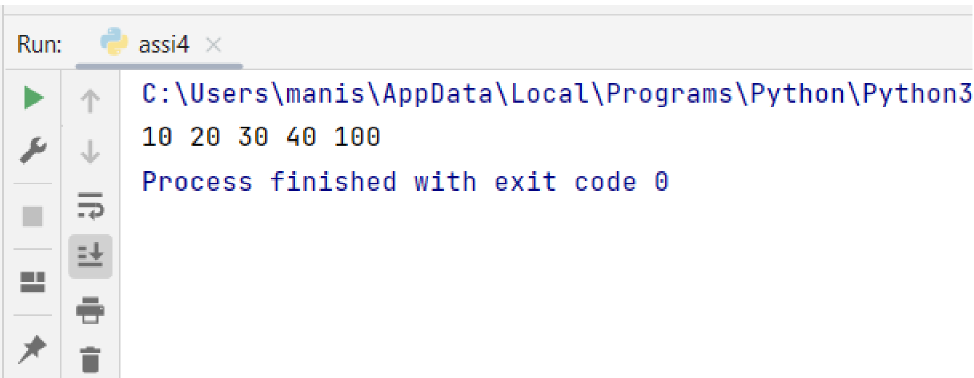
b. Passing list to a function:

Ans.

As list is a mutable object. A programmer can pass a list to a function and can perform various operations on it. We can change the contents of a list after passing it to a function. Since a list is an object, passing a list to a function is just like passing an object to a function.

Consider the following example to print the contents of a list after the list is passed to a function.

```
def print_list(lst):
    for num in lst:
        print(num, end=" ")
lst = [10,20,30,40,100]
print_list(lst)
```



c. Returning list from a function.
Ans.

We can pass a list while invoking a function. Similarly, a function can return a list. When a functionreturns a list, the list's reference value is returned.

Consider the following example to pass a list to a function. After passing, reverse the elementsof the list and return the list.

```
def Reverse_List(Lst):
    print('List Before Reversing = ',Lst)
    Lst.reverse()   # The reverse() to reverse the contents of list
    return Lst      #  Return List
Lst=[10,20,30,40,3]
print('List after Reversing = ',Reverse_List(Lst))
```

Output

```
List Before Reversing = [10, 20, 30, 40, 3]
List after Reversing = [3, 40, 30, 20, 10]
```

Q10. Define Searching techniques.

Ans.

Searching is a technique of quickly finding a specific item from a given list of items, such as a number in a telephone directory. Each record has one or more fields such as name, address and number. Among the existing fields, the field used to distinguish records is called the key. Imagine a situation where you are trying to find the number of a friend. If you try to locate the record corresponding to the name of your friend, the 'name' will act as the key. Sometimes there canbe various mobile numbers allotted to the same name. Therefore, the selection of the key playsa major role in defining the search algorithm. If the key is unique, the records are also identified uniquely. For example, mobile number can be used as the key to search for the mobile number ofa specific person. If the search results in locating the desired record then the search is said to be successful, else the search is said to be unsuccessful. Depending on way the information is stored for searching a particular record, search techniques are categorised as:

(a) Linear or sequential search
(b) Binary search

Q11. Explain Linear Search in details.

Ans.

In linear search, elements are examined sequentially starting from the first element. Element to besearched, i.e. the (key element), is compared sequentially with each element in a list. The search process terminates when the element to be searched, i.e. the key element, matches with the element present in the list or the list is exhausted without a match being found in it. If the element to be searched is found within the list, the linear search returns the index of the matching element. If the element is not found, the search returns -1.

```
def Linear_Search(My_List,key):
    for i in range(len(My_List)):
        if(My_List[i]==key):
            #print(key,"is found at index",i)
            return i
            break
    return -1
My_List=[12,23,45,67,89]
```

```
print("Contents of List are as follows:")
print(My_List)
key=(int(input("Enter the number to be searched:")))
L1=Linear_Search(My_List,key)
if(L1!=-1):
    print(key ," is found at position",L1+1)
else:
    print(key," is not present in the List")
```

Output

```
#Test Case 1
Contents of List are as follows:
[12, 23, 45, 67, 89]
Enter the number to be searched:23
23  is found at position 2
#Test Case 2
Contents of List are as follows:
[12, 23, 45, 67, 89]
Enter the number to be searched:65
65  is not present in the List
```

Explanation In the above program, we have defined the function called `Linear _ Search()`. The list and element to be searched, i.e. the key, is passed to the function. The comparison starts from the first element of the list. The comparison goes on sequentially till the key element matches the element present within the list or the list is exhausted without a match being found.

Unordered List—Analysis of Sequential Search

Table 9.1 shows that the analysis has been made with respect to the unordered list, i.e. if the content of the list is not in any order, either ascending or descending.

Table 9.1 Sequential search analysis in an unordered list

Case	*Best Case*	*Worst Case*	*Average Case*
Element is present in the list	1	N	N/2
Element is not present in the list		N	N N

Sorted List—Analysis of Sequential Search

Expected number of comparisons required for an unsuccessful search can be reduced if the list is sorted.

Example

```
List1[] =   10 15 20 25 50 60 70 80

Element to be searched = 30
```

Search should terminate here.

Assuming the elements are stored in an ascending order, the search should terminate as soon asthe value of the element in the list is greater than value of the element (key) to be searched or the key (element to be searched) is found (Table 9.2).

Table 9.2 Sequential search analysis on a sorted list

	Best Case	*Worst Case*	*Average Case*
Element is present in the list	1	N	N/2
Element is not present in the list		1	N N/2

Q12. Explain Binary search in details.

Ans.

Linear search is convenient for a small list. What happens if the size of a list is large? Let us consider the size of the list is 1 Million (2^{20}). So, if we want to search using a sequential search algorithm then in the worst case we require 2^{20} comparisons. This means that a sequential search algorithm is not suitable for a large list. Thus, we require more efficient algorithms. In this section, we will explore how binary search is a simpleand efficient algorithm.

For binary search, the elements in a list must be in a sorted order. Let us consider the list is in ascending order. The binary search compares the element to be searched, i.e. the key element withthe element in the middle of the list. The binary search algorithm is based on the following three conditions:

1. If the key is less than the list’s middle element then a programmer has to search only in thefirst half of the list.

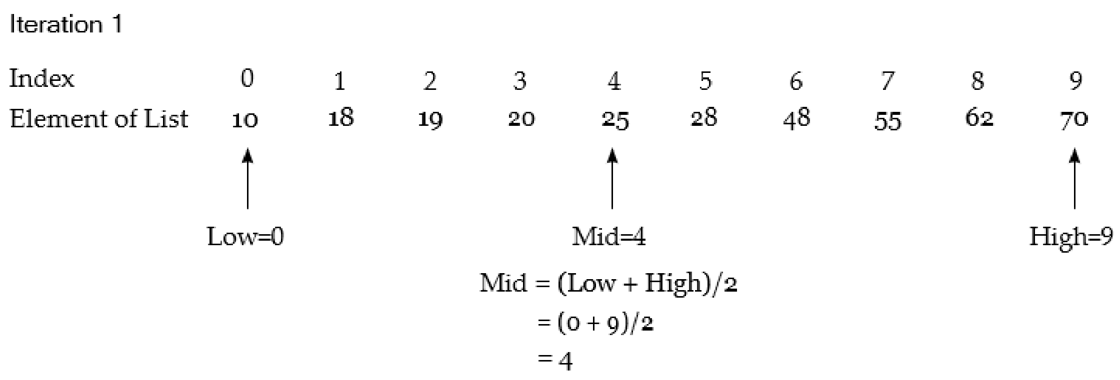
2. If the key is greater than the list's middle element then a programmer has to search only inthe second half of the list.
3. If the element to be found, i.e. the key element is equal to the middle element in the list thenthe search ends.
4. If the element to be found is not present within the list then the it returns None or -1 whichindicates the element to be searched is not present in the list.

Example of Binary Search

Consider the sorted list of 10 integers given below.

10 18 19 20 25 28 48 55 62 70

Element to be searched = 48



Now we will compare the middle element which is 25 with the element that we want to search,
i.e. 48.

Since 48>25,

we will eliminate the first half of the list and we will search again in the second half of the list. Now,

Low = Mid + 1 = 4 + 1 = 5 #Change the Position of Low

High = 9 # High will remain as earlier.

Iteration 2

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Element of List	10	18	19	20	25	28	48	55	62	70
						↑		↑		↑
						Low=5		**Mid=7**		**High=9**

Mid = (Low +High)/2

= (5 + 9)/2

= 7

Now we will compare the middle element which is **55** with element we want to search, i.e. 48.

Since **48 < 55**,

we will search the element in the left half of the list.

Now,

Low = 5 **#It will remain as it is.**

High = Mid-1 = 7-1 **#Change the Position of High**

= 6

Iteration 3

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Element of List	10	18	19	20	25	28	48	55	62	70
						↑	↑			
						Low=5 **Mid=5**	**High=6**			

Mid = (Low +High)/2

= (5 + 6)/2

= 5

Now we will compare the middle element which is **28** with the element we want to search, i.e. 48.

Since **28 < 48**,

we will search the element in the right portion of the mid of the list.

Now,

Low = mid+1 = 6 **#Change the Position of Low**

High = 6 **#High will remain as it is.**

Iteration 4

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Element of List	10	18	19	20	25	28	48	55	62	70

Low=6 High= Mid=6

Mid = (Low + High)/2

= 6

Now we will compare the middle element which is 48 with the element we want to search, i.e. 48.

Since 48=48, the number is found at index position 6.

Q13. Describe Sorting and types of sorting.
Ans.

Consider a situation where a user wants to pick up a book from a library but finds that the books are stacked in no particular order. In this situation, it will be very difficult to find any book quicklyor easily. However, if the books were placed in some order, say alphabetically, then a desired titlecould be found with little effort. As in this case, sorting is used in various applications in general to retrieve information efficiently.

Sorting means rearranging the elements of a list, so that they are organised in some relevant order. The order can be either ascending or descending. Consider a list L1, in which the elements are arranged in an ascending order in a way that L1[0] < L1[1] < ....<L1[N].

Example

If a list is declared and initialised as:

L1 = [9, 3, 4, 2, 1]

The sorted list in an ascending

order can be:L1 = [1, 2,

3, 4, 9]

Form the above example, it is clear that sorting is a process of converting an unordered set ofelements into an ordered set.

Types of Sorting

Sorting algorithms are divided into two main categories, viz.

1. Internal sorting
2. External sorting

If all the records to be sorted are kept internally in the main memory then they can be sorted using internal sort. However, if a large number of records are to be sorted and kept in secondary storage then they have to be sorted using external sort.

1. ***Internal sorting algorithms**:* Any sorting algorithm which uses the main memory exclusively during sorting is called an internal sort algorithm. It takes advantage of the random access nature of the main memory. Internal sorting is faster than external sorting.
2. ***External sorting algorithms:*** External sorting is carried on secondary storage. Therefore, any sorting algorithm which uses external memory, such as tape or disk during sorting is called external sort algorithm. It is carried out if the number of elements to be stored is too large to fit in the main memory. Transfer of data between secondary and main memory is bestdone by moving blocks of contiguous elements.

