SE CASE STUDY
TASK 4

Team:

Dhanvin S	*PES1UG20CS125*
Ghanashyam Mahesh Bhat	*PES1UG20CS153*
Jeevan R	*PES1UG20CS179*

Unit Test Code:

The below mentioned code has been modified for conducting manual tests. The manual input will be replaced by the sensor inputs in the end product.

```
import serial
import time
import schedule
import json

def main_func():

    # arduino = serial.Serial('com5', 115200)
    # print('Established serial connection to Arduino')
    # arduino_data = arduino.readline()

    # decoded_values = str(
    #     arduino_data[0:len(arduino_data)].decode("utf-8"))
    # list_values = decoded_values.split(':')

    # for item in list_values:
    #     list_in_floats.append(item)

    # print(f'Collected readings from Arduino: {list_in_floats}')
    # for i in beacons.keys():
    #     if(i in list_in_floats[1]):
    #         beacons[i] = int(list_in_floats[0])
    # print("beacons:", beacons)
    l = ['U1\r\n', 'U2\r\n', 'U3\r\n']
    temp = []
    for i in range(3):
        temp.append(int(input()))
    global beacons
    x = 0
    for i in beacons.keys():
        beacons[i] = temp[x]
        x += 1
```

```
    temp_max = max(temp)
    res = ""
    for i in beacons.keys():
        if temp_max==beacons[i]:
            res = i
    source = -1
    # temp = max(beacons.values())
    # print(temp)
    # res = [key for key in beacons if beacons[key] == temp]
    # # print(res[0])

    # if(res[0] == l[0]):
    #     source = 1
    # elif(res[0] == l[1]):
    #     source = 2
    # elif(res[0] == l[2]):
    #     source = 3
    if(res == l[0]):
        source = 1
    elif(res == l[1]):
        source = 2
    elif(res == l[2]):
        source = 3


    # db.collection("Person").document("P-1").update({"Room": source})
    print("Room number: ",source)

    # arduino_data = 0
    # list_in_floats.clear()
    # list_values.clear()
    # arduino.close()
    print('Connection closed')
    print('<------------------------------>')

# ----------------------------------------Main
Code-----------------------------------
# Declare variables to be used
list_values = []
list_in_floats = []
```

```
beacons = {"U1\r\n": -10000, "U2\r\n": -10000, "U3\r\n": -10000}

print('Program started')

# Setting up the Arduino
schedule.every(0.5).seconds.do(main_func)

while True:
   schedule.run_pending()
   time.sleep(0.5)
```

Problem Statement – 1: Unit Testing

A unit is the smallest block of code that functions individually. The first level of testing is Unit testing and this problem statement is geared towards the same.

1. Discuss with your teammates and demarcate units in your code base [Note: discuss why the code snippet you have chosen can be classified as a unit]

   The above mentioned code block makes up a single unit as it is independent of all other part of the project and it can be classified as a component of the product (As we are using component based software development architecture)

2. Develop test cases for both valid and invalid data
   We are developing different type of testcases of the given code block. The code expects three integers as input in each iteration.

   Positive test cases:
   5 8 7

7 6 3
5 8 9
5 4 1
1 1 3
2 2 2
Negative test cases:
3.0 5.3 8
'3m' 8 's'

3. Ideate how you could further modularize larger blocks of code into compact units with your teammates.

The given code block is mainly for taking the sensor input and calculating the room in which the user is residing in. The above code can be further modularized into compact units by splitting the code and making it into smaller functions, i.e a function to take the input from the sensor and a separate function to calculate the room.

Thus the above code can be split further into smaller unit, thus increasing the chance of reusing and decreasing the probability of having bugs (since, with the decrease in the size of unit, it is easier to test, analyze and solve the problems)

Problem Statement – 2: Dynamic Testing

Dynamic testing involves execution of your code to analyze errors found during execution. Some common techniques are Boundary Value Analysis and Mutation Testing.

