ASSIGNMENT-1

1. Program for Prime Number.

```
num = int(input("Enter a Number :"))
if num > 1:
    for i in range(2, (num/2)+1):
        if (num % i) == 0:
            print(num, "is not a prime number")
            break
        else:
            print(num, "is a prime number")
    else:
        print(num, "is not a prime number")
```

2. Program for factorial.

```
def factorial(n):
    return 1 if (n==1 or n==0) else
                n * factorial(n-1)

n = int(input("Enter the Number:"))
print("factorial of", n, "is", factorial(n))
```

3. LCM of a given Number.

```
import math
def lcm_using_gcd(a,b):
    gcd = math.gcd(a,b)
    lcm = (a*b)//gcd
    return lcm
n1 = int(input("Enter 1st Number:"))
n2 = int(input("Enter 2nd Number:"))
print("LCM of", n1, "and", n2, "is:", lcm_using_gcd
                                    (n1, n2))
```

4. Reverse a string.

```
def reverse(s):
    str = s[::-1]
    return str
s = "Ajinkya"
print("The original string is :", end=" ")
print(s)
print("The reversed string is :", end=" ")
print(reverse(s))
```

5. Switch case Program.

```
def num_to_str(argument):
    match argument:
        case 1:
            add = a+b
            return add
        case 2:
            sub = a-b
            return sub
        case 3:
            mult = a*b
            return mult.
        case 4:
            div = a/b
            return div
        case default:
            return "Try Again"
```

```
print("Enter 1 for Addition")
print("Enter 2 for Subtraction")
print("Enter 3 for Multiplication")
print("Enter 4 for Division")
a = int(input("Enter First Number:"))
b = int(input("Enter Second Number:"))
ch = int(input("Enter Your Choice:"))
head = num_to_str(ch)
print(head).
```

6. Palindrome Number.

→
```
num = int(input("Enter a value:"))
temp = num
rev = 0
while (num > 0):
    dig = num % 10
    rev = rev * 10 + dig
    num = num // 10
    if (temp == rev):
        print("This value is a palindrome
                number!")

    else:
        print("This value is not a
                Palindrome Number
```

ASSIGNMENT-2

1. Water-Jug Problem:

```
-> From collections import deque
def wat_jug(C1, C2, T1, T2):
    queue = deque()
    visited = set()
    queue.append((0,0,[]))
    visited.add((0,0))
    while queue:
        A1, A2, path = queue.popleft()
        if A1 == T1 and A2 == T2:
            return path+[(A1, A2)]
        next_states = [
        (C1, A2),
        (A1, C2),
        (0, A2),
        (A1, 0),
        (A1-min(A1, C2-A2), A2+min(A1, C2-A2)),
        (A1+min(A2, C1-A1), A2-min(A2, C1-A1)),
        ]
        for state in next_states:
            if state not in visited:
                visited.add(state)
                queue.append((state[0], state[1],
                              path+[(A1, A2)]))
    return "No Solution"
C1, C2, T1, T2 = 3, 4, 2, 0
res = wat_jug(C1, C2, T1, T2)
print(res)
```

ASSIGNMENT-3

1) SET OPERATIONS:

i) append() →

```
veg = ['corn', 'Tomato', 'Broccoli']
veg.append('Cucumber')
print(veg).
```

ii) Extend() →

```
color = ['Red', 'Pink', 'Orange']
color2 = ['Blue', 'Yellow']
color.extend(color2)
print(color)
```

iii) insert() →

```
name = ['Rita', 'Pinky', 'Babita']
name.insert(1, 'Mini')
print(name)
```

iv) remove() →

```
roll = [1,2,3,4,5]
roll.remove(3)
print(roll)
```

v) count() →

```
place = ['Delhi', 'Banglore', 'Delhi']
value = place.count('Delhi')
print(value)
```

vi) clear() →

```
place = ['Delhi', 'Banglore', 'Kollrata']
place.clear()
print(place)
```

vii) sort() →

```
bike = ['Splendor', 'Royal Enfield', 'Pulsor']
bike.sort(reverse = false)
print(bike)
```

viii) reverse() →

```
emp_id = [12, 13, 14, 15]
emp_id.reverse()
print(emp_id)
```

ix) index() →

```
flower = ['Flora', 'Hana', 'Rose']
name = flower.index('Hana')
print(name).
```

x) copy() →

```
chocolate = ['Kitkat', 'Bournville', 'Snickers']
chocolate.copy()
print(chocolate.
```

) set of vowels →

```
vowels = {'a', 'e', 'i', 'o', 'u'}
print('vowels(before clear):', vowels)
```

Lab

> clearing vowels →
vowels.clear()
print("Vowels (after clear):", vowels)

> set of vowels →
vowels = {'a', 'e', 'i', [redacted], 'u'}

> adding 'o' →
vowels.add('o')
print("Vowels are:", vowels)

> adding 'a' again →
vowels.add('a')
print("Vowels are:", vowels)

> language set →
lang = {"English", "French", "German"}

> removing "German" from language →
lang.remove("German")

