

1a

```
num = 12
for i in range(1,11):
    print(num,'x',i,'=',num*i)
num = 11
```

1b

```
num = 11
if num > 1:
    for i in range(2, int(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")
```

1c

```
num=int(input('enter the number'))
fact=1
for i in range(1,num+1):
    fact=fact*i
print(f"The Factorial of {num} is:{fact}")
```

2 (a) list operation

```
my_list = [1, 2, 3, 4, 5]

print("Original List:", my_list)

print("Element at index 2:", my_list[2])
# Slicing
print("Sliced List (index 1 to 3):", my_list[1:4])
# Modifying elements
my_list[2] = 6
print("Modified List:", my_list)
# Appending and extending
my_list.append(7)
print("List after appending 7:", my_list)
my_list.extend([8, 9])
print("List after extending with [8, 9]:", my_list)
# Removing elements
my_list.remove(6)
print("List after removing 6:", my_list)
popped_element = my_list.pop(2)
print(f"Popped element at index 2: {popped_element}, Updated List: {my_list}")
# Finding index of an element
index_of_5 = my_list.index(5)
print("Index of 5:", index_of_5)
# Length of the list
list_length = len(my_list)
print("Length of the list:", list_length)
# Check if an element is in the list
element_to_check = 8
if element_to_check in my_list:
    print(f"{element_to_check} is in the list.")
else:
    print(f"{element_to_check} is not in the list.")
```

2(b) List methods

```
my_list = [1, 2, 3, 4, 5]

my_list.append(6)
print("List after append(6):", my_list)

my_list.extend([7, 8])
print("List after extend([7, 8]):", my_list)

my_list.insert(2, 10)
print("List after insert(2, 10):", my_list)

my_list.remove(4)
print("List after remove(4):", my_list)

popped_element = my_list.pop(1)
print(f"Popped element at index 1: {popped_element}, Updated List: {my_list}")

index_of_3 = my_list.index(3)
print("Index of 3:", index_of_3)

count_of_6 = my_list.count(6)
print("Count of 6:", count_of_6)

my_list.reverse()
print("Reversed List:", my_list)

my_list.sort()
print("Sorted List:", my_list)

copied_list = my_list.copy()
print("Copied List:", copied_list)

my_list.clear()
print("Cleared List:", my_list)
```

3 Chabot

```
import time
def simple_chatbot(user_input):
    # Get the current time every time the function is called
    now = time.ctime()

    conversations = {
        "hi": "Hello! How can I help you?",
        "how are you": "I'm doing well, thank you. How about you?",
        "name": "I'm a chatbot. You can call me ChatPy!",
        "age": "I don't have an age. I'm just a program.",
        "bye": "Goodbye! Have a great day.",
        "python": "Python is a fantastic programming language!",
        "weather": "I'm sorry, I don't have real-time data. You can check
a weather website for updates.",
        "help": "I'm here to assist you. Ask me anything!",
        "thanks": "You're welcome! If you have more questions, feel free
to ask.",
        "what is the time now": now
    }

    user_input_lower = user_input.lower()
    response = conversations.get(user_input_lower, "I'm not sure how to
respond to that. You can ask me something else.")
    return response

# Chatbot interaction loop
print("Hello! I'm ChatPy, your friendly chatbot.")
print("You can start chatting. Type 'bye' to exit.")

while True:
    user_input = input("You: ")

    if user_input.lower() == 'bye':
        print("ChatPy: Goodbye! Have a great day.")
        break

    response = simple_chatbot(user_input)
    print("ChatPy:", response)
```

4. Write a python program to Illustrate Different Set Operations

```
set1={1,2,3,4,5}
set2={3,4,5,6,7}

union_set=set1.union(set2)
print(union_set,"union set")

intersection_set=set1.intersection(set2)
print(intersection_set,"intersection set")

difference_set1=set1.difference(set2)
print(difference_set1,"set1-set2")

symmetric_difference_set=set1.symmetric_difference(set2)
print(symmetric_difference_set,"symmetric difference set")

have_common_elements=set1.isdisjoint(set2)
print("Do set1 and set2 have any common elements?",not
have_common_elements)

set1.add(6)
print("set1 after adding element:",set1)

set1.remove(3)
print("set1 after removing element:",set1)
```

5 (A) number of times a string (s1) occurs in another string(s2)

```
def count_occurrences(main_string, substring):
    count = 0
    start_index = 0
    while start_index < len(main_string):
        index = main_string.find(substring, start_index)
        if index == -1:
            break
        count += 1
        start_index = index + 1
    return count

main_string = "ababababab ab ab"
substring = "ab"
result = count_occurrences(main_string, substring)
print(f"The substring '{substring}' occurs {result} times in the main string.")
print(main_string.count(substring))
```

5 (b)

```
sample_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
key_to_check = 'b'
if key_to_check in sample_dict:
    print(f'The key "{key_to_check}" exists in the dictionary.')

print("Traversing the dictionary:")
for key, value in sample_dict.items():
    print(f'Key: {key}, Value: {value}')

keys_list = list(sample_dict.keys())
values_list = list(sample_dict.values())
items_list = list(sample_dict.items())
print("\nUsing dictionary methods:")
print(f'Keys: {keys_list}')
print(f'Values: {values_list}')
print(f'Items: {items_list}')
```

