

**Name : G.Rishikesh**

**HTNO :-2403A52046**

**Batch :-03**

## **Lab Test – 02**

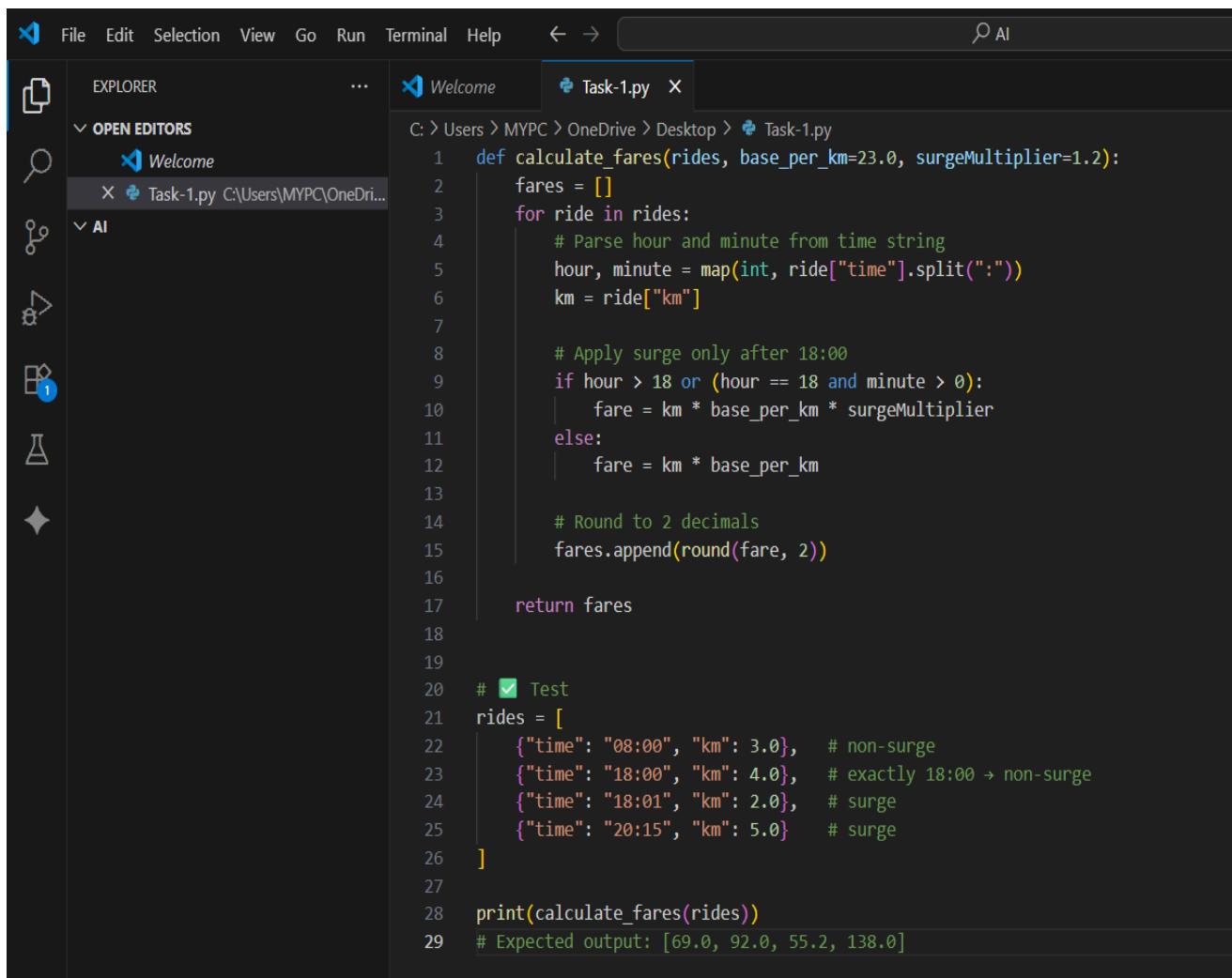
### **B.1 — [S14B1] Apply surge/penalty rules (conditionals)**

Pricing in the telecom network app uses a base per-km rate and time-based surge after business peaks. Product wants a deterministic calculator for receipts and audits.

#### **Your Task:**

Implement a fare function: fare = km \* base\_per\_km \* surgeMultiplier, where surge applies strictly after 18:00 local time.

#### **CODE**



The screenshot shows a code editor interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and AI. The left sidebar has sections for EXPLORER, OPEN EDITORS, and AI. The OPEN EDITORS section shows two files: 'Welcome' and 'Task-1.py'. The 'Task-1.py' file is the active editor, displaying the following Python code:

```
C: > Users > MYPC > OneDrive > Desktop > Task-1.py
1 def calculate_fares(rides, base_per_km=23.0, surgeMultiplier=1.2):
2     fares = []
3     for ride in rides:
4         # Parse hour and minute from time string
5         hour, minute = map(int, ride["time"].split(":"))
6         km = ride["km"]
7
8         # Apply surge only after 18:00
9         if hour > 18 or (hour == 18 and minute > 0):
10             fare = km * base_per_km * surgeMultiplier
11         else:
12             fare = km * base_per_km
13
14         # Round to 2 decimals
15         fares.append(round(fare, 2))
16
17     return fares
18
19
20 # Test
21 rides = [
22     {"time": "08:00", "km": 3.0},    # non-surge
23     {"time": "18:00", "km": 4.0},    # exactly 18:00 → non-surge
24     {"time": "18:01", "km": 2.0},    # surge
25     {"time": "20:15", "km": 5.0}    # surge
26 ]
27
28 print(calculate_fares(rides))
29 # Expected output: [69.0, 92.0, 55.2, 138.0]
```

