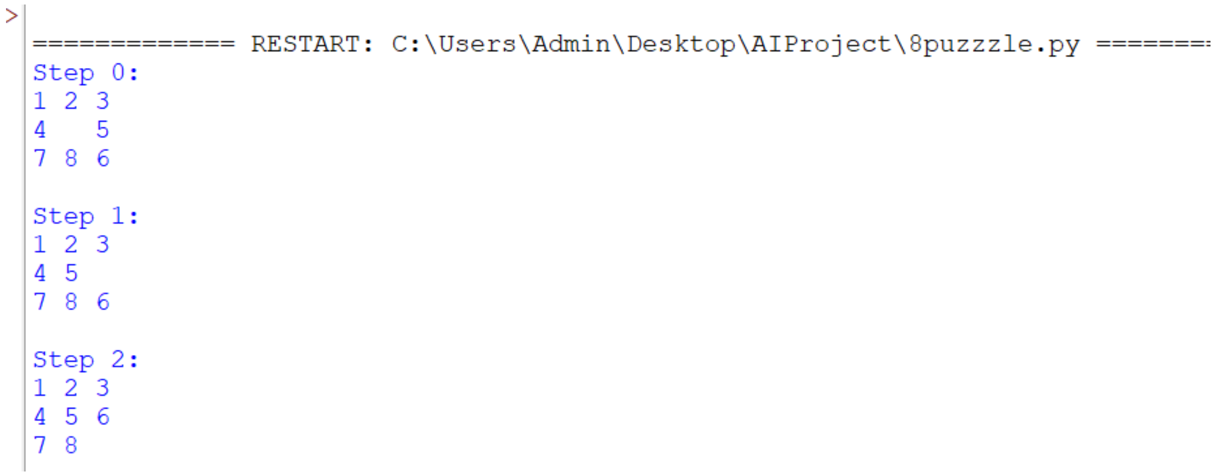
DAY-1 Experiments

1 Write the python program to solve 8-Puzzle problem

Program:

```
n=3
a=[[1,7,4],[2,8,6],[5,3,0]]
k=1
for i in range(0,3):
    for j in range(0,3):
        if(a[i][j]!=k):
            a[i][j]=k
        k=k+1
a[2][2]=0
print(a)
```

Sample output:



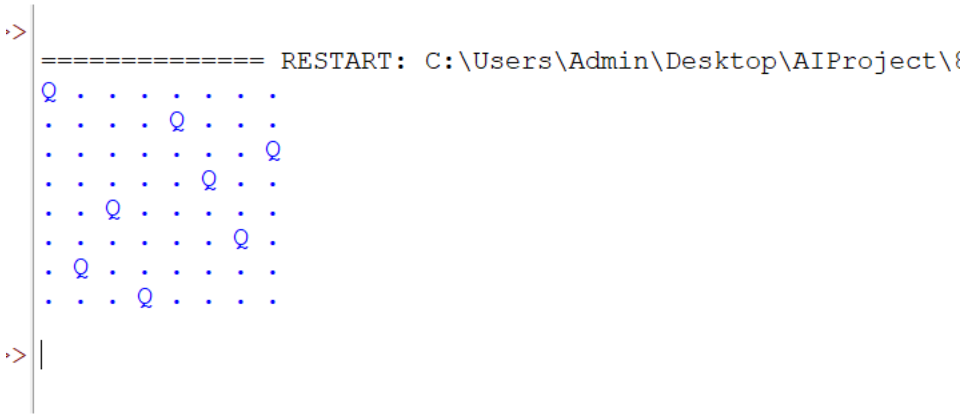
```
>
  ============= RESTART: C:\Users\Admin\Desktop\AIProject\8puzzzle.py =======
  Step 0:
  1 2 3
  4   5
  7 8 6

  Step 1:
  1 2 3
  4 5
  7 8 6

  Step 2:
  1 2 3
  4 5 6
  7 8
```

2 Write the python program to solve 8-Queen problem

Program:

```
def solve(n):
    board = [-1] * n
    def is_safe(r, c):
        return all(board[i] != c and abs(board[i] - c) != r - i for i in range(r))
    def place_queens(r):
        if r == n:
            print_board()
            return True
        for c in range(n):
            if is_safe(r, c):
                board[r] = c
                if place_queens(r + 1):
                    return True
        return False
    def print_board():
        for r in range(n):
            print(" ".join("Q" if board[r] == c else "." for c in range(n)))
        print()
    place_queens(0)
solve(8)
```

