Name : G.Rishikesh

HTNO :-2403A52046

Batch :-03

Lab Test – 02

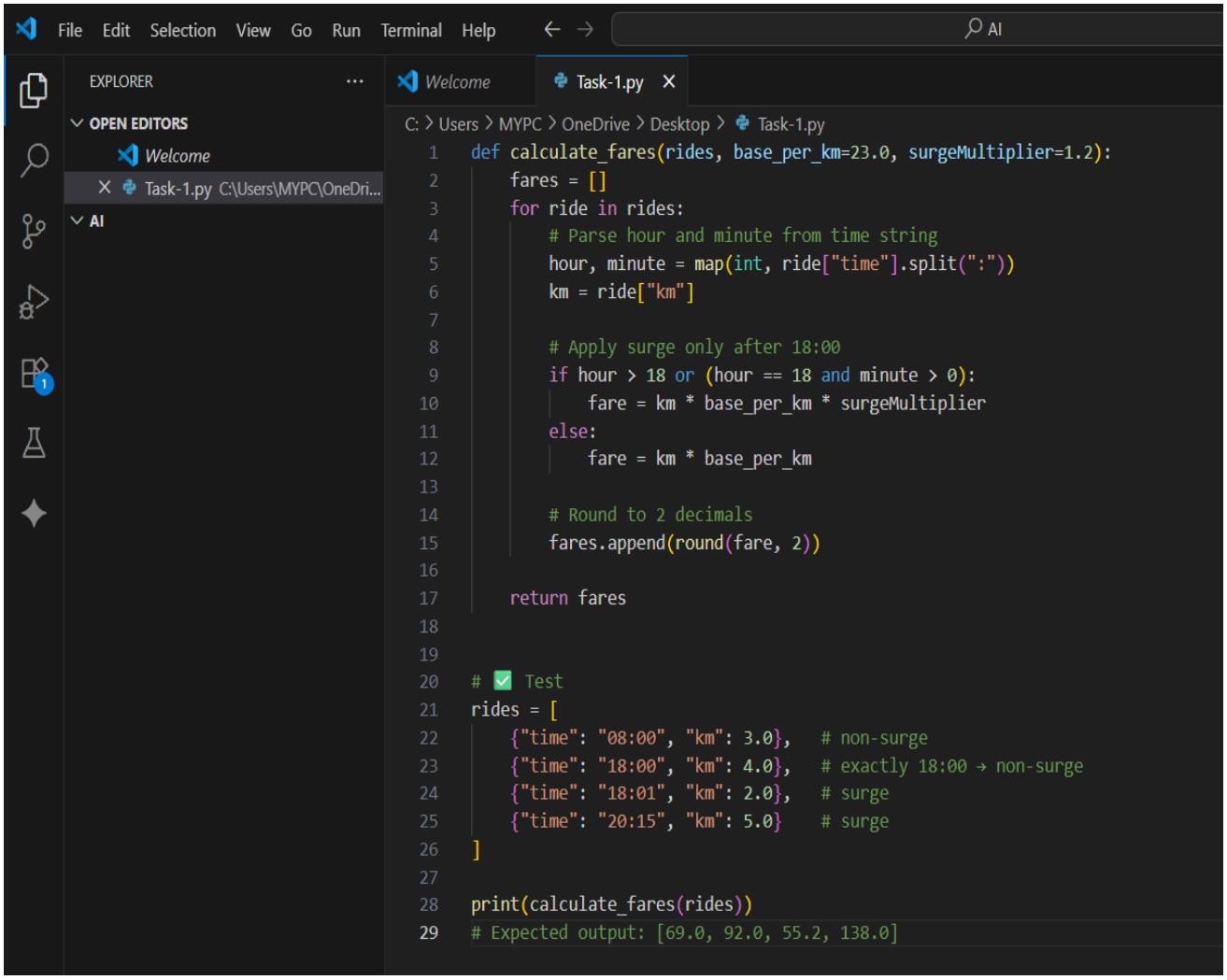
B.1 — [S14B1] Apply surge/penalty rules (conditionals)

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.

