22/12/23

Date __/__/__
Page ____

LAB - 5

VACUUM CLEANER

Consider a grid of m×n. This is the configuration of the room. Clean status is given as dirty - 1, clean - 0.

Algorithm : (Grid)

- Cleaning of dirty cell :
  → Consider a room.
  → For each row of the room iterate from left to right if its even row & vice versa.
  → If floor[i][j] == 1, then the cell is dirty. Make floor[i][j] = 0 to make it clean.

- Check if all the cells of a room are clean.
  → if all cells of a room are clean return true. So that it can proceed to next room
  → if any one cell is atleast is dirty return false.

- Main function :
  → Input the number of rows and columns for each room. Enter clean/dirty status,
  → Pass room 1 to Cleaning().
  → Once all the cells are clean of both room, check it by passing to check_status() function.
  → If ~~the~~ all the rows are clean then ~~move~~ end the process else call the clean() function again

Algorithm: (For single cell room)

1. Clean-room (room_name, isdirty)
   → It will check if the room is dirty : if so it updates the status to 0 (clean = 0, dirty = 1).
   → If it is already clean, the status is retained

2. Main function.
   → Input room status of both room.
   → Pass room1 first followed by room2 to clean.
   → Once both the rooms are clean, call the clean-room function to check if it is dirty again.
   → If it is clean ~~return~~ to ~~room~~ exit the process

Code:

```
def clean_room (room_name, is_dirty):
    if is_dirty:
        print (f"Cleaning {room_name} (Room was
                                        dirty)")
        print (f"{room_name} is now clean")
        return 0
    else :
        print (f"{room_name} is already clean")
        return 0

def main() :
    rooms = ["Room1", "Room2"]
    row_status = []
    for room in rooms :
        status = int(input (f"Enter Clean Status
                    for {room} (1 for dirty,
                                        0 for clean): "))
    room_status.append ((room, status))
```

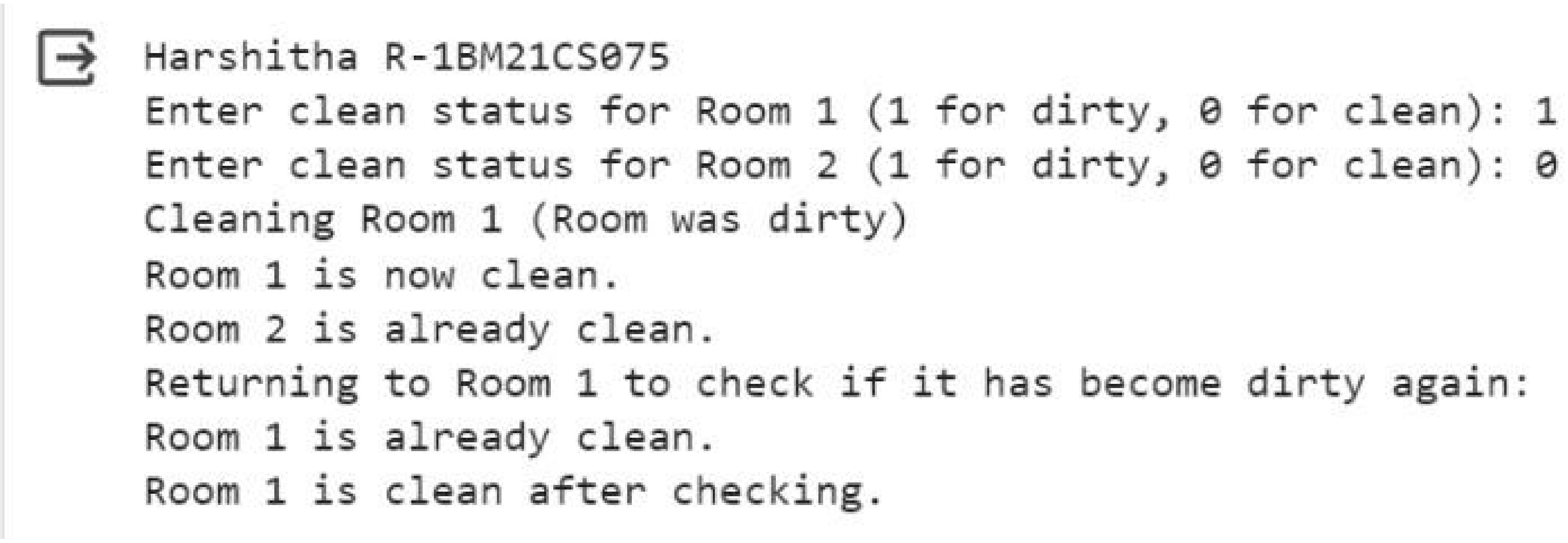
```
for i, (room, status) in enumerate(room_statuses
    room_statuses[i] = (room, clean_room
                         (room, status))

print(f"Returning to {rooms[0]} to
        check if has become dirty again")
room_statuses[0] = (rooms[0], clean_room
                     (rooms[0], room_statuses[0]

print(f"{rooms[0]} is {'dirty' if
        room_statuses[0][1] else 'clean'}
        after checking")
```

[illegible]