The various sorting algorithms are Bubble sort, Selection sort, Insertion sort, Quick sort and Merge sort.

Q14. Explain Bubble sort in details.

Ans.

Bubble sort is the simplest and oldest sorting algorithm. Bubble sort sorts a list of elements by repeatedly moving the largest element to the highest index position of the list. The consecutive adjacent pair of elements in both the lists is compared with each other. If the element at lower indexis greater than the element at higher index then the two elements are interchanged so that the element with the smaller value is placed before the one with a higher value. The algorithm repeats this process till the list of unsorted elements is exhausted. This entire procedure of sorting is calledbubble sort. The algorithm derives its name as bubble sort because the smaller elements bubble tothe top of the list.

Example of Bubble Sort

Consider the elements within a list as:

L1 = [30, 50, 45, 20, 90, 78]

Sort the list using bubble sort.

Solution

Iteration 1

30	**40**	45	20	90	78	**No Exchange**
30	**40**	**45**	20	90	78	**No Exchange**
30	40	**45**	**20**	90	78	**Exchange**
30	40	20	**45**	**90**	78	**No Exchange**
30	40	20	45	**90**	**78**	**Exchange**
30	40	20	45	78	**90**	**Output of bubble sort after the first iteration. But still the output is not in a sorted order. Repeat the above steps for iteration 2.**

Iteration 2

Apply bubble sort to the output of the first iteration.

30	**40**	20	45	78	90	**No Exchange**
30	**40**	**20**	45	78	90	**Exchange**
30	20	**40**	**45**	78	90	**No Exchange**
30	20	40	**45**	**78**	90	**No Exchange**
30	20	40	45	78	90	**Output of bubble sort after the second iteration. But still the output is not in a sorted order. Repeat the above steps for iteration 3.**

Iteration 3

Apply bubble sort to the output of the second iteration.

30	**20**	40	45	78	90	**Exchange**
20	**30**	**40**	45	78	90	**No Exchange**
20	30	**40**	**45**	78	90	**No Exchange**
20	30	40	**45**	**78**	90	**No Exchange**
20	30	40	45	78	90	**No Exchange**

Thus, in third iteration itself we have obtained a sorted list of elements in an ascending order.

Working of Bubble Sort

In the above example, the working of bubble sort can be generalised as:

1. In each iteration, the first element of the list, i.e. L[1] is compared with the second element ofthe list, i.e. L[2] then L[2] is compared with L[3], L[3]is compared with L[4] and so on. Finally, L[N-1] is compared with L[N]. This process is continued till we obtain the list in a sorted order.

2. In the second iteration, L[1] is compared with L[2], L[2]is compared with L[3] and so on. Finally, L[N-2] is compared with L[N-1]. The iteration 2 just requires N-2 comparisons. Therefore, at the end of the second iteration, the second biggest element is placed at the second highest index position of the list.
3. Similarly, the above process is continued for the subsequent iterations. Therefore, in the lastiteration we obtain all the elements within the list in a sorted order.

Q15. Explain Insertion Sort in details.

Ans.

Insertion sort is based on the principle of inserting an element in its correct place in a previously sorted list. It always maintains a sorted sublist in the lower portion of the list. Each new element isinserted back into the previous sub list. Thus, insertion sort sorts a list of elements repeatedly by inserting a new element into a sorted sublist until the whole list is sorted. An example of insertion sort is given below.

Example

Consider the unsorted list as

MyList = [15,0,11,19,12,16,14]

Initially, the sorted sublist contains the first element in the list, i.e. 15

15	0	11	19	12	16	14

⊙ **STEP 1:** Initially, the sorted sublist contains the first element in the list, i.e. 15. Now insert the next element from the list, i.e. 0 into the sublist.

0	15	11	19	12	16	14

⊙ **STEP 2:** The sorted sublist is [0, 15]. Insert 11 into the sublist.

0	11	15	19	12	16	14

⊙ **STEP 3:** The sorted sublist is [0,11,15]. Insert 19 into the sublist.

0	11	15	19	12	16	14

⊙ **STEP 4:** The sorted sublist is [0,11,15,19]. Insert 12 into the sublist.

0	11	12	15	19	16	14

⊙ **STEP 5:** The sorted sublist is [0,11,12,15,19]. Insert 16 into the sublist.

0	11	12	15	16	19	14

⊙ **STEP 6:** The sorted sublist is [0,11,12,15,16,19]. Insert 14, into the sublist.

0	11	12	15	16	19	14

⊙ **STEP 7:** The sorted sublist is [0,11,12,14,16,19].

0	11	12	14	15	16	19

Finally, we obtain the sorted list of elements in Step 7.

Q16. Explain Quick Sort in details.

Ans.

Quick sort is one of the fastest internal sorting algorithms. It is based on the following three main strategies:

1. Split or Partition: Select a random element called pivot from the sequence of elements to be sorted. Suppose the selected element is X, where X is any number. Now split (divide) the list into the two small lists, viz. Y and Z such that:
   - All the elements of the first part Y are less than the selected element pivot.
   - All the elements of the second part Z are greater than the selected element pivot.
2. Sort the sub-arrays.
3. Merge (join/concatenate) the sorted sub-array.

The split divides the lists into two smaller sublists. When these sublists are ultimately sorted recursively using quick sort these sublists are called conquered. Therefore, the quick sort algorithm is also so called the divide and conquer algorithm.

Suppose there are N elements as a[0], a[1], a[2],......a[N-1]. The steps for using the quick sort algorithm are given below.

- STEP 1: Select any element as the pivot. For example, select the element stored at the first position in a list as the pivot element. Although there are many ways to choose the pivot element, we will use the first item from the list. It helps to split a list into twoparts.

  Pivot = a[First] //Select Pivot Element

  where the value of First is 0.

- **STEP 2:** Initialize the two pointers i and j.

  i = First+1 (The first (low) index of a list)

  j = Last (The last (upper) index of a list)

- **STEP 3:** Now increase the value of i until we locate an element that is greater than the pivot element.

  while i<=j and a[i] <= Pivot

  i++

- **STEP 4:** Decrease the value of j until we find a value less than the pivot element.

  while i<=j and a[j]>=Pivot

  j--

- STEP 5: If i<j interchange i a[i] and a[j].
- STEP 6: Repeat Steps 2 to 4 until i>j.
- STEP 7: Interchange the selected data element pivot and a[j].

Q17. Explain Merger sort in details.

Ans.

All sorting algorithms are mainly used for internal sorting where the data to be sorted fits in the main memory. When the data to be sorted resides in a file oron a disk and does not fit in the available memory, the merge sort method is used. Merge sort is a well-known and efficient method for external sorting.

Like quick sort, merge sort is also based on three main strategies:

(a) Split the list into two sub lists (Split or Divide): Split implies partitioning the n elements of a list into two sublists, where each sublist contains n/2 elements.

(b) Sort sublists (Conquer): Sorting two sub-arrays recursively using merge sort.

(c) Merge the sorted sublists (Combine): Combine implies merging two sorted sublists, each of size n/2, to produce a sorted list of n elements.

Example of Merge Sort

Consider the following elements within a list.

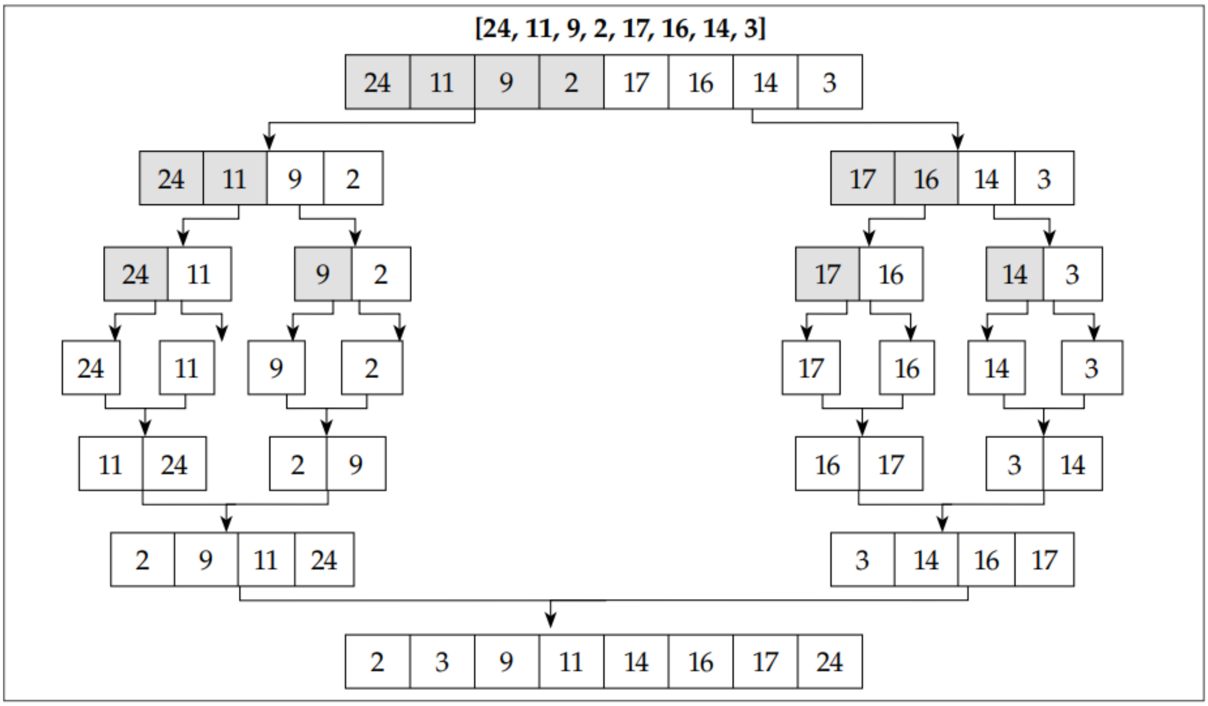


Figure 9.1 Example of merge sort

The list in Figure 9.1 has 8 elements. The index of the first element is i=0 and the index of the last element is j=7. In order to divide the above list around the middle element, the index of the middleelement is found, i.e. mid=(i+j)/2.