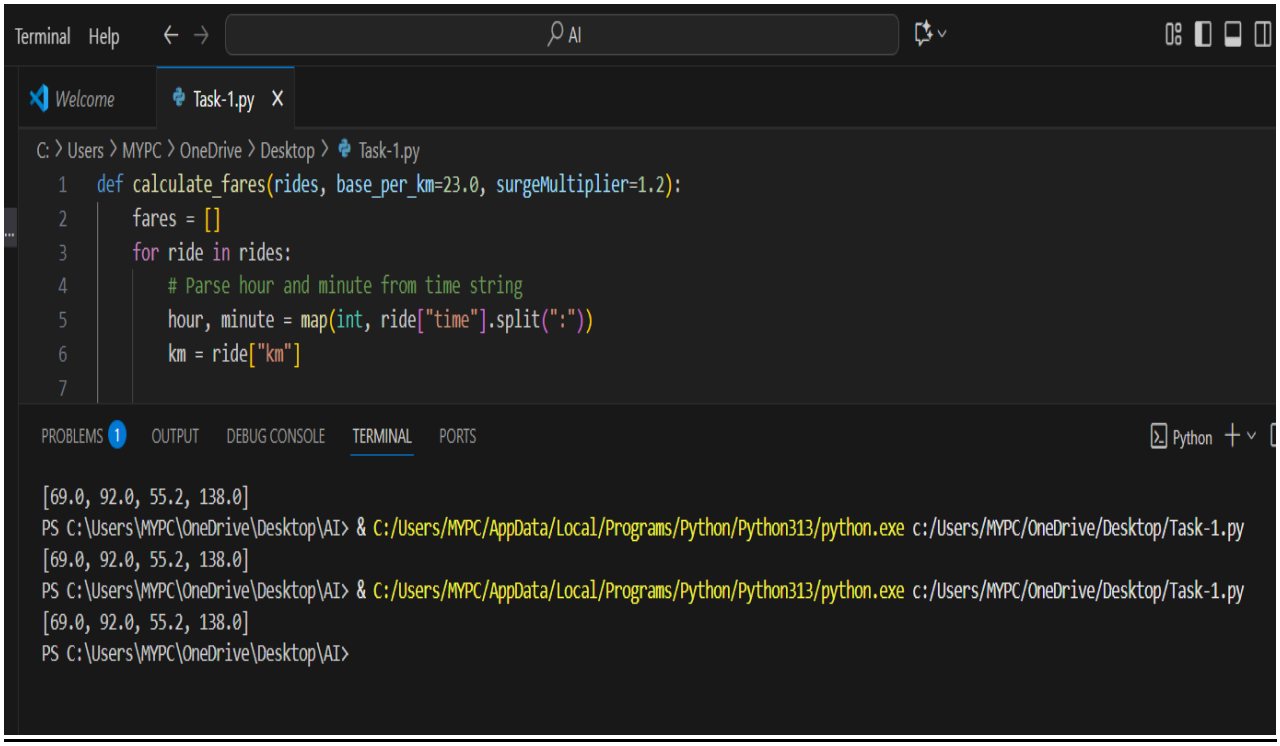
Your Task:

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

CODE



OUTPUT



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

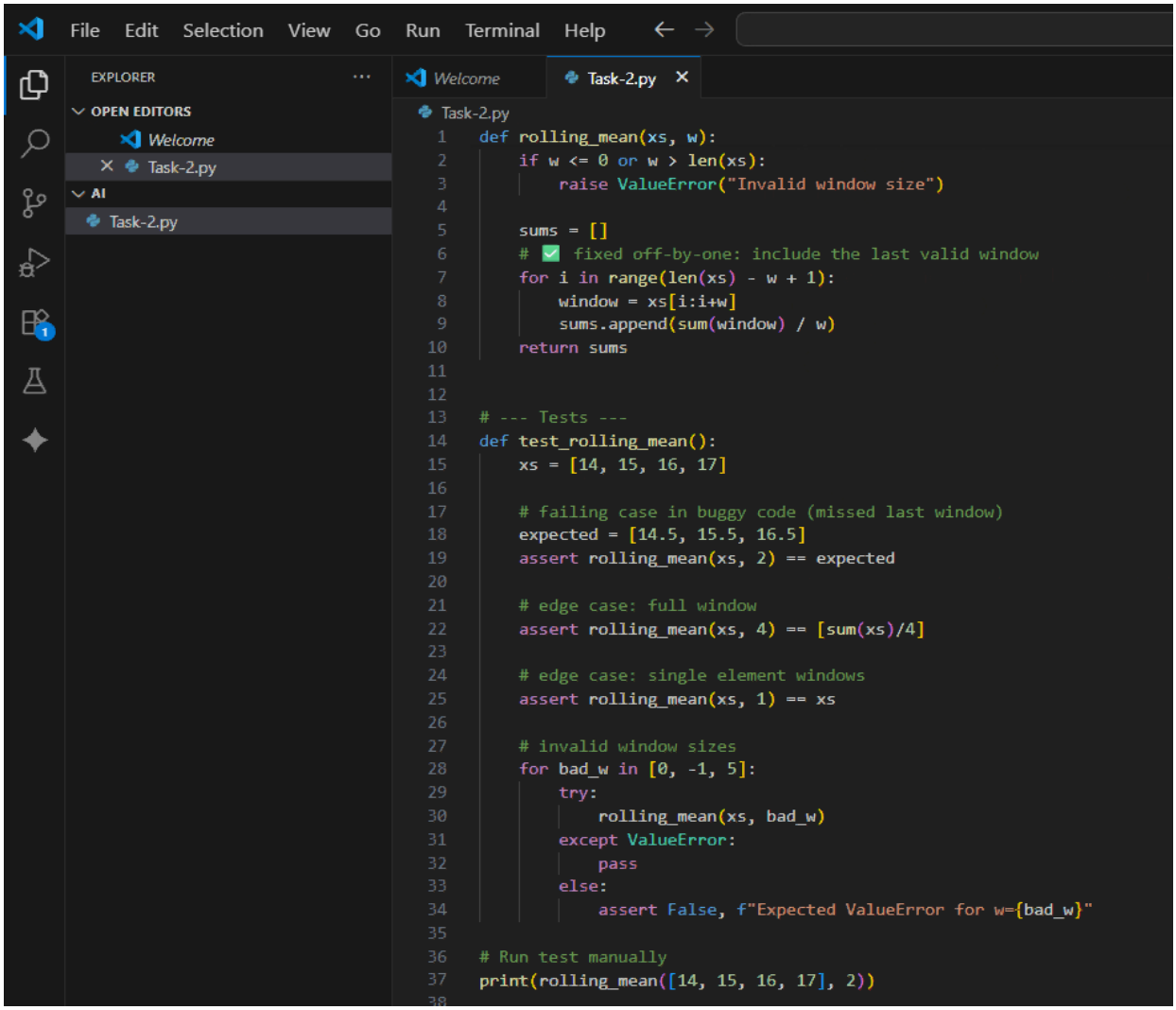
B.2 — [S14B2] Debug rolling mean (off-by-one)