-Problem Statement – 2.a: Boundary Value Analysis

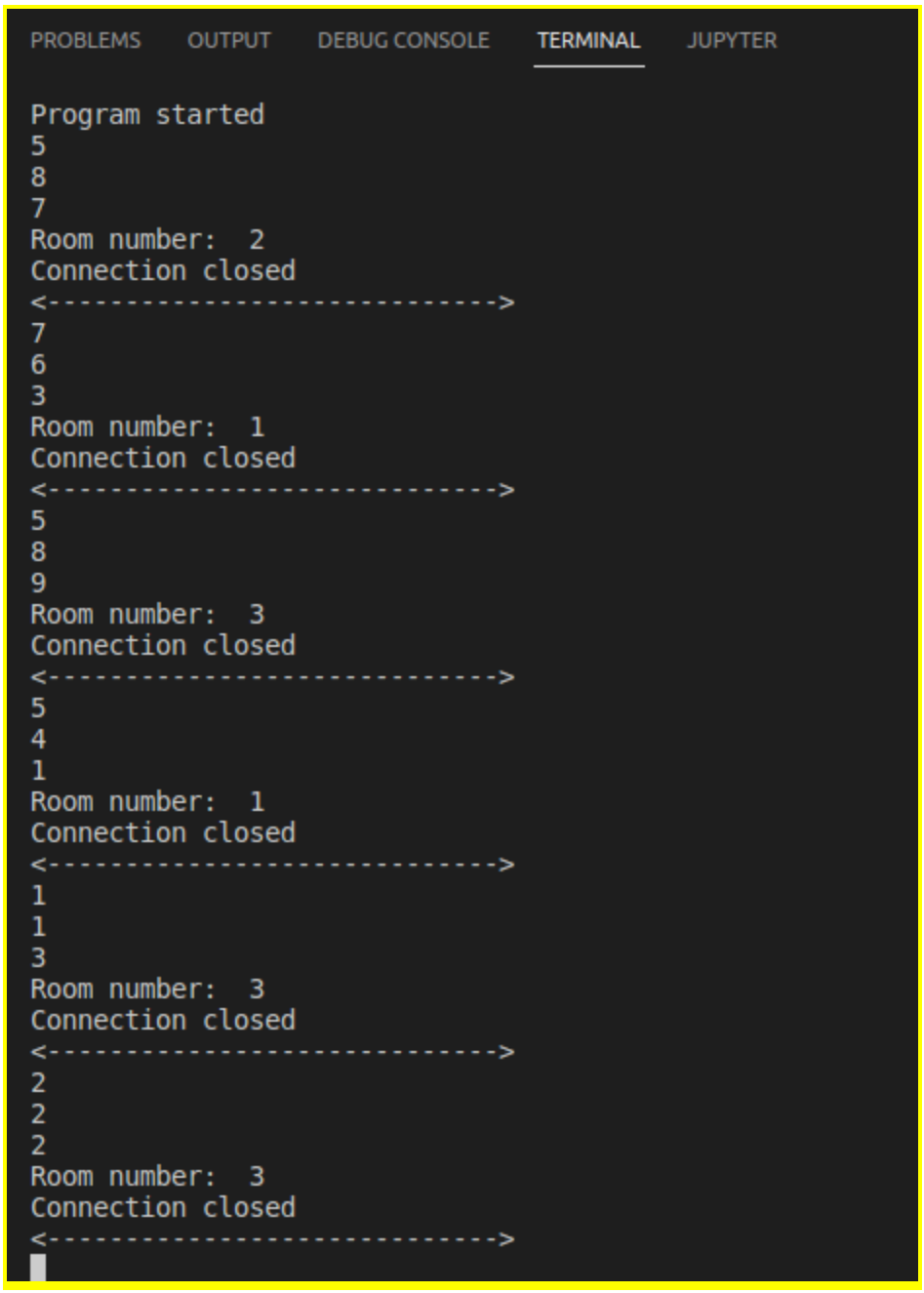
When it comes to finding errors in your code base, they are often found at locations where a condition is being tested. Due to this, developers often use Boundary Value tests to reduce defect density.

• How would you define a boundary test? Note: Simple relational conditions are a basic example

With respect to our project, we are finding the max value from the input and giving the corresponding room number as output (for determining the room in which the employee is in).
Therefore there is no specific boundary value condition in the code base, but we are testing for different input types and different input values.

