Perspectives on Computational Analysis

Problem Set 7, Q1

Unit Test Descriptions

Prob1:

On running the tests, answers initially seem to come out correctly for 1,2, and 3. Errors begin to occur at 4 (the answer comes out as 4, instead of 2). Initially, I checked if this was a problem with perfect squares. The square root of 9 and larger numbers did not cause any such problems. I also checked for different combinations of odd and even factors. Problems occurred only for 4. On a closer inspection of the code, it became clear that the issue was a particularity of the range expression in Python, where the second argument is not included in the iteration. For the edge case of 4, the range would thus become range(2,2), which would not produce any meaningful output. This would lead the programme to simply execute the last line, and return 4 itself. To remedy this, I added '+1' to the second argument of the range expression. After this modification, the code passed all tests

Prob 2

The unit tests check for matching the correct month with the correct number of days (either 31 or 30, based on the dictionaries provided). For this purpose, I tested the code's results against 'December' and 'June' respectively from the two groups. In the case of February, we check for the correct of 29 or 28 days given that the year is a leap year or not respectively. Finally, the code also ensures that incorrect spellings of the months or an empty argument are not processed, and simply return 'None' as the output.

Problem 3

The code first ensures that the third argument of the function is of type string. In case that is not true, the pytest.raises method ensures that if a TypeError is based on the written raise statement. The code then tests out each of the first three operations- addition, multiplication and subtraction on the numbers 3 and 4. The case of division gets special attention. First, we check for 0 in the denominator. The pytest.raises method ensures that if a ZeroDivisionError is based on the written raise statement. Secondly, we ensure that the division operator works correctly for both float and integer results. Finally, we use pytest.raises again- this time with the ValueError – to ensure that our raise statement comes into play whenever any operator other than the ones designated in the codes is used.