MTH 3300
Spring 2018
E. Fink
November 3, 2018

Homework 8

Directions: Download the template files I have provided on Blackboard. Then open Spyder, load these template files, and write the following programs. Submit your source code to me via Blackboard, in `.py` format; do NOT send any other files. READ THE INSTRUCTIONS on how to submit your work in the Course Documents section of Blackboard.

Be sure to read the SPECIFICATIONS carefully for all problems! And write comments!

---

1) abc.py

The *abc conjecture* is a hot topic in mathematics right now. A mathematician named Shinichi Mochizuki claimed to have proven it in 2012, but unfortunately, the proof is so complicated that there are only a very small number of people on the planet who have read it and understand it (less than 100?), and many top mathematicians who have read it suspect that at least one part is nonsense. Two of these mathematicians went to Kyoto in March just to talk to Mochizuki about his proof, but Mochizuki and the visitors couldn't come to an agreement on whether or not the proof was flawed. The two visitors wrote a paper saying, essentially, "this proof is incomplete." Mochizuki accused the two of fundamental misunderstandings. Drama!

The abc conjecture is related to triples of numbers a, b and c such that $a + b = c$ and $GCD(a, b) = 1$. Imagine you take all the prime numbers that are factors of either a, b or c, and multiply them together. The abc conjecture states, roughly, that this product – which we'll call the *radical* of a, b and c – is usually greater than c. (I'm leaving out some details here.)

For example: consider $a = 4$, $b = 45$, and $c = 4 + 45 = 49$. The primes that go into the numbers 4, 45 and 49 are: 2 (which goes into 4), 3 and 5 (which go into 45), and 7 (which goes into 49). The radical of $4, 45$ and 49 is the product of the primes, which is $2 \cdot 3 \cdot 5 \cdot 7 = 210$. The radical here is way bigger than $c = 49$.

Consider $a = 5, b = 27$, and $c = 5 + 27 = 32$. The primes that go into the numbers 5, 27 and 32 are $5, 3$ and 2, so the radical of $5, 27$ and 32 is $5 \cdot 3 \cdot 2 = 30$. This is a rare exception to the rule: the radical 30 is *less* than $c = 32$.

In this problem, you are going to verify that the rule "radical $> c$" does indeed almost always hold for a collection of numbers that I have in a file.

Your first mission is to create a function called `is_prime()`. This function should take an integer as input, and return `true` if it has exactly two factors, and `false` otherwise. (Remember that 1 is not prime!) In my `abc.py` file, I have put some test code, so that you can see if your function is working properly – all the lines should say `LOOKING GOOD!`, and none of the lines should say `ERROR!`.

Once your function is working, delete my tests. Your *next* mission is to create a function called `radical()`. This function should take three positive integers as input, and should return the radical of those three numbers. The program should identify every prime number that goes into any of the three arguments, multiply them together, and return the product. I suggest that for each integer between 2 and c, you check if that integer is a prime which is a factor of a, b or c.

This function should contain a call (or calls) to `is_prime()`!.

Finally, open my file `ab.txt`, which has 200 lines, each of which consist of an a and a b. For each line, I want you to print out a, b and $GCD(a, b)$ (you may use my GCD function, which I have included in the code). If $GCD(a, b) = 1$, calculate $c = a + b$ and the radical of a, b and c. The first few lines of your output should look like

```
a: 104, b: 33, GCD: 1, c: 137, Radical: 117546
a: 3, b: 105, GCD: 3
a: 33, b: 146, GCD: 1, c: 179, Radical: 862422
a: 96, b: 153, GCD: 3
```

Note that only the lines where the GCD is 1 get printed.

Finally, at the end, you should print out

$$\frac{\text{number of lines where } GCD(a,b) = 1 \text{ and radical } > c}{\text{number of lines where GCD(a,b) = 1}}.$$

Let me emphasize: use the functions thoughtfully! The point of functions is to break down your code into easy-to-comprehend/write/test bits. Take advantage of this, and don't rewrite the same code repeatedly!

Specifications: your program must

- write a function `is_prime()`, which should accept a positive integer as input and return `True` if it is prime, `False` if it is not. You need not concern yourself with the behavior for non-positive integers.
- write a function `radical()`, which should accept three positive integers as input, and return the product of all prime numbers which go into any of the inputs. You may assume, if you like, that the third input is the largest. This function should call `is_prime()`!
- open a file named `ab.txt`, and produce a print out as shown above: for each line in the file, if the two numbers a and b have GCD equal to 1, show their sum c, and the radical of a, b and c, utilizing the `radical()` function.
- print out the fraction $\dfrac{\text{number of lines where } GCD(a,b) = 1 \text{ and radical } > c}{\text{number of lines where GCD(a,b) = 1}}$.

***Challenge**: read this article* *https://www.nature.com/news/peculiar-pattern-found-in-random-prime-numbers-1.19550. The second section,* Not so random*, describes a surprising recent observation about the last digits of consecutive primes. Write a program that will confirm these observations – only look at the primes less than 10,000,000.*

---

2) scrabble.py

Which word in the English dictionary is worth the most points in Scrabble?

In `scrabble.py`, I have included a dictionary called `points` which contains the point value for each letter. The value of a word is gotten by taking each letter, figuring out its corresponding point value in the dictionary, and adding up all the points.

In `words_eng.txt`, there is a fairly exhaustive list of English words (each word contains only letters – no apostrophes or hyphens or spaces). Write code which reads the file, computes the Scrabble value for each word, and keeps track of the highest valued-word.

Depending on how you read the `words_eng.txt` file, you may end up with all your words having a newline character at the end. One way to fix this is `.strip()`, which words as follows: if `x` is a string variable, then `y = x.strip()` assigns `y` to be the same string, except with the newline character at the end removed if there is one.

Specifications: your program must

- open and read `words_eng.txt`.
- compute the Scrabble value for each word, using the dictionary `points`, by adding the point values corresponding to each letter in a word.
- display the word with the highest point value. (There is in fact a single highest-valued word.)

***Challenge**: have your program allow the user to enter seven lowercase letters, and then print out the highest-valued valid word that can be made out of those letters (in any order, using each letter at most once).*