Sample output:



```
>
============== RESTART: C:\Users\Admin\Desktop\AIProject\
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .

>
```

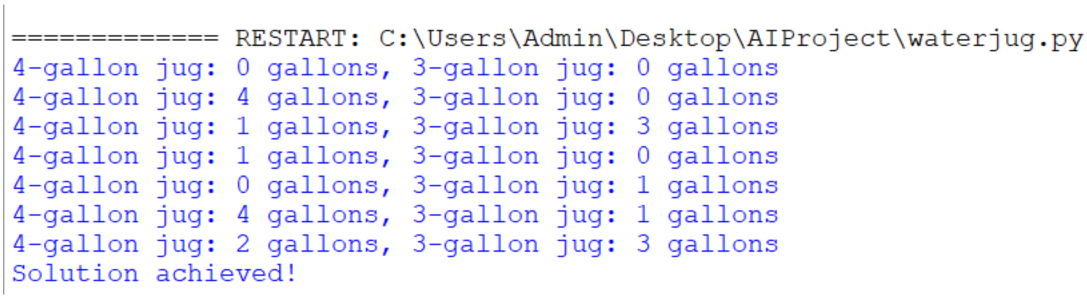
3 Write the python program for Water Jug Problem

Program:

```
from collections import deque
MAX_JUG_4 = 4
MAX_JUG_3 = 3
GOAL = 2
def print_solution(path):
    for step in path:
        print(f"4-gallon jug: {step[0]} gallons, 3-gallon jug: {step[1]} gallons")
    print("Solution achieved!\n")
```

```
def bfs():
    queue = deque([((0, 0), [])])
    visited = set()
    while queue:
        (jug4, jug3), path = queue.popleft()
        if jug4 == GOAL:
            print_solution(path + [(jug4, jug3)])
            return True
        if (jug4, jug3) in visited:
            continue
        visited.add((jug4, jug3))
        next_states = [
            (MAX_JUG_4, jug3),
            (jug4, MAX_JUG_3),
            (0, jug3),
            (jug4, 0),
            (jug4 - min(jug4, MAX_JUG_3 - jug3), jug3 + min(jug4, MAX_JUG_3 - jug3)),
            (jug4 + min(jug3, MAX_JUG_4 - jug4), jug3 - min(jug3, MAX_JUG_4 - jug4))
        ]
        for state in next_states:
            queue.append((state, path + [(jug4, jug3)]))
    print("No solution found.")
    return False
bfs()
```

Sample output:



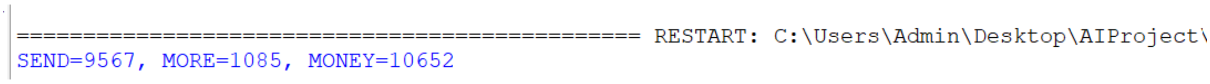
```
============= RESTART: C:\Users\Admin\Desktop\AIProject\waterjug.py
4-gallon jug: 0 gallons, 3-gallon jug: 0 gallons
4-gallon jug: 4 gallons, 3-gallon jug: 0 gallons
4-gallon jug: 1 gallons, 3-gallon jug: 3 gallons
4-gallon jug: 1 gallons, 3-gallon jug: 0 gallons
4-gallon jug: 0 gallons, 3-gallon jug: 1 gallons
4-gallon jug: 4 gallons, 3-gallon jug: 1 gallons
4-gallon jug: 2 gallons, 3-gallon jug: 3 gallons
Solution achieved!
```

4 Write the python program for Cript-Arithmetic problem

Program:

```
import itertools
def solve_cryptarithmetic():
    for perm in itertools.permutations(range(10),8):
        s, e, n, d, m, o, r, y = perm
        if s and m and (s * 1000 + e * 100 + n * 10 + d) + (
            m * 1000 + o * 100 + r * 10 + e) == m * 10000 + o * 1000 + n * 100 + e * 10 + y:
            print(f"SEND={s}{e}{n}{d}, MORE={m}{o}{r}{e}, MONEY={m}{o}{n}{e}{y}")
            break
solve_cryptarithmetic()
```

Sample output:



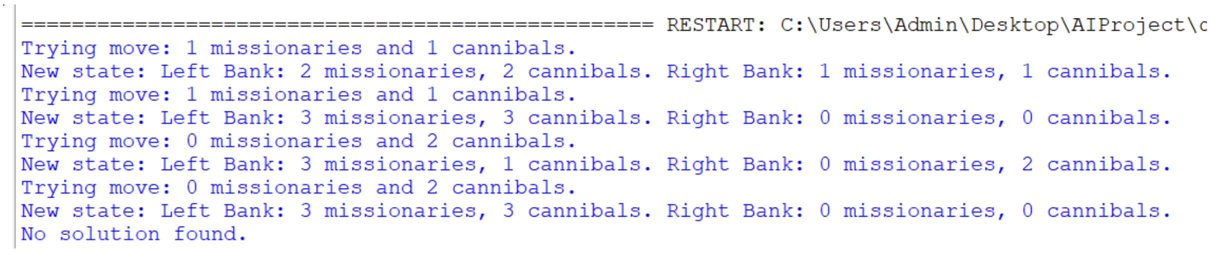
```
============================================== RESTART: C:\Users\Admin\Desktop\AIProject\
SEND=9567, MORE=1085, MONEY=10652
```

5 Write the python program for Missionaries Cannibal problem

Program:

```
def is_valid(m_left, c_left, m_right, c_right):
    return not ((m_left > 0 and m_left < c_left) or (m_right > 0 and m_right < c_right))
def solve(m_left, c_left, boat_on_left, visited, path):
    if m_left == 0 and c_left == 0:
        print("Solution found!")
        for move in path:
            print(f"Move {move[0]} missionaries and {move[1]} cannibals.")
        return True
    state = (m_left, c_left, boat_on_left)
    if state in visited: return False
    visited.add(state)
    boat_moves = [(2, 0), (1, 1), (0, 2)]
    for m_move, c_move in boat_moves:
        new_m_left, new_c_left = (m_left - m_move, c_left - c_move) if boat_on_left else (m_left + m_move, c_left + c_move)
        new_m_right, new_c_right = 3 - new_m_left, 3 - new_c_left
        new_boat_on_left = not boat_on_left
    if new_m_left >= 0 and new_c_left >= 0 and new_m_right >= 0 and new_c_right >= 0 and is_valid(new_m_left, new_c_left, new_m_right, new_c_right):
      print(f"Trying move: {m_move} missionaries and {c_move} cannibals.")
      print(f"New state: Left Bank: {new_m_left} missionaries, {new_c_left} cannibals. "
         f"Right Bank: {new_m_right} missionaries, {new_c_right} cannibals.")
      if solve(new_m_left, new_c_left, new_boat_on_left, visited, path + [(m_move, c_move)]):
        return True
  return False
visited, path = set(), []
if not solve(3, 3, True, visited, path):
  print("No solution found.")
```

Sample output:



```
==================================================== RESTART: C:\Users\Admin\Desktop\AIProject\c
Trying move: 1 missionaries and 1 cannibals.
New state: Left Bank: 2 missionaries, 2 cannibals. Right Bank: 1 missionaries, 1 cannibals.
Trying move: 1 missionaries and 1 cannibals.
New state: Left Bank: 3 missionaries, 3 cannibals. Right Bank: 0 missionaries, 0 cannibals.
Trying move: 0 missionaries and 2 cannibals.
New state: Left Bank: 3 missionaries, 1 cannibals. Right Bank: 0 missionaries, 2 cannibals.
Trying move: 0 missionaries and 2 cannibals.
New state: Left Bank: 3 missionaries, 3 cannibals. Right Bank: 0 missionaries, 0 cannibals.
No solution found.
```

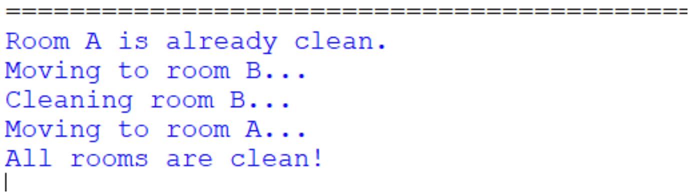
6 Write the python program for Vacuum Cleaner problem

Program:

```
def vacuum_cleaner(room_status, start_position):
  position = start_position
  while 'dirty' in room_status.values():
    if room_status[position] == 'dirty':
      print(f"Cleaning room {position}...")
      room_status[position] = 'clean'
    else:
      print(f"Room {position} is already clean.")
    position = 'B' if position == 'A' else 'A'
    print(f"Moving to room {position}...")
  print("All rooms are clean!")
room_status = {'A': 'clean', 'B': 'dirty'}
start_position = 'A'
```

vacuum_cleaner(room_status, start_position)

Sample output:



```
==========================================
Room A is already clean.
Moving to room B...
Cleaning room B...
Moving to room A...
All rooms are clean!
```