Homework 0. *Due 11:59pm Fri Sept 1, via Gradescope.com.*

(1) *[1 point]* Fun fact about yourself.
(2) *[1 point]* What do you like most about computers?
(3) *[1 point]* What do you want to do after you graduate?

Prime numbers are fundamental to computer science. They are the "atomic" integers in the sense that all other integers can be built up from them by multiplication.

Recall that a *prime number* is an integer ≥ 2 which cannot be written as a product of two smaller positive integers. For example, 7 is prime because it's impossible to write $7 = m \cdot n$ with m and n positive integers < 7. But 8 is composite because $8 = 2 \cdot 4$.

(4) *[10 points]* Write a computer program (in Python) which checks, by brute force, whether a given integer n is prime by checking that n is not divisible by any integer d less than n but greater than 1. In other words, by checking that n/d is *not* an integer for each d with $1 < d < n$. [You can test whether d divides n in Python by evaluating `n%d==0`. Here `n%d` returns the remainder of the division n/d, and `==` returns `True` if its two arguments are equal and otherwise returns `False`.]
[There are various shortcuts you can use to speed up your code: for example, do you really have to check every possible integer d with $1 < d < n$? Later this semester, we will learn a ***much*** *faster method for primality testing.]*

Your program should be named `is_prime.py` and define a function `is_prime(n)` which, given an *integer* `n`, returns `True` (if n is prime) or `False` (otherwise). Your program should use this function to read integers from `stdin` (printing a `>` prompt using for example

```
int(input('> '))
```

(beware of the space after the `>`) and write `True` (if prime) or `False` (otherwise) to `stdout` for each. Here is a sample transcript (alternating input/output):

```
> 2
True
> 4
False
> 7
True
> 8
False
```

```
> 13
True
> 15
False
> 7919
True
```

Your program must follow this format exactly in order to pass the autograder's tests.

So that your code can be loaded as a library without getting stuck in an infinite loop, your main input/output loop should be wrapped within an if statement like

```
if __name__ == "__main__":
    # main loop
```

(5) *[10 points]* Recall the Fundamental Theorem of Arithmetic: every integer n greater than 1 can be represented uniquely as a product of prime numbers

$$n = p_1 p_2 \cdots p_r$$

up to the order of the factors. So if we sort the factors in, say, ascending order $p_1 \leq p_2 \leq \cdots \leq p_r$, then the list of factors

$$[p_1, p_2, \ldots, p_r]$$

is unique.

Write a computer program (in Python) which computes this prime factorization, by searching (in ascending order) for integers d less than n but greater than 1 which divide n.

Your program should be named `prime_factorization.py` and define a function `prime_factorization(n)` which, given a *positive integer* **n**, returns the (unique, ascending) prime factorization of n, as a Python array. Your program should use this function to read integers n from `stdin`, printing a `>` prompt using for example

```
int(input('> '))
```

and write the prime factorization of n to `stdout` for each. Here is a sample transcript (alternating input/output):

```
> 3
[3]
> 4
[2, 2]
> 6
[2, 3]
> 8
```

```
[2, 2, 2]
> 22
[2, 11]
> 341234
[2, 61, 2797]
> 456234532
[2, 2, 13, 23, 381467]
> 3412341325
[5, 5, 43, 193, 16447]
```

So that your code can be loaded as a library without getting stuck in an infinite loop, your main input/output loop should be wrapped within an if statement like

```
if __name__ == "__main__":
    # main loop
```

(6) *[10 points]* Write a computer program (in Python) which computes the nth prime for any positive integer n. For example, the first prime ($n = 1$) is 2, the second prime ($n = 2$) is 3, the fourth prime ($n = 4$) is 7, etc.

Your program should be named `nth_prime.py` and define a function `nth_prime(n)` which returns the nth prime number. Your program should use this function to read integers n from `stdin` (printing a `>` prompt using `int(input('> '))`) and write the nth prime to `stdout` for each. Here is a sample transcript (alternating input/output):

```
> 1
2
> 4
7
> 10
29
> 20
71
> 1000
7919
```

So that your code can be loaded as a library without getting stuck in an infinite loop, your main input/output loop should be wrapped within an if statement like

```
if __name__ == "__main__":
    # main loop
```

[Hint: You can speed up your code by memoizing (caching) the list (or, perhaps better, the dictionary) of your program's known primes and using these cached primes to more quickly test new candidate primes.]