AI ASSISTED CODING

LAB TEST -2

Question:B.1

```
import re

def extract_and_multiply_times(data):
    results = []
    for item in data:
        time_str = item.get('time', '')
        km = item.get('km', 0)

        # Match both hh:mm and hh.mm
        match = re.match(r'(\d{1,2})[:\.](\d{2})', time_str)
        if match:
            hours = int(match.group(1))
            minutes = int(match.group(2))
            decimal_hours = hours + minutes / 60.0
            product = round(decimal_hours * km, 2)
            results.append(product)
        else:
            results.append(None)

    return results

# Use the correct input to get [42.0, 84.0]
sample_input = [{'time':'14:00','km':3.0},
{'time':'16:48','km':5.0}]
result = extract_and_multiply_times(sample_input)
print(result)
```

Output:

```
[42.0, 84.0]
```

Explanation:

1. **Import `re`**: The code starts by importing the `re` module, which is Python's regular expression library. This is used to parse the time string.
2. **Define the function**: The function `extract_and_multiply_times` is defined to accept one argument, `data`, which is expected to be a list of dictionaries.
3. **Initialize results list**: An empty list called `results` is created. This list will store the calculated products for each data entry.
4. **Iterate through data**: The code then loops through each `item` (dictionary) in the input `data list`.
5. **Extract time and km**: Inside the loop, it extracts the `time` string and `km` value from the current `item`. It uses `.get()` with a default value (`''` for time, `0` for km) to handle cases where a key might be missing.
6. **Parse time with regex**: It uses a regular expression `r'(\d{1,2})[:\.](\d{2})'` to match time strings in either `hh:mm` or `hh.mm` format.
    - `(\d{1,2})` captures one or two digits for the hours.
    - `[:\.]` matches either a colon or a dot separating hours and minutes.
    - `(\d{2})` captures exactly two digits for the minutes.
7. **Process valid time**: If a match is found:
    - It extracts the captured hours and minutes as integers.
    - It converts the time to decimal hours by adding the hours to the minutes divided by 60.0.
    - It calculates the `product` by multiplying the `decimal_hours` by the `km` value.
    - It rounds the `product` to two decimal places using `round()`.
    - The rounded `product` is appended to the `results` list.
8. **Handle invalid time**: If no time match is found for an item, `None` is appended to the `results` list.
9. **Return results**: After processing all items in the `data` list, the function returns the `results` list.
10. **Test case**: The code includes a sample `sample_input` list and calls the function with this input, then prints the returned `result`.

Question:B.2

```
xs=[5,6,7,8]
w= 2
def rolling_means(xs,w):
    sums=[]
    for i in range(len(xs)-w + 1):
        window=xs[i:i+w]
        sums.append(sum(window)/w)
    return sums

print(rolling_means(xs, w))
```

Output:

```
[5.5, 6.5, 7.5]
```

Explanation:

- `xs = [5, 6, 7, 8]`: This line initializes a list named `xs` with the numbers 5, 6, 7, and 8.
- `w = 2`: This line initializes a variable `w` with the value 2. This represents the window size for the rolling mean calculation.
- `def rolling_means(xs, w):`: This line defines a function named `rolling_means` that takes two arguments: `xs` (the list of numbers) and `w` (the window size).
- `sums = []`: Inside the function, an empty list called `sums` is created to store the calculated rolling means.
- `for i in range(len(xs) - w + 1):`: This is a `for` loop that iterates through the `xs` list. The `range()` function generates a sequence of numbers starting from 0 up to (but not including) `len(xs) - w + 1`. This range ensures that the loop iterates through all possible starting positions for a window of size `w`.
- `window = xs[i:i+w]`: In each iteration, this line creates a "window" by slicing the `xs` list from index `i` up to (but not including) `i+w`. This extracts a sublist of size `w`.
- `sums.append(sum(window) / w)`: This line calculates the sum of the numbers in the current `window` using `sum()` and then divides it by the window size `w` to get the mean. This mean is then appended to the `sums` list.
- `return sums`: After the loop finishes, the function returns the `sums` list containing all the calculated rolling means.
- `print(rolling_means(xs, w))`: This line calls the `rolling_means` function with the `xs` list and window size `w` and prints the returned list of rolling means.