Therefore, i = 0 and j = 7

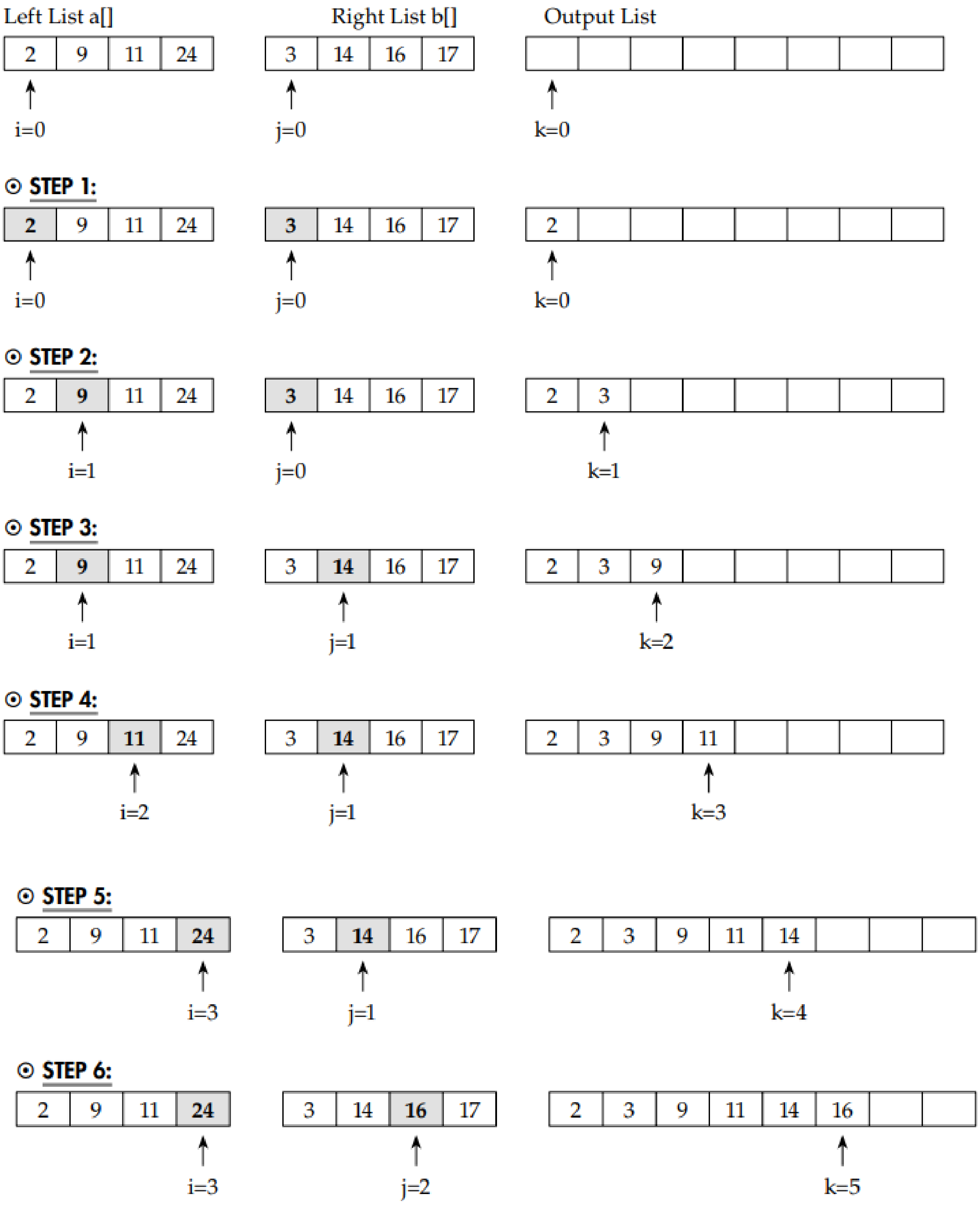
$$Mid = (i + j)/ 2 = (0+7)/2 = 3.$$

Merge sort is applied recursively to the left part of the list from i = 0 to j = 3. After sorting of the left half of the list, the right half of the list is sorted from i= 4 to j = 7 recursively using merge sort. After sorting the left half of the list from i = 0 to j = 3 and right half of the list from i=4 to j=7, the two lists are merged to produce a single sorted list.

Merging Operation in Merge Sort

A fundamental operation in the merge sort algorithm is merging of two sorted lists. The merging algorithm takes two sorted lists a[] and b[], i.e. (left and right list) as the input and the third list c[] as the output list. Each element of a list, i.e. (LeftList) a[i] is compared with the elements of the(RightList) b[j]. The smaller element among a[i] and b[j] is copied to the output list c[k]. When either of the input list is exhausted, the remainder of the other list is copied to the output list c.

In the above example, we obtained two sorted sublists, viz. a[] as the left list and b[] as the rightlist, as shown.



⊙ **STEP 7:**

2	9	11	**24**
			i=3

3	14	16	**17**
			j=3

2	3	9	11	14	16	17	
						k=6	

⊙ **STEP 8:** In Step 7, the list b[] is exhausted. Therefore, the remaining elements of the list a[] are added to the output list.

2	9	11	**24**
			i=3

3	14	16	17

2	3	9	11	14	16	17	24
							k=7

Finally, in Step 8, all the elements of the list are sorted.

Python Assignment No. 5

Q1. How to define a class. Write a program to create a simple class and print message "Welcome to Object Oriented Programming" and print the address of the instance of class.

Ans. As discussed above, class is another name for type in Python. A class may contain data in the form of fields. Fields are also called attributes and coded in the form of procedures known as methods. Finally, a programmer has to create an object of its own class, where the object representsan entity which can be easily identified. For example, person, vehicle, fan, book etc. represent realobjects. Each object has a unique identity, state and behaviour. The state of the object is also calledproperty or attribute. For example, a circular object has a data field radius which is a property that characterises a circle. The syntax to define a class in Python is given as follows:

```
Class Class_Name:
    Initializer
    attributes
    methods()
    Statement(s)
```

Program:

```
class MyFirstProgram:
    print(' Welcome to Object-oriented Programming')
C=MyFirstProgram()     #Instance of class.
print(C)
```

Output

```
Welcome to Object-oriented Programming
<__main__.MyFirstProgram object at 0x028B6C90>
```

Explanation Name of the class is MyFirstProgram. The instance, i.e. 'C' of the class is created. Inside the class, the print statement is used to display the welcome message. Additionally, the lastprint statement is used to display the address of the computer's memory where the object 'C' is stored.

Q2. Write a program to calculate the area of a rectangle by assigning the value to the attributes ofa rectangle, i.e. length and breadth.

Ans.

```
class Rectangle:
    length=0;          #Attribute  length
    breadth=0;         #Attribute  breadth
R1 = Rectangle ()      #Instance of a class
print(R1.length)       #Access attribute length
print(R1.breadth)      #Access attribute breadth
```

Output

```
0
0
```

Q3. Write a program to calculate area of rectangle by assigning the value to the attributes of rectangle, i.e. length and breadth.

Ans.

```
class Rectangle:
    length=0;          #Attribute  length
    breadth=0;         #Attribute  breadth
R1 = Rectangle ()      #Instance of a class
print('Initial values of Attribute')
print('Length   = ',R1.length)       #Access attribute length
print('Breadth  = ',R1.breadth)      #Access attribute breadth
print('Area of Rectangle = ',R1.length * R1.breadth )
R1.length   =  20   #Assign value to attribute length
R1.breadth  =  30   #Assign  value to attribute breadth
print('After reassigning the value of attributes')
print('Length = ',R1.length )
print('Breadth = ',R1.breadth )
print('Area of Rectangle is ',R1.length * R1.breadth)
```

Output

```
Initial values of Attribute
Length   =  0
Breadth  =  0
Area of Rectangle =  0
After reassigning the value of attributes
Length =  20
Breadth =  30
Area of Rectangle is  600
```

Q4. Explain self parameter.

Ans.

The Self-parameter

To add methods to an existing class, the first parameter for each method should be self. There is only one difference between class methods and ordinary functions. The self-parameter is used in the implementation of the method, but it is not used when the method is called. Therefore, the self-parameter references the object itself. Program illustrates the self-parameter and addition of methods to an existing class.

```
class MethodDemo:
    def Display_Message(self):
        print('Welcome to Python Programming')
ob1 = MethodDemo()      #Instance of a class
ob1.Display_Message()   #Calling Method
```

Output

```
Welcome to Python Programming
```

Explanation In the above program, the Display_Message() method takes no parameters but still the method has the self-parameter in the function definition. Therefore, the self-parameter refers to the current object itself. Finally, the method is called and the message is displayed.

Q5. Write a program to create a class named circle. Pass the parameter radius to the method named Calc_Area() and calculate area of circle.

Ans.

```
import math
class Circle:
    def Calc_Area(self,radius):
        print('radius = ',radius)
        return math.pi*radius**2
ob1 = Circle()
print('Area of circle is ',ob1.Calc_Area(5))
```

Output

```
radius =  5
Area of circle is  78.53981633974483
```

Explanation The class with name Circle is created as shown above. The extra parameter radius is passed to a method defined inside the class Calc_Area(). The instance ob1 of a class is created and used to call the method of the existing class. Even though the method Calc_Area() contains two parameters, viz. self and radius, only one parameter should be passed, viz. the radius of the circle while calling the method.

Q6. Write a program to display attributes in a given class.

Ans.

```
class DisplayDemo:
    Name = '';          #Attribute
    Age = ' ';          #Attribute
    def read(self):
        Name=input('Enter Name of student: ')
        print('Name = ',Name)
        Age=input('Enter Age of the Student:')
        print('Age = ',Age)
D1 = DisplayDemo()
D1.read()

#Display attributes using dir() on the interactive mode
>>>(dir( DisplayDemo )
['Age', 'Name', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
'__init__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__
reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__
str__', '__subclasshook__', '__weakref__', 'read']
```

Q7. Explain __init__ method(constructor). Write a simple program using init method.

Ans.

There are many inbuilt methods in Python. Each method has its own significance. The importanceof the_init method is explained ahead.

The_init method is known as an initialiser. It is a special method that is used to initialise theinstance variable of an object. This method runs as soon as an object of a class is instantiated. The syntax of adding_init method to a class is given as follows:

```
class Class_Name:
    def __init__(self):     #__init__ method
        ..............................
        ..............................
```

init needs to be preceded and followed by two underscores. Also_init method must have self as the first argument. As self refers to the object itself, it refers to the object that invokes the method. The self-parameter within the_init_ method automatically sets the reference forthe object just created. The_init method can also have positional and/or keyword arguments.

```
class Circle:
    def __init__(self,pi):
        self.pi = pi
    def calc_area(self,radius):
        return self.pi*radius**2
C1=Circle(3.14)
print(' The area of Circle is ',C1.calc_area(5))
```

Output

```
The area of Circle is 78.5
```

Explanation In the above program we have created a class named Circle. The class contains two different methods, viz. one is__init_ method and another calc_area() to calculate the area of a circle. Notice that in the above program we do not explicitly call the_init method. We have created an instance of the class Circle, i.e. C1. While creating the instance of the class we have passed the arguments following the class name to initialise the instance variable of an object.

Q8. Write a program to pass object as a parameter to a method.

Ans.

```
class Test:
    a = 0
    b = 0
    def __init__(self, x , y):
        self.a = x
        self.b = y
    def equals(self, obj):
        if(obj.a == self.a and obj.b == self.b):
            return True
        else:
            return False
Obj1 = Test(10,20)
Obj2 = Test(10,20)
Obj3 = Test(12,90)
print(' Obj1 == Obj2 ',Obj1.equals(Obj2))
print(' Obj1 == Obj3 ',Obj1.equals(Obj3))
```

Output

```
Obj1 == Obj2  True
Obj1 == Obj3  False
```

Q9. Explain __del__ (destructor) method. Write a program to illustrate the use of __del__ method.

Ans.