Part B

1. Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem

```

def water_jug_dfs(capacity_x, capacity_y, target):
    stack = [(0, 0, [])]
    visited_states = set()

    while stack:
        x, y, path = stack.pop()

        if (x, y) in visited_states:
            continue

        visited_states.add((x, y))

        if x == target or y == target:
            return path + [(x, y)]

        # Define possible jug operations
        operations = [
            ("fill_x", capacity_x, y),
            ("fill_y", x, capacity_y),
            ("empty_x", 0, y),
            ("empty_y", x, 0),
            ("pour_x_to_y", max(0, x - (capacity_y - y)), min(capacity_y,
y + x)),
            ("pour_y_to_x", min(capacity_x, x + y), max(0, y -
(capacity_x - x))),
        ]

        for operation, new_x, new_y in operations:
            if 0 <= new_x <= capacity_x and 0 <= new_y <= capacity_y:
                stack.append((new_x, new_y, path + [(x, y, operation)]))

    return None

# Example usage:
capacity_x = 4
capacity_y = 3
target = 2

solution_path = water_jug_dfs(capacity_x, capacity_y, target)
if solution_path:
    print("Solution found:")
    for state in solution_path:
        print(f"({state[0]}, {state[1]})")
else:
    print("No solution found.")

```

2. Implement and Demonstrate Breadth First Search Algorithm on any AI problem

BFS Tree Traversal

```
tree = {
    1: [2, 9, 10],
    2: [3, 4],
    3: [],
    4: [5, 6, 7],
    5: [8],
    6: [],
    7: [],
    8: [],
    9: [],
    10: []
}

def breadth_first_search(tree, start):
    q = [start]
    visited = []

    while q:
        print("Before:", q)
        node = q.pop(0)
        visited.append(node)

        for child in tree[node]:
            if child not in visited and child not in q:
                q.append(child)
        print("After:", q)

    return visited

# Run BFS from node 1
result = breadth_first_search(tree, 1)
print("BFS Traversal:", result)
```

4. Solve 8-Queens Problem with suitable assumptions

```
print("Enter the number of queens")
N = int(input())
board = [[0]*N for _ in range(N)]
def attack(i, j):
    for k in range(0,N):
        if board[i][k] == 1 or board[k][j] == 1:
            return True
    for k in range(0,N):
        for l in range(0,N):
            if (k + l == i + j) or (k - l == i - j):
                if board[k][l] == 1:
                    return True
    return False

def N_queens(n):
    if n == 0:
        return True
    for i in range(0,N):
        for j in range(0,N):
            if (not(attack(i, j))) and (board[i][j] != 1):
                board[i][j] = 1
                if N_queens(n - 1)==True:
                    return True
                board[i][j] = 0
    return False

if N_queens(N):
    for i in board:
        print(i)
else:
    print("No solution exists.")
```

5. Implementation of TSP using heuristic approach

```
from itertools import permutations
def calculate_total_distance(tour, distances):
    total_distance = 0
    for i in range(len(tour) - 1):
        total_distance += distances[tour[i]][tour[i + 1]]
    total_distance += distances[tour[-1]][tour[0]] # Return to the
starting city
    return total_distance

def traveling_salesman_bruteforce(distances):
    cities = range(len(distances))
    min_distance = float('inf')
    optimal_tour = None

    for tour in permutations(cities):
        distance = calculate_total_distance(tour, distances)
        if distance < min_distance:
            min_distance = distance
            optimal_tour = tour
    return optimal_tour, min_distance

distances_matrix = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]

optimal_tour, min_distance =
traveling_salesman_bruteforce(distances_matrix)
print("Optimal Tour:", optimal_tour)
print("Minimum Distance:", min_distance)
```

8 Implement any Game and demonstrate the Game playing strategies

```
board = [" " for x in range(9)]
def print_board():
    row1 = "| {} | {} | {} |".format(board[0], board[1], board[2])
    row2 = "| {} | {} | {} |".format(board[3], board[4], board[5])
    row3 = "| {} | {} | {} |".format(board[6], board[7], board[8])
    print()
    print(row1)
    print(row2)
    print(row3)
    print()

def player_move(icon):
    if icon == "X":
        number = 1
    elif icon == "O":
        number = 2
    print("Your turn player {}".format(number))
    choice = int(input("Enter your move (1-9): ").strip())
    if board[choice - 1] == " ":
        board[choice - 1] = icon
    else:
        print()
        print("That space is taken! Try again.")

def is_victory(icon):
    if (board[0] == icon and board[1] == icon and board[2] == icon) or \
        (board[3] == icon and board[4] == icon and board[5] == icon) or \
        (board[6] == icon and board[7] == icon and board[8] == icon) or \
        (board[0] == icon and board[3] == icon and board[6] == icon) or \
        (board[1] == icon and board[4] == icon and board[7] == icon) or \
        (board[2] == icon and board[5] == icon and board[8] == icon) or \
        (board[0] == icon and board[4] == icon and board[8] == icon) or \
        (board[2] == icon and board[4] == icon and board[6] == icon):
        return True
    else:
        return False

def is_draw():
    if " " not in board:
        return True
    else:
```

```
    return False
```

```
while True:
    print_board()
    player_move("X")
    print_board()
    if is_victory("X"):
        print("X wins! Congratulations!")
        break
    elif is_draw():
        print("It's a draw!")
        break
    player_move("O")
    if is_victory("O"):
        print_board()
        print("O wins! Congratulations!")
        break
    elif is_draw():
        print("It's a draw!")
        break
```