NAME:- PARAM KANADA
ADMISSION NUMBER:- U24AI047
LAB ASSIGNMENT 7 AND 8

QUESTION 1:

```
# Linked LIst

class node:

    def __init__(self,data):
        self.data = data
        self.next = None

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

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

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" ")
            temp = temp.next

        print()

    def delete(self, num):
        current = self.head

        if current and current.data == num:
            self.head = current.next
            current = None
            return

        prev = None
        while current and current.data != num:
            prev = current
            current = current.next

        if current is None:
            print("Key not found in the list.")
            return
```

```
        prev.next = current.next
        current = None

a = int(input("Enter How Many LL You want = \n"))
o = LL()

for i in range(a):
    c = int(input("Enter Data = \n"))
    o.insert(c)

print("\nDisplay...\n\n")
o.display()

d = int(input("\n\nEnter the node you wanna delete = \n"))
o.delete(d)

print("\nDisplay...\n\n")
o.display()
```

OUTPUT:-

```
PS F:\SVNIT PRACTICALS\WPP> python -u "f:\SVNIT PRACTICALS\WPP\ASSIGNMENT 7_8\q1.py"
10 -> 20 -> 30 -> None
10 -> 30 -> None
Key not found
PS F:\SVNIT PRACTICALS\WPP>
```

QUESTION 2:-

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

    def __init__(self):
        self.queue = []

    def enqueue(self,data):
        self.queue.append(data)

    def dequeue(self):
        if not self.isempty():
            removed = self.queue.pop(0)
            print(f"Dequeued: {removed}")
            return removed
        else:
```

```
            print("Queue is empty!")
            return None

    def isempty(self):
        return len(self.queue) == 0

    def display(self):
        if self.isempty():
            print("Queue is empty!")
        else:
            print("Queue = ", " ".join(map(str, self.queue)))

a = int(input("Enter How Many Queue You want = \n"))
o = queue()

for i in range(a):
    c = int(input("Enter Data = \n"))
    o.enqueue(c)

print("\nDisplay...\n\n")
o.display()

d = int(input("\n\nEnter How many times you wanna dequeue = \n"))

for i in range(d):
    o.dequeue()

print("\nDisplay...\n\n")
o.display()
```

OUTPUT:-

```
Enter How Many Queue You want =
2
Enter Data =
1
Enter Data =
2

Display...


Queue =  1 2


Enter How many times you wanna dequeue =
3
Dequeued: 1
Dequeued: 2
Queue is empty!

Display...


Queue is empty!
PS F:\SVNIT PRACTICALS\WPP>
```

QUESTION 3:-

```python
# Bank

class bankacc:

    def __init__(self, accnumber, accholder, balance=0.0):
        self.accnumber = accnumber
        self.accholder = accholder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited {amount} into account {self.accnumber} ,
New balance: {self.balance}\n")
        else:
            print("Invalid Operation !!!\n")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
```

```
            print(f"Withdrew {amount} from account {self.accnumber} ,
New balance: {self.balance}\n")
        elif amount > self.balance:
            print("Insufficient balance !\n")
        else:
            print("Invalid Operation")

    def getbalance(self):
        return self.balance

    def displaydetails(self):
        print(f"Account Number: {self.accnumber}\nAccount Holder:
{self.accholder}\nBalance: {self.balance}\n")

class bank:

    def __init__(self, bank_name):
        self.bank_name = bank_name
        self.accounts = {}

    def createaccount(self, accnumber, accholder, initialbalance=0.0):
        if accnumber in self.accounts:
            print("Account number already exists !\n")
        else:
            self.accounts[accnumber] = bankacc(accnumber, accholder,
initialbalance)
            print(f"Account for {accholder} created successfully \n")

    def getaccount(self, accnumber):
        return self.accounts.get(accnumber, None)

    def deposit(self, accnumber, amount):
        account = self.getaccount(accnumber)
        if account:
            account.deposit(amount)
        else:
            print("Account not found !!!\n")

    def withdraw(self, accnumber, amount):
        account = self.getaccount(accnumber)
        if account:
            account.withdraw(amount)
        else:
```