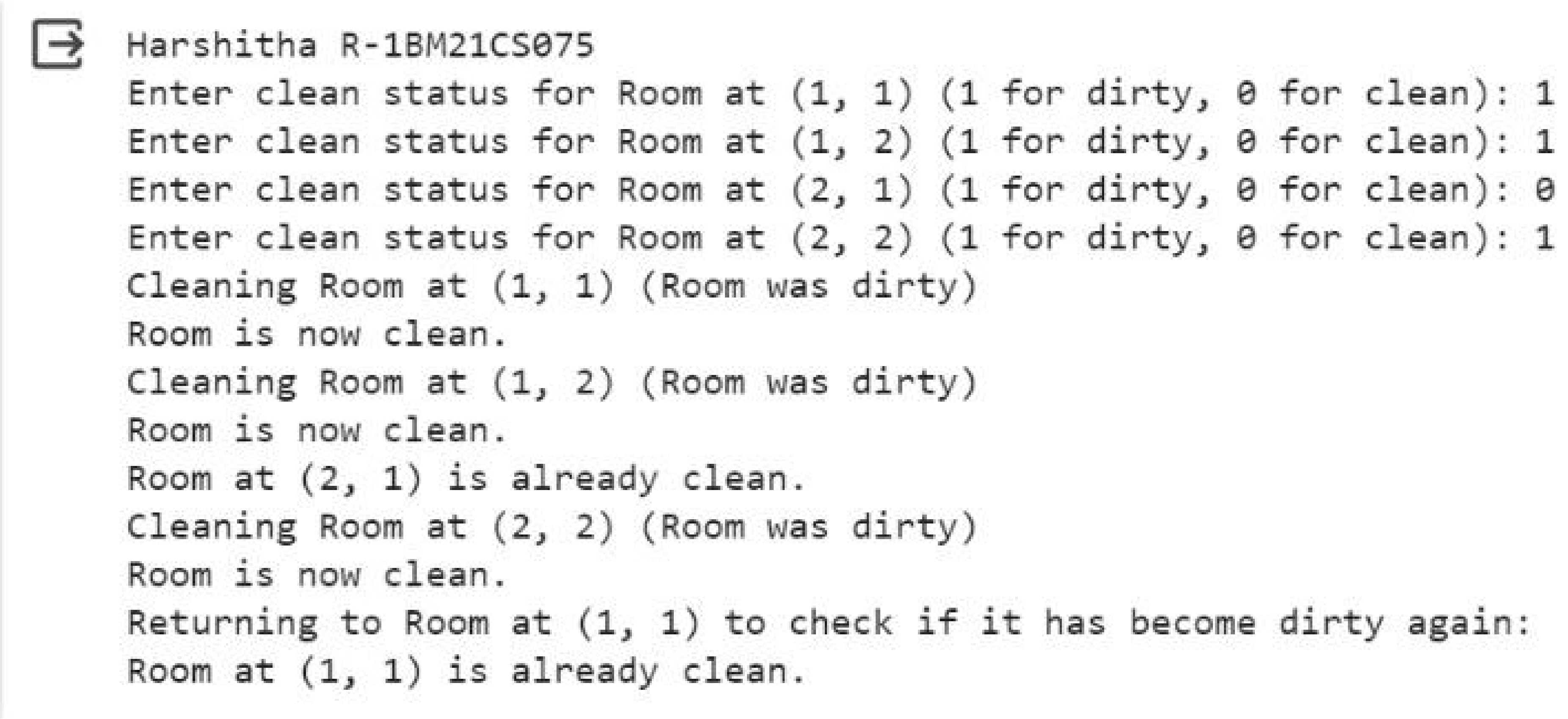
4 rooms:

```
def clean_room(floor, room_row, room_col):
    if ~~f~~ floor[room_row][room_col] == 1:
        print~~f~~("Room is Dirty")
        print("Cleaning Room")
        floor[room row][room_col] = 0
    else:
        print("Room is already clean")

def main():
    rows = 2
    cols = 2
    floor = [[0,0], [0,0]]

    for i in range(rows)
        for j in range(cols)
        status = int (input (f"Enter clean status
                                  for Room"))
        floor[i][j] = status

for i, in range (rows):
    for j in range (cols);
        clean_room (floor, i, j)

# check if the initial room is dirty again
    clean_room (floor, 0, 0)
```

[illegible]



```
Harshitha R-1BM21CS075
Enter clean status for Room 1 (1 for dirty, 0 for clean): 1
Enter clean status for Room 2 (1 for dirty, 0 for clean): 0
Cleaning Room 1 (Room was dirty)
Room 1 is now clean.
Room 2 is already clean.
Returning to Room 1 to check if it has become dirty again:
Room 1 is already clean.
Room 1 is clean after checking.
```



```
Harshitha R-1BM21CS075
Enter clean status for Room at (1, 1) (1 for dirty, 0 for clean): 1
Enter clean status for Room at (1, 2) (1 for dirty, 0 for clean): 1
Enter clean status for Room at (2, 1) (1 for dirty, 0 for clean): 0
Enter clean status for Room at (2, 2) (1 for dirty, 0 for clean): 1
Cleaning Room at (1, 1) (Room was dirty)
Room is now clean.
Cleaning Room at (1, 2) (Room was dirty)
Room is now clean.
Room at (2, 1) is already clean.
Cleaning Room at (2, 2) (Room was dirty)
Room is now clean.
Returning to Room at (1, 1) to check if it has become dirty again:
Room at (1, 1) is already clean.
```