Q.1

```
#LinkedList in python using classes and objects
class node:
    def __init__(self,data):
        self.data = data
        self.next = None

class Linkedlist:
    def __init__(self):
        self.head = None

    def insert(self,data):
        newNode = node(data)
        newNode.data = data
        newNode.next = self.head
        self.head = newNode

    def delete(self,key):
        if self.head == None:
            print("Empty linked list")
            return
        else:
            temp = self.head
            prev = None
            while(temp != None and temp.data != key):
                prev = temp
                temp = temp.next
            if temp.data == key and prev == None:
                self.head = temp.next
            elif temp.data == key and prev != None:
                prev.next = temp.next
            elif temp == None:
                print("Match not found")




    def display(self):
```

```
        temp = self.head
        while(temp!= None):
            print(temp.data,end="->")
            temp = temp.next


ll = Linkedlist()
for i in range(1,11):
    ll.insert(i)

ll.delete(9)
ll.display()
```

Q.2

```
# Write a Python program to create a class representing a
queue data structure. Include methods
# for enqueueing and dequeuing elements.

class node:
    def __init__(self,data):
        self.data = data
        self.next = None
class Queue:
    def __init__(self):
        self.front = None
        self.rear = None

    def enqueue(self,data):
        newNode = node(data)
        if(self.front == self.rear == None):
            self.front = self.rear =newNode
        else:
            self.rear = newNode
            newNode.next = self.rear
    def dequeue(self):
        if(self.rear == None):
            print("Queue is empty")
            return
```

```
        else:
            temp = self.rear
            num = temp.data
            self.rear = temp.next
            temp = None
            return num



q1 = Queue()
q1.enqueue(1)
print(q1.dequeue())
q1.dequeue()

Q.3
# Write a Python program to create a class representing a
bank. Include methods for managing
# customer accounts and transactions.

class bank:
    def __init__(self):
        self.accounts = {}
        self.__pasword = {}

    def addAcc(self,accNo,password,balance = 0):
        self.accounts[accNo] = balance
        self.__pasword[accNo] = password

    def deposit(self,accNo,password,amount):
        if accNo in self.accounts:
            if password == self.__pasword[accNo]:
                self.accounts[accNo] += amount
                print("Balance Updated: ",self.accounts[accNo])
            else:
                print("Incorrect Password")
        else:
            print("Account Not found")
```

```
    def withdraw(self,accNo,password,amount):
        if accNo in self.accounts:
            if password == self.__pasword[accNo]:
                if self.accounts[accNo]>= amount:
                    self.accounts[accNo] -= amount
                    print("Balance Updated: ",self.accounts[accNo])
                else:
                    print("insufficient balance")
            else:
                print("Incorrect Password")
        else:
            print("Account Not found")

    def displayBalance(self,accNo,password):
        if accNo in self.accounts:
            if self.__pasword[accNo] == password:
                print(self.accounts[accNo])

b = bank()
b.addAcc(123,"abc")
b.deposit(123,"abc",100)
b.withdraw(123,'abc',90)
```

Q.4

```
# Create a class "Employee" with attributes name and salary.
Implement overloaded operators +
# and - to combine and compare employees based on their
salaries.

class Employee:
    def __init__(self,name,salary):
        self.name = name
        self.salary = salary

    def __add__(self,other):
        return (self.name + "&" +other.name, self.salary +
other.salary )
```

```
    def __sub__(self,other):
        return self.salary - other.salary

emp1 = Employee("Kavya",120000)
emp2 = Employee("Someone",500)
print(emp1 + emp2)
print(emp1 - emp2)
```

Q.5

```
# Create a base class "Shape" with methods to calculate the area and perimeter. Implement
# derived classes "Rectangle" and "Circle" that inherit from "Shape" and provide their own area
# and perimeter calculations.

class shape:
    def area(self):
        pass
    def perimeter(self):
        pass

class rectangle(shape):
    def __init__(self,length,width):
        super().__init__()
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
    def perimeter(self):
        return 2*(self.length+self.width)

class circle(shape):
    def __init__(self,radius):
        super().__init__()
        self.radius = radius

    def perimeter(self):
        return 6.28 *self.radius
```

```
    def area(self):
        return 3.14 *self.radius*self.radius
```