```
            print("Account not found !!!\n")

    def displayaccounts(self):
        if not self.accounts:
            print("No accounts found !!!\n")
        else:
            print(f"\nAccounts in {self.bank_name}:")
            for account in self.accounts.values():
                account.displaydetails()
                print("-" * 30)

o = input("Enter bank name = \n")
a = bank(o)

b = int(input("Enter Acc Number = \n"))
c = input("Enter Acc Holder Name = \n")
d = int(input("Enter Acc balance = \n"))

a.createaccount(b,c,d)

e = int(input("Enter How much you wanna deposit = \n"))
a.deposit(b,e)

f = int(input("Enter How much you wanna withdraw = \n"))
a.withdraw(b,f)

print("\nDisplay...\n")
a.displayaccounts()
```

OUTPUT:-

```
PS F:\SVNIT PRACTICALS\WPP> python -u "f:\SVNIT PRACTICALS\WPP\ASSIGNMENT 7_8\tempCodeRunnerFile.py"
Enter bank name =
KOTAK
Enter Acc Number =
54354
Enter Acc Holder Name =
PARAM
Enter Acc balance =
50000
Account for PARAM created successfully

Enter How much you wanna deposit =
500
Deposited 500 into account 54354 , New balance: 50500

Enter How much you wanna withdraw =
16000
Withdrew 16000 from account 54354 , New balance: 34500


Display...


Accounts in KOTAK:
Account Number: 54354
Account Holder: PARAM
Balance: 34500

-----------------------------
PS F:\SVNIT PRACTICALS\WPP>
```

Help us improve our support for C++

QUESTION 4:-

```python
# Employee

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

    def __add__(self, other):
        if isinstance(other, emp):
            combined_salary = self.salary + other.salary
            return emp(f"{self.name} & {other.name}", combined_salary)

    def __sub__(self, other):
        if isinstance(other, emp):
            salary_difference = self.salary - other.salary
            return abs(salary_difference)

    def __str__(self):
        return f"Employee(Name = {self.name}, Salary = {self.salary})"

a = input("Enter Employee One Name = \n")
b = int(input("Enter Salary of Employee One = \n"))

emp1 = emp(a,b)
```

```
c = input("Enter Employee Two Name = \n")
d = int(input("Enter Salary of Employee Two = \n"))

emp2 = emp(c,d)

print("\nCombined Salary of Employees = \n")
combine = emp1 + emp2
print(combine)

print("\nCompared Salary of Employees = \n")
Diff = emp1 - emp2
print(Diff)
```

OUTPUT:-

```
PS F:\SVNIT PRACTICALS\WPP> python -u "f:\SVNIT PRACTICALS\WPP\ASSIGNMENT 7_8\q4.py"
Enter Employee One Name =
KANA
Enter Salary of Employee One =
5000
Enter Employee Two Name =
PARA
Enter Salary of Employee Two =
21231

Combined Salary of Employees =

Employee(Name = KANA & PARA, Salary = 26231)

Compared Salary of Employees =

16231
PS F:\SVNIT PRACTICALS\WPP>
```

QUESTION 5:-

```
# Shape

import math

class Shape:
    def area(self):
        pass

    def perimeter(self):
        pass

class Rectangle(Shape):
```

```
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)

    def __str__(self):
        return f"Rectangle(width = {self.width} , height =
{self.height})"

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

    def perimeter(self):
        return 2 * math.pi * self.radius

    def __str__(self):
        return f"Circle(radius = {self.radius})"

a = int(input("Enter Width for Rectangle = \n"))
b = int(input("Enter Height for Rectangle = \n"))
rec = Rectangle(a,b)

c = int(input("Enter radius for Circle = \n"))
cir = Circle(c)

print("\nDisplay...\n")

