1.

a. List and Explain use of comparison operators in Python. Write the step by step execution of the following expression in Python. 6M
3/2*4 + 3 + (10/4)3 – 2**

Ans:
The comparison operators are relation operators those return a Boolean True or False.
The six comparison operators are
i) == equal to,
ii) != not equal to,
iii) > greater than,
iv) >= greater than or equal to
v) < less than, and
vi) <= or less than or equal to.
They can be used to compare different values in Python, such as integers or strings.
Examples of relational/comparison operators with output:

```
print(n>20)       #False
print(n<20)       #False
print(n!=20)      #False
print(n==20)      #True
print(n>=20)      #True
print(n<=20)      #True
print(n is 20)    #True
print(n is not 20)#False
```

Answer of the expression of 3/2*4 + 3 + (10/4)**3 – 2 is **22.625**
Step-by-Step as per precedence of operation (PEDMAS)
Within Parenthesis (10/4) will be computed first with output 2.50
2.5 is raised to power 3 (2.5)**3 will result 15.625
So, 3/2*4 + 3 + 15.625 – 2
Now Division and multiplication of 3/2*4 will result 6
So, 6+3+15.625-2 will do addition and subtraction and will result 22.625

b. Explain control statements if, else, elif with proper syntax and examples. 6M

Ans:

c. Develop a Python Program to calculate area and circumference of a circle. Input the value of radius and print the result. 8M
Ans:

```
# Area and circumference of a circle
import math
radius=float(input("Enter radius : "))
```

```
area=math.pi*radius*radius
circum=2*math.pi*radius
print("Area = %.3f Square unit" % area)
print("Circumference = %.3f unit" % circum)
```

Expected output:
Enter radius : 3.0
Area = 28.274 Square unit
Circumference = 18.850 unit

2.

a. Explain the string concatenation and string replication operator with an example. 6M

Ans:

Strings are immutable in Python. However string concatenation and string replication are allowed to display or to store in another string variable.
For concatenation + operator is allowed
For replication * operator is allowed.
Examples:

```
>>> species1='Python'
>>> species2='Anaconda'
>>> bigsnakes=species1+species2
>>> print(bigsnakes)
PythonAnaconda
>>> print(species1*3)
PythonPythonPython
>>> print(species1*3+species2*3)
PythonPythonPythonAnacondaAnacondaAnaconda
```

b. Explain local and global scope of variable with example 6M

Ans:

Local or function scope: It is the code block or body of any Python function or lambda expression. This Python scope contains the names that you define inside the function. These names will only be visible from the code of the function
Global or Module scope: It is the top-most scope in a Python program, script, or module. This scope contains all of the names that are defined at the top level of a program or a module. **To access global variable within function we will have to use keyword global.**

Examples of global, local, enclosing and enclosing:

```
var1 = 50 #var1 is in the global namespace
def ABC( ):
  var2 = 60 # var2 is in the local namespace
  def XYZ( ):
    var3 = 70 # var3 is in the nested local namespace
```

c. Develop a program to read students details like Name,USN and Marks in three subjects. Display the student's detail, total marks and percentage with suitable messages. 8M

Ans:

```
#Student's detail and Marks percentage
```

```
name=input("Enter student's name : ")
USN=input("Enter USN : ")
sub1=int(input("Enter marks in subject 1 in 100 : "))
sub2=int(input("Enter marks in subject 2 in 100 : "))
sub3=int(input("Enter marks in subject 3 in 100 : "))
total=sub1+sub2+sub3
percent=total/3
print("Name: ",name,"USN: ",USN)
print("Subject1: ",sub1,"Subject2: ",sub2,"Subject3: ",sub3)
print("Percentage of marks = ",percent)

'''Expected output:
Enter student's name : Sultan
Enter USN : 1CR24CS001
Enter marks in subject 1 in 100 : 56
Enter marks in subject 2 in 100 : 76
Enter marks in subject 3 in 100 : 79
Name:  Sultan USN:  1CR24CS001
Subject1:  56 Subject2:  76 Subject3:  79
Percentage of marks =  70.33333333333333
'''
```

3.