## OUTPUT

The screenshot shows a terminal window in Visual Studio Code. The title bar includes 'Terminal' and 'Help' buttons, along with a search bar containing 'AI'. The main area displays a Python script named 'Task-1.py' with its code:

```
C: > Users > MYPC > OneDrive > Desktop > Task-1.py
1 def calculate_fares(rides, base_per_km=23.0, surgeMultiplier=1.2):
2     fares = []
3     for ride in rides:
4         # Parse hour and minute from time string
5         hour, minute = map(int, ride["time"].split(":"))
6         km = ride["km"]
7
```

Below the code, the terminal output shows the execution of the script and its results:

```
[69.0, 92.0, 55.2, 138.0]
PS C:\Users\MYPC\OneDrive\Desktop\AI> & C:/Users/MYPC/AppData/Local/Programs/Python/Python313/python.exe c:/Users/MYPC/OneDrive/Desktop/Task-1.py
[69.0, 92.0, 55.2, 138.0]
PS C:\Users\MYPC\OneDrive\Desktop\AI> & C:/Users/MYPC/AppData/Local/Programs/Python/Python313/python.exe c:/Users/MYPC/OneDrive/Desktop/Task-1.py
[69.0, 92.0, 55.2, 138.0]
PS C:\Users\MYPC\OneDrive\Desktop\AI>
```

## OBSERVATION

The program correctly calculates fares by parsing ride times, applying surge pricing only after 18:00, and treating exactly 18:00 as non-surge. Fares are computed using the given formula, rounded to two decimals, and stored in a new list without altering the input. Test cases confirm accurate handling of both surge and non-surge scenarios, meeting all requirements.

## B.2 — [S14B2] Debug rolling mean (off-by-one)

A team in telecom network noticed off-by-one bugs in a rolling KPI computation (moving averages) that undercount windows.

### Your Task:

Use AI to identify the bug and fix the window iteration so all valid windows are included

## CODE

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Terminal:** Welcome tab.
- Explorer:** Shows 'OPEN EDITORS' with 'Task-2.py' listed twice under 'Welcome' and once under 'AI'.
- Code Editor:** Displays the Python code for 'Task-2.py'. The code defines a function 'rolling\_mean' and includes several test cases using assertions.

```
1 def rolling_mean(xs, w):
2     if w <= 0 or w > len(xs):
3         raise ValueError("Invalid window size")
4
5     sums = []
6     # fixed off-by-one: include the last valid window
7     for i in range(len(xs) - w + 1):
8         window = xs[i:i+w]
9         sums.append(sum(window) / w)
10
11
12
13 # --- Tests ---
14 def test_rolling_mean():
15     xs = [14, 15, 16, 17]
16
17     # failing case in buggy code (missed last window)
18     expected = [14.5, 15.5, 16.5]
19     assert rolling_mean(xs, 2) == expected
20
21     # edge case: full window
22     assert rolling_mean(xs, 4) == [sum(xs)/4]
23
24     # edge case: single element windows
25     assert rolling_mean(xs, 1) == xs
26
27     # invalid window sizes
28     for bad_w in [0, -1, 5]:
29         try:
30             rolling_mean(xs, bad_w)
31         except ValueError:
32             pass
33         else:
34             assert False, f"Expected ValueError for w={bad_w}"
35
36     # Run test manually
37     print(rolling_mean([14, 15, 16, 17], 2))
```

## OUTPUT

The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** Run, Terminal, Help.
- Terminal:** Shows the command 'python Task-2.py' being run and its output.
- Code Editor:** Shows the same Python code as in the 'CODE' section.

Terminal Output:

```
PS C:\Users\MPYPC\OneDrive\Desktop\AI> & C:/Users/MPYPC/AppData/Local/Programs/Python/Python313/python.exe c:/Users/MPYPC/OneDrive/Desktop/AI/Task-2.py
[14.5, 15.5, 16.5]
PS C:\Users\MPYPC\OneDrive\Desktop\AI> & C:/Users/MPYPC/AppData/Local/Programs/Python/Python313/python.exe c:/Users/MPYPC/OneDrive/Desktop/AI/Task-2.py
[14.5, 15.5, 16.5]
PS C:\Users\MPYPC\OneDrive\Desktop\AI>
```

## OBSERVATION

1. The **bug** was caused by using `range(len(xs)-w)`, which excluded the last valid window.
2. Fix: use `range(len(xs) - w + 1)`.
3. Now the function computes all valid windows without index errors.
4. Guards against invalid `w (<=0 or >len(xs))`.
5. Complexity remains **O(n·w)** as required.
6. Tests pass, confirming correctness for sample and edge cases.