Q.6 **Create a class "BankAccount" with attributes account number and balance. Implement methods to deposit and withdraw funds, and a display method to show the account details.**

```
class bank:
    def __init__(self):
        self.accounts = {}
        self.__pasword = {}

    def addAcc(self,accNo,password,balance = 0):
        self.accounts[accNo] = balance
        self.__pasword[accNo] = password

    def deposit(self,accNo,password,amount):
        if accNo in self.accounts:
            if password == self.__pasword[accNo]:
                self.accounts[accNo] += amount
                print("Balance Updated: ",self.accounts[accNo])
            else:
                print("Incorrect Password")
        else:
            print("Account Not found")

    def withdraw(self,accNo,password,amount):
        if accNo in self.accounts:
            if password == self.__pasword[accNo]:
                if self.accounts[accNo]>= amount:
                    self.accounts[accNo] -= amount
                    print("Balance Updated: ",self.accounts[accNo])
                else:
                    print("insufficient balance")
            else:
                print("Incorrect Password")
        else:
```

```
            print("Account Not found")

    def displayBalance(self,accNo,password):
        if accNo in self.accounts:
            if self.__pasword[accNo] == password:
                print(self.accounts[accNo])

b = bank()
b.addAcc(123,"abc")
b.deposit(123,"abc",100)
b.withdraw(123,'abc',90)
```

Q.7

```
# Create a class for representing any 2-D point or vector. The methods inside this class include
# its magnitude and its rotation with respect to the X-axis. Using the objects define functions for
# calculating the distance between two vectors, dot product, cross product of two vectors. Extend
# the 2-D vectors into 3-D using the concept of inheritance. Update the methods according to 3-
# D.
import math
class two_D:
    def __init__(self,x,y):
        self.x =x
        self.y =y

    def mag(self):
        return (self.x**2 + self.y**2)**0.5
    def distance(self,other):
        return ((self.x-other.x)**2 + (self.y-other.y)**2)**0.5
    def dot(self,other):
        return (self.x*other.x + self.y*other.y)
    def cross(self,other):
        return self.x*other.y - self.y*other.x
    def rotate(self,theta):
```

```
        return two_D(self.x*math.cos(theta)-
self.y*math.sin(theta),self.x*math.sin(theta)
+self.y*math.cos(theta))


class three_D(two_D):
    def __init__(self, x, y,z):
        super().__init__(x, y)
        self.z =z

    def mag(self):
        xy_mag = super().mag()
        return math.sqrt(xy_mag**2 + self.z**2)

    def dot(self,other):
        return self.x*other.x + self.y*other.y + self.z*other.z

    def cross(self,other):
        x_new = self.y * other.z - self.z * other.y
        y_new = self.z * other.x - self.x * other.z
        z_new = self.x * other.y - self.y * other.x
        return three_D(x_new, y_new, z_new)
```

Q.8

```
# Decode the message:
# A message containing the letters from A-Z can be encoded
into the numbers using the mapping
# A-> 1, B-> 2, C-> 3, ..., Z-> 26. To decode an encoded
message, you need to group the digits
# and do the reverse mapping. You are required to display all
the possible decoded messages.
# For example: "11106" can be decoded into:
# a. "AAJF" with the grouping (1 1 10 6)
# b. "KJF" with the grouping (11 10 6)

def decode_ways(s, result=""):
    if not s:
        print(result)
        return
```

```
    if s[0] != "0":
        decode_ways(s[1:], result + chr(int(s[0]) + 64))
    if len(s) > 1 and "10" <= s[:2] <= "26":
        decode_ways(s[2:], result + chr(int(s[:2]) + 64))

decode_ways(input("Enter the code: “))
```

Q.9# Create a tokenizer for your own language (mother tongue you speak). The tokenizer should
tokenize punctuations, dates, urls, emails, numbers (in all different forms such as “33.15”,

“3,22,243”, “313/77”), social media usernames/user handles. Use regular expressions to design
this. [Hint: Use unicode blocks for your language, check wikipedia pages]

```
import re

def custom_tokenizer(text):
    patterns = {
        "dates": r"\b\d{1,2}[-/]\d{1,2}[-/]\d{2,4}\b",
        "urls": r"https?://[^\s]+",
        "emails": r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]
{2,}",
        "numbers": r"\b\d{1,3}(?:,\d{3})*(?:\.\d+)?\b",
        "usernames": r"@[a-zA-Z0-9_]+",
        "punctuation": r"[,.!?;:]"
    }

    tokens = []
    for category, pattern in patterns.items():
        matches = re.findall(pattern, text)
        if matches:
            tokens.extend(matches)

    return tokens

print(custom_tokenizer("www.google.com"))
```