Like other object-oriented programming languages, Python also has a destructor. The method
__del_ denotes the destructor and the syntax to define destructor is.

```
def __del__(self)
     block
```

Python invokes the destructor method when the instance is about to be destroyed. It is invokedone per instance. The self refers to the instance on which the __del__() method is invoked. In other words, Python manages garbage collection of objects by reference counting. This functionis executed only if all the references to an instance object have been removed.

```
class Destructor_Demo:
    def __init__(self):        #Constructor
        print('Welcome')
    def __del__(self):         #Destructor
        print('Destructor Executed Successfully')
Ob1=Destructor_Demo()          #Instantiation
Ob2 = Ob1
Ob3 = Ob1     #Object Ob2 and Ob3 refers to same object
print(' Id of Ob1 = ',id(Ob1))
print(' Id of Ob2 = ',id(Ob2))
print(' Id of Ob3 = ',id(Ob3))
```

```
del Ob2      #Remove reference Ob2
del Ob1      #Remove reference Ob1
del Ob3      #Remove reference Ob3
```

Output

```
Welcome
 Id of Ob1 =  47364272
 Id of Ob2 =  47364272
 Id of Ob3 =  47364272
Destructor Executed Successfully
```

Explanation In the above example, we have used constructor **__init__** and destructor **__del__** functions. Initially we have instantiated the object **Ob1** and then assigned aliases **Ob2** and **Ob3** to it. The inbuilt function **id()** is used to confirm that all the three aliases reference to the same object. Finally, all the aliases are removed using the del statement.

Q10. Explain Inheritance in details. Write a simple program on inheritance.

Ans. Inheritance is one of the most useful and essential characteristics of object-oriented programming.The existing classes are the main components of inheritance. New classes are created from the existing ones. The properties of the existing classes are simply extended to the new classes. Anew class created using an existing one is called a derived class or subclass and the existingclass is called a base class or super class. An example of inheritance is shown in Figure 10.2. The relationship between base and derived class is known as kind of relationship. A programmer can define new attributes, i.e. (member variables) and functions in a derived class.

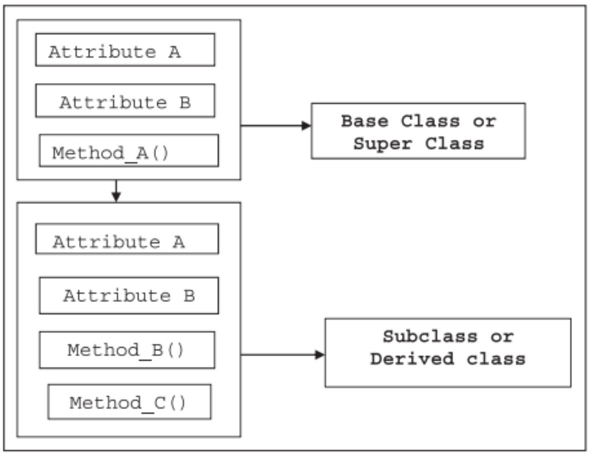


Figure 10.2 Simple example of inheritance

```
class A:
    print('Hello I am in Base Class')
class B(A):
    print('Wow!! Great ! I am Derived class')
ob2 = A()  #Instance of class B
```

Output

```
Hello I am in Base Class
Wow!! Great! I am Derived class
```

Q11. Explain types of inheritance in details with suitable block diagram.

Ans.

Theprocess of inheritance can be simple or complex according to the following:

1. *Number of base classes:* A programmer can use one or more base classes to derive a singleclass.
2. *Nested derivation:* The derived class can be used as the base class and a new class can bederived from it. This is possible at any extent.

Inheritance can be classified as: (i) single inheritance, (ii) multilevel inheritance and (iii) multipleinheritance. Each of these has been described in detail as follows:

(i) ***Single inheritance:*** Only one base class is used for deriving a new class. The derived class is notused as the base class.

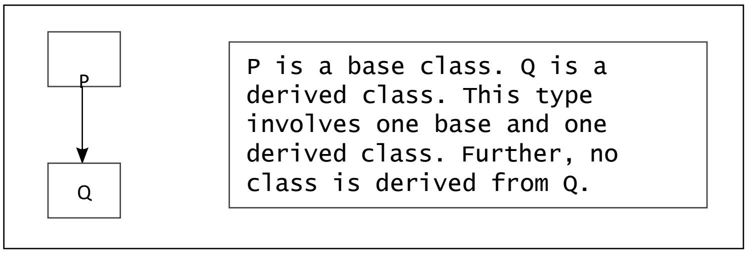


Figure 10.3 Single inheritance

(ii) ***Multilevel inheritance:*** When a class is derived from another derived class, the derived classacts as the base class. This is known as multilevel inheritance.

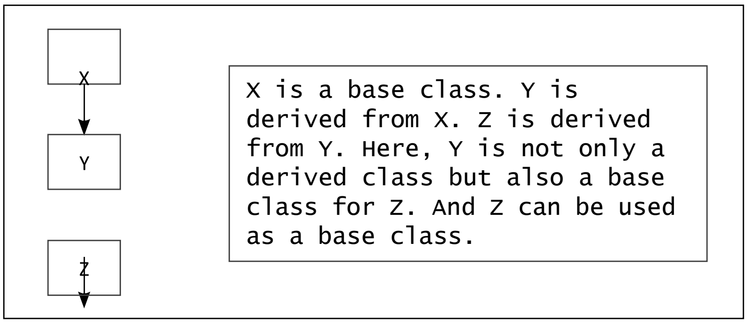


Figure 10.4 Multilevel inheritance

(iii) ***Multiple inheritance:*** When two or more base classes are used for deriving a new class, it iscalled multiple Inheritance.

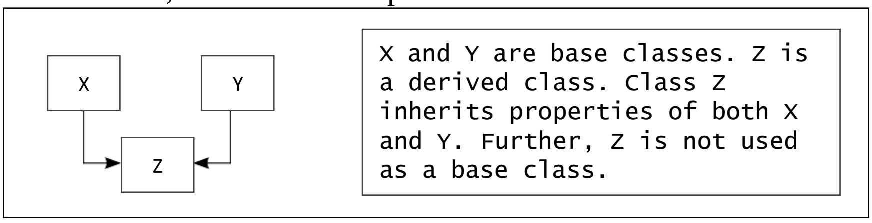


Figure 10.5 Multiple inheritance

Q12. Explain Tuple. Describe inbuilt function for tuple along with example.

Ans.

Tuples work exactly like lists. A tuple contains a sequence of items of many types. The elements oftuples are fixed. Once a tuple has been created, we cannot add or delete elements, or even shuffle their order. Hence, tuples are immutable. This means that once created, they cannot be changed. Since tuples are immutable, their length is also fixed. A new tuple must be created to grow or shrink an earlier one.

Inbuilt Functions for Tuples

Python provides various inbuilt functions that can be used with tuples. Some of these are shownin Table 11.1.

Table 11.1 Inbuilt functions that can be used with tuples

Inbuilt Functions	*Meaning*
len()	Returns the number of elements in a tuple
max()	Returns the element with the greatest value
min()	Returns the element with the smallest value
sum()	Returns the sum of all the elements of a tuple
index(x)	Returns the index of element x
count(x)	Returns the number of occurrences of element x

Example

```
>>> t1=("APPLE")
>>> len(t1)  #Return the length of tuple t1
5
>>> max(t1)  #Return Element from tuple with Maximum Value
'P'
>>> min(t1)  #Return Element from tuple with Minimum Value
'A'
>>> t1.index('A')
0
>>> t1.count('P')
2
```

Q13. Define Python Set. How to create set(creating empty set, creating set of 4 elements).

Ans.

A set is an unordered collection of unique elements without duplicates. A set is mutable. Hence, we can easily add or remove elements from a set. The set data structure in Python is used to support mathematical set operations.

Creating Sets

A programmer can create a set by enclosing the elements inside a pair of curly brackets {}. The elements within a set can be separated using commas. We can also create a set using the inbuilt set()function, or from an existing list or tuple.

Examples

```
>>>S1 =set()   # Creates an empty Set
>>>S1          # Print Set S1
set()
>>> type(S1)   # Check type of S1
<class 'set'>


>>> S1={10,20,30,40}    # Create set of 4 elements
>>> S1                  # Print Set S1
{40, 10, 20, 30}
```

```
>>> S2=[1,2,3,2,5]            # Create List
>>> S2                        # Print List
[1, 2, 3, 2, 5]
>>> S3=set(S2)                # Convert List S2 to Set
>>> S3                        #Print S3 (Removes duplicate from the List)
{1, 2, 3, 5}
```

```
>>> S4=(1,2,3,4)            # Create Tuple
>>> S5=set(S4)              # Convert Tuple to Set
>>> S5                      # Print S5
{1, 2, 3, 4}
```

Q14. Explain Set operations in details.

Ans.

In mathematics or everyday applications we often use various set operations, such as union(), intersection(), difference()and symmetric_difference(). All these methods are partof the set class.

- ***The union()Method***

The union of two sets A and B is a set of elements which are in A, in B or in both A and B. We canuse the union method or the |operator to perform this operation.

Example

```
>>> S1={1,2,3,4}
>>> S2={2,4,5,6}
>>> S1.union(S2)
{1, 2, 3, 4, 5, 6}
```

```
>>>S1 | S2
{1, 2, 3, 4, 5, 6}
```

- ***The intersection()Method***

The intersection of two sets A and B is a set which contains all the elements of A that also belong to B. In short, intersection is a set which contains elements that appear in both sets. We can use intersection methods or the & operator to perform this operation.

Example

>>> S1={1,2,3,4}

>>> S2={3,4,5,6}

>>> S1.intersection(S2)

{3, 4}

>>> S1 & S2

{3, 4}

- ***The difference()Method***

The difference between two sets A and B is a set which contains the elements in set A but not in set

A. We can use the difference method or the – operator to perform the difference operation.

Example

>>> A={1,2,3,4}

>>> B={3,4,5,6}

>>> A.difference(B)

{1, 2}

>>>>>> A-B

{1, 2}

- ***The symmetric_difference()***

The symmetric difference is a set which contains elements from the either set but not in both sets. We can use symmetric_difference method or the ^ (exclusive) operator to perform this operation.

Example

>>> S1={1,2,3,4}

>>> S2={3,4,5,6}

>>> S1.symmetric_difference(S2)

{1, 2, 5, 6}

>>> S1^S2

{1, 2, 5, 6}

Q15. Explain basics of dictionaries along with suitable diagram.

Ans.

In Python, a dictionary is a collection that stores values along with keys. The sequence of such keyand value pairs is separated by commas. These pairs are sometimes called entries or items. All entries are enclosed in curly brackets { and }. A colon separates a key and its value. Sometimes, items within dictionaries are also called associative arrays because they associate a key with a value.

Simple examples of dictionaries are given as follows:

Phonebook - {“Amit”:“918624986968”, “Amol”:“919766962920”}

Country Code Information - {“India”:“+91”,“USA”:“+1”,“Singapore”: “+65”}

The structure of a dictionary is shown in Figure 11.1a. The above phonebook example is illustrated in Figure 11.1b.

