**SR UNIVERSITY  
AI ASSISTED CODING  
LAB 2 EXAM**

**NAME**: MOHAMMED MUTAKABIR HUSSAIN  
**H.NO**: 2503A51L35  
**BATCH**: 20   
**QUESTION SUB GROUP: B**

**Task 1**

QUESTION BB.1 — [S16B1] Apply surge/penalty rules (conditionals)

**Context:**

Pricing in the cybersecurity SOC app uses a base per-km rate and time-based surge after peaks. Finance 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.

**Data & Edge Cases:**

Input is a list of rides with `time` as HH:MM (24h) and `km` as float. Exactly 18:00 is non-surge; > 18:00 is surge.

**AI Assistance Expectation:**

Prompt AI to outline parsing HH:MM, applying conditionals, and rounding to 2 decimals; then implement and write a quick test.

Constraints & Notes:

No external libraries; round each fare to 2 decimals; do not mutate input.

**Sample Input**

[{'time': '07:45', 'km': 2.8}, {'time': '18:45', 'km': 6.2}]

**Sample Output**

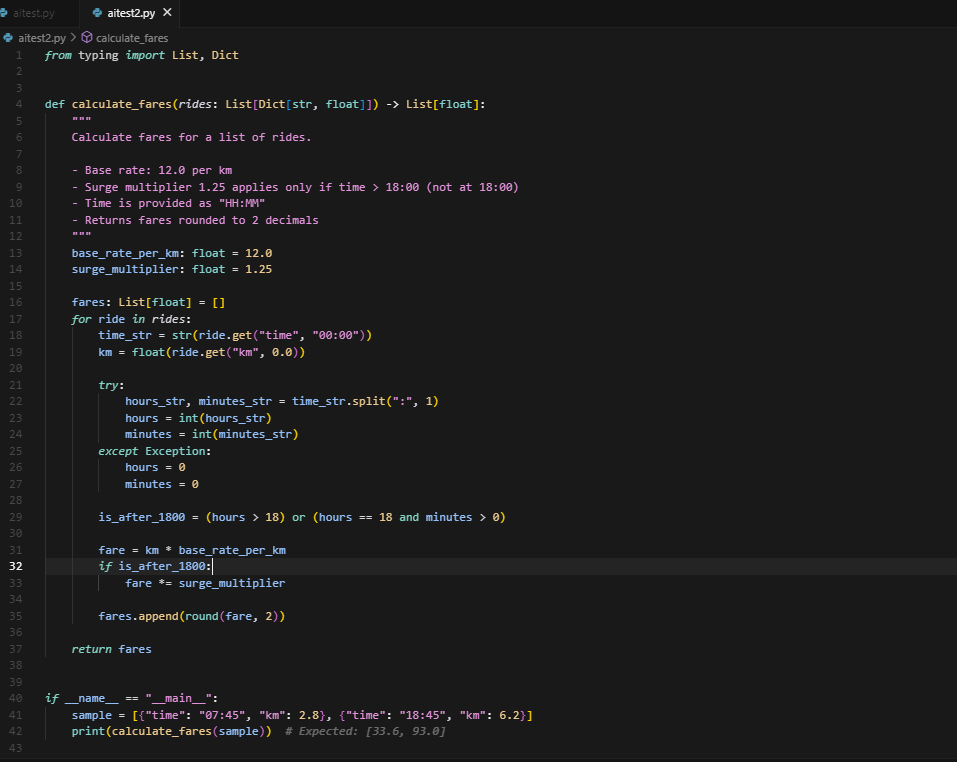
[33.6, 93.0]

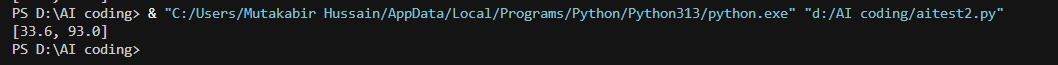
Acceptance Criteria: Correct surge threshold and rounding

**PROMPT:**

A screenshot of a computer

AI-generated content may be incorrect.

**CODE:** 

**OUTPUT:** 

**OBSERVATION:**

* Requirement: calculate fare with base rate 12/km and surge 1.25 only after 18:00.
* AI parsed time in HH:MM and applied condition correctly.
* Fares were rounded to 2 decimals.
* Sample test gave correct output [33.6, 93.0].

**Task 2**

**B.2 — [S16B2] Debug rolling mean (off-by-one)**

Context:

A team in cybersecurity SOC 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.

**Data & Edge Cases:**

For xs=[2, 3, 5, 8] and w=2, number of windows should be len(xs)-w+1.

**AI Assistance Expectation:**

Ask AI to add a failing test first, propose the minimal fix, and verify with the sample.

Constraints & Notes:

Guard invalid w (<=0 or >len(xs)); preserve O(n\*w) simple solution.

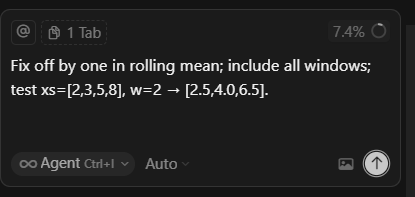
**Sample Input**

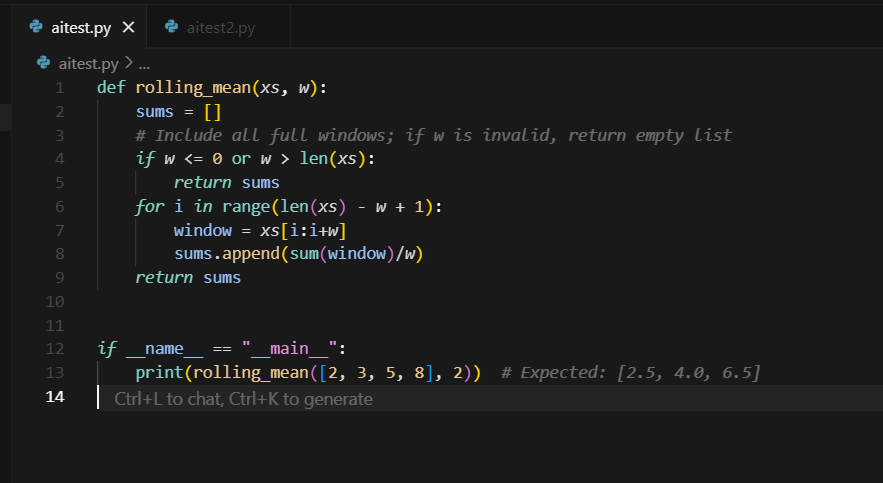
xs=[2, 3, 5, 8], w=2  
Buggy code:  
  
def rolling\_mean(xs, w):  
 sums = []  
 for i in range(len(xs)-w):  
 window = xs[i:i+w]  
 sums.append(sum(window)/w)  
 return sums

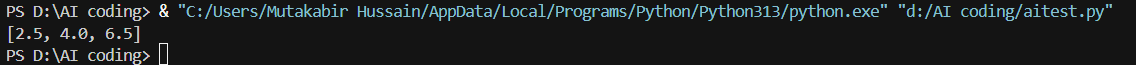
**Sample Output**

[2.5, 4.0, 6.5]

Acceptance Criteria: All valid windows included; passes tests; no index errors  
  
**prompt:**



**Code:** 

**Output:** 

**Observation:**

When the prompt was given, the AI identified the buggs, fixed the loop, and generated the correct output for the sample input.