print(f"Area of Rectangle = {rec.area()} , Perimeter of Rectangle =
{rec.perimeter()}\n\n")
print(f"Area of Circle = {cir.area()} , Perimeter of Circle =
{cir.perimeter()}\n")
```

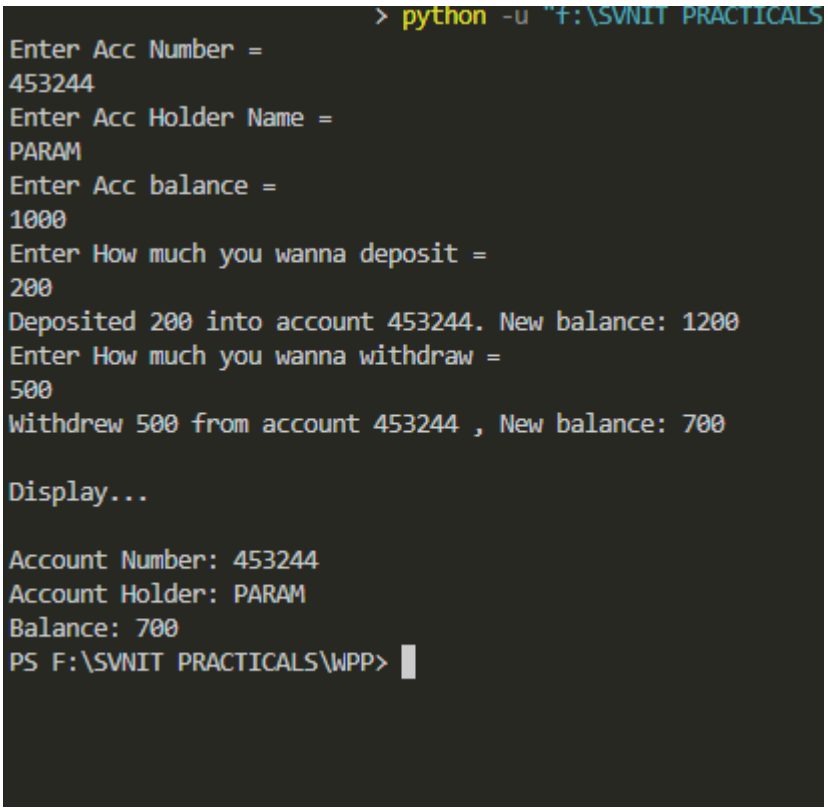
OUTPUT:-

QUESTION 6:-

```
# bank Acc

class bankacc:

    def __init__(self, accnumber, accholder, balance=0.0):
        self.accnumber = accnumber
        self.accholder = accholder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited {amount} into account {self.accnumber}. New balance:
{self.balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew {amount} from account {self.accnumber} , New balance:
{self.balance}")
        elif amount > self.balance:
            print("Insufficient balance!")
        else:
            print("Withdrawal amount must be positive.")

    def getbalance(self):
        return self.balance

    def displaydetails(self):
        print(f"Account Number: {self.accnumber}\nAccount Holder: {self.accholder}\nBalance:
{self.balance}")

b = int(input("Enter Acc Number = \n"))
c = input("Enter Acc Holder Name = \n")
d = int(input("Enter Acc balance = \n"))

a = bankacc(b,c,d)

e = int(input("Enter How much you wanna deposit = \n"))
a.deposit(e)

f = int(input("Enter How much you wanna withdraw = \n"))
a.withdraw(f)
```

```
print("\nDisplay...\n")
a.displaydetails()
```

OUTPUT:-



QUESTION 7:-

```
# Vectors

import math

class Vector2D:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def magnitude(self):
        return math.sqrt(self.x ** 2 + self.y ** 2)

    def rotation_angle(self):
        return math.degrees(math.atan2(self.y, self.x))

    def distance(self, other):
        return math.sqrt((self.x - other.x) ** 2 + (self.y - other.y) ** 2)

    def dotproduct(self, other):
        return self.x * other.x + self.y * other.y
```

```
    def crossproduct(self, other):
        return self.x * other.y - self.y * other.x

    def __str__(self):
        return f"Vector2D({self.x}, {self.y})"


class Vector3D(Vector2D):

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

    def magnitude(self):
        return math.sqrt(self.x ** 2 + self.y ** 2 + self.z ** 2)

    def distance(self, other):
        return math.sqrt((self.x - other.x) ** 2 + (self.y - other.y) ** 2 + (self.z - other.z) ** 2)