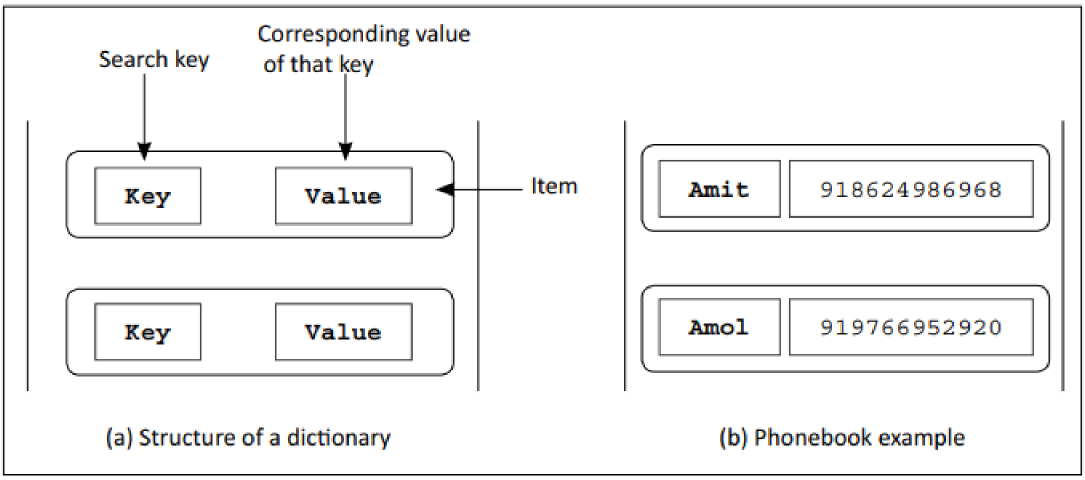


Figure 11.1 a and b Dictionary—structure and example

Keys are like an index operator in a dictionary. A key can be of any type. Therefore, a dictionarymaps a set of objects, i.e. keys to another set of objects, i.e. values. It is a mapping of unique keys to values, i.e. each key is mapped to one value. Also, dictionaries do not contain any duplicate keys.

Q16. Describe 4 different ways to create dictionaries.

Ans. We can create a dictionary by enclosing the items inside a pair of curly brackets {}. One way to starta dictionary is to create an empty dictionary first and then add items to it.

Creating an Empty Dictionary

Example

```
>>>D1 = {}                # Create Empty Dictionary
>>>D1                     # Print Empty Dictionary
{}
>>> type(D1)              # Check the type of D1
<class ‘dict’>
```

Creating a Dictionary with Two Items

To create a dictionary of two items, the items should be in the form of key:value and separated bycommas.

```
>>> P={“Amit”:”918624986968”, “Amol”:”919766962920”}
>>> P             #Display P
{‘Amit’: ‘918624986968’, ‘Amol’: ‘919766962920’}
```

Creating Dictionaries in Four Ways:

Example

```
#Way 1:
>>>D1={‘Name’:’Sachin’,’Age’:40}
>>> D1
{‘Name’: ‘Sachin’, ‘Age’: 40}

#Way 2:
>>> D2={}
>>> D2[‘Name’]=’Sachin’
>>> D2[‘Age’]=40
>>> D2
{‘Name’: ‘Sachin’, ‘Age’: 40}

#Way 3:
>>> D3=dict(Name=’Sachin’,Age=40)
>>> D3
{‘Name’: ‘Sachin’, ‘Age’: 40}
```

#Way 4:

>>> dict([('name','Sachin'),('age',40)])

{'age': 40, 'name': 'Sachin'}

Explanation

In the above example, we have created dictionaries in four different ways. We can select the first way if we know all the contents of a dictionary in advance. The second way, if we want to add onefield at a time. The third way requires all keys to string. The fourth way is good if we want to buildkeys and values at runtime.

Python Assignment No. 6

Q1. Explain Turtle module in details.

Ans.

To start, a programmer can use interactive mode (command line) or script mode of Python. The steps required to start graphics programming using the Turtle module in interactive mode of Python are given as follows:

- STEP 1: Launch Python by pressing the start button in Windows and writing Python in the search box. Click on Python IDLE to start the interactive mode. The following windowwill then appear (Figure 12.1).

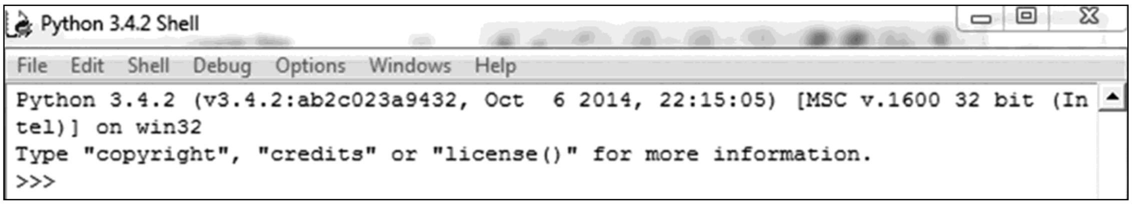


Figure 12.1

- STEP 2: At the Python's statement prompt >>> type the following command to import theTurtle module.

  >>> import Turtle #import Turtle module

- STEP 3: Type the following command to show the current location and direction of the Turtle.

  >>> Turtle.showTurtle()

After the execution of the above statement, Python's Turtle graphics window will be displayedas shown in Figure 12.2.

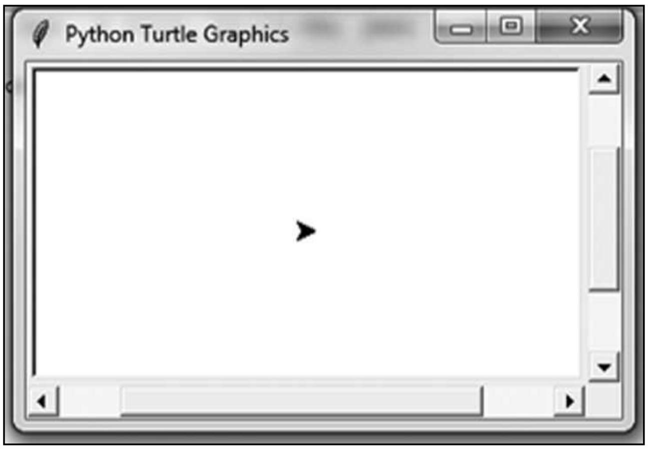


Figure 12.2 Python's Turtle Graphics Window

The Turtle is like a pen. The arrowhead indicates the current position and direction of the pen.

Initially, the Turtle is positioned at the center of the window.

Q2. Explain moving turtle in any direction.

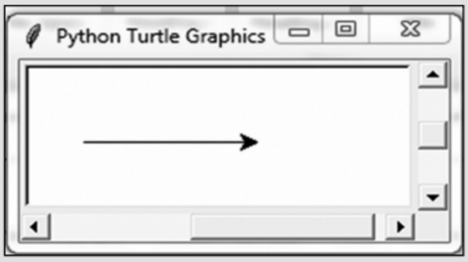
Ans.

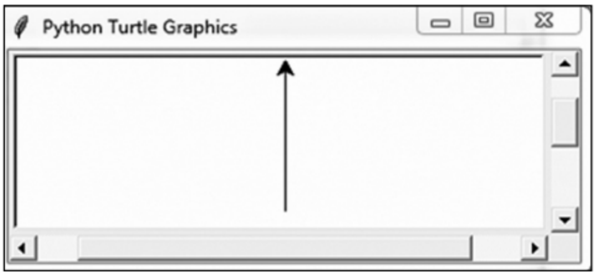
The Turtle is an object which is created when we import the Turtle module. As soon as the object is created its position is set at (0, 0), i.e. at the center of the Turtle graphics window. Also by default its direction is set to go straight to the right.

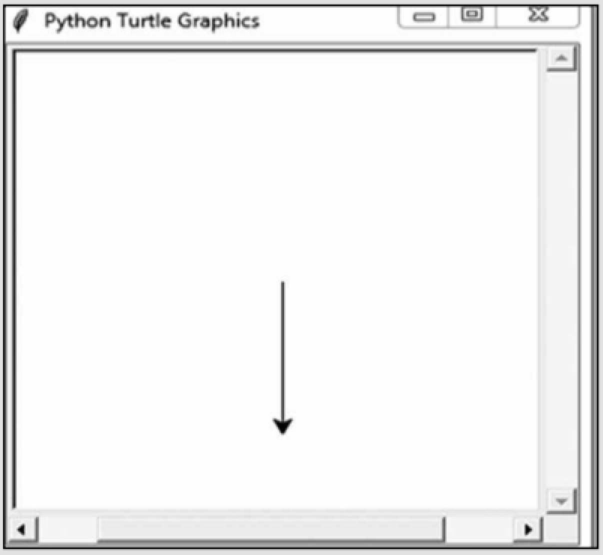
The imported Turtle module uses a pen to draw shapes. It can be used to move and draw lines in any direction. Python contains methods for moving the pen, setting the pen's size, lifting and putting the pen down. By default, the pen is down, i.e. it draws a line from the current position to the new position. Table 12.1 shows a list of methods to move the Turtle in specified directions.

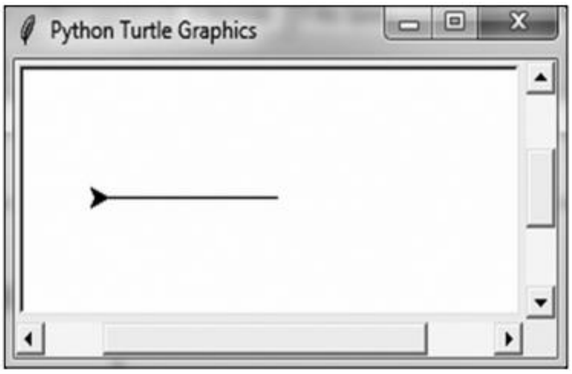
Table 12.1 Turtle methods related to directions

Method	*Meaning*

`Turtle.forward(P)`
Example:
`>>> import Turtle`
`>>> Turtle.forward(100)`
	Moves the Turtle P pixels in the direction of its current heading.
	Output:
	

`Turtle.left(angle)`
Example:
`>>> import Turtle`
`>>> Turtle.left(90)`
`>>> Turtle.forward(100)`
	Rotates the Turtle left by the specified angle.
	Output:
	
	Explanation: Initially the Turtle is placed at the centre by default. The command **Turtle.left(90)** changes the direction of the Turtle to the left by 90 degrees. Finally, the arrowhead moves 100 pixels forward.

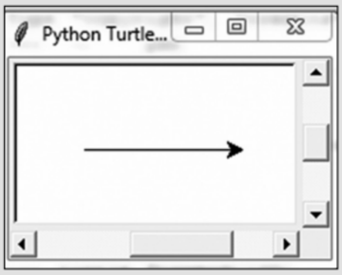
`right(P)`
Example:
`>>> import Turtle`
`>>> Turtle.right(90)`
`>>> Turtle.forward(100)`
	Rotates the Turtle in place a degree clockwise.
	Output:
	
	
	Explanation: Initially the Turtle is placed at the centre by default. The command **Turtle.right(90)** changes the direction of the Turtle to the right by 90 degrees. Finally, the arrowhead moves 100 pixels forward.