a. What is list? Explain the concept of list indexing and slicing with examples. 6M

Ans:

A list in python is collection of different data types. A list is mutable(changeable) and ordered sequence of elements or items. A list can be created by [ ] or list() built-in method

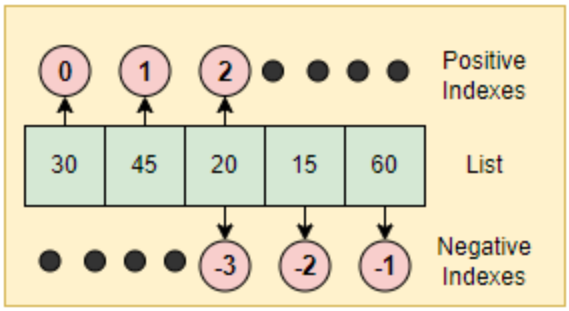
Examples:

```
Lst1 = [101,"Raman",76.5,'A',False]
numlist=[30,45,20,15,60]
lst2=[] or lst2=list() #to create empty list
```

Indexing

List elements are accessed by indexes from 0 to total elements – 1 in positive indexing.

In negative indexing elements are from the last element. we use negative indexes starting from -1 and adding -1 each time to get the next number from the end.




```
numlist=[30,45,20,15,60]
print(numlist[1]) # will display 45
print(numlist[-1]) # will display 60
```

Slicing: In order to access a range of elements we need to slice a list. The general format of slicing is:
Lst[start: beforeend: indexJump]
lst1 = [30,45,20,15,60]
lst1[1:3] # [45,20] 1 to 2 (just before index 3)
lst1[-2::1] #[15, 60]

b. With suitable examples explain the list methods append(), extend(), sort(), count() and pop() 8M

Ans:

append(): append() method call adds the argument to the end of the list.

```
>>> marks=[65,67,87,76]
>>> marks.append(78)
>>> print(marks)
[65, 67, 87, 76, 78]
```

extend():extend function adds multiple elements (from an iterable) to the list taking iterable as argument

```
>>> marks1=[45,43,49,47,38,50]
>>> marks2=[40,42,46]
>>> marks1.extend(marks2)
>>> print(marks1)
[45, 43, 49, 47, 38, 50, 40, 42, 46]
```

sort(): sorts the list item either in ascending or descending order

```
>>> marks1.sort()
>>> marks1
[38, 40, 42, 43, 45, 46, 47, 49, 50]
>>> marks1.sort(reverse = True)
>>> marks1
[50, 49, 47, 46, 45, 43, 42, 40, 38]
```

count(): to count number of occurances of an element in a list

```
>>> num=[3,4,5,6,2,3,6,7,6,5,4]
>>> num.count(5)
2
```

pop() – deleting elements, if element number is not given then it pops & deletes last element. Popped value can be stored to other variable.

```
>>> namelist=['Akrur', 'Vinay', 'Vinayak', 'ajay', 'kunti', 'paras', 'samartha']
>>> namelist.pop(5)
'paras'
>>> namelist
['Akrur', 'Vinay', 'Vinayak', 'ajay', 'kunti', 'samartha']
>>> namelist.pop() #by default last element
'samartha'
```

c. Read N numbers from the console and create a list. Develop a program to print mean, variance and standard deviation with a suitable message 6M

Ans:

```
def varstdev(x):
   mean = sum(x) / N
   var = sum((item - mean)**2 for item in x) / (N - 1)
   stdev=var **.5
   print("Mean = %.2f variance = %.3f Standard Deviation = %.3f"\
       %(mean,var,stdev))

N=int(input("How many numbers : "))
lst=[]
for i in range(N):
   num=int(input("Enter number %d:" % (i+1)))
   lst.append(num)
varstdev(lst)
```

Expected output:

```
How many numbers : 5
Enter number 1:2
Enter number 2:3
Enter number 3:4
Enter number 4:5
Enter number 5:6
Mean = 4.00 variance = 2.500 Standard Deviation = 1.581
```

4.

a. Define tuple data type. List out difference between tuple and list. 6M

Ans:

Tuples although list-like in some respects, are immutable and cannot be changed.
Syntactically, a tuple is a comma-separated list of values:
Examples:

```
t1=(20,34,'Naresh',4.5,True)
>>> print(t1[2])
Naresh
```

Lists are useful collections of different types of data since they allow us to write code that works on a modifiable number of values in a single variable.
Lists are mutable, meaning that their contents can change. But tuples are immuable
tuples are typed with parentheses ( ), lists are within [ ].
tuple() is used create a tuple. list() is used to create a list
list( ) and tuple( ) functions are used to convert to list and tuple respectively. Examples:

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list(('cat', 'dog', 5))
['cat', 'dog', 5]
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

b. Identify and explain dictionary methods like get(), item(), keys() and values() in Python with examples.

Ans:

get( ) method:

get() method of dictionary takes two arguments: the key of the value to retrieve and a fallback value to return if that key does not exist.

Example:

```
>>> examitems = {'pen':2, 'pencil':2, 'scale':1}
>>> print('I am bringing ' + str(examitems.get('pencil', 0)) + ' pencils.')
I am bringing 2 pencils.
```

The keys(), values(), and items() Methods

There are three dictionary methods that will return list-like values of the dictionary's keys, values, or both keys and values: keys(), values(), and items(). The values returned by these methods are not true lists: They cannot be modified and do not have an append() method. But these data types (dict_keys, dict_values, and dict_items, respectively) can be used in for loops.

Examples:

items() method : Keys and values both

```
>>> marks={'DSA':76, 'SE': 78, 'CO': 80, 'Python':91}
>>> for a in marks.items():
        print(a)

('DSA', 76)
('SE', 78)
('CO', 80)
('Python', 91)
```

keys() method : Only keys

```
>>> for b in marks.keys():
        print(b)

DSA
SE
CO
Python
```

values() method : Only values

```
>>> for c in marks.values():
        print(c)

76
78
80
91
```

c. Develop a Python program to swap 2 numbers without using intermediate variable. Prompt the user for input. **6M**

Ans:

```
#Swapping 2 numbers without using intermediate variable
a=int(input("Enter value of a : "))
b=int(input("Enter value of b : "))
print("a = ",a," b= ",b)
#now swapping without any 3rd intermediate variable
a=a+b
b=a-b
a=a-b
print("Swapped value a = ",a," b= ",b)

'''
Output:
Enter value of a : 20
Enter value of b : 30
a =  20  b=  30
Swapped value a =  30  b=  20
'''
```

5.

a. Write the output for the following:

i) 'HeLLo'.upper().isupper()

ii) 'HeLLo'.upper().lower()

iii) ' '.join('There can be only one'.split()) **6M**

Ans:

i) True

ii) 'hello'

iii) 'There can be only one'

b. With examples explain any five string methods. **6M**

Ans:

i) join() : converts the element of an iterable into a string

```
>>> namesurname=["Dr. P. N.", "Singh"]
>>> name=" ".join(namesurname)
>>> print(name)
Dr. P. N. Singh
```

ii)islower(): returns True if all the characters of a string are in Upper case else False

```
>>> name.islower()
False
```

iii) strip() : Trims leading and trailing blanks

```
>>> name=' Dr. P. N. Singh '
>>> print("Your name is ", name.strip())
Your name is Dr. P. N. Singh
```

iv) **center() :** Centre alignment of the string in number of characters

```
>>> name.center(20)
' Dr. P. N. Singh
```

v) upper(): Displays all characters of the string in upper case. Original string does not change.

```
>>> inst='CMR Institute of Technology'
>>> print(inst.upper())
CMR INSTITUTE OF TECHNOLOGY
>>> print(inst)
CMR Institute of Technology
```

c. Develop a program to count total number of vowels, consonants in a string. 8M

Ans:

```
#total vowels and consonants in a string
vowels='aeiouAEIOU'
sent="A quick brown fox jumps right over a lazy dog"
totvowels=sum(sent.count(v) for v in vowels)  #totvowels = 12
spaces=sent.count(' ') #counting blank spaces #spaces = 9
consonants=len(sent)-totvowels-spaces #45-12-9 = 24
print("Total vowels = ",totvowels," Total consonants = ",consonants)

# output : Total vowels =  12  Total consonants =  24
```

6.

a. Make use of the concept of file handling and explain Reading and writing process with suitable Python programs. 7M

Ans:

Python provides built-in functions for creating, writing, and reading files. Each line of text is terminated with a special character called EOL (End of Line), which is the new line character ('\n') in Python by default.

