Lab 00: A Grade Calculator

The purpose of this lab is primarily to get you started writing little python programs. The other goal is to make sure that everyone can run python code and submit it through the Mimir system.

Part 1: Assign a letter grade.

Usually, a raw grade will come as a number (either an `int` or a `float`) and we will turn it into a letter grade. Write a function that takes a number as a parameter and returns the corresponding letter grade. The assignment works as follows. If the score is at least 90, the grade should be A. That is, return the string `'A'`. If the score is at least 80 but is less than 90, it's a `'B'`. If the score is at least 70, then its a `'C'`. If it's at least 65, then it's a `'D'`. Below 65, the grade returned should be `'F'`.

Call your function `assigngrade`. For example, here's a version that might be written by a teacher who has quit trying to teach.

```
def assigngrade(score):
    return `'A'`
```

Your version will have a little more logic.

Part 2: Test your code.

After you have written your function, you want to know if it works. Some test code has been included. It tests some aspects of the code that we haven't written yet, but that's okay. The tests are in a file called `testgrader.py`. Make sure its in the same folder as your code and run it. Look at the output. Do any of the tests pass? Look at the tests. Did you expect them to pass?

Part 3: Make it customizable

Right now, 64 is an F, but so is 3. Maybe it's time to invent a new grade, F-, for scores less than 13. Or if you find that too negative, you can think about adding A+ or grades

over 100. Either way, it would be nice to have different grade assignments without changing the code for each new grade assignment rule. We'll do this by adding two new parameters to the `assigngrade` function. We will pass it a list of `grades` and a list of `cutoffs`. Assume that the grades are ordered from highest to lowest. Then, we can check each cutoff in a loop to see what grade to return. As an example, the following code would call `assigngrade` using the cutoffs that we previously hardcoded.

```
grades = ['A', 'B', 'C', 'D', 'F']
cutoffs = [90, 80, 70, 65, 0]
assigngrade(80, grades, cutoffs)
```

In this case, it would return `'B'`.

An alternative grading scheme might look like the following.

```
grades = ['Pass', 'Fail', 'Srsly?']
cutoffs = [60, 7, float('-inf')]
assigngrade(80, grades, cutoffs)
```

In this case, the grade returned would be `'Pass'`. The expression `float('-inf')` is negative infinity. Yes, that's a valid number in python.

Part 4: Add default values.

If you tried to run the tests, you may have found that some of the old tests didn't work. You probably get an error saying that you are missing positional arguments. This is because our old tests didn't pass these new arguments. Rather than modifying all the old tests, we can make the original scheme a default value. This way, if those arguments are not present, they will just use the default values.

In python, to add default values to a parameter, you use the equals sign in the definition. In this case, we would do the following.

```
def assigngrade(score, grades = ['A', 'B', 'C', 'D', 'F'],
                cutoffs = [90, 80, 70, 65, 0]):
    # body of the function goes here....
```

The line wraps around to keep the overall line length under 80 characters. This is good style convention.

Part 5: Compute the average.

Write a function to compute the average of a list of scores. Call it `average`. You will use this to compute the average grade. The input to the function is a list and the output should be the average.

It will be useful to remember that `len(L)` returns the length of the list `L`.

Part 6: Drop the lowest grade.

Now, we are going to allow the lowest grade to be dropped. So, we'll want to figure out which is the lowest and remove it from the list.

Write a function that takes a list as input and removes the smallest entry. Do not assume the entries are sorted. You may want to use the method `list.remove` to remove the item from the list. Note that this really removes the entry, i.e. it modifies the list. This is okay for our purposes.

Here is an example of `list.remove` in action.

```
L = [1,2,3,4,3,2,1]
print(L)
L.remove(3)
print(L)
```

Notice that it only removes the first occurrence of `3` in the list.

Now, your function should be called `droplowest`. It should accept a nonempty list as an argument and it should remove the lowest element from that list. It should only remove one element. It is not expected to return anything.