• Build your boundary test cases and execute them

Since the test cases have already been generated in the previous step, we use that to test the code after running it.
The output of the program for the given input looks like this:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
Program started
5
8
7
Room number: 2
Connection closed
<------------------------------>
7
6
3
Room number: 1
Connection closed
<------------------------------>
5
8
9
Room number: 3
Connection closed
<------------------------------>
5
4
1
Room number: 1
Connection closed
<------------------------------>
1
1
3
Room number: 3
Connection closed
<------------------------------>
2
2
2
Room number: 3
Connection closed
<------------------------------>

```
gb@Ubuntu-PC:~$ /bin/python3 /home/gb/tracking.py
Program started
3.0
Traceback (most recent call last):
  File "/home/gb/tracking.py", line 80, in <module>
    schedule.run_pending()
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 780, in run_pending
    default_scheduler.run_pending()
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 100, in run_pending
    self._run_job(job)
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 172, in _run_job
    ret = job.run()
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 661, in run
    ret = self.job_func()
  File "/home/gb/tracking.py", line 27, in main_func
    temp.append(int(input()))
ValueError: invalid literal for int() with base 10: '3.0'
gb@Ubuntu-PC:~$
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
  File "/home/gb/tracking.py", line 27, in main_func
    temp.append(int(input()))
ValueError: invalid literal for int() with base 10: '3.0'
gb@Ubuntu-PC:~$ /bin/python3 /home/gb/tracking.py
Program started
Hello
Traceback (most recent call last):
  File "/home/gb/tracking.py", line 80, in <module>
    schedule.run_pending()
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 780, in run_pending
    default_scheduler.run_pending()
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 100, in run_pending
    self._run_job(job)
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 172, in _run_job
    ret = job.run()
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 661, in run
    ret = self.job_func()
  File "/home/gb/tracking.py", line 27, in main_func
    temp.append(int(input()))
ValueError: invalid literal for int() with base 10: 'Hello'
gb@Ubuntu-PC:~$ /bin/python3 /home/gb/tracking.py
Program started
1/0
Traceback (most recent call last):
  File "/home/gb/tracking.py", line 80, in <module>
    schedule.run_pending()
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 780, in run_pending
    default_scheduler.run_pending()
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 100, in run_pending
    self._run_job(job)
  File "                                    n3.10/site-packages/schedule/__init__.py", line 172, in _run_job
    ret
  File "/home/gb/.local/lib/python3.10/site-packages/schedule/__init__.py", line 661, in run
    ret = self.job_func()
  File "/home/gb/tracking.py", line 27, in main_func
    temp.append(int(input()))
ValueError: invalid literal for int() with base 10: '1/0'
gb@Ubuntu-PC:~$
```

Open file in editor (ctrl + click)

We can conclude from the test that the program gives valid output for the integer input, but it crashes when the input type is different from the integer.
This bug has to be handled to prevent failing of the program.

```
while True:
    e = "ERROR"
    try:
        schedule.run_pending()
        time.sleep(0.5)
    except:
        print(e)
```

The exception can be handled by making the above changes to the code block so that the crashing of the program can be prevented and the wrong input can be handled effectively.

```
gb@Ubuntu-PC:~$ /bin/python3 /home/gb/tracking.py
Program started
5.0
ERROR
3
2
6
Room number:  3
Connection closed
<---------------------------->
5
"hello"
ERROR
5
9
-3
Room number:  2
Connection closed
<---------------------------->
```

-Problem Statement – 2.b: Mutation Testing

1. Using your isolated units from the first problem statement, ideate with your teammates on how to mutate the code

For developing 3 different kind of mutation, we chose one value mutation, one decision mutation and one statement mutation.

2. Develop at least 3 mutants of the functioning code and test all 4 code bases using the test case from the first problem statement

1. Value mutation:

```
list_values = []
list_in_floats = []
beacons = {"U1\r\n": -10000, "U5\r\n": -10000, "U8\r\n": -10000}
```

In line 77 of the code, we changed the room names as a part of value mutation.

```
gb@Ubuntu-PC:~$ /bin/python3 /home/gb/tracking.py
Program started
5
8
7
Room number:  -1
Connection closed
<---------------------------->
7
6
3
Room number:  1
Connection closed
<---------------------------->
5
8
9
Room number:  -1
Connection closed
<---------------------------->
```

We can see that, since the value of Room 2 and Room 3 are mutated, we get invalid output when the output is supposed to be Room 2 or Room 3, but for Room 1, the output is what we had expected

2. Decision mutation

```
if(res != l[0]):
    source = 1
elif(res != l[1]):
    source = 2
```

```
   elif(res == l[2]):
        source = 3
```

The code block is mutated in line 50 and 52, and the condition is inverted.

```
gb@Ubuntu-PC:~$ /bin/python3 /home/gb/tracking.py
Program started
5
8
7
Room number:  1
Connection closed
<---------------------------->
7
6
3
Room number:  2
Connection closed
<---------------------------->
5
8
9
Room number:  1
Connection closed
<---------------------------->
```

The room with the maximum value had to be selected, but the mutation in the condition made the code to behave differently, and hence we are getting wrong outputs.