Opening the file: file_object = open("filename", "access mode")

File access modes: Access modes govern the type of operations possible in the opened file:

"r" - read mode: Opens the file in read mode, It is also default mode

"w" - write mode: Opens the file for writing/creating. If the file exists, it will overwrite the contents. If the file does not exist, it will create a new file.

"a" - append mode: Opens the file to add contents at the end of file.

Other modes: "r+" – read and write, "w+" – write and read, "a+" – append and read mode

Example1:

```
#to display the contents of file: dikhao.py
fname=input("Enter filename : ")
file=open(fname,"r")
#if no such file then FileNotFoundError
for line in file:
   print(line,end=" ")
file.close()
```

Example2:

```
fname=input("Enter filename : ")
file=open(fname,"w")
```

```
print("Write contents done to end:")
while True:
   line=input()
   if line.lower()=="done":
      break
   file.write(line+"\n")
file.close()
```

Sample output:
Enter filename : gana.txt
Write contents done to end:
Kudi Punjaban
Dil chura ke lai gayee
Sona Sona
Dil mera sona
done

We can see the file contents by example1 program

b. Explain the concept of file path, also discuss absolute and relative path. 7M

Ans:

File path: file path is a way to locate our file under which directory it exists. The path starts in our file system from drive and root directory with **backward slash \ or forward slash /** like **C:\Windows\Python39\myprog.py**. The back slash \ can be made literal by double backslash: C:\\Windows\\System32

The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules.
OS.path module contains some useful functions on pathnames.

Absolute Path: Absolute file paths are full path from drive label from the root of the file system.
For example: C:\\Windows\\System32

Relative path: Relative file paths are interpreted from the perspective of current directory. If you use a relative file path from the wrong directory, it will refer to no file at all.
For example to open a file myprog.py from singh folder in current directory:
file1=open("singh myprog.py",”r”)

c. Briefly explain saving variables with shelve module. 6M

Ans:

We can save variables in our Python programs to binary shelf files using the shelve module. This way, our program can restore data to variables from the hard drive. The shelve module will let us **add, Save, and Open** features to our program. For example, if we ran a program and entered some configuration settings, we could save those settings to a shelf file and then have the program load them the next time it is run.

```
>>> import shelve
>>> shelfFile = shelve.open('mydata')
```

```
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()
```

Saving Variables with the pprint.pformat() Function:

The pprint.pprint() function will "pretty print" the contents of a list or dictionary to the screen, while the pprint.pformat() function will return this same text as a string instead of printing it.

```
>>> import pprint
>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]
>>> pprint.pformat(cats)
"[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]"
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
83
>>> fileObj.close()
```

7.

a. Explain the following file operations in Python 6M

i) Copying files and folders

ii) Moving files and folders

iii) Permanently deleting files and folders

Ans:

The shutil (or shell utilities) module has functions to let you copy, move, rename, and delete files in our Python programs. To use the shutil functions, we will first need to use import shutil.

```
>>> import shutil
```

i) Copying files and folders

shutil.copy(source, destination)

```
>>> shutil.copy('e:\\5A-ADP-2021\\Assign1.docx','d:\\pnsback')
'd:\\pnsback\\Assign1.docx'
>>>shutil.copy('e:\\5C-ADP-2020\\Assign1.docx','d:\\pnsback\\5AAssign1.docx')
'd:\\pnsback\\5AAssign1.docx'
```

shutil.copytree() copies the directory tree

Moving files and folders

```
>>>shutil.move('d:\\pnsback\\5aAssign1.docx', 'e:\\5A-ADP-2021') #moving
#existing file in target directory will be overwritten
#if there is no folder 5A-ADP-2021 then file will be moved with name 5A-ADP-2021
>>>shutil.move('d:\\pnsback\\5aAssign1.docx', 'e:\\%B-ADP-2020\\Assign15c.docx')
```

#Moving & Renaming : if the folder/sub-folder is not in the mentioned path, python will throw an exception

Permanently deleting files and folders

We can delete a single file or a single empty folder with functions in the os module, whereas to delete a folder and all of its contents, you use the shutil module.

```
>>> import os, shutil
```

- Calling os.unlink(*path) will delete the file at path.*
- Calling os.rmdir(*path) will delete the folder at path. This folder must be* empty of any files or folders.