> Updated language set →
print("Updated language set:", lang)

```
A = {'a', 'b', 'c', 'd'}
B = {'c', 'f', 'g'}

> Equivalent to A-B:
    print(A.difference(B))
> Equivalent to B-A
    print(B.difference(A))

    print(A-B)
    print(B-A)

A = {100, 7, 8}
B = {200, 4, 5}
C = {300, 2, 3, 7}
D = {100, 200, 300}
    print(A & C)
    print(A & D)
    print(A & C & D)
    print(A & B & C & D)

A = {2, 3, 5, 4}
B = {2, 5, 100}
C = {2, 3, 8, 9, 10}
print(B.intersection(A))
print(B.intersection(C))
print(A.intersection(C))
print(C.intersection(A, B))
```

```
A = {'a', 'c', 'd'}
B = {'c', 'd', 2}
C = {1, 2, 3}
print('A U B =', A | B)
print('B U C =', B | C)
print('A U B U C =', A | B | C)

A = {'a', 'c', 'd'}
B = {'c', 'd', 2}
C = {1, 2, 3}
print('A U B =', A.union(B))
print('B U C =', B.union(C))
print('A U B U C =', A.union(B, C))
print('A.union() =', A.union())
```

> Chatbot :

```
import random
grt = ['hola', 'hello', 'hi', 'Hi', 'Hey']
ran_grt = random.choice(grt)
ques = ['How are you?', 'What are you doing']
res = ['okay', 'I'm fine']
ran_que = random.choice(res)
while True:
    userInput = input(">>> ")
    if userInput in greetings:
        print(random_greeting)
    elif userInput in question:
        print(random_response)
    else:
        print("I did not understand what you said")
```

ASSIGNMENT -4

→ Breadth First Search:

```
graph = { 'A' : ['B', 'C'],
          'B' : ['D', 'E'],
          'C' : ['F']
          'D' : []
          'E' : ['F']
          'F' : []
        }
visited = []
queue = []

def bfs (visited, graph, node):
    visited.append (node)
    queue.append (node)

while queue:
    s = queue.pop (0)
    print (s, end = " ")

    for neighbour in graph[s]:
        if neighbour not in visited:
            visited.append (neighbour)
            queue.append (neighbour)
bfs (visited, graph, 'A').
```

classmate
Date
Page

ASSIGNMENT - 5

→ Depth-First Search :

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited

graph = {'0': set(['1', '2']),
         '1': set(['0', '3', '4']),
         '2': set(['0']),
         '3': set(['1']),
         '4': set(['2', '3'])
        }

dfs(graph, '0').
```

Ans

Assignment - 6

1 Linear Regression:

```
import numpy as np
import matplotlib.pyplot as plt
def estimat_coef(x,y):
# no. of observations / Points
n = np.size(x)
# mean of x & y vector
m_x = np.mean(x)
m_y = np.mean(y)
calculating cross-deviation & deviation about
x
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
calculating regression coefficients
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
return (b_0, b_1)

def plot_reg_line(x,y,b):
Plotting actual points as scatter plot
    plt.scatter(x,y, color = "m", marker
                = "o", s = 30)
Predicted response vector.
    y_pred = b[0] + b[1]*x
plotting regression line
    plt.plot(x,y_pred, color = "g")
```

putting labels
plt.xlabel('x')
plt.ylabel('y')
function to show plot
plt.show()

```
def main():
    # observations / Data
    x = np.array([0,1,2,3,4,5,6,7,8,9])
    y = np.array([1,3,2,5,7,8,8,9,10,12])
    # estimating coefficients
    b = estimate_coef(x,y)
    print("Estimated coefficients:
    b_0 = {b[0]} and b_1 = {b[1]}")
    # plotting regression line
    plot_reg_line(x,y,b)

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

2) Random Forest:

```
from sklearn.model_selection import train_test_split
from sklearn import datasets
iris = datasets.load_iris()
print(iris.target_names)
print(iris.feature_names)
X, y = datasets.load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split
                    (X, y, test_size=0.20)
from sklearn.ensemble import RandomForestClassifier
import pandas as pd.
data = pd.DataFrame({
                    'sepallength': iris.data[:, 0],
                    'sepalwidth': iris.data[:, 1],
                    'petallength': iris.data[:, 2],
                    'petalwidth': iris.data[:, 3],
                    'species': iris.target})
print(data.head())
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
from sklearn import metrics
print("ACCURACY of MODEL:", metrics.accuracy
                    _score(y_test,
                    y_pred))
```

(c) v. good
Akash
1111019

```
clf.predict([[3, 3, 2, 2]])
feature_imp = pd.Series(clf.feature_importances_,
                    index = iris.feature_names).sort_value
                    (ascending = False)
feature_imp.
```

3] K-nearest Neighbours

```
from sklearn.neighbours import kNeighboursClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
# loading data
irisData = load_iris()
# Create feature & target arrays
X = irisData.data
y = iris.Data.target
# Split data into train & test part
X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size = 0.2, random_state = 42)
knn = KNeighboursClassifier(n_neighbours = 7)
knn.fit(X_train, y_train)
#Predict on dataset which model has not seen before
print(knn.predict(X_test))
neighbours = np.arange(1, 9)
train_accuracy = np.empty(len(neighbours))
test_accuracy = np.empty(len(neighbours))
#loop over k values
for i, k in enumerate(neighbours):
    knn = KNeighboursClassifier(n_neighbours = k)
    knn.fit(X_train, y_train)
#Compute train & test data accuracy
```

(C)