3. Statement Mutation

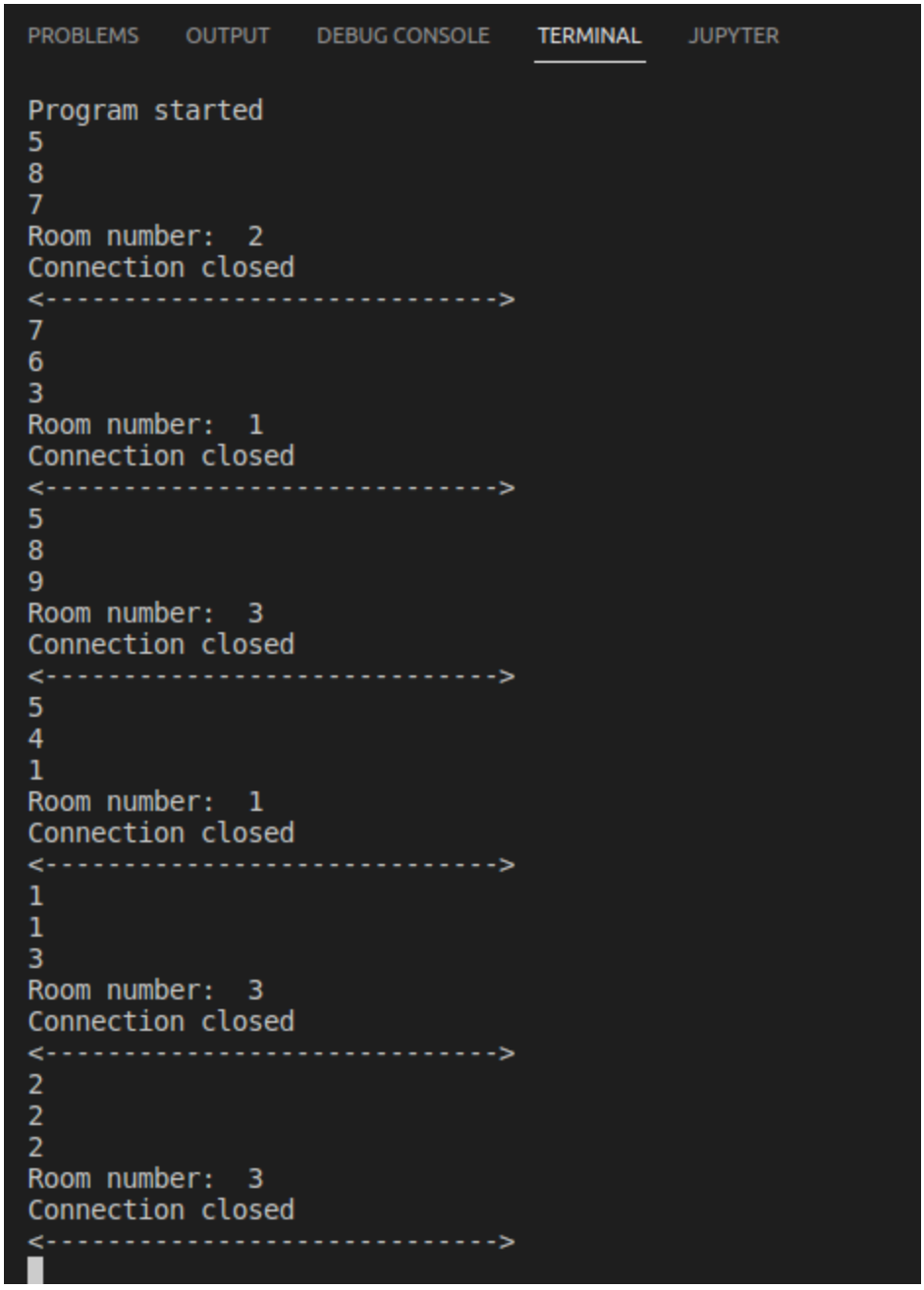
```
x = 0
    for i in beacons.keys():
        beacons[i] = temp[x]
        x -= 1
```

In line 32, x updation statement has been mutated.

```
gb@Ubuntu-PC:~$ /bin/python3 /home/gb/tracking.py
Program started
5
8
7
Room number:  3
Connection closed
<---------------------------->
7
6
3
Room number:  1
Connection closed
<---------------------------->
5
8
9
Room number:  2
Connection closed
<---------------------------->
```

Again, we can see that the output obtained are wrong after the mutation has been made into the code base.

4. Original Code



The above output is for original code, which are giving the expected result for the given input.

Hence only original code passed the test and all three mutated code blocks failed the test.