- Calling shutil.rmtree(*path) will remove the folder at path, and all files* and folders it contains will also be deleted.

```
>>> os.unlink('average.py') # will delete the file permanently
>>> os.rmdir('c:\\windows\\temp') # will delete folder temp permanently
```

b. List out the benefits of compressing file with zip file module, also explain the concept of walking a directory tree. 8M

Ans:

A zip file can hold the compressed contents of many other files. Compressing a file reduces its size, which is useful when transferring it over the Internet.
Since a ZIP file can also contain multiple files and subfolders, it's a handy way to package several files into one. This single file, called an archive file, can then be, say, attached to an email. Our Python programs can both create and open (or extract) ZIP files using functions in the zipfile module.

Creating and Adding to ZIP Files : To create your own compressed ZIP files, you must open the ZipFile object in write mode by passing 'w' as the second argument. (This is similar to opening a text file in write mode by passing 'w' to the open() function.)

```
>>> import zipfile
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

walk() :
os.walk() generate the file names in a directory tree by walking the tree either top-down or bottom-up. For each directory in the tree rooted at directory top (including top itself), it yields a 3-tuple (dirpath, dirnames, filenames).

```
import os #importing os module
for (root,dirs,files) in os.walk('Myprogs', topdown=true):
print (root)
print (dirs)
print (files)
print ('--------------------------------')
```

c. List out difference between shutil.copy() and shutil.copytree() methods 6M

Ans:

shutil.copy(source, destination) The shutil. copy() method is used to copy a specified source without the metadata to the destination file or directory and then return the path to the newly created file

```
>>> shutil.copy('e:\\1stsemJ\\Assign1.docx','d:\\pnsback')
'd:\\pnsback\\Assign1.docx'
>>>shutil.copy('e:\\1stsemI\\Assign1.docx','d:\\pnsback\\1stsemJ\Assign1.docx')
'd:\\pnsback\\5AAssign1.docx'
```

shutil.copytree() copies the directory tree from source to destination. The destination directory, named by (dst) must not already exist.

```
src = 'C:/Singh/1stsemJ' # Source path
```

```
dest = 'C:/Singh/1stseml' #Destination path
# Copy the content of from source to destination
destination = shutil.copytree(src, dest)
```

8.

a. Briefly explain Assertion and raising a exception. 6M

Ans:

An assertion is a sanity check to make sure your code isn't doing something obviously wrong.
These sanity checks are performed by assert statements.
If the sanity check fails, then an AssertionError exception is raised.
An assert statement consists of the following:

- The assert keyword
- A condition (that is, an expression that evaluates to True or False)
- A comma
- A string to display when the condition is False
- In plain English, an assert statement says, "I assert that the condition holds true, and if not, there is a bug somewhere, so immediately stop the program."

```
#test if condition returns True
x = "hello"
assert x == "hello"
#if condition returns False, AssertionError is raised:
assert x == "goodbye"

#a message if condition is false
x = "hello"
#if condition returns False, AssertionError is raised:
assert x == "goodbye", "x should be 'hello'"
```

Raising and exception (raise) is typically used when we have detected an error condition or some condition does not satisfy. assert is similar but the exception is only raised if a condition is met. Both raise an AssertionError.

