CSCI 150 HW: Dictionary and Class Homework practice

Due: Wednesday, April 13

To receive full credit, for each exercise you should do the following:

1. **Design**: First, write a function or design a Python class as requested in the exercise.
2. **Check**: Run the provided test code. Does your actual output agree with the given correct output?
3. **Evaluate**: If the actual output does not match the expected output, keep experimenting, consult the textbook or Python documentation, ask a friend or TA or professor, *etc.* until you can fix your class definition and explain what your misunderstanding(s) were. (You do not need to do anything for step 3 if the outputs already agree exactly.)

You should consider the code in each exercise separately from the other exercises.

Upload your code to the class Teams page.

1. Write a function `vowel_count` which takes in a list of strings (all lower case) and returns a dictionary, keyed on the five vowels a, e, i, o, and u. The value for each should be the total number of entries in the list which contain at least a single copy of that vowel. For example,

```
vowel_count(['time', 'is', 'the', 'end']) returns
{'a': 0, 'e': 3, 'i': 2, 'o': 0, 'u': 0}


vowel_count(['aardvark', 'facetious', 'too', 'to', 'two']) returns
{'a': 2, 'e': 1, 'i': 1, 'o': 4', 'u': 1}

(note that we are not counting the total number of a's, or e's,
but how many individual entries in the list contain at least one a)
```

2. Write a function `char_remaining` which takes a string as input and returns a dictionary whose keys are the characters in the string and values are the number of characters in the string remaining after the *first* occurance of the keyed-character. For example:

```
char_remaining('Hello') returns
 {'H': 4, 'e': 3, 'l': 2, 'o': 0}


char_remaining('CSCI 150 class') returns
 {'C': 13, 'S': 12, 'I': 10, ' ': 9, '1': 8, '5': 7, '0': 6, 'c': 4, 'l': 3, 'a': 2, 's':1}
```

3. Write a function `dict_count` which takes in two parameters: `d`, which is a dictionary keyed on strings with integer values, and `s` a single character string. The function should return an integer which is the sum of all values which correspond to keys which contain `s`. For example:

```
if d = {'hi': 3, 'there': 2, 'bye': 6} and s = 'h', then
dict_count(d,s)) returns 5

if d = {'hi': 3, 'there': 2, 'bye': 6} and s = 'a', then
dict_count(d,s)) returns 0

if d = {'hi': 3, 'there': 2, 'bye': 6} and s = 'e', then
dict_count(d,s)) returns 8
```

4. Write a Python class `BouncyBall`, which represents a bouncy ball containing a certain amount of air.

   - When a `BouncyBall` object is first created, it should have 10 units of air.
   - There should be a method `bounce()` which normally prints the word `Bounce!` and decreases the amount of air in the ball by two units. However, if the amount of air is less than or equal to three, then `bounce()` does not decrease the amount of air and prints `Thupp.` instead of `Bounce!`.
   - There should be a method `inflate()` which increases the amount of air by three units. If the amount of air ever becomes greater than 12, then the ball explodes by printing `BANG!!!`.
   - You cannot bounce or inflate an exploded ball. After a ball explodes, calling `bounce()` or `inflate()` should just cause a message to be printed such as `Sorry, you cannot bounce this ball! It has exploded.`

   To test your class, you can type in and run the following code:

```
def main():
    b = BouncyBall()

    for i in range(6):
        b.bounce()

    b.inflate()
    b.bounce()
    b.bounce()
```

```
    for i in range(5):
        b.inflate()

    b.bounce()

main()
```

If your definition of `BouncyBall` is correct, `main()` should produce the following output:

```
Bounce!
Bounce!
Bounce!
Bounce!
Thupp.
Thupp.
Bounce!
Thupp.
BANG!!!
Sorry, you cannot inflate this ball!  It has exploded.
Sorry, you cannot bounce this ball!  It has exploded.
```

5. Write a Python class `Gradebook` which works as follows:

   - When a new `Gradebook` object is first created, it should start out with an empty list of grades, and zero points of extra credit.
   - There should be a method `add_grade(g: int)` which adds the grade `g` to the end of the list.
   - There should be a method `add_ec(ec: int)` which adds `ec` points of extra credit to the current amount of extra credit.
   - There should be a method `average()` which computes and returns the average of all the grades so far (the sum of all the grades, plus the extra credit score, divided by the number of grades).

   You can test your implementation of `Gradebook` by running the code below:

```
def main():
    gb = Gradebook()
    gb.add_grade(90)
    gb.add_grade(83)
    gb.add_grade(97)
    gb.add_ec(10)

    print(gb.average())

main()
```

If your definition of `Gradebook` is correct, this should print `93.33333333333333`.

Specifications:

To earn a **Complete** on this assignment, you need to:

- All five problems are attempted
- Four of the five are completely correct; the fifth has a minor, non-obvious error not easily detectible through simple testing.
- There are no `return` vs `print` issues.

To earn a **Partially Complete**, you need to:

- At least four of five are attempted
- At least two are completely correct
- At least one class problem has a correct `__init__` method
- There are no `return` vs `print` issues.