Problems 2 and 3 on this assignment have corresponding code elements in the repository this week. You will need to scan and upload the pdf for Problem 1 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/tPTllkFo

1. Below are three snippets of code along with what their author was hoping to achieve. For each determine if anything is wrong. If something is wrong, write down how you could fix the error while still meeting their desired objective.

   (a) **Objective:**

   To count the number of times a word in a list of words shows up in a different list of words.

   Code:

```
1  def count_words(all_words, to_count):
2      def is_word(all_words, word):
3          global count
4          for a_word in all_words:
5              if a_word == word:
6                  count += 1
7
8      count = 0
9      for word in to_count:
10         is_word(all_words, word)
11
12 all_words = ('My', 'name', 'is', 'Bob', 'and', 'I',
13              'like', 'bacon', 'and', 'burgers')
14 print(count_words(all_words, ('is', 'and')))
```

(b) **Objective:**

To count the total number of times the letter `'a'` appears in 'File.txt'.

Code:

```
def count_letters(string, letter, count):
    global count
    for L in string:
        if L == letter:
            count += 1

count = 0
f = open('File.txt', 'r')
for line in f:
    count_letters(line.strip(), 'a', count)
f.close()
print(count)
```

(c) **Objective:**

Count the number of times the `fib()` function is called in the course of computing the 15th Fibonnaci number.

Code:

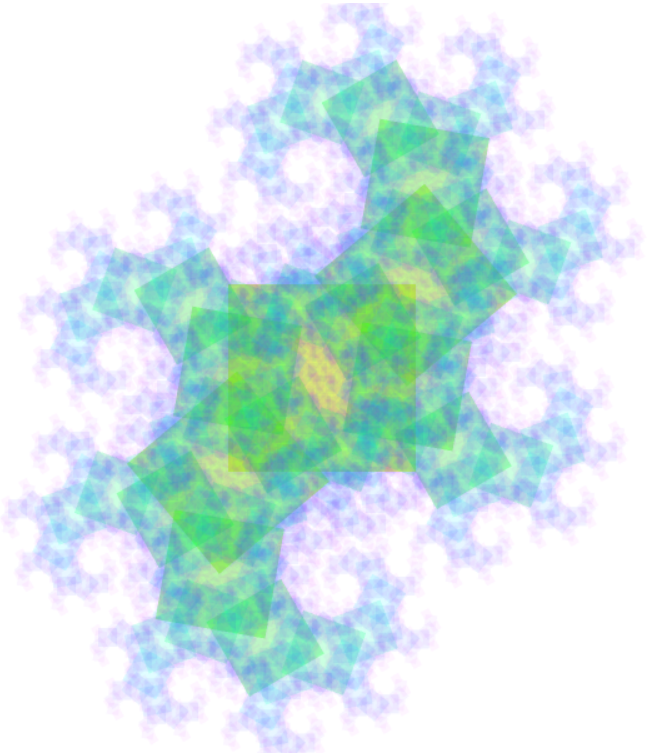
```
def fib_sum(max_n = 15):
    def fib(n):
        global fsum
        fsum += n
        if n == 1 or n == 2:
            return 1
        else:
            return fib(n-1) + fib(n-2)

    global fsum
    fsum = 0
    fib(max_n)
fib_sum(15)
print(fsum)
```

2. We saw last Friday how one could use recursion to make a complicated graphic like the Sierpinski Triangle. Now it is your turn!

   Using arcade, write a recursive function which will repeatedly call a function to draw a repeating pattern on the screen. You are free to be more abstract with this than in the past. If you want some interesting ideas to try to implement, you could look here, which has a nice gallery. Since we have thus far only looked at how to draw primitive shapes in Arcade, you'll be limited to those unless you want to look up how to include Sprites (images) (which we will be getting to this semester, fret not!). The only requirements I put on your code are:

   - It needs to render in under a minute on your computer. Details are nice in recursive drawings like these, but I can't wait 30 minutes to see your final image!
   - Your recursive function should take some arguments that change slightly each recursion. Maybe the position charges, or the size, or the color, all any combination of all of the above.
   - Include docstrings on any functions you write.
   - Fill out the top comment block describing to me what you were aiming for with your code, and whether you were able to reach your vision.

   My example that I came up with while preparing this assignment is below, and is comprised purely of rotated boxes drawn at the opposite corners of each previous box.



3. We have worked with prime numbers a few times already, and you should already have a function available to determine if a particular number is prime. Interestingly, any natural number can be broken down into a unique product of prime numbers. For instance, 10 can be written as 2×5 or 12 written as $2 \times 2 \times 3$. For this problem, write a function that returns a tuple of the prime factors of any provided number. Your function should take advantage of your previously written `is_prime` function for checking if certain values are prime and thus whether to add them to the tuple of prime factors. Make sure your function correctly finds multiples of the same prime factor (such as the two 2's found for 12 above). There are plenty of sources on the web for how to find prime factors. I'm looking for *YOUR* version here and one that uses your previous `is_prime` function.

   Hints:

   - *When finding prime factors you essentially want to check two things:*
     - *Does the number evenly divide your number in question? (ie. is it actually a factor?)*
     - *Is the factor prime?*
   - *If both conditions are met, then that number is a prime factor! It should be added to the tuple list and you should divide your original number by the factor to continue finding new factors. (You want to find things that multiply together to get the desired number. So once you find one, dividing the original number by that ensures that going forward you'll keep finding prime factors that work.)*