```
if condition:
    raise Exception('User Text')
```

b. Develop a Python program with a function named DivExp which takes two parameters a,b and returns a value c (c=a/b). Write a suitable assertion for a > 0 in function DivExP and raise an exception for when b=0. Program has to read two values from console and call the function DivExP. 8M

Ans:

```
def DivExp(a,b):
try:
if a>0:
print("Okay a is greater than zero.")
c=a/b
print("The result is",c)
except:
print("Divide error!!! divider is 0")
```

```
n1=int(input("Enter dividend : "))
n2=int(input("Enter divider : "))
DivExp(n1,n2)
```

#Output1
Enter dividend : 5
Enter divider : 0
Okay a is greater than zero.
Divide error!!! divider is 0
#output2
Enter dividend : 5
Enter divider : 2
Okay a is greater than zero.
The result is 2.5

c. Briefly explain different logging levels. 6M

Ans:

- Python has a built-in module logging which allows writing status messages to a file or any other output streams.
- Logging is a means of tracking events that happen when some software runs.
- The software's developer adds logging calls to their code to indicate that certain events have occurred.

There are six levels for logging in Python; each level is associated with an integer that indicates the log severity:

- **Notset = 0**: This is the initial default setting of a log when it is created. It is not really relevant and most developers will not even take notice of this category. In many circles, it has already become nonessential. The root log is usually created with level WARNING.
- **Debug = 10**: This level gives detailed information, useful only when a problem is being diagnosed.
- **Info = 20:** This is used to confirm that everything is working as it should.
- **Warning = 30:** This level indicates that something unexpected has happened or some problem is about to happen in the near future.
- **Error = 40:** As it implies, an error has occurred. The software was unable to perform some function.
- **Critical = 50**: A serious error has occurred. The program itself may shut down or not be able to continue running properly.

Example code:

```
import logging
logging.basicConfig(filename="logfile.log",format='%(asctime)s %(message)s',filemode="w")
logger=logging.getlogger() #creating an object
logger.setLevel(logging.DEBUG)
#test Messages
logger.debug ("Harmless debut message")
logger.info("Just for information")
logger.warning("It is a warning")
logger.error("tried to divide by zero?")
logger.critical("Internet is slow")
```

9.

a. Define classes and objects in Python. Construct the class called rectangle and initialize its height = 100, width =200, starting point as (x=0, y=0) and write the method to display the centre point coordinates of a rectangle. 8 Marks

Ans:

```
class rectangle():
   def __init__(self, h, w,x,y):
      self.height = h
      self.width  = w
      self.x=0
      self.y=0

   def centrepoint(self):
      self.cx =(self.x + self.height)/2
      self.cy = (self.y + self.width)/2
      print("Centre points = ",self.cx,self.cy)


rect1 = rectangle(100,200,0,0)
rect1.centrepoint()

#output
Centre points =  50.0 100.0
```

b. Briefly explain concept of prototyping and planning. 6M

Ans:

Prototype and Patch

- "Prototype and Patch" is A development plan that involves writing a rough draft of a program, testing, and correcting errors as they are found. For each function, a prototype that performed the basic calculation and then tested it, patching errors along the way. Prototype method is creational design pattern which aims to reduce the number of classes used for an application.
- Incremental corrections can generate code that is unnecessarily complicated—since it deals with many special cases—and unreliable. Python provides its own interface of Prototype via `copy.copy` and `copy.deepcopy` functions. And any class that wants to implement custom implementations have to override these member functions.

Planning:

- A development plan that involves high-level insight into the problem and more planning than incremental development or prototype development.
- For example of time objects with 3 data members hour minute and time there must be planning to write functions with concept that seconds and minutes should have 60 base. Minutes and seconds are resulting more than 60 in adding 2 time objects will not be so good. We should plan to write functions which is more reliable and more likely to be correct.

c. Explain clearly __init__() and __str__() methods with examples 6M

Ans:

The __init__() method is like constructor member function of C++ and Java.
It is used to initialize the objects data.
The __init__() method is called automatically as the objects are declared in memory.
self argument refers to the current object.
The __str__() method returns whole expression with objects data as string.
The __str__() returns a human-readable, or informal, string representation of an object.
This method is called by the built-in print(), str() and format() functions.