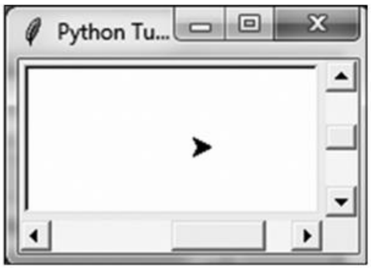
`backward(P)`
Example:
`>>> import Turtle`
`>>> Turtle.backward(100)`
	Moves the Turtle P pixels in a direction opposite to its current heading.
	Output:
	

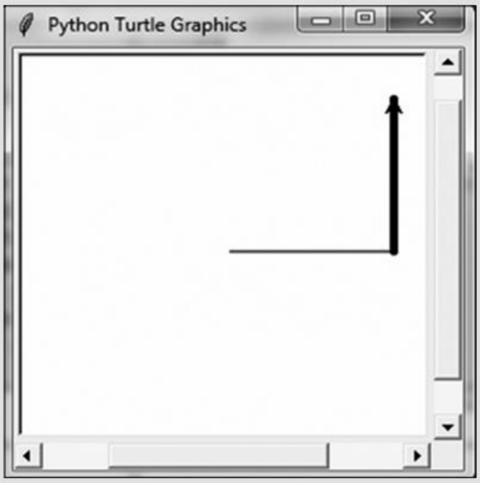
In Table 12.1, we used various methods to move the Turtle from one position to the other. As discussed above, the Turtle draws a line from one position to the other with the help of the pen. Table 12.2 illustrates various methods related to the state of a pen.

Table 12.2 Methods related to the state of a pen

Method	*Meaning*

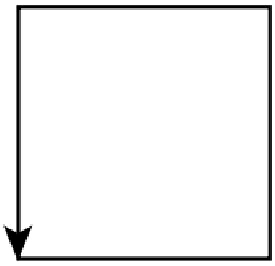
`Turtle.pendown()`
Example:
`>>> import Turtle`
`>>> Turtle.pendown()`
`>>> Turtle.forward(100)`
	Pulls the pen down. Draws when it moves from one place to the other.
	Output:
	
	Explanation: In the above example, the method **Turtle.pendown() draws different** shapes when it moves from one place to the other.

`Turtle.penup()`
Example:
`>>> import Turtle`
`>>> Turtle.penup()`
`>>> Turtle.forward(100)`
	Pulls the pen up. In this state, it just moves from one place to the other without drawing anything.
	Output:
	
	Explanation: The import Turtle method places the pen at the center of the circle. The **Turtle.penup()** doesn't allow a programmer to draw things, it just moves from one place to the other. When the statement Turtle.forward(100) is executed immediately after the `penup()` statement, it moves 100 pixels forward without drawing any line or shape.

`Turtle.pensize(width)`
Example:
`>>> import Turtle`
`>>> Turtle.forward(100)`
`>>> Turtle.pensize(5)`
`>>> Turtle.pensize(5)`
`>>> Turtle.left(90)`
`>>> Turtle.forward(100)`
	Sets the line thickness to the specified width.
	Output:
	
	Explanation: In the above code, initially the line is drawn 100 pixels ahead in the forward direction. The statement **Turtle.pensize(5)** increases the thickness to draw the figure from here onwards.

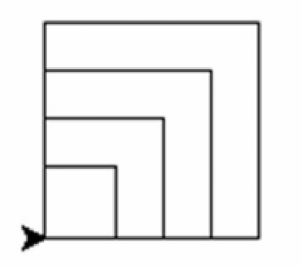
Q3. Write a program to draw square as shown below using Python turtle module.

a.



```
import Turtle             #import Turtle module
Turtle.forward(100)       #Move Turtle in forward direction
Turtle.left(90)  #Change the direction of Turtle to left by 90 degree
Turtle.forward(100)
Turtle.left(90)
Turtle.forward(100)
Turtle.left(90)
Turtle.forward(100)
```

b.



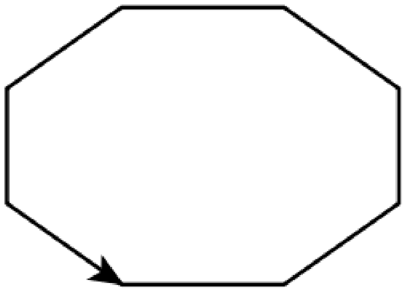
```
import Turtle
def  square(side):
    for i in range(4):
        Turtle.forward(side)
        Turtle.left(90)

square(20)
square(30)
square(40)
square(50)
```

Q4. Write a program to display polygon.

Ans.

```
import Turtle          #import Turtle module
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
Turtle.left(45)
Turtle.forward(50)
```



Q5. Explain moving turtle to any location.

Ans.

When a programmer tries to run Python's Turtle graphics program by default, the Turtle's arrowhead (Cursor or Pen) is at the center of the graphics window at coordinate(0, 0) as shown inFigure 12.3.

>>> import Turtle   #import Turtle module

>>> Turtle.showTurtle ()

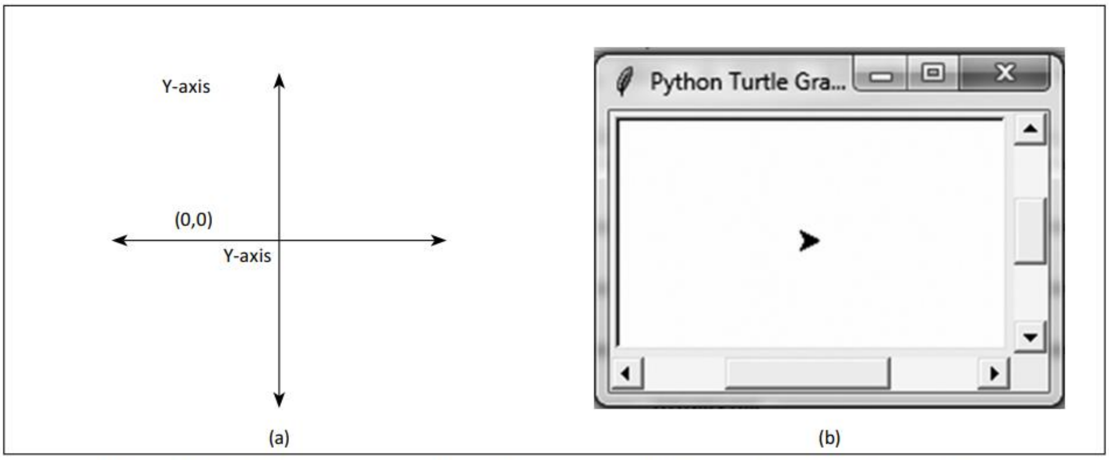


Figure 12.3 (a) Representation of coordinate system (b) Centre of Turtle graphics at (0, 0)

The method goto(x, y) is used to move the Turtle at specified points (x, y). The following exampleillustrates the use of goto(X, Y) method.

Example

>>> import Turtle

>>> Turtle.showTurtle ()

>>> Turtle.goto(0,-50)

Output

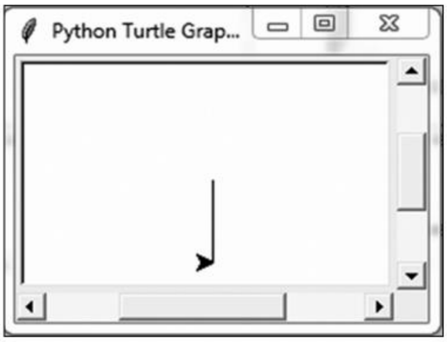


Figure 12.4

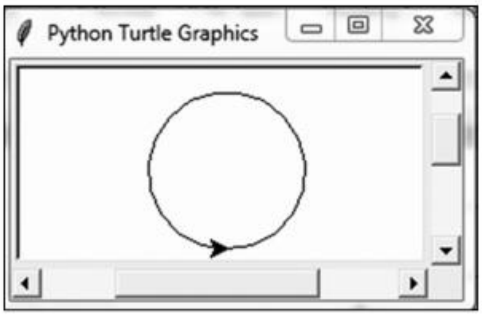
Explanation

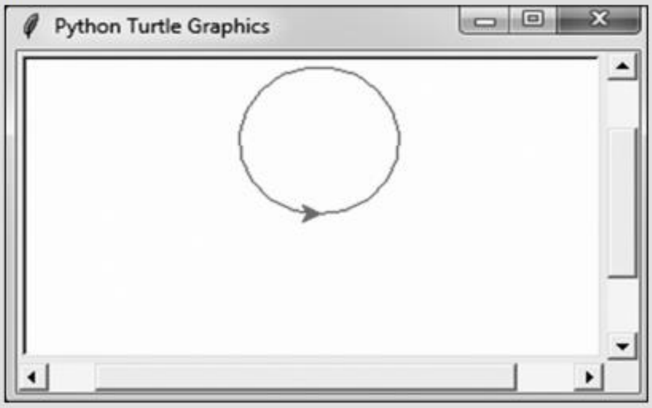
In the above example, the statement goto(0,-50) will move towards coordinate (0, -50).

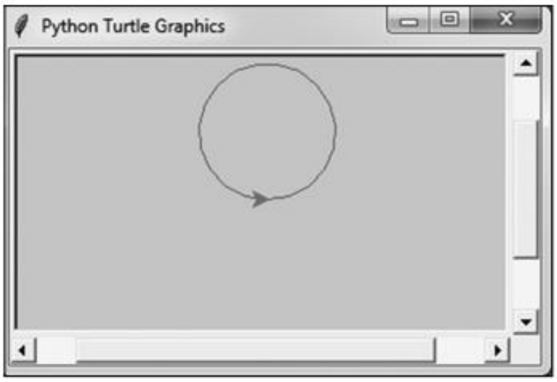
Q6. Explain the color, bgcolor, circle and speed method of turtle.

Ans.

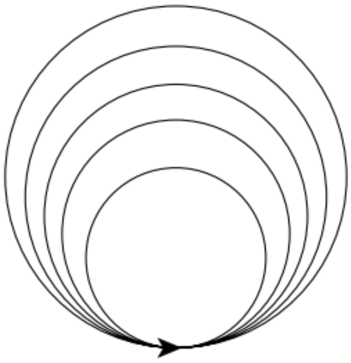
Method	*Meaning*
`Turtle.speed(integer _ paramter)`	The drawing speed of the Turtle must be in the range int 1 (slowest) to 10 (fastest) or 0 (instantaneous).

`Turtle.circle(radius, extent=None)`
Example:
`>>> import Turtle`
`>>> Turtle.circle (45)`
	Draws a circle with the given radius. The center is radius units left of the Turtle. The extent determines which part of the circle is drawn. If it is not given, the entire circle is drawn.
	Output:
	
	Explanation: The statement Turtle.circle (45) is used to draw a circle of radius 45 in an anti-clockwise direction.

`Turtle.color(*args)`
Example:
`>>> import Turtle`
`>>> Turtle.color("red")`
`>>> Turtle.circle (45)`
	The color method is used to draw colorful animations.
	Output:
	
	Explanation: The above statement draws a circle in red color.

`Turtle.bgcolor(*arg)`
Example:
`>>> import Turtle`
`>>> Turtle.color("red")`
`>>> Turtle.`
`bgcolor("pink")`
	Returns the background color of the Turtle screen.
	Output:
	
	
	Explanation: Changes the background color of the Turtle graphics window to pink.

