Page No.
Date / /

Q1

```
def sub_lists (L) :
    ## store all the sublists
    lists = [[]]

    for i in range (len(L) + 1 ) :
        for j in range (i) :
            Lists.append (L[j:i])
    return lists

# driven code

if __name__ == "__main__" :

mainlist = list ()
size = int (input (" Enter size of list = "))
print ("Enter elements of the list :")
for i in range (int (size)) :
    K = int (input (" "))
    mainlist. append (K)
print (" SUBLISTS are : ", sub_lists (mainlist))
```

→ SAMPLE OUTPUT

```
Enter size of list = 3
Enter elements of the list : 1, 2, 3
SUBLISTS are : [[],[1],[2],[3],[1,2],[2,3]
```

Page No.
Date / /

Q2. SOURCE CODE FOR GUI TO CONVERT KM to MILE

```
import tkinter as tk

def main()
    root = tk.TK()
    root.title("tk")
    root.geometry("400x200")

    label1 = tk.label(root, text = "Enter a
                distance in kilometer :")
    label1.place (x=50, y=30)

    label2 = tk.label(root, text =" Converted to
                                miles :")
    label2.place (x=50, y=100)

    textbox = tk.Entry(root, width =12, borderwidth,
                relief = "sunken")
    textbox.place (x=220, y=30)

    label3 = LK.label(root, text =" ")
    label3.place (x=120, y=100)

def calc_dist():
    miles = round (textbox.get()) * 0.621)
    label3.configure (text = str(miles) + 'miles')
```

Page No.
Date / /

```
convert1 = tk.Button (root, text = "convert",
                      command = calc_dist )
convert1.place (x = 125, y = 150)
convert1.pack (side = tk.left)

quit1 = tk.Button (root, text = "Quit", command
                   = quit)
quit1.place (x = 200, y = 150)
quit1.pack (side = tk.Right)

root.mainloop()

main()
```

Page No.
Date / /

Q3.

```
from random import randint

SHORTEST = 7
LONGEST = 10
MIN_ASCII = 33
MAX_ASCII = 126

## generating random password

def randomPassword():
    randomlength = randint(SHORTEST, LONGEST)
    result = ""
    for i in range(randomLength):
        randomchar = chr(randint(MIN_ASCII, MAX_ASCII))
        result = result + randomchar
    return result

## displaying random password

def main
    print("Random Password:", randomPassword())

## call main function only when solution has not
    been imported into another file.

if __name__ == "__main__":
    main()
```

→ SAMPLE OUTPUT:

Random Password: J"$aSn/

Page No.
Date / /

Q4.

```
## reading string from the user

str = input (" Enter a string:" )

## adding each character to a dictionary with the
   value of true and once we are done, the no. of
   keys in the dictionary will be the number of
   uniaue characters in the string

characters = { }
for ch in str :
    characters [ch] = True

## displaying the result

print (" No. of uniaue characters: ", len(characters))
print (characters)
```

→ SAMPLE INPUT = "Hello, World!"
SAMPLE OUTPUT = No. of uniaue characters = 10
{'h', 'e', 'l', 'o'}

→ SAMPLE INPUT = "ddddd"
SAMPLE OUTPUT = No. of uniaue characters = 1
{'d'}

Page No.
Date / /

Q5.

	ABSTRACT CLASS	INTERFACE
→	Abstract class can have abstract and non-abstract methods.	Interface can only have abstract methods
→	We can use an abstract base class to define and enforce an interface	An interface is a set of methods and attributes on that object.
→	Abstract classes are used when there is some common featured shared by all the objects as they are	We use interface if all the features need to be implemented differently for different objects.
→	Abstract class are faster	Interface is slow in compare to abstract class.
→	Doesn't support multiple inheritance	Supports multiple inherit-ance
→	Can have final, non-final, static and non-static variables	Only static and final variables
→	Abstract keyword is used	Interface keyword is used.

