Only the last problem has a corresponding portion in the repository this week. You will need to scan and upload the pdf for just the first problem this week. Do not forget to adjust the README to indicate you have completed the assignment before your final commit!

Get Assignment link: https://classroom.github.com/a/dp0c8byZ

1. Below are three snippets of code along with what their author was hoping to achieve. For each determine if an error would occur, if the output would not be as desired, or if nothing is wrong. If an error is occurring or the output would not be as desired, write down how you could fix the problem while still meeting their desired objective.

   (a) **Objective:**
   To construct a list of all integers less than or equal to 100 which are divisible by both 2 and 5 but not by both 2 and 6

   Code:

```python
div_by_two = [i for i in range(1,101) if i % 2 == 0]

div_by_two_five = div_by_two
# Looping backwards to not mess up indices
for i in range(len(div_by_two_five)-1,-1,-1):
    if div_by_two_five[i] % 5 != 0:
        div_by_two_five.remove(div_by_two_five[i])

div_by_two_six = div_by_two
# Looping backwards to not mess up indices
for n in div_by_two_six[::-1]:
    if n % 6 != 0:
        div_by_two_six.remove(n)

good_vals = []
for n in div_by_two_five:
    if n not in div_by_two_six:
        good_vals.append(n)
print(good_vals)
```

(b) **Objective:**
To construct a dictionary with user names as keys and corresponding ages.

Code:

```python
names = ['Sally', 'Barry', 'Barney', 'Arnold']
ages = [34,45,23,45,25]

age_dict = {}
for i,name in enumerate(names):
    age_dict[name] = ages[i]
print(age_dict)
```

(c) **Objective:**
To generate a list of prime numbers less than 100 by removing all numbers which have a divisor from a list.

Code:

```python
def is_prime(num):
    i = 2
    while i ** 2 <= num:
        if num % i == 0:
            return False
        i += 1
    return True

nums = [x for x in range(2,100)]
for x in nums:
    if not is_prime(x):
        nums.remove(x)
print(nums)
```

2. In this problem we'll be implementing a version of a little word game. There are a lot of moving parts in this problem, but many of them have been broken up into smaller portions. Read the instructions, read the docstrings, and THEN start working on various portions of the problem.

   The basic design of the game is as such:

   Playing:

   - A player is dealt a hand of `HAND_SIZE` letters of the alphabet, chosen at random. The player is ensured about one-third of the letters will be vowels, and letters can appear multiple times.
   - The player arranges the hand into as many words as they can out of the given letters, but using each letter only once.
   - Not all letters need to be played (and in some cases it might prove impossible). The size of the hand when each word is played though factors into its score.

   Scoring:

   - The score for the hand is the sum of the score for each word formed
   - The score of a word is the product of two components:
     - The sum of the points for the letters in the word
     - Either

       $$7 \times (\text{ word length }) - 3 \times (n - \text{ word length }) \qquad \text{or} \qquad 1$$

       whichever is larger. Here n is the number of cards in the hand when the word is formed.
   - Words are scored as in Scrabble, with A worth 1 point, B worth 3 points, etc. A dictionary has already been provided that gives the score for each letter of the alphabet.
   - Examples:
     - In a hand of 6, say the word played was "weed". This would result in 176 points:

       $$(4 + 1 + 1 + 2) \times (7 \times 4 - 3 \times (6 - 4)) = 176$$
     - In a hand of 7, say the word "it" was played. This would result in 2 points:

       $$(1 + 1) \times (1) = 2$$

       since $7 \times (2) - 3 \times (7 - 2) = -1$.

   This problem is structured so that you will complete a number of modular functions which you can then glue together to form the scaffolding for the entire game. Test each function you are finish it! Catching small errors in the functions as you go will save you lots of time in debugging later.

I have provided all the code necessary for the functions `load_words` and `deal_hand`. Below I'll attempt to walk you through what in necessary in each of the other functions and the order you should likely approach them.