Q7. Explain a program to display circles shown.



Ans.

```
import Turtle
Turtle.circle (45)
Turtle.circle (55)
Turtle.circle (65)
Turtle.circle (75)
Turtle.circle (85)
```

Explanation In the above program, the 5 circles are drawn with different radius, viz. 45, 55, 65, 75 and 85, respectively.

Q8. Explain methods for drawing turtles with colors.

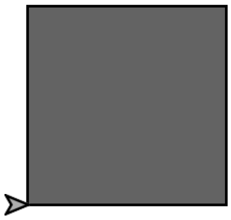
Ans.

A Turtle object contains methods for setting a color. In the above section, we have learnt how to draw different shapes. Table 12.4 lists methods to draw different shapes with different colors.

Table 12.4 More methods of Turtle related to color

Method	*Meaning*
`Turtle.color(c)`	Sets the pen's color
`Turtle.fillcolor(C)`	Sets the pen's fill color to **'C'**
`Turtle.begin _ fill()`	Calls this method before filling a shape
`Turtle.end _ fill()`	Fills the shape drawn before the last call to **begin_fill**
`Turtle.filling()`	Returns the fill state. True is filling, False if not filling.
`Turtle.clear()`	Clears the window. The state and position of window is not affected.
`Turtle.reset()`	Clears the window and resets the state and position to its original default value
`Turtle.screensize()`	Sets the width and height of the canvas
`Turtle.showTurtle()`	Makes the Turtle visible
`Turtle.hideTurtle()`	Makes the Turtle invisible
`Turtle.write(msg, move,align,font=fontname, fontsize, fonttype)`	Writes a message on the Turtle graphics window

- **Write a program to draw a color filled square box as shown.**

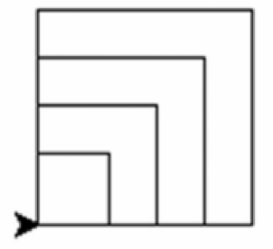


```
import Turtle
Turtle.fillcolor ("gray") #Fill gray color inside the square
Turtle.begin_fill ()
Turtle.forward(100)
Turtle.left(90)
```

```
Turtle.forward(100)
Turtle.left(90)
Turtle.forward(100)
Turtle.left(90)
Turtle.forward(100)
Turtle.left(90)
Turtle.end_fill()
```

Q9. Explain drawing basic shapes using iteration.

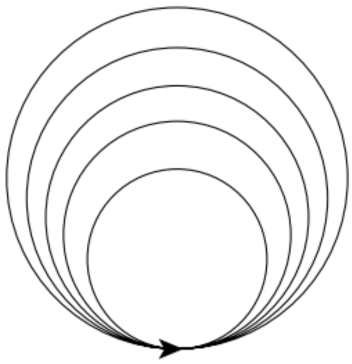
Ans.



```
import Turtle
def  square(side):
    for i in range(4):
        Turtle.forward(side)
        Turtle.left(90)

square(20)
square(30)
square(40)
square(50)
```

b.



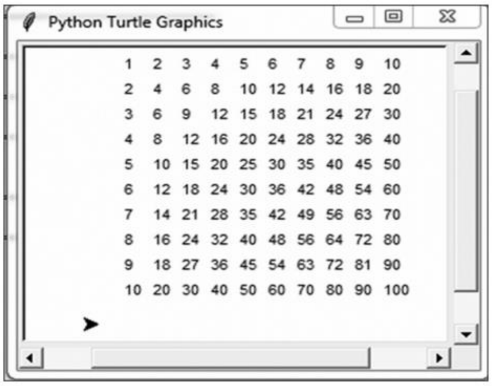
```
import Turtle
Turtle.circle (45)
Turtle.circle (55)
Turtle.circle (65)
Turtle.circle (75)
Turtle.circle (85)
```

Explanation In the above program, the 5 circles are drawn with different radius, viz. 45, 55, 65, 75 and 85, respectively.

Q9. Write a program to display the multiplication table from 1 to 10 in turtle graphics window.

```
import Turtle as t
t.penup()
```

```
x =  -100
y =  100
t.goto(x,y)  #Move pen at location x and y
t.penup()
for i in range(1,11,1):   # value of i varies from 1 to 10
    y = y - 20
    for j  in range(1,11,1):  # Value of j varies from 1 to 10
        t.penup()
        t.speed(1)
        t.forward(20)
        t.write(i*j)
   t.goto(x, y)
```



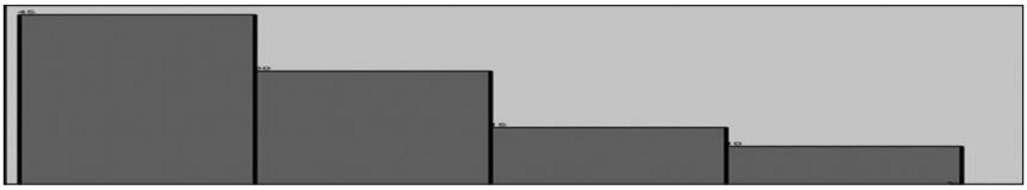
Q10. Write a program to draw a barchart using turtle for sample data given below.

Table 12.5 Sample data to draw a chart

Web Browser	*Percentage*
Mozilla Firefox	45%
Google Chrome	30%
Internet Explorer	15%
Others	10%

Ans.

```
import Turtle
def Draw_Bar_Chart(t, height):
    t.begin_fill()                # start filling this shape
    t.left(90)
    t.forward(height)
    t.write(str(height))
    t.right(90)
    t.forward(40)
    t.right(90)
    t.forward(height)
    t.left(90)
    t.end_fill()                  # stop filling this shape
Mozilla_Firefox = 45
Chrome =  30
IE = 15
Others = 10
S = [Mozilla_Firefox, Chrome, IE,Others]  # Sample Data
maxheight = max(S)
num_of_bars = len(S)
border = 10
w = Turtle.Screen()              # Setting up attributes of Window
w.setworldcoordinates(0,0,40*num_of_bars + border, maxheight + border)
w.bgcolor("pink")
T1 = Turtle.Turtle()
T1.color("#000000")
T1.fillcolor("#DB148E")
T1.pensize(3)
for a in S:
    Draw_Bar_Chart(T1,a)
```



Q11. Describe the need of file handling.

Ans.

Often the output screen of a laptop or monitor is not enough to display all the data. This usually happens when the data is large and only a limited amount can be displayed on the screen and stored in the memory. Computer memory is volatile, so even if a user tries to store the data in the memory, its contents would be lost once a program is terminated. If the user needs the same data again, either it has to be entered through a keyboard or regenerated programmatically. Obviously, both these operations are tedious. Therefore, to permanently store the data created in a program, a user needs to save it in a File on a disk or some other device. The data stored in a file is used to retrieve the user's information either in part or whole.

Various operations carried out on a file are

(a) Creating a file

(b) Opening a file
(c) Reading from a file
(d) Writing to a file
(e) Closing a file

Q12. Explain opening file in Python in details.

Ans.

A file needs to open before we can perform read and write operations on it. To open a file, a user needs to first create a file object which is associated with a physical file. While opening a file, a user has to specify the name of the file and its mode of operation. The syntax to open a file is:

file object = open(File_Name, [Access_Mode],[Buffering])

The above syntax to open a file returns the object for file name. The mode operation used in the syntax above is a string value which indicates how a file is going to be opened. Table 13.1 describes the various modes used to open a file. The third parameter within the open function is an optional parameter, which controls the buffering of a file. If this parameter is set to 1, line buffering is performed while accessing the file. If the buffering value is set to 0 then no buffering takes place. If we specify the buffering value as an integer greater than 1 then the buffering action is performed with the indicated buffer size.

Mode	*Description*
R	Opens a file for reading
W	Opens a new file for writing. If a file already exists, its contents are destroyed.
A	Opens a file for appending data from the end of the file
Wb	Opens a file for writing binary data
Rb	Opens a file for reading binary data

```
F1 = open ("Demo.txt","r") #Open File from Current Directory
F2 = open("c:\Hello.txt","r")
```

The above example opens a file named Hello.txt located at C: in read mode.

Q13. Explain Writing text to a file along with methods and program to write the sentences.

Ans. The open function creates a file object. It is an instance of _io.TextIOWrapper class. This class contains the methods for reading and writing data. Table 13.2 lists the methods defined in the _io.TextIOWrapper class.

Table 13.2 Methods for reading and writing data

_io.TextIOWrapper	*Meaning*
str readline()	Returns the next line of a file as a string
list readlines()	Returns a list containing all the lines in a file

str read([int number])	Returns a specified number of characters from a file. If the argument is omitted then the entire content of the file is read.
Write (str s)	Writes strings to a file
close()	Closes a file

Once a file is opened, the write method is used to write a string to a file.

PROGRAM 13.1 | Write a program to write the sentences given below the file **Demo1.txt**.

Hello, How are You?
Welcome to The chapter File Handling.
Enjoy the session.

```
def main():
    obj1 = open("Demo1.txt","w") #Opens file in Write mode
    obj1.write(" Hello, How are You ? \n")
    obj1.write(" Welcome to The chapter File Handling. \n ")
    obj1.write(" Enjoy the session. \n ")
main()   # Call to main function
```

Explanation In the above program, initially the file **Demo1.txt** is opened in **'w'** mode, i.e. **write mode**. If the file **Demo1.txt** does not exist, the **open** function creates a new file. If the file already exists, the contents of the file will be over written with new data.

When a file is opened for reading or writing, a special pointer called file pointer is positioned internally in the file. Reading and writing operation within the file starts from the pointer's location. When a file is opened, the file pointer is set at the beginning of the file. The file pointer moves forward as soon as we start reading from the file or write the data to the file.

The step-wise execution and position of the file pointer is updated in the following manner by the Python interpreter.

Initially, a call is made to the main() function. The statement obj1 = open("Demo1.txt","w") opens Demo1.txt in write mode. The file is created and initially the file pointer is at the starting ofthe file as shown in Figure 13.1.

Figure 13.1 Initial position of the file pointer

The following statement within the program invokes the write method on the file object to writestrings into the file.

obj1.write(" Hello, How are You ? \n")

After successful execution of the above statement, the file pointer is located as shown inFigure 13.2.

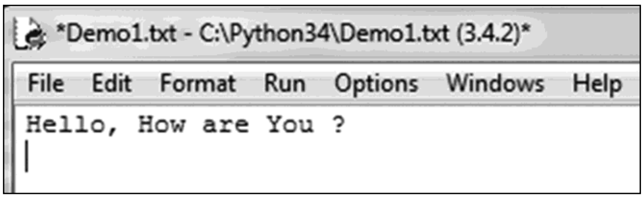


Figure 13.2

After successful execution of a second statement, i.e. obj1.write(“Welcome to The chapter FileHandling. \n “), the file pointer is located as shown in Figure 13.3.

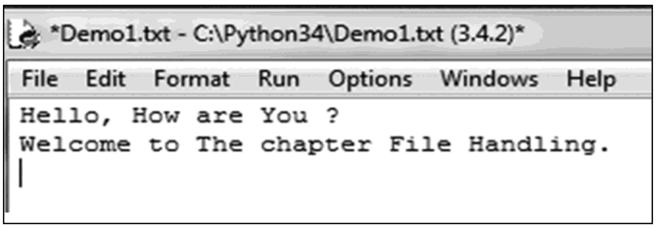


Figure 13.3

Finally, after the execution of the third statement, i.e. obj1.write(“ Enjoy the session.\n”), thecontents of the file are updated as shown in Figure 13.4.

