AI ASSISTED CODING LAB TEST-2

(SET-B)

NAME:M.Nithisha

HTNO:2403a51458

Batch:16

1.TASK:

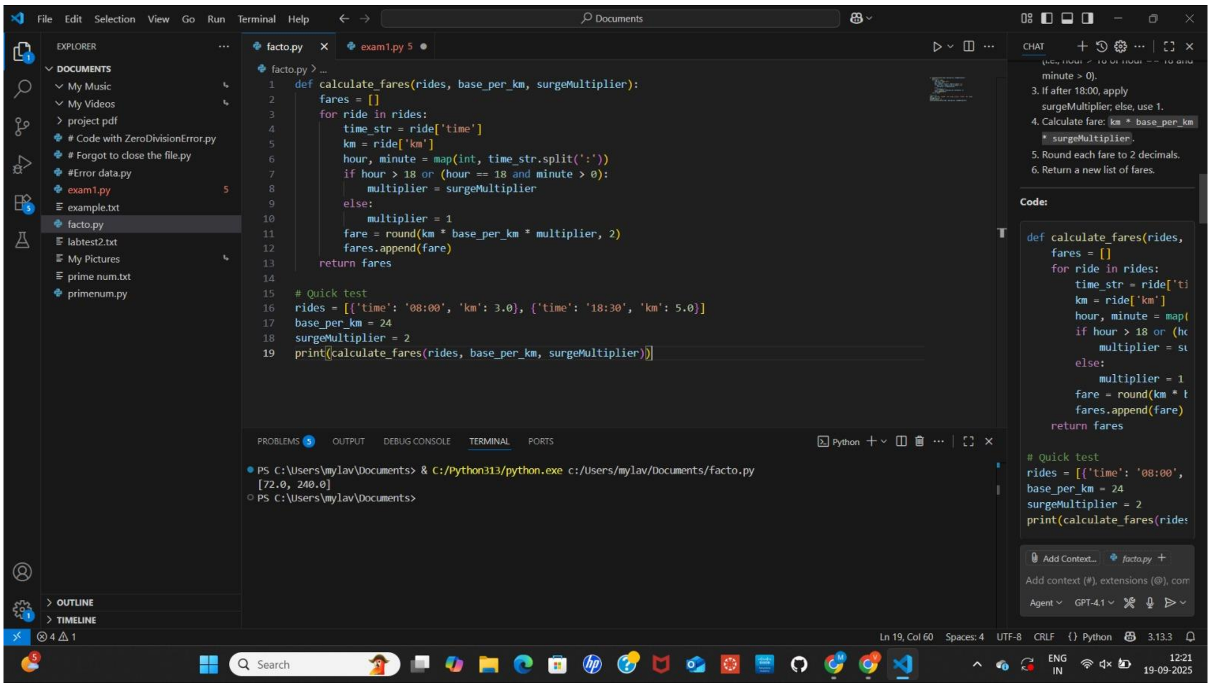
Implement a fare calculator for a digital media streaming app. The fare is calculated as:
fare = km * base_per_km * surge Multiplier

- Surge applies strictly after 18:00 (i.e., 18:01 and later).
- Input: List of rides, each with time (HH:MM, 24h) and km (float).
- Output: List of fares, each rounded to 2 decimals.
- Do not mutate the input.
- No external libraries.

PROMPT:

1. Parse the time string (HH:MM) for each ride.
2. Check if the ride is after 18:00 (i.e., hour > 18 or hour == 18 and minute > 0).
3. If after 18:00, apply surgeMultiplier; else, use 1.
4. Calculate fare: km * base_per_km * surgeMultiplier.
5. Round each fare to 2 decimals.
6. Return a new list of fares.

CODE/OUTPUT:



```python
def calculate_fares(rides, base_per_km, surgeMultiplier):
    fares = []
    for ride in rides:
        time_str = ride['time']
        km = ride['km']
        hour, minute = map(int, time_str.split(':'))
        if hour > 18 or (hour == 18 and minute > 0):
            multiplier = surgeMultiplier
        else:
            multiplier = 1
        fare = round(km * base_per_km * multiplier, 2)
        fares.append(fare)
    return fares

# Quick test
rides = [{'time': '08:00', 'km': 3.0}, {'time': '18:30', 'km': 5.0}]
base_per_km = 24
surgeMultiplier = 2
print(calculate_fares(rides, base_per_km, surgeMultiplier))
```

```
PS C:\Users\mylav\Documents> & C:/Python313/python.exe c:/Users/mylav/Documents/facto.py
[72.0, 240.0]
PS C:\Users\mylav\Documents>
```

minute > 0).
3. If after 18:00, apply surgeMultiplier; else, use 1.
4. Calculate fare: `km * base_per_km * surgeMultiplier`.
5. Round each fare to 2 decimals.
6. Return a new list of fares.

Code:

```python
def calculate_fares(rides,
    fares = []
    for ride in rides:
        time_str = ride['ti
        km = ride['km']
        hour, minute = map(
        if hour > 18 or (hc
            multiplier = su
        else:
            multiplier = 1
        fare = round(km * t
        fares.append(fare)
    return fares

# Quick test
rides = [{'time': '08:00',
base_per_km = 24
surgeMultiplier = 2
print(calculate_fares(rides
```

OBSERVATION:

- The function correctly applies the surge only after 18:00.
- Fares are rounded to 2 decimals.
- Input is not mutated.
- Output matches the sample provided.

