1. **Two water jug problem**

**Input:**

a = int(input("Enter capacity of Jug A: "))

b = int(input("Enter capacity of Jug B: "))

ai = int(input("Enter initial water in Jug A: "))

bi = int(input("Enter initial water in Jug B: "))

af = int(input("Enter final water level required in Jug A: "))

bf = int(input("Enter final water level required in Jug B: "))

print("\nList of operations:")

print("1: Fill jug A completely")

print("2: Fill jug B completely")

print("3: Empty jug A")

print("4: Empty jug B")

print("5: Pour water from jug A to B until full")

print("6: Pour water from jug B to A until full")

print("7: Pour all water from jug A to B")

print("8: Pour all water from jug B to A")

while (ai != af or bi != bf):

    print(f"\nCurrent state: Jug A = {ai}, Jug B = {bi}")

    op = int(input("Choose an operation: "))

    if op == 1:

        ai = a

    elif op == 2:

        bi = b

    elif op == 3:

        ai = 0

    elif op == 4:

        bi = 0

    elif op == 5:

        if (b - bi > ai):

            bi = bi + ai

            ai = 0

        else:

            ai = ai - (b - bi)

            bi = b

    elif op == 6:

        if (a - ai > bi):

            ai = ai + bi

            bi = 0

        else:

            bi = bi - (a - ai)

            ai = a

    elif op == 7:

        bi = bi + ai

        ai = 0

    elif op == 8:

        ai = ai + bi

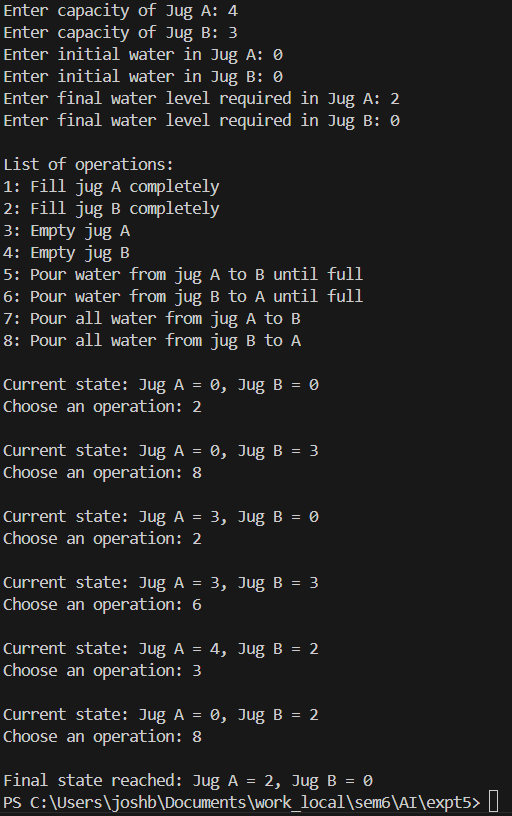
        bi = 0

    else:

        print("Invalid operation. Please choose a number between 1 and 8.")

print(f"\nFinal state reached: Jug A = {ai}, Jug B = {bi}")

**Output:**



1. **Three water jug problem**

**Input:**

#3 water jug problem

x = int(input("Enter the maximum capacity of Jug A: "))

y = int(input("Enter the maximum capacity of Jug B: "))

z = int(input("Enter the maximum capacity of Jug C: "))

goalx = int(input("Enter the goal state in Jug A: "))

goaly = int(input("Enter the goal state in Jug B: "))

visited = {}

path = []

def all\_states(state):

    jugA, jugB, jugC = state

    if jugA == goalx and jugB == goaly:

        path.append(state)

        return True

    if state in visited:

        return False

    visited[state] = True

    if jugA > 0:

        if jugA + jugB <= y:

            if all\_states((0, jugA + jugB, jugC)):

                path.append(state)

                return True

        else:

            if all\_states((jugA - (y - jugB), y, jugC)):

                path.append(state)

                return True

        if jugA + jugC <= z:

            if all\_states((0, jugB, jugA + jugC)):

                path.append(state)

                return True

        else:

            if all\_states((jugA - (z - jugC), jugB, z)):

                path.append(state)

                return True

    if jugB > 0:

        if jugA + jugB <= x:

            if all\_states((jugA + jugB, 0, jugC)):

                path.append(state)

                return True

        else:

            if all\_states((x, jugB - (x - jugA), jugC)):

                path.append(state)

                return True

        if jugB + jugC <= z:

            if all\_states((jugA, 0, jugB + jugC)):

                path.append(state)

                return True

        else:

            if all\_states((jugA, jugB - (z - jugC), z)):

                path.append(state)

                return True

    if jugC > 0:

        if jugA + jugC <= x:

            if all\_states((jugA + jugC, jugB, 0)):

                path.append(state)

                return True

        else:

            if all\_states((x, jugB, jugC - (x - jugA))):

                path.append(state)

                return True

        if jugB + jugC <= y:

            if all\_states((jugA, jugB + jugC, 0)):

                path.append(state)

                return True

        else:

            if all\_states((jugA, y, jugC - (y - jugB))):

                path.append(state)

                return True

    return False

initial\_state = (x, 0, 0)

print("\nStarting computation...\n")

if all\_states(initial\_state):

    path.reverse()

    for state in path:

        print(state)

else:

    print("No solution found.")

**Output:**

