Assignment #2

Math 1800: Mathematical Programming in Python

Instructions: Choose 3 of these problems. Some are hard, some easy. Some are boring, and some are interesting. As your answer, submit Python text files, with the extension `.py`. Each file should include your name and the problem number.

- *Problem 2.0:* Use Python commands to determine which of the following 8 integers are prime:

$$31; 331; 3331; 33331; 333331; 3333331; 33333331; 33333331.$$

- *Problem 2.1:* Euler announced that 1,000,009 is a prime number. Was he right? Use Python to investigate his claim.

- *Problem 2.2:* Euler found a formula $p(n) = n^2 + n + 41$ which he suspected would produce many prime values. Write a function called *euler41(n)* which evaluates this formula for any value of n. Start with $n = 0$, for which $p(n) = 41$, which is prime. Increase n by ones, evaluate $p(n)$. Keep increasing n until the formula $p(n)$ returns a nonprime (composite) value. What are the values of n and $p(n)$ when this happens?

- *Problem 2.3:* We have discussed four versions of an `is_prime()` function, trying to make an efficient one.

  1. The basic code
  2. The code with a `break` statement
  3. The code that only goes up to $\sqrt(n)$
  4. A code that skips divisors that are multiples of 2 or 3. See the Wikipedia page on `Primality_test`

  Modify each function so that it counts the number of times the modulo function ($n\%d$) is used. Run each function on the input $n = 27644437$ and report how much work was done, that is, how many times the modulo function was called. Does each code on the list seem more efficient than the previous ones?

- *Problem 2.4:* In number theory, the prime factorization of an integer n can be written as a product of k prime numbers p_i raised to exponents e_i:

$$n = p_1^{e_1}\, p_2^{e_2} \ldots p_k^{e_k}$$

  Rather than use exponents, we can simply write a repeated factor several times. Thus, $740 = 2 \times 2 \times 5 \times 37$ and $337 = 337$ (it's a prime!). Write a Python program `factor(n)` which accepts a number n and prints out the prime factorization. This is actually a tricky program to write. Your best bet would be to use a `while()` statement, something like this:

```
for every divisor from 2 up to and including n (careful here)!
  while n is bigger than 1
    if n is divisible by i
      print i
      divide n by i
```

  Note that when you divide n by i, you want to write `n = n // i`, otherwise Python will make the result a real number and mess everything up! Use your program to factor the three numbers $n = 1120, 2023, 314159265$.

- *Problem 2.5:* In number theory, the function $\pi(n)$ counts all the primes less than or equal to the integer n. Thus, $\pi(10) = 4, \pi(11) = 5, \pi(12) = 5$. Write a Python function *prime_pi(n)* that can evaluate $\pi(n)$ for any integer n. Compute $\pi(n)$ for each of the three values $n = 64, 256, 1024$.

- *Problem 2.6:* Although he tried, Gauss never found a simple formula for $\pi(n)$. Instead, he ended up looking for mathematical approximations. One such estimate is $\pi(n) \approx \frac{n}{\log(n)}$. For instance, $\pi(128) = 31$ while $\frac{n}{\log(n)} = 26.38....$ The relative error is $\frac{31-26.28}{31} \approx 0.15$. Make a table of the relative error of Gauss's approximation as n increases; you might look at $n = 256, 512, 1024$ and a few more values, and decide if Gauss was on the right track.

- *Problem 2.7:* In number theory, the function $\sigma(n)$ counts the sum of all the distinct divisors of n, including 1 and n itself. Thus, $\sigma(10) = 18, \sigma(11) = 12, \sigma(12) = 28$. Write a Python function *sigma(n)* that can evaluate $\sigma(n)$ for any integer n. Compute $\sigma(n)$ for each of the four values $n = 617, 816, 1000, 1024$.

- *Problem 2.8:* In number theory, the function $\tau(n)$ counts all the divisors of the integer n, including 1 and n itself. Thus, $\tau(10) = 4, \tau(11) = 2, \tau(12) = 6$. Write a Python function *tau(n)* that evaluates $\tau(n)$ for any integer n. Compute $\tau(n)$ for each of the four values $n = 521, 610, 832, 960$.

- *Problem 2.9:* In number theory, the function known as $gpf(n)$ returns the greatest prime factor of the integer n, that is, the largest prime p that is a factor of n. Thus, $gpf(12) = 3, gpf(13) = 13, gpf(14) = 7$. Write a Python function `gpf(n)` that evaluates the greatest prime factor for $n = 25, 698, 751, 364, 526$.

- *Problem 2.10:* Verify the formula $key = p_1 * p_2$ for the RSA example. Your program will be very simple, having the form

  ```
  p1 = ...
  p2 = ...
  key = ...
  p1p2 = p1 * p2
  diff = key - p1p2
  print ( 'diff = ', diff )
  ```

  The difference should be zero (unless you or I made a typing mistake in listing the values!) I am asking you to do this mainly to convince you that integer arithmetic in Python allows you to work with very large values. This is **not** the case in most other computer languages.