```
# __init__() and __str__() methods
class student:
    def __init__(self,usn,name,marks):
        self.usn=usn
        self.name=name
        self.marks=marks
    def __str__(self):
        return "%12s%20s%5d" %(self.usn,self.name,self.marks)
s1=student("1CR23CS001","Roma Ritambhara",78)
print(s1)

#output: 1CR23CS001 Roma Ritambhara 78
```

10.

a. Explain the term Objects are mutable with an example. 6M

Ans:

An object is instance of class. An object has behavior and characteristics. All the real world entities are objects. The class is also sometimes called object. The class object is like a factory for creating objects. To create a Point, we call Point as if it were a function.

```
>>> blank = Point()
>>> blank
<__main__.Point object at 0xb7e9d3ac>
```

The return value is a reference to a Point object, which we assign to blank. Creating a new object is called instantiation, and the object is an instance of the class. When we print an instance, Python tells us what class it belongs to and where it is stored.

Objects are mutable.

Mutable objects in Python are those that can be changed after they are created, like lists or dictionaries. Immutable objects, on the other hand, cannot be changed after they are created, such as strings, integers, or tuples.

b. Explain the concept of polymorphism with examples. 8M

Ans:

The word polymorphism means one thing having many forms. In terms of Object Oriented programming, polymorphism means the functions having the same name performs different task according to number and type of arguments (function signature). This is also know as function overloading.

“Function overloading in the same class is not permitted in Python.”. However we can show two classes having the same function name with different number of arguments.

```
#polymorphism
class rectangle:
   def area(self,l,w):
      print("Area of rectangle = ",l*w, "square unit")
class circle:
```

```
    def area(self,r):
        print("Area of circle = ",3.14*r*r, "square unit")

r1=rectangle()
c1=circle()
r1.area(9,6)
c1.area(4.5)
```

Operator overloading is also a type of Polymorphism. Changing the behavior of an operator so that it works with programmer-defined types is called **operator overloading**. For every operator in Python there is a corresponding special method, like __add__ for +

+ operator of Python can add numbers and string both. So, this is operators overloading and can be treated as polymorphism.

```
name=”Harishit”
surname=” Raj”
fullname=name+ surname
print(fullname)
```

Similary * multiplication operator can be used for replication of string also.

c. Explain briefly pure functions and modifiers with examples. 6M

Ans:

Pure function:

The function creates a new object, initializes its attributes, and returns a reference to the new object. This is called a pure function because it does not modify any of the objects passed to it as arguments and it has no effect, like displaying a value or getting user input, other than returning a value.

Here is a simple prototype of add_time:

```
def add_time(t1, t2):
    sum = Time()
    sum.hour = t1.hour + t2.hour
    sum.minute = t1.minute + t2.minute
    sum.second = t1.second + t2.second
    return sum
>>> start = Time()
>>> start.hour = 9
>>> start.minute = 45
>>> start.second = 0
>>> duration = Time()
>>> duration.hour = 1
>>> duration.minute = 35
>>> duration.second = 0
>>> done = add_time(start, duration)
>>> print_time(done)
10:80:00
```

The problem is that this function does not deal with cases where the number of seconds or minutes adds up to more than sixty.

Modifiers:

Sometimes it is useful for a function to modify the objects it gets as parameters. In that case, the changes are visible to the caller. Functions that work this way are called **modifiers**.

```
#addtime.py
class Time:
    def gettime(self):
        self.hr=int(input("Enter hours: "))
        self.min=int(input("Enter minutes: "))
        self.sec=int(input("Enter seconds: "))
    def add_time(self,t1,t2):
        sumt=Time()
        sumt.sec=t1.sec+t2.sec
        xmin, rsec = divmod(sumt.sec, 60)
        sumt.sec=rsec
        sumt.min=t1.min+t2.min+xmin
        xhr, rmin = divmod(sumt.min, 60)
        sumt.min=rmin
        sumt.hr=t1.hr+t2.hr+xhr
        return sumt

t1=Time()
t1.gettime()
t2=Time()
t2.gettime()
t3=Time()
t3=t3.add_time(t1,t2)
print("Adding both times : ",t3.hr,t3.min,t3.sec)
```

#expected output

```
Enter hours: 23
Enter minutes: 45
Enter seconds: 54
Enter hours: 9
Enter minutes: 32
Enter seconds: 51
Adding both times : 33 18 45
```