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

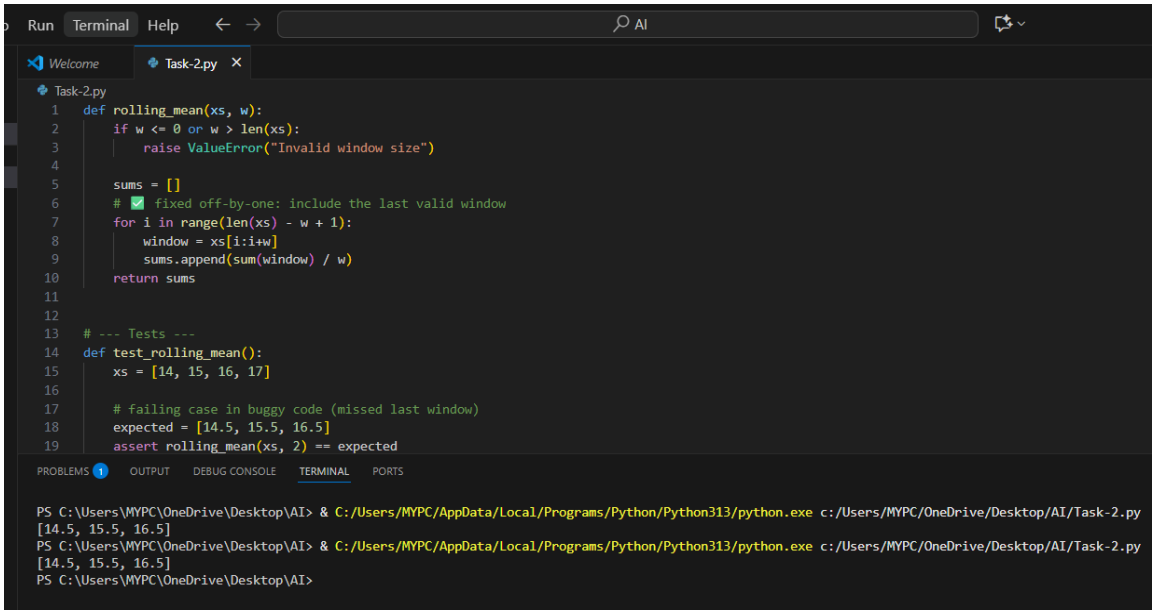
Your Task:

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

CODE


File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
Welcome
Task-2.py
AI
Task-2.py
Welcome
Task-2.py
Task-2.py
def rolling_mean(xs, w):
if w <= 0 or w > len(xs):
raise ValueError("Invalid window size")
sums = []
# ✅ fixed off-by-one: include the last valid window
for i in range(len(xs) - w + 1):
window = xs[i:i+w]
sums.append(sum(window) / w)
return sums
# --- Tests ---
def test_rolling_mean():
xs = [14, 15, 16, 17]
# failing case in buggy code (missed last window)
expected = [14.5, 15.5, 16.5]
assert rolling_mean(xs, 2) == expected
edge case: full window
assert rolling_mean(xs, 4) == [sum(xs)/4]
edge case: single element windows
assert rolling_mean(xs, 1) == xs
# invalid window sizes
for bad_w in [0, -1, 5]:
try:
rolling_mean(xs, bad_w)
except ValueError:
pass
else:
assert False, f"Expected ValueError for w={bad_w}"
# Run test manually
print(rolling_mean([14, 15, 16, 17], 2))

OUTPUT


Run Terminal Help
AI
Welcome
Task-2.py
Task-2.py
def rolling_mean(xs, w):
if w <= 0 or w > len(xs):
raise ValueError("Invalid window size")
sums = []
# ✅ fixed off-by-one: include the last valid window
for i in range(len(xs) - w + 1):
window = xs[i:i+w]
sums.append(sum(window) / w)
return sums
# --- Tests ---
def test_rolling_mean():
xs = [14, 15, 16, 17]
# failing case in buggy code (missed last window)
expected = [14.5, 15.5, 16.5]
assert rolling_mean(xs, 2) == expected
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\MYPC\OneDrive\Desktop\AI> & C:/Users/MYPC/AppData/Local/Programs/Python/Python313/python.exe c:/Users/MYPC/OneDrive/Desktop/AI/Task-2.py
[14.5, 15.5, 16.5]
PS C:\Users\MYPC\OneDrive\Desktop\AI> & C:/Users/MYPC/AppData/Local/Programs/Python/Python313/python.exe c:/Users/MYPC/OneDrive/Desktop/AI/Task-2.py
[14.5, 15.5, 16.5]
PS C:\Users\MYPC\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.