1. Practical on basic programs using python for introducing and using python environment such as,

a) Program to print multiplication table for given no.

```
def Multiple(N):
  for i in range(1,11):
    print(N*i)


a=int(input("Enter number :"))

if(a<0):
  print("Please enter Positive number.")

else:
  Multiple(a)
```

b) Program to check whether the given no is prime or not

```
a=int(input("Enter a number:"))
count=0
for i in range(1,a+1):
  if(a%i==0):
   count=count+1


if(count==2):
 print("It is prime number")

else:
  print("It is not prime number")
```

c) Program to find factorial of the given no and similar programs

```
a=int(input("Enter a number:"))
fact=1

for i in range(1,a+1):
  fact=fact*i

print("Factorial of given number is",fact)
```

2. Write a program to implement List Operations

```
# Creating a nested list
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# Iterating through the nested list
print("Iterating through the nested list:")
for row in nested_list:
    for element in row:
        print(element, end=" ")
    print()

#Print length of string
print(len(nested_list))

#Concatenation of string
a=[1,2,3,4,5]
b=[6,7,8,9]
print(a+b)

#Membership(find particular element present in list or not)
if(2 in a):
        print(“Number is present ”)
else:
        print(“Number is not present”)

#Iteration
c=[10]*4
```

```
print(c)

#Indexing and slicing
# Creating a list
my_list = ['apple', 'banana', 'sweet lime', 'strawberry', 'watermelon']

# Accessing elements using positive indexing
print("Positive indexing:")
print("Element at index 0:", my_list[0])
print("Element at index 2:", my_list[2])

# Accessing elements using negative indexing
print("\nNegative indexing:")
print("Last element:", my_list[-1])
print("Second-to-last element:", my_list[-2])

# Slicing a list
print("\nSlicing:")
sliced_list = my_list[1:4]
print("Sliced list:", sliced_list)

# Modifying elements using indexing
print("\nModifying element at index 1:")
my_list[1] = 'blueberry'
print("Modified list:", my_list)
```

Write a program to implement List Methods

```
# Create a list
list1 = [1, 2, 3, 4, 5]

# Print the original list
print("Original List:", list1)

# Append an element to the end of the list
list1.append(6)
print("After appending 6:", list1)

# Extend the list by appending elements from another list
list2 = [7, 8, 9]
list1.extend(list2)
print("After extending with [7, 8, 9]:", list1)

# Remove an element by value
list1.remove(3)
print("After removing 3:", list1)

# Remove an element by index
removed_element = list1.pop(2)
print(list1)
```

3 Write a program to Illustrate Different Set Operations.

```
s1={67,34,56,32,90,2}
s2={45,20,2,67,89,56}

print(s1)
print(s2)

print("Union of Set s1 and s2: ",s1|s2)

print("Union of Set s1 and s2 using function: ",s1.union(s2))

print("Symmetric diffrence of two set: ",(s1-s2)|(s2-s1))

print("Symmetric diffrence of two set using function: ",
        s1.symmetric_difference(s2))

print("intersection of two set : ",s1&s2)

print("intersection of two set using function: ",s1.intersection(s2))

print("Difference of two set: ",s1-s2)

print("Difference of two set using function: ",s1.difference(s2))

print(“Adding element 6 in set 1:”, s1.add(6))

print(“Adding multiple element:”,s1.update(99,30))

print(“Removing element 67 from set 1:”,s1.remove(67))

print(“Poping element from set 2:”, s2.pop())

print(“Checking set1 is super set of set2 or not: ”,s1.issuperset(s2))

print(“Disjoint function:”, s1.isdisjoint(s2))

print(“clears set :”,s1.clear())
```