Page No.
Date / /

PYTHON PROGRAM

```
from abc import ABC, abstractmethod

class Car (ABC):
    @abstractmethod
    def company (self):
        pass

class Maruti (Car):
    def company (self):
        print ("MARUTI SUZUKI")

class Santro (Car):
    def company (self):
        print ("HYUNDAI")

M = Maruti ()
M. company ()

S = Santro ()
S. company ()
```

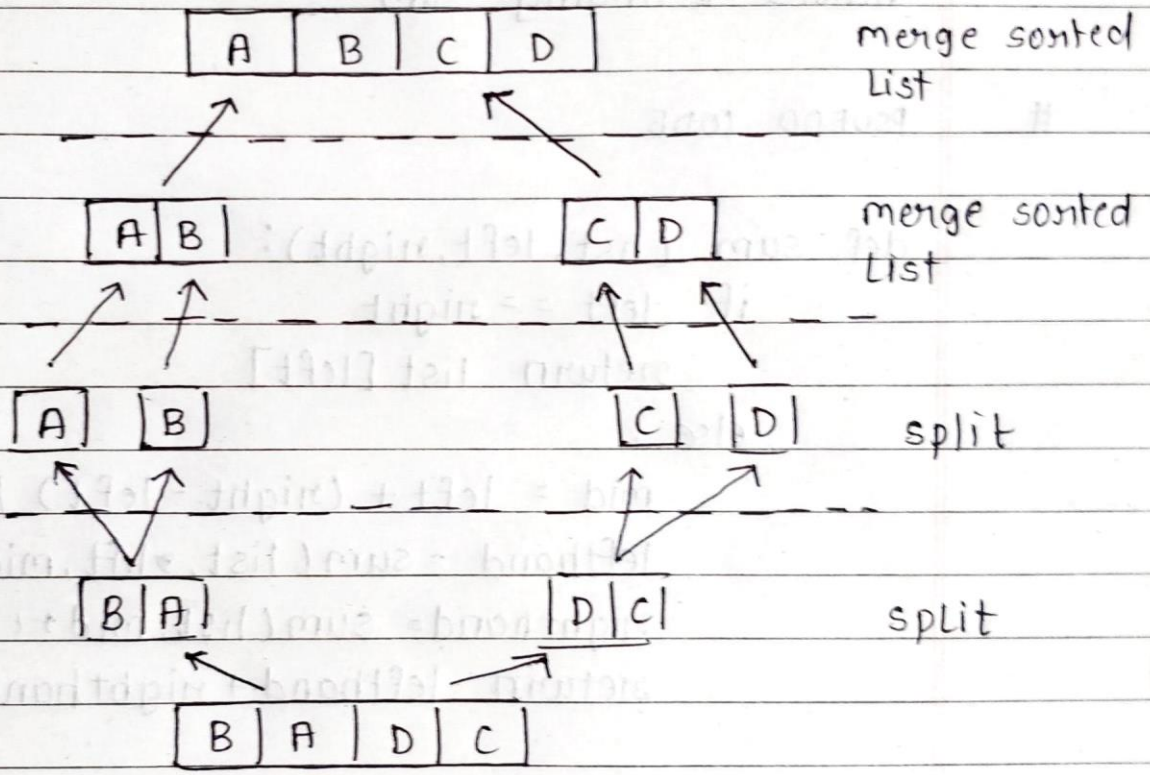
→ OUTPUT :

```
MARUTI SUZUKI
HYUNDAI
```

Page No.
Date / /

Q6. MERGE SORT, Recursive sum of all elements

Merge sort works by splitting the input list into the two halves, repeating the process on those halves and finally merging the two sorted halves together.



→ The algorithm first moves from top to bottom dividing the list into smaller and smaller parts until only separate or individual elements remain

→ From there, it moves back up ensuring that the merging lists are sorted.

Page No.
Date / /

→ This method is used when we want to decrease time complexity & also when we don't want to use loops

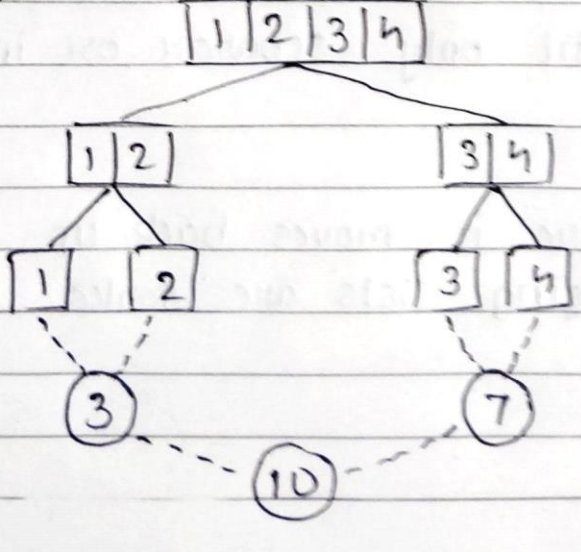
→ Now, similar approach we can use for summing up elements of a list by dividing them in equal halves & finding sum

PSUEDO CODE

```
def sum (list, left, right):
    if left == right
        return list [left]
    else :
        mid = left + (right - left) //2
        lefthand = sum ( list, left, mid )
        righthand = sum (list, mid+1, right)
        return lefthand + righthand
```

