Due: Wednesday, January 24, 2018 at 10 am (No late assignments accepted)

Assignment Guidelines

- This assignment covers material in Module 02.
- Submission details:
  - Solutions to these questions must be placed in files a02q1.py, a02q2.py, a02q3.py, and a02q4.py.
  - You must be using Python 3 or higher. Do NOT use Python 2.
  - Download the interface file from the course Web page to ensure that all function names are spelled correctly and each function has the correct number and order of parameters.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the Python section of the CS116 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page
- Download the testing module from the course web page, and save it in the same folder as your solutions. Include import check in each solution file.
  - When a function returns a floating point value, you must use check.within for your testing. Unless told otherwise, you may use a tolerance of 0.001 in your tests.
- Restrictions:
  - Do not import any modules other than math and check.
  - Do not use Python constructs from later modules (e.g. loops and lists). Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post on Piazza. You are always allowed to define your own helper functions, as long as they meet the assignment restrictions.
  - While you may use global constants in your solutions, do not use global variables for anything other than testing.
  - Read each question carefully for additional restrictions.

The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.
1. *The following function will help you calculate your final grade, just like on Assignment 1. This time, however, they do not assume that you passed the weighted exam average!*

Write a Python function `earned_grade`, that consumes five parameters: the first four parameters are of type Float between 0 and 100 (inclusive), corresponding to (in order) your grades on the assignments, the midterm exam, the final exam, and the clicker questions asked in class. The final parameter is a natural number between 0 and 12 (inclusive) corresponding to the number of weeks you attended a tutorial.

The function returns the basic grade (rounded to an integer using the built-in Python function `round`), calculated using the grading scheme on slide 4 of Module 01.

You may be thinking this is exactly the same as A1Q3, and you’re almost correct! However, remember that you must pass the weighted exam average in order to pass the course! If your weighted exam average is at least 50%, `earned_grade` will return the same value that `basic_grade` does. If your weighted exam average is less than 50%, then your grade will be reduced to 45, if it is not already less than that.

For example, `earned_grade(60.0, 55.8, 40.0, 55.5, 9)` => 45, and `earned_grade(10.0, 5.0, 40.0, 12.0, 0)` => 22

2. *Cold enough for you? The following function will help you determine what the weather “feels like”, just like the Weather Network does!*

You may have heard on the radio (or read on your phone) that current temperature is -15°C, but it “feels like” -30! This is because the wind blows away the pocket of warm air that usually surrounds our faces. This is known as the “wind-chill” factor. In the summer months, the humidity can lessen the ability of our bodies to cool themselves off. In Canada we measure this with the “humidex” (there are many other competing ways of measuring “feels like”).

In Canada the wind-chill and humidex equations are

\[
windchill = 13.12 + 0.6125T - 11.37W^{0.16} + 0.3965TW^{0.16}
\]

\[
humidex = T + \frac{5}{9}(V - 10)
\]

Where T is the temperature in Celsius, W is the wind speed in km/h and V is the (water) vapour pressure. Weather stations rarely record vapour pressure directly, but they do record relative humidity (H), and that can be used to compute vapour pressure.

\[
V = 6.112 \times 10^{\frac{7.5T}{237.7+T}} \times \frac{H}{100}
\]
Write a Python function `feels_like`, that consumes three parameters: The first is a Float corresponding to the current temperature in degrees Celsius, the second is a Float corresponding to the current wind speed in km/h (a non-negative value), and the third is a natural number corresponding to the relative humidity percentage (between 0 and 100 inclusive).

The function returns what it currently “feels like”, as a Float value. If T at least 15, and the humidex is above T by at least a full degree, then it “feels like” the current humidex value. If the T is less than 15, and the windchill is at least a full degree below T, then it “feels like” the current wind-chill value. In all other cases, it “feels like” T.

For example, `feels_like(-9.5,1.9,91)` => -9.5 as -9.5 is too cold to use the humidex calculation, and the wind is not fast enough for a wind-chill, so the function returns the temperature.

3. In this question you will predict the future. Or, at least, what the weather will be like in the future.

If you have a barometer, you can use it to tell the weather. For example: If the pressure is high, it is almost always sunny. If the pressure is low, it is almost always cloudy. Additionally, if the barometer is wet, it is raining. A barometer is much more useful if you keep track of how the pressure has been changing lately, as this will tell you how the weather will be changing in the short term!

The procedure is simple: Most barometers let you leave a marker where the needle currently is. Failing that, you can always write it down on a piece of paper (or even use a computer). After waiting a while, you see if the pressure has gone up, gone down, or stayed about the same. Using this information, plus the needle’s absolute position, you can look up the prediction in a table. Here is the table for you to use on this question:

<table>
<thead>
<tr>
<th>Pressure</th>
<th>Trend</th>
<th>Forecast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>Steady</td>
<td>No Change</td>
</tr>
<tr>
<td>Low</td>
<td>Rising Slowly</td>
<td>No Change</td>
</tr>
<tr>
<td>Low</td>
<td>Rising Quickly</td>
<td>Sunny</td>
</tr>
<tr>
<td>Low</td>
<td>Falling Slowly</td>
<td>Storms</td>
</tr>
<tr>
<td>Low</td>
<td>Falling Quickly</td>
<td>Tornado Watch</td>
</tr>
<tr>
<td>Moderate</td>
<td>Rising Slowly</td>
<td>Sunny</td>
</tr>
<tr>
<td>Moderate</td>
<td>Rising Quickly</td>
<td>Windy</td>
</tr>
<tr>
<td>Moderate</td>
<td>Falling Slowly</td>
<td>Rain</td>
</tr>
<tr>
<td>Moderate</td>
<td>Falling Quickly</td>
<td>Storms</td>
</tr>
<tr>
<td>High</td>
<td>Rising Slowly</td>
<td>Sunny</td>
</tr>
<tr>
<td>High</td>
<td>Rising Quickly</td>
<td>Sunny</td>
</tr>
<tr>
<td>High</td>
<td>Falling Slowly</td>
<td>No Change</td>
</tr>
<tr>
<td>High</td>
<td>Falling Quickly</td>
<td>Cloudy</td>
</tr>
</tbody>
</table>
(Real forecast tables are larger, also include the direction and speed of the wind, and are region specific, meaning a table for Waterloo is not as accurate in Toronto)

Write a Python function `weather_forecast` that consumes two parameters: The first is a Float value corresponding to the current pressure (in kPa, between 80.0 and 120.0), and the second is a string corresponding to how the pressure is currently trending (one of “Steady”, “Rising Slowly”, “Rising Quickly”, “Falling Slowly”, or “Falling Quickly”).

The function returns the weather forecast according to the provided chart. Pressures greater than 103.0 kPa are considered “High”, pressures less than 100.0 kPa are considered “Low” and all other pressures are considered “Moderate”.

For example `weather_forecast(101.7, "Falling Slowly")` => "Rain" as 101.7 kPa is “Moderate” Pressure.

This example courtesy of the UW Weather Station, at 2018-01-03 9:00AM. It did in fact snow later that day! The system works, if you count snow as a kind of rain, anyway.

4. *Recursion is a wonderful thing*

Write a Python function `count_primes` that consumes two positive integers (`start`, and `end`). The function returns the number of prime numbers between `start` and `end`, inclusive. In the case where `start` is greater than `end`, there are no prime numbers in that range by definition.

For example, `count_primes(1, 10)` => 4 (the prime numbers in that range are 2,3,5, and 7)

As mentioned in class, there is a limit to how many times a Python function can recurse. For that reason, you may assume that `end < 500`. 