4 Write a program to implement Simple Chatbot.

```
import nltk
from nltk.chat.util import Chat, reflections
"""reflections = {
"i am" : "you are",
"i was" : "you were",
"i" : "you",
"i'm" : "you are",
"i'd" : "you would",
"i've" : "you have",
"i'll" : "you will",
"my" : "your",
"you are" : "I am",
"you were" : "I was",
"you've" : "I have",
"you'll" : "I will",
"your" : "my",
"yours" : "mine",
"you" : "me",
"me" : "you"
}"""

pairs = [
[r"my name is (.*)",["Hello %1, How are you today ?",]],

[r"hi|hey|hello",["Hello", "Hey there",]],

[r"what is your name ?",["I am a bot creat. you can call me crazy!",]],

[r"how are you ?",["I'm doing good and How about You ?",]],

[r"sorry (.*)",["Its alright","Its OK, never mind",]],

[r"I am fine",["Great to hear that, How can I help you?",]],

[r"i'm (.*) doing good",["Nice to hear that","How can I help you?:)",]],

[r"what (.*) want ?",["Make me an offer I can't refuse",]],

[r"how is weather in (.*)?",["Weather in %1 is awesome like always","Too
hot here in %1","Too cold here in %1","Never even heard about %1"]],
```

```
[r"how (.*) health(.*)",["I'm a computer program, so I'm always healthy
",]],

[r"(.*) (sports|game) ?",["I'm a very big fan of Football",]],

[r"quit",["BBye take care. See you soon :) ","It was nice talking to you.
See you soon :)"]],
]

def chat():
  print("Hi! I am a chatbot created for your service")
  chat = Chat(pairs, reflections)
  chat.converse()

#initiate the conversation
if __name__ == "__main__":
  chat()
```

5 Write a program to implement Breadth First Search Traversal

```
visited=[]
queue=[]
tree={'1':['2','3'],'2':['4','5'],'3':['6'],'4':[],'5':[],'6':[]}

def BFS(tree,visited,s):
  visited.append(s);
  queue.append(s);

  while queue:
      q=queue.pop(0);


      for k in tree[q]:
        if k not in visited:
          visited.append(k);
          queue.append(k);
  print(visited);


BFS(tree,visited,'1');
```

6 Write a program to implement Depth First Search Traversal

```
visited=[]
tree={'1':['2','3'],'2':['4','5'],'3':['6'],'4':[],'5':[],'6':[]}

def DFS(tree,visited,s):
  visited.append(s);
  print(s,end=' ');
  for k in tree[s]:
    if k not in visited:

      DFS(tree,visited,k)


DFS(tree,visited,'1');
print()
```

7 Write a program to implement Water Jug Problem

```
def water_jug_problem(jug1_cap, jug2_cap, target_amount):

    # Initialize the jugs and the possible actions
    j1 = 0
    j2 = 0
    actions = [("fill", 1), ("fill", 2), ("empty", 1), ("empty", 2), ("pour", 1, 2),
("pour", 2, 1)]

    # Create an empty set to store visited states
    visited = set()

    # Create a queue to store states to visit
    queue = [(j1, j2, [])]

    while queue:
        # Dequeue the front state from the queue
        j1, j2, seq = queue.pop(0)

        # If this state has not been visited before, mark it as visited
        if (j1, j2) not in visited:
            visited.add((j1, j2))

            # If this state matches the target amount, return the sequence
                                of actions taken to get to this state

            if j1 == target_amount:
                return seq

            # Generate all possible next states from this state

            for action in actions:
                if action[0] == "fill":
                    if action[1] == 1:
                      next_state = (jug1_cap, j2)
                    else:
                      next_state = (j1, jug2_cap)

                elif action[0] == "empty":
                    if action[1] == 1:
                       next_state = (0, j2)
                    else:
```

```
                    next_state = (j1, 0)

                else:
                    if action[1] == 1:
                        amount = min(j1, jug2_cap - j2)
                        next_state = (j1 - amount, j2 + amount)
                    else:
                        amount = min(j2, jug1_cap - j1)
                        next_state = (j1 + amount, j2 - amount)

            # Add the next state to the queue if it has not been visited before
                if next_state not in visited:
                    next_seq = seq + [action]
                    queue.append((next_state[0], next_state[1], next_seq))

        # If the queue becomes empty without finding a solution, return
None
        return None
result = water_jug_problem(4, 3, 2)
print(result)
```

8 Write a program to implement K -Nearest Neighbor algorithm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn import metrics
import seaborn as sn

data=pd.read_csv('/home/mitacsc/FG216/Breast Cancer Detection
Classification Master.csv')
print(data.shape)
data.head()

data.isnull().sum()

x=data.iloc[:,:-1].values
y=data.iloc[:,-1].values

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3,
random_state=0)
# display(x_train.shape, y_train.shape, x_test.shape, y_test.shape)