Problem Statement – 4: Acceptance Testing

1. Assume your neighboring team is the client for your code. Give them an idea of what your product is and the software requirements for the product.
2. Exchange your code base and test each others projects to see if it meets user requirements
3. If you identify a bug in the project you are testing, inform the opposing team of the bug
4. As a team, based in clients experience, ideate modifications to the existing project that could improve client experience

CODE:

```
def printSolution( sol ):

    for i in sol:
        for j in i:
            print(str(j) + " ", end ="")
        print("")

def isSafe( maze, x, y ):

    if x >= 0 and x < N and y >= 0 and y < N and maze[x][y] == 1:
        return True

    return False

def solveMaze( maze ):

    # Creating a 4 * 4 2-D list
    sol = [ [ 0 for j in range(4) ] for i in range(4) ]

    if solveMazeUtil(maze, 0, 0, sol) == False:
        print("Solution doesn't exist");
        return False

    printSolution(sol)
    return True
```

```python
def solveMazeUtil(maze, x, y, sol):

    # if (x, y is goal) return True
    if x == N - 1 and y == N - 1:
        sol[x][y] = 1
        return True

    # Check if maze[x][y] is valid
    if isSafe(maze, x, y) == True:
        # mark x, y as part of solution path
        sol[x][y] = 1

        # Move forward in x direction
        if solveMazeUtil(maze, x + 1, y, sol) == True:
            return True

        # If moving in x direction doesn't give solution
        # then Move down in y direction
        if solveMazeUtil(maze, x, y + 1, sol) == True:
            return True

        # If none of the above movements work then
        # BACKTRACK: unmark x, y as part of solution path
        sol[x][y] = 0
        return False









N=int(input("Enter size of maze:"))

# Driver program to test above function
if __name__ == "__main__":
    # Initializing the maze
    maze = [ [1, 0, 0, 0],
```

```
            [1, 1, 0, 1],
            [0, 1, 0, 0],
            [1, 1, 1, 1] ]

    solveMaze(maze)
```

FEW TEST CASES ARE AS FOLLOWS:

```
PS C:\Users\My PC\Desktop> python REST.py
Enter size of maze:-1
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
PS C:\Users\My PC\Desktop> python REST.py
Enter size of maze:9999999
Traceback (most recent call last):
  File "C:\Users\My PC\Desktop\REST.py", line 63, in <module>
    solveMaze(maze)
  File "C:\Users\My PC\Desktop\REST.py", line 20, in solveMaze
    if solveMazeUtil(maze, 0, 0, sol) == False:
  File "C:\Users\My PC\Desktop\REST.py", line 40, in solveMazeUtil
    if solveMazeUtil(maze, x + 1, y, sol) == True:
  File "C:\Users\My PC\Desktop\REST.py", line 45, in solveMazeUtil
    if solveMazeUtil(maze, x, y + 1, sol) == True:
  File "C:\Users\My PC\Desktop\REST.py", line 40, in solveMazeUtil
    if solveMazeUtil(maze, x + 1, y, sol) == True:
  File "C:\Users\My PC\Desktop\REST.py", line 40, in solveMazeUtil
    if solveMazeUtil(maze, x + 1, y, sol) == True:
  File "C:\Users\My PC\Desktop\REST.py", line 40, in solveMazeUtil
    if solveMazeUtil(maze, x + 1, y, sol) == True:
  File "C:\Users\My PC\Desktop\REST.py", line 35, in solveMazeUtil
    if isSafe(maze, x, y) == True:
  File "C:\Users\My PC\Desktop\REST.py", line 10, in isSafe
    if x >= 0 and x < N and y >= 0 and y < N and maze[x][y] == 1:
IndexError: list index out of range
```

```
PS C:\Users\My PC\Desktop> python REST.py
Enter size of maze:qwe
Traceback (most recent call last):
  File "C:\Users\My PC\Desktop\REST.py", line 53, in <module>
    N=int(input("Enter size of maze:"))
ValueError: invalid literal for int() with base 10: 'qwe'
```

```
PS C:\Users\My PC\Desktop> python REST.py
Enter size of maze:4
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1
PS C:\Users\My PC\Desktop>
```

```
PS C:\Users\My PC\Desktop> python REST.py
Enter size of maze:4
Solution doesn't exist
```

```
maze = [ [1, 0, 0, 0],
         [1, 1, 0, 1],
         [0, 7, 0, 0],
         [1, 1, 1, 1] ]
```

1. *The Project which was tested is **"RAT IN A MAZE" game.***
   So this project shows the rat the path to move such that it can reach the destination.
2. *All the user requirements as mentioned by the team are satisfied.*
3. *All edge cases are taken care of.*
4. *To improve the clients' experience an UI can be made and games can be interactive and visually appealing and Error handling(type errors or out of range errors etc) can be done in order to handle errors gracefully and also helping the execution to resume when interrupted.*