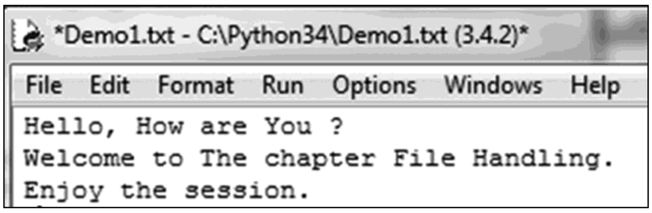
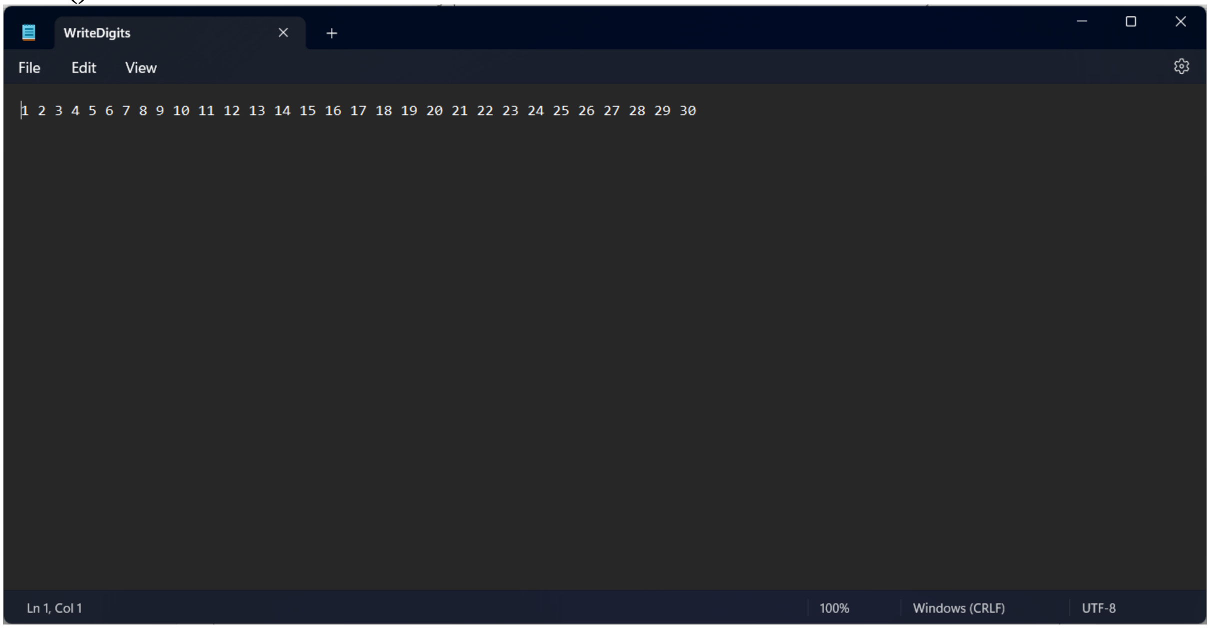


Figure 13.4

Q14. Write a program to print numbers from 1 to 30 to the output file writedigits.txt.

Ans.

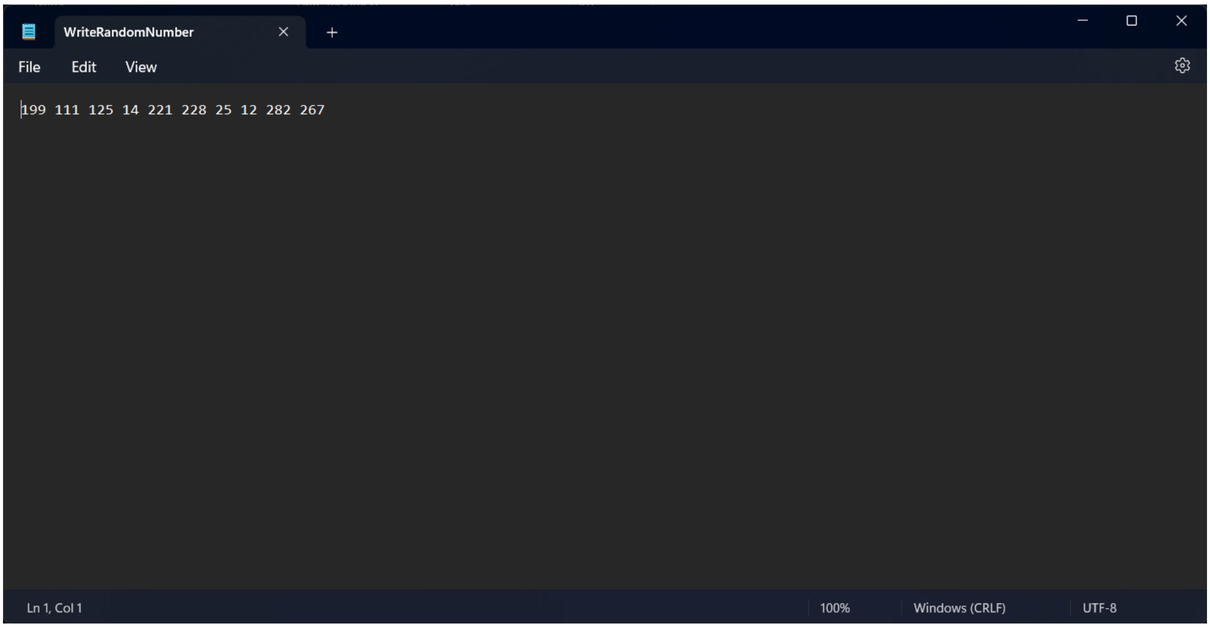
```
def main():
    obj1 = open("WriteDigits.txt", "w")
    for x in range (1,31):
        x = str(x)
        obj1.write(x)
        obj1.write(" ")
    obj1.close()
main()
```



Q15. Generate 10 random numbers within a range 1 to 30 and write them to a file WriteRandomNumbers.txt.

Ans.

```
from random import randint
fp1 = open("WriteRandomNumber.txt", "w")
for x in range(10):
    x = randint(1, 300)
    x = str(x)
    fp1.write(x + " ")
fp1.close()
```



Q16. Explain reading text from a file and write down a program using read method.
Ans.

Once a file is opened using the open () function, its content is loaded into the memory. The pointer points to the very first character of the file. To read the content of the file, we open the file in 'r' (read) mode. The following code is used to open the file ReadDemo1.txt.

>>> fp1 = open("ReadDemo1.txt","r")

There are several ways to read the content of a file. The two common approaches are:

a. Use read()method to read all the data from a file and return as one complete string.
b. Use readlines()method to read all data and return as a list of strings.

The following program demonstrates the use of the read()method to read the content of the file ReadDemo1.txt. The content of the file is as shown in Figure.

Program 1:

```
fp = open("ReadDemo1.txt","r") #Open file in read mode
text = fp.read()          # Read Whole File exactly  once
print(text)           #Print the contents of file
```

Output

```
I
Love
Python
Programming
Language
```

Explanation Initially the file ReadDemo1.txt is opened in read mode. The content of the file is read using the `read()` method. It reads all the content of the file exactly once and returns all the data as a single string.

Program 2:

```
fp = open("ReadDemo1.txt","r")
for line in fp:
    print(line)
```

Output

```
I

Love

Python

programming

Language
```

Explanation In the above program, the for loop views the file object as a sequence of lines of text. In each iteration of the for loop, the loop variable **line** is bound to the next line from the sequences of lines present in the text file. Note the output of above program. The **print**() statement prints one extra new line. This is because each line of the input file retains its new line character.

Q17. Explain reading multiple items on one line.
Ans.

Many text files contain multiple items in a single line. The method split () for strings allows us to read more than one piece of information in a line. The split () returns all the items in a list. In short, it splits a string into separate items and all the items are separated by spaces or tabs.

The following example written in Python IDLE interpreter gives more details about the split() method.

```
>>> str = 'I am Loving The Concepts of File Handling'
>>> str.split()
['I', 'am', 'Loving', 'The', 'Concepts', 'of', 'File', 'Handling']
#
>>> for i in range(len(str)):
         print(str[i])

I
am
Loving
The
Concepts
of
File
Handling
```

Explanation The above example simply splits the string and stores the content to a list. Finally, the for loop is used to access and display each item of the list.

Q18. Describe the following in short.

a. Appending Data.

Ans.

The append 'a' mode of a file is used to append data to the end of an existing file. The following program demonstrates the use of append mode.

PROGRAM 13.10 | Write a program to append extra lines to a file name **appendDemo.txt**.

The content of **appendDemo.txt** file is as shown in Figure 13.10.

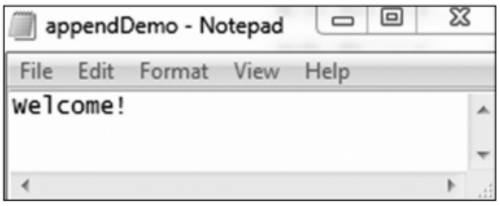
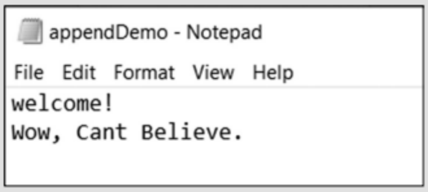


Figure 13.10

```
fp1=open('appendDemo.txt','a') # Open file in append file
fp1.write('\nWow, Cant Believe.')# Append contents to a file
fp1.close() #Close file
```

Output



b. The seek() function.

Ans.

So far, we have learnt that data is stored and subsequently read from a file in which it is stored. When a file is opened, we can imagine an imaginary pointer positioned at the beginning of the file. What about reading the content of files from random positions? Python provides an inbuilt function called seek()for moving the pointer explicitly to any position in a file.

Thus, the seek()method is used to set the file pointer to a specific position in a file. The syntaxfor seek()function is:

File_object.seek(offset, whence)

where offset indicates the number of bytes to be moved from the current position of the pointer and whence indicates the point of reference from where the bytes are to be moved from. The valueof whence can be determined from Table 13.3.

Value	*Meaning*
0	The position is relative to the start of the file, i.e. it sets the pointer at the beginning of the file. This is a default setting if we don't supply '0' as the second argument to the seek() function.
1	The position is relative to the current position.
2	The position is relative to the end of the file.

Examples

```
#Create Seek_Demo1.txt file in write mode
>>> fp1= open('Seek_Demo1.txt','w+')
#Write some data to the file
>>> fp1.write('Oh!God!SaveEarth!')
17  #returns number of characters written in a file
#By default second argument of seek function is zero
>>> fp1.seek(3)
2
>>> fp1.readline()
'God!SaveEarth!'
```

Explanation

In the above example the file Seek_Demo.txt contains 17 characters. The statement **fp1.seek(3)** tells Python to read the content of the file from the third position.