AI ASSISTED CODING LABTEST-2:

NAME:T.GOWRI SHANKAR REDDY

HT.NO:2403A51295

BATCH:12

B.1 — Apply Surge/Penalty Rules (Conditionals)

Context

Pricing in the real estate listings platform app uses a base per-km rate and time-based surge after peaks. Finance wants a deterministic calculator for receipts and audits.

Formula: fare = km × base_per_km × surgeMultiplier

- Surge applies strictly after 18:00 local time.
- Exactly 18:00 is non-surge.
- Round to 2 decimals.
- Do not mutate input.

Steps to Solve

1. Parse time string HH:MM into hours and minutes.
2. Check surge: If hour > 18 or (hour == 18 and minute > 0) → surge applies.
3. Use base_per_km = 7.0, surgeMultiplier = 2.0 if surge else 1.0.
4. Calculate fare = km × base_per_km × surgeMultiplier.
5. Round fare to 2 decimals and collect in results.

Implementation

```
def calculate_fares(rides, base_per_km=7.0, surgeMultiplier=2.0):
    results = []
    for ride in rides:
        time_str = ride['time']
        km = ride['km']
        hh, mm = map(int, time_str.split(':'))
        if hh > 18 or (hh == 18 and mm > 0):
            surge = surgeMultiplier
```

```
        else:
            surge = 1.0
        fare = km * base_per_km * surge
        results.append(round(fare, 2))
    return results
```

Quick Test

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

print(calculate_fares(rides))
```

Expected Output: [39.2, 108.5]

B.2 — Debug Rolling Mean (Off-by-One)

Context

A team in the platform noticed off-by-one bugs in a rolling KPI computation (moving averages). Buggy code undercounts windows.

Requirement: Number of windows should be len(xs)-w+1 for window size w.

Buggy Code

```
def rolling_mean(xs, w):
  sums = []
  for i in range(len(xs)-w):   # Off-by-one here
    window = xs[i:i+w]
    sums.append(sum(window)/w)
  return sums
```

Problem: The loop runs only until len(xs)-w (exclusive). This misses the last valid window.

Steps to Fix

6. Check invalid cases: if w <= 0 or w > len(xs), return [].
7. Correct loop to iterate until len(xs)-w+1 inclusive.
8. Keep O(n*w) simple solution.

Fixed Implementation

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

Quick Test

```
xs = [4, 5, 7, 10]
w = 2
print(rolling_mean(xs, w))
```

Expected Output: [4.5, 6.0, 8.5]