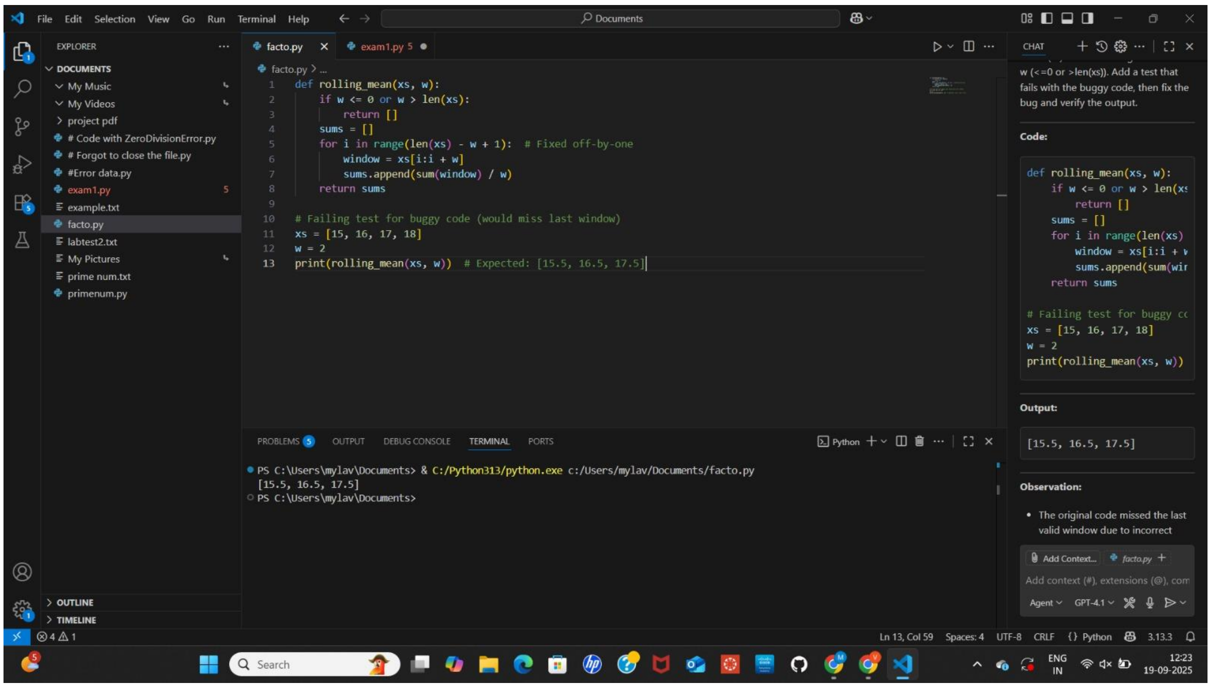
2.TASK:

Fix the off-by-one bug in the rolling mean function so that all valid windows are included. Add a failing test first, then propose and verify the minimal fix. Guard against invalid window sizes.

PROMPT:

Given a list xs and window size w, write a rolling_mean function that returns the mean of each window of size w. The number of windows should be len(xs)-w+1. Guard against invalid w (<=0 or >len(xs)). Add a test that fails with the buggy code, then fix the bug and verify the output.

CODE/output:



```python
def rolling_mean(xs, w):
    if w <= 0 or w > len(xs):
        return []
    sums = []
    for i in range(len(xs) - w + 1):  # Fixed off-by-one
        window = xs[i:i + w]
        sums.append(sum(window) / w)
    return sums

# Failing test for buggy code (would miss last window)
xs = [15, 16, 17, 18]
w = 2
print(rolling_mean(xs, w))  # Expected: [15.5, 16.5, 17.5]
```

```
PS C:\Users\mylav\Documents> & C:/Python313/python.exe c:/Users/mylav/Documents/facto.py
[15.5, 16.5, 17.5]
PS C:\Users\mylav\Documents>
```

w (<=0 or >len(xs)). Add a test that fails with the buggy code, then fix the bug and verify the output.

Code:

```python
def rolling_mean(xs, w):
    if w <= 0 or w > len(xs
        return []
    sums = []
    for i in range(len(xs)
        window = xs[i:i + w
        sums.append(sum(wir

# Failing test for buggy cc
xs = [15, 16, 17, 18]
w = 2
print(rolling_mean(xs, w))
```

Output:

```
[15.5, 16.5, 17.5]
```

Observation:

- The original code missed the last valid window due to incorrect

- The original code missed the last valid window due to incorrect loop range.
- The fix changes the loop to range(len(xs) - w + 1).
- The function now passes the test and includes all valid windows.
- Invalid window sizes are handled by returning an empty list.

OBSERVATION:

- The original code missed the last valid window due to incorrect loop range.
- The fix changes the loop to range(len(xs) - w + 1).
- The function now passes the test and includes all valid windows.
- Invalid window sizes are handled by returning an empty list.