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()

x_train=sc.fit_transform(x_train)

x_test=sc.transform(x_test)

results=[]

for i in [1,2,3,4,5]:
        model=KNeighborsClassifier(n_neighbors=i,metric='minkowski',p
=2)

        model.fit(x_train,y_train)
        y_pred=model.predict(x_test)
        Accuracy_score=metrics.accuracy_score(y_test,y_pred)
        results.append(Accuracy_score)

print('KNN[minkowski]')
print('for n_neighbor=5:')
```

```
conf_mat=metrics.confusion_matrix(y_test,y_pred)
print('\n confussion matrix:',conf_mat)
print('Accuracy Score:',Accuracy_score)
print('Accuracy in percentage:',int(Accuracy_score*100),'%')
print('\n',classification_report(y_pred,y_test))
print(results)

conf_mat=pd.crosstab(y_test, y_pred, rownames=['Actual'],
colnames=['Predicted'])

sn.heatmap(conf_mat, annot=True).set(title='KNN [ minkowski,
neighbor=5 ]')

models = pd.DataFrame({ 'n_neighbors': ['1', '2','3','4','5'], 'Accuracy
Score': [results[0],results[1],results[2],results[3],results[4]]})

models.sort_values(by='Accuracy Score')
print(models.to_string(index=False))
```

9 Write a program to implement Regression algorithm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data=pd.read_csv('/home/bcslab-204/FG216/Salary_Data.csv')
print(data.head())

print(data.shape)

print(data.isnull().sum())

x=data.iloc[:,:1].values
y=data.iloc[:,1:2].values

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_st
ate=42)
model=LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(y_pred)
print(y_test)

plt.scatter(x_train,y_train,color='blue')
plt.plot(x_train,model.predict(x_train),color='red')
plt.title('SALARY VS EXPERIENCE')
plt.xlabel('Experience')
plt.ylabel('salary')
plt.show()

plt.scatter(x_train,y_train,color='blue')
plt.plot(x_train,model.predict(x_train),color='red')
plt.title('SALARY VS EXPERIENCE')
plt.xlabel('Experience')
plt.ylabel('salary')
plt.show()

from sklearn.metrics import r2_score,mean_squared_error
print('R2 score %2.f' %r2_score(y_test,y_pred))
print(x_test)
print(y_pred)
```

10 Write a program to implement Random Forest Algorithm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import classification_report
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn import tree

#Importing (Reading) Datasets
data=pd.read_csv('/home/mitacsc/FG216/Breast Cancer Detection
Classification Master.csv')

print(data.head)
x=data.iloc[:,:-1].values
y=data.iloc[:,-1].values
#Splitting the dataset into Training and Testing Dataset
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_st
ate=41)

#Preprocessing Data with StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)

#Fitting the Model Random Forest Classifier:
model=RandomForestClassifier(n_estimators=10,
criterion='entropy',random_state=0)
model.fit(x_train,y_train)
y_pred=model.predict(x_test)

#Evaluation Metrics
print('Random Forest Classifier')
conf_mat=metrics.confusion_matrix(y_test, y_pred)
print('\n Confusion Matrix : \n', conf_mat)
Accuracy_score=accuracy_score(y_test,y_pred)
print('Accuracy Score : ', Accuracy_score)
print('Accuracy in Percentage : ', int(Accuracy_score*100),'%')
print('\n',classification_report(y_pred,y_test))
```

```
conf_mat=pd.crosstab(y_test,y_pred, rownames=['Actual'], colnames=['Predicted'])

sn.heatmap(conf_mat, annot=True).set(title='Random Forest Classifier')

import pydotplus
from IPython.display import Image

# Create DOT data
dot_data = tree.export_graphviz(model.estimators_[0], out_file=None, feature_names=data.columns[:- 1], class_names=['0', '1'], filled=True, rounded=True, special_characters=True)

# Create graph from DOT data
graph = pydotplus.graph_from_dot_data(dot_data)
 # Generate image
Image(graph.create_png())
```

11 Develop a program to solve the eight queens problem. (Uninformed Search)