SAMPLE INPUT : arr [1,2,3,4]
SAMPLE OUTPUT : sum = 10

FLOWGRAPH



Page No.
Date / /

Q7. ARRAY A - (8,3)
ARRAY B - (2,2)

CODE:

```
import numpy as np
from random import randint
arrayA = np.random.randint(0,6,(8,3))
arrayB = np.random.randint(0,6,(2,2))
print("Array A \n")
print(arrayA)
print("\n", arrayB)

temp = (arrayA[..., np.newaxis, np.newaxis] == arrayB
rows = (temp.sum(axis=(1,2,3)) >= arrayB.
                    shape[1]).nonzero()[0]

print("\n Rows of array A that contain elements
      of each row of another array B : ")

print(rows)
```

→ SAMPLE OUTPUT :

```
Array A [[2 4 1]      Array B [[3 5]
 [0 2 1]                       [5 5]]
 [3 4 2]
 [3 0 4]      Rows of array A that contain elements
 [3 1 0]      of each row of array B are: [5 6 7]
 [5 3 2]
 [5 0 5]
 [5 4 1]]
```

Page No.
Date / /

Q8. PUBLIC INTERFACE - the public interface of class specifies the functionality supported by the class but does not disclose any details of how the functionality is implemented

→ In contrast, the implementation of a class is the code that accomplishes the task to support the class's functionality

→ Interface just defies what an object can do it, but actually won't do it

→ Implementation carries out the instructions defined in the Intstructions defined in interface

→ Interface and implementation can be in 2 different languages

→ Interface offers a means of expressing design without ~~or~~ worrying about implementation

→ Interface can be used to achieve loose coupling

→ Interface abstracts & protects details from client

→ In some languages like Java, intrface is used to support functionality of multiple inheritance.

Page No.
Date / /

Q9. NEGATIVE INDEX IN PYTHON

→ Python arrays and list items can be accessed with positive or negative number.

→ For instance our array / list of size n, then for positive index 0 is first index, 1 second last index will be n-1.

→ For negative index, -n is the first index, -(n-1) second, last negative index will be -1

→ A negative index access elements from the end of list counting backwards

For eg arr = [10,20,30,40,50]

```
print (arr[-1])  = 50
print (arr[-3])  = 30
print (arr[-2])  = 40
```

Example using list comprehension :

→ After getting list, we can get part of it using python's slicing operation which has the following syntax

```
[ start : stop : steps ]
```

So [:stop] will slice the list from starting till stop index and [start :] will slice list from start index till end.

Page No.
Date / /

→ Negative value of steps check shows right to left traversal instead of left to right traversal that is why [::-1] prints list in reverse order

Eg.

```
lst = list (range (1,11))
print (lst)

lst_rev = lst [::-1]
print (lst_rev)

lst_rev_new = lst [9:4:-2]
print (lst_rev_new)
```

→ OUTPUT :

```
[1,2,3,4,5,6,7,8,9,10]
[10,9,8,7,6,5,4,3,2,1]
[10,8,6]
```

Page No.
Date / /

Q10. PROJECT STRUCTURE OF DJANGO PROJECT :

→ Django makes use of a directory structure to arrange different parts of the web application. It creates a project and an app folder for this
→ Creating a proper project and organizing it helps in keeping the project DRY (don't repeat yourself) and clean.
→ When we create a Django project, it itself creates a root directory of the project with the project name you have given on it. It contains necessary files that would provide basic functionalities to your web applications.

→ Default Django has 6 files and functions of each are given below

(1) manage.py - this file is used basically as a command line utility and for development, deploying, debugging or running our web applications.
- this file contains code for runservers makemigrations or migrations etc that we use in shell.

↳ runserver
↳ migration
↳ makemigration

these are important commands under manage.py

Page No.
Date / /

(2) init.py - this file remains empty & is present their only to tell that this particular directory is a package

(3) setting.py - this file is present for adding all the applications and the middleware application present. Also, it has information about templates and database Overall, this is main file of our Django web application

(4) urls.py - this file handles all the URLs of our web application. This file has lists of all endpoints that we need for website.

URL - universal resource locator is used to provide the address of resources that are on internet

(5) wsgi.py - this file mainly concers with our ~~application~~ WSGI servers and is used for displaying our applications on to servers like apache.

WSGI stands for web Server Gateway Interface

(6) asgi.py - this file is newer version of django. ASGI ca be considered succeder interface to WSGI.

ASGI stands for Asynchronous Server Gateway Interface.