Figurate Numbers Project

Purposes of this assignment

- To familiarize you with functions
- To get you started pair programming

General Idea

In this assignment you will write a number of functions for testing characteristics of positive integers, plus one special function named **main**. Each function other than **main** will take *any* single positive integer as an argument, test whether it has some property, and return a value of either **True** or **False**.

Details

The run() function

In the **run** function, define a variable **limit = 100** and test each of the numbers **1** through **limit**, inclusive. Do not use the number 100 elsewhere; use the variable **limit** instead. This makes the program easier to change if you later want some limit other than 100.

The **run** function will call each of the other functions, for each of the numbers 1 through **limit**, to determine the properties of those numbers. It will then print out each number, one per line, along with a list of its properties (on the same line).

Your output should look approximately like this:

```
1: composite, triangular, square, not prime oblong
2: prime, not triangular, not square,  not prime oblong
3: prime, triangular, not square, not prime oblong
4: composite, not triangular, square, not prime oblong
5: prime, not triangular, not square, not prime oblong
 . . .
```

The **run** function doesn't return a value. Each of the other functions is a **predicate**, that is, a function that returns **True** or **False**.

The is_prime(n) function

A positive number is **prime** if its only positive divisors are itself and one. For example, **7** is prime because it is evenly divisible by **1** and **7**, but not by any number in between (in particular, it is not evenly divisible by **2**, **3**, **4**, **5**, or **6**).

A positive number is **composite** if it is not prime. For example, **10** is composite because it is evenly divisible by **2** and **5**; **12** is composite because it is evenly divisible by **2**, **3**, **4**, and **6**. As a special case, **1** is considered to be composite.

This function should return **True** if its argument is a prime number, and **False** otherwise. Similar statements hold for the other functions.

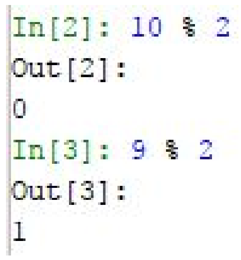
Important Lesson from Aaron: “the mod operator (%)”

How do we determine if n is prime? The most basic way is trial division. You know, just divide n by every number up to n - 1. That would be extremely tedious though, wouldn’t it? Not to mention, if limit was some massive number, your computer would really slow down and possibly never be able to calculate if n is prime.

So we can save time and processing power by using a bit of math sense. Instead of checking all numbers 1 to n - 1, we’ll check all numbers from 2 to the square root of n. Think about why…

The procedure would be to divide n by every number from 2 all the way up to sqrt(n). If we don’t find a number in this range that evenly divides n, then n is prime.

We won’t do “normal” division, though. We’ll do something call “mod”, which gives us the remainder of one number divided by another. In Python and many other languages we use the “%” symbol as the “mod operator”, in the following way:



```
In[2]: 10 % 2
Out[2]:
0
In[3]: 9 % 2
Out[3]:
1
```

As you may remember from elementary school math (or may not, don’t feel too guilty), 10 divided by 2 gives you 5 with a remainder of 0. Therefore, we say 10 % 2 equals 0. While division will give us the quotient, the mod operator gives us the remainder.

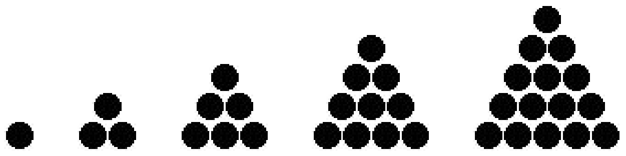
Even with this strategy, we’ll wind up testing more numbers than we need to. So another way we could save time is to use certain tricks. For example, we could skip all even numbers greater than 2, since if 2 does not divide n, none of the other multiples of 2 will divide n either. Can you think of any other tricks?

By the way, the task of determining the primality of very large numbers is one of the toughest in CS. It is also the core of certain encryption algorithms. You’ll learn more about this next summer.

Anyway, back to the project. ^_^

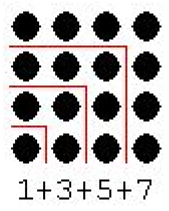
The is_triangular(*n*) function

A **triangular number** is a number of the form **1 + 2 + 3 + ... + n**, for some positive **n**. These numbers are called triangular because, if you have that many objects, you can arrange them in an equilateral triangle (see figure).
The first few triangular numbers are **1, 3, 6, 10, 15**....



The is_square(*n*) function

A **square number** is a number of the form **1 + 3 + 5 + 7 + ... + n**, for some odd positive **n**. (The figure should help you understand why this definition is the same as the usual definition of square numbers.)



The is_prime_oblong(*n*) function

Prime oblong numbers are those that have exactly two different prime divisors. Thus, 10 (2 * 5) is prime oblong, 12 is not (too many divisors), 25 is not (two equal prime divisors).

Done?

At the end of your Python program (after all your function definitions), insert the following line:

run()

And click the green "run" button in your PyCharm IDE (looks like a "play" button). Ask your classmates and teacher if you can't find it.

This project is adapted for SMASH from the work of my professor and friend, David Matuszek: http://www.cis.upenn.edu/~matuszek/cit591-2014/Assignments/02_figurate_numbers.html