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

    def crossproduct(self, other):
        return Vector3D(
            self.y * other.z - self.z * other.y,
            self.z * other.x - self.x * other.z,
            self.x * other.y - self.y * other.x
        )

    def __str__(self):
        return f"Vector3D({self.x}, {self.y}, {self.z})"

b = int(input("\nEnter X coordinate for vector 1 = \n"))
c = int(input("Enter Y coordinate for vector 1 = \n"))
v1 = Vector2D(b,c)

d = int(input("\nEnter X coordinate for vector 2 = \n"))
e = int(input("Enter Y coordinate for vector 2 = \n"))
v2 = Vector2D(d,e)

print("\nFor 1and 2 \n")
print(f"Magnitude: {v1.magnitude()}")
print(f"Rotation Angle: {v1.rotation_angle()}°")
print(f"Distance to v2: {v1.distance(v2)}")
print(f"Dot Product: {v1.dotproduct(v2)}")
print(f"Cross Product: {v1.crossproduct(v2)}")
```

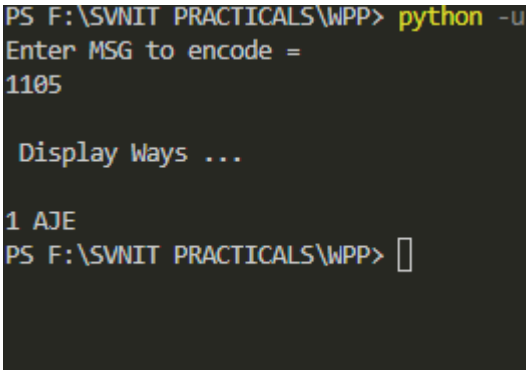
```
f = int(input("\nEnter X coordinate for vector 3 = \n"))
g = int(input("Enter Y coordinate for vector 3 = \n"))
h = int(input("Enter Z coordinate for vector 3 = \n"))
v3 = Vector3D(f,g,h)

i = int(input("\nEnter X coordinate for vector 4 = \n"))
j = int(input("Enter Y coordinate for vector 4 = \n"))
k = int(input("Enter Z coordinate for vector 4 = \n"))
v4 = Vector3D(i,j,k)

print("\nFor 3 and 4 \n")
print(f"Magnitude: {v3.magnitude()}")
print(f"Distance to v4: {v3.distance(v4)}")
print(f"Dot Product: {v3.dotproduct(v4)}")
print(f"Cross Product: {v3.crossproduct(v4)}")
```

SOLUTION:-

QUESTION 8:-

```
# Decode

def decode(encoded, index=0, current_decoding="", results=None):
    if results is None:
        results = []

    if index == len(encoded):
        results.append(current_decoding)
        return

    num1 = int(encoded[index])
    if 1 <= num1 <= 9:
        decode(encoded, index + 1, current_decoding + chr(64 + num1), results)

    if index + 1 < len(encoded):
        num2 = int(encoded[index:index + 2])
        if 10 <= num2 <= 26:
            decode(encoded, index + 2, current_decoding + chr(64 + num2), results)

    return results

a = input("Enter MSG to encode = \n")

b = decode(a)

print("\n Display Ways ...\n")
for i,j in enumerate(b,1):
    print(f"{i} {j}")
```

OUTPUT:-



QUESTION 9:-

```
# unicode

import re

def hindi_tokenizer(text):
    patterns = {
        "url": r"https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+",
        "email": r"[\w\.-]+@[\w\.-]+\.[a-zA-Z]{2,6}",
        "date": r"\d{1,2}[-/.]\d{1,2}[-/.]\d{2,4}",
        "number": r"\d{1,3}(?:,\d{2,3})*(?:\.\d+)?|\d+/\d+|[\u0966-\u096F]+",  # Hindi numerals
        "username": r"@[\w_]+",
        "punctuation": r"[.,!?;:\"'()\[\]{}]",
        "hindi_word": r"[\u0900-\u097F]+"  # Hindi Devanagari block
    }

    combined_pattern = "|".join(f"(?P<{key}>{pattern})" for key, pattern in patterns.items())

    tokens = []
    for match in re.finditer(combined_pattern, text):
        for key, value in match.groupdict().items():
            if value:
                tokens.append((key, value))

    return tokens

a = input("Enter text = \n\n")
tokens = hindi_tokenizer(a)
print()

for i in tokens:
    print(i,end = '\t')
```

OUTPUT:-