```
n=8
board=[[0]*n for _ in range(n)]

def check(i,j):
        for x in range(0,n):
                if board[i][x]=='Q' or board[x][j]=='Q':
                        return True

        for x in range(0,n):
                for y in range(0,n):
                        if(x+y==i+j) or (x-y==i-j):
                                if board[x][y]=='Q':
                                        return True
        return False

def put_queen(countOfQueen):
        if countOfQueen==0:
                return True

        for i in range(0,n):
                for j in  range(0,n):
                        if(not(check(i,j))) and board[i][j]!='Q':
                                board[i][j]='Q'
                                if put_queen(countOfQueen-1)==True:
                                        return True
                                board[i][j]=0
        return False
put_queen(n)
for i in board:
    print (i)
```

14 Develop a program to solve the N queens puzzle using forward checking. Show in steps how the constraints are handled. (Constraint Satisfaction Problem)

```
def is_safe(board, row, col, n):
    # Check if there is a queen in the same row to the left
    for i in range(col):
        if board[row][i] == 1:
            return False

    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check lower diagonal on left side
    for i, j in zip(range(row, n, 1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solve_nqueens(board, col, n):
    if col >= n:
        return True  # All queens are placed successfully

    for i in range(n):
        if is_safe(board, i, col, n):
            board[i][col] = 1  # Place queen
```

```
            # Recur to place the rest of the queens
            if solve_nqueens(board, col + 1, n):
                return True

            # If placing queen in the current position doesn't lead to a
solution, backtrack
            board[i][col] = 0

    return False  # If no queen can be placed in this column

def print_board(board):
    for row in board:
        print(" ".join(["Q" if x else "." for x in row]))

def main():
    n = int(input("Enter number"))
    board = [[0] * n for _ in range(n)]

    if solve_nqueens(board, 0, n):
        print("Solution:")
        print_board(board)
    else:
        print("No solution exists.")

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

15 Write a computer program to play tic-tac-toe game. (Game Theory)

```
import random


class TicTacToe:

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

    def create_board(self):
        for i in range(3):
            row = []
            for j in range(3):
                row.append('-')
            self.board.append(row)

    def get_random_first_player(self):
        return random.randint(0, 1)

    def fix_spot(self, row, col, player):
        self.board[row][col] = player

    def is_player_win(self, player):
        win = None

        n = len(self.board)

        # checking rows
        for i in range(n):
            win = True
            for j in range(n):
                if self.board[i][j] != player:
                    win = False
                    break
            if win:
                return win

        # checking columns
        for i in range(n):
            win = True
```

```
            for j in range(n):
                if self.board[j][i] != player:
                    win = False
                    break
            if win:
                return win

        # checking diagonals
        win = True
        for i in range(n):
            if self.board[i][i] != player:
                win = False
                break
        if win:
            return win

        win = True
        for i in range(n):
            if self.board[i][n - 1 - i] != player:
                win = False
                break
        if win:
            return win
        return False

        for row in self.board:
            for item in row:
                if item == '-':
                    return False
        return True

    def is_board_filled(self):
        for row in self.board:
            for item in row:
                if item == '-':
                    return False
        return True

    def swap_player_turn(self, player):
        return 'X' if player == 'O' else 'O'

    def show_board(self):
        for row in self.board:
```

```
            for item in row:
                print(item, end=" ")
            print()

    def start(self):
        self.create_board()

        player = 'X' if self.get_random_first_player() == 1 else 'O'

        while True:
            print(f"Player {player} turn")

            self.show_board()

            # taking user input
            row, col = list(map(int, input(
                    "Enter row and column numbers to fix spot:").split()))
            print()

            # fixing the spot
            self.fix_spot(row - 1, col - 1, player)

            # checking whether current player is won or not
            if self.is_player_win(player):
                print(f"Player {player} wins the game!")
                break

            # checking whether the game is draw or not
            if self.is_board_filled():
                print("Match Draw!")
                break

            # swapping the turn
            player = self.swap_player_turn(player)

        # showing the final view of board
        print()
        self.show_board()


# starting the game
tic_tac_toe = TicTacToe()
tic_tac_toe.start()
```