(a) **Word scores:** One of the more fundamental parts of this game in the necessity to calculate a score for each submitted word. The function `get_word_score` is dedicated to this purpose. Follow the scoring conventions given above (and also in the docstring). You can use the `LETTER_VALUES` dictionary defined at the top of `Prob2.py` to get the necessary letter scores for the first portion of the score. Keep things general here, so n is the total number of letters in the hand when the word was entered. Also, make sure to convert the case of whatever word is entered to lowercase, since that is what form all the letters in the dictionary are in.

(b) **Representing hands:** All hands in the game are represented as a list of single character strings. For example, one hand of 5 letters might be:

```
hand = ['a', 'e', 'f', 's', 't']
```

Printing the letters to the screen in this format is not particularly nice however, so the function `display_hand` exists to take any hand and print its contents to the screen in a more human readable and friendly way. The function `display_hand` will *not* return anything, it is just a shorthand way to print the necessary characters to the screen when we want them.

The function `deal_hand` I have already done, and takes care of creating a semi-random hand of the desired number of letters. Semi-random in this case because we do take steps to ensure about a third of any starting hand is comprised of vowels. This function takes care of creating each initial hand and returning it in the above specified format.

The function `update_hand` is your responsibility. Each time the user plays a word, whether the word is valid or not, those letters are removed from their pool of available letters. The function `update_hand` is supposed to take an provided hand and played word and return a *new* hand which contains only the left-over letters from the initial hand. Be careful that your function does **not** modify the existing hand, but instead returns a **new** hand.

(c) **Valid words:** A player is free to type in anything for the word they want to make, but only certain words are actually found in the dictionary and are made up of letters in the players hand. The `is_valid_word` function (which is your responsibility) exists to check both these conditions. If the word is made up only of letters in the hand and also exists in the list of valid english words, then True should be returned, otherwise return False.

(d) **Playing a hand:** Now you can begin writing code that finally interacts with the player. The `play_hand` function is responsible for playing a single hand, which will likely involve multiple words. The hand should end whenever there are no letters left for the player to utilize or the player enters "!!". The docstring here walks you through more the individual steps of what you'll likely want to compute, display, and request. A round might look something like:

```
>>> play_hand(hand, word_list)
c s t h a u i
Enter a word, or "!!" to indicate that you are finished: cats
cats earned 114 points! Total: 114 points
h u i
Enter a word, or "!!" to indicate that you are finished: hi
hi earned 55 points! Total: 169 points
u
Enter a word, or "!!" to indicate that you are finished: !!
Total score is: 169
```

or

```
>>> play_hand(hand, word_list)
e a g p p i n
Enter a word, or "!!" to indicate that you are finished: ping
ping earned 133 points! Total: 133 points
e a p
Enter a word, or "!!" to indicate that you are finished: pea
pea earned 105 points! Total: 238 points
Total score is: 238
```

(e) **Playing the game:** Finally, each game consists of multiple hands. The `play_game` function implements this overall functionality. The user should be prompted for how many hands they want to play and then the `play_hand` function should be called the correct number of times. Moreover, you want to keep track of an overall score across multiple hands which will be output upon the conclusion of the game.

An example of a game might look like:

```
>>> play_game(word_list)
How many hands do you want to play? 2

---------------------------Round 1---------------------------
l s u p i z o
Enter a word, or "!!" to indicate that you are finished: soul
soul earned 76 points! Total: 76 points

p i z
Enter a word, or "!!" to indicate that you are finished: zip
zip earned 294 points! Total: 370 points
Total score is: 370

---------------------------Round 2---------------------------
o a o q g b l
Enter a word, or "!!" to indicate that you are finished: boo
boo earned 45 points! Total: 45 points

a q g l
Enter a word, or "!!" to indicate that you are finished: lag
lag earned 72 points! Total: 117 points

q
Enter a word, or "!!" to indicate that you are finished: !!
Total score is: 117
Your overall score was: 487!
```