CIT 590: Fall 2018
Homework 2

HW deadline as per Canvas.

This homework deals with the following topics:

- loops
- functions

This HW deals primarily with loops and functions. It will introduce you to some interesting number theory as well. You have probably heard of prime numbers and composite numbers before, but have you ever heard of abundant numbers, or narcissistic numbers? This assignment will ask you to write code to identify different kinds number properties.

We also want you to learn how to reuse code in this assignment. This is a basic strategy for reducing complexity in a program. Why? Because things change, and if you have the same thing in several places, when you change one, you have to change all the others. And you can't always find them all! In order to reuse code, you must take common pieces of code and put them into functions. Failure to do so will result in loss of points.

You will submit just one file for this assignment called *numberProperties.py*. We also want you to add comments to your code and docstrings to your functions.

You can assume that all inputs in this assignment will be positive integers.

Background on Number Theory

You'll use the following number theory properties in this assignment.

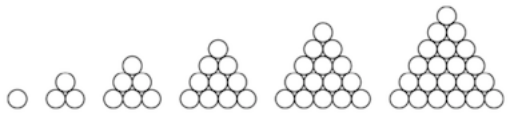
Prime number - A number with exactly 2 factors -- 1 and itself. For example, 2 is a prime number because its only 2 factors are 1 and 2. 41 is also a prime number because it can only be divided by 1 and 41.

Composite number - A number with more than 2 factors. For example, 9 is a composite number because its factors are 1, 3, and 9. Please note that number 1 is neither prime nor composite.

Perfect number - A number is said to be perfect if it is equal to the sum of all its factors (for obvious reasons the list of factors being considered does not include the number itself). 6 = 3 + 2 + 1, hence 6 is perfect. 28 is another example since 1 + 2 + 4 + 7 + 14 is 28. Please note that the number 1 is not a perfect number.

Abundant number - A number is considered to be abundant if the sum of its factors (aside from the number) is greater than the number itself. For example, 12 is abundant since 1 + 2 + 3 + 4 + 6 = 16 > 12. However, a number like 15, where the sum of the factors is 1 + 3 + 5 = 9, is not abundant.

Triangular number - The triangular number T_n is a number that can be represented in the form of a triangular grid of points where the first row contains a single element and each subsequent row contains one more element than the previous one (see figure below).

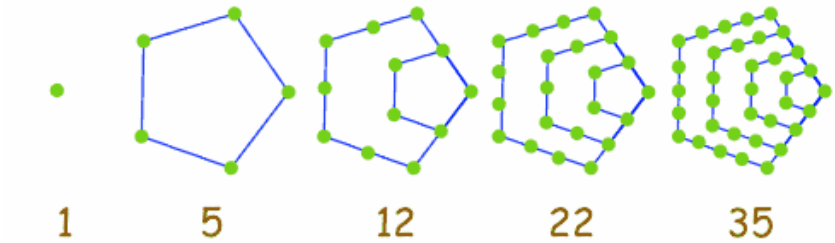


For the purposes of this assignment, we use the fact that the nth triangular number can be found by using the following formula:

$$T_n = \frac{n(n+1)}{2}$$

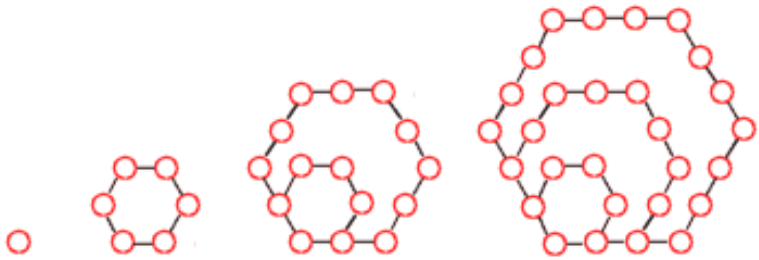
For example, 3 is the 2nd triangular number, since 2 (3) / 2 = 3. As another example, 15 is the 5th triangular number, since 5 (6) / 2 = 15. However, 14 is not triangular, since there is no x such that x (x + 1) / 2 = n.

Pentagonal number - Every pentagonal number is 1/3 of a triangular number. More specifically, the nth pentagonal number is one-third of the 3n − 1th triangular number.



For example, 5 is the 2nd pentagonal number, since 3 (5) = 15, which is the 5th triangular number, and 5 = 3 (2) – 1. 22 is the 4th pentagonal number, since 3 (22) = 66, which is the 11th triangular number, and 11 = 4 (3) – 1.

Hexagonal number - A number that can be expressed in the form $2n^2 - n$. The nth hexagonal number will be the number of points in a hexagon with n regularly spaced points on a side as shown in the figure below.



For example, 6 is the 2_{nd} hexagonal number, since 2 (2 ** 2) - 2 = 6. 15 is the 3^{rd} hexagonal number, since 2 (3 ** 2) - 3 = 15.

Narcissistic number - A positive integer is called narcissistic if it is equal to the sum of its own digits each raised to the power of the number of digits. For example, 153 is narcissistic because $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$. Note that by this definition all single digit numbers are narcissistic.

The Assignment

Write a Python program with at least the following functions. We want you to demonstrate your understanding of code reusability so you should write a few others -- we call these helper functions. For example, you might want to write a function that calculates and returns all the factors of a given number.

Please name the required functions exactly as they are named below. We are going to use automated tests for checking them and any incorrect spelling will result in a loss of points.

isPrime(x) - function that returns whether or not the given number x is prime. This function returns a boolean.

isComposite(x) - function that returns whether or not the given number x is composite. This function returns a boolean. We intend for you to reuse code from the prime number function here.

isPerfect(x) - function that returns whether or not the given number x is perfect. This function returns a boolean.

isAbundant(x) - function that returns whether or not the given number x is abundant. This function returns a boolean.

isTriangular(x) - function that returns whether or not a given number x is triangular. This function returns a boolean.

isPentagonal(x) - function that returns whether or not a given number x is pentagonal. This function returns a boolean. We intend for you to reuse code from the triangular number function here.

isHexagonal(x) - function that returns whether or not a given number is hexagonal. This function returns a boolean.

isNarcissistic(x) - function that returns whether or not a given number is Narcissistic. This function returns a boolean.

Define and implement these functions, along with a *main* function, to get a complete program. We suggest commenting out sections of the *main* function in order to check each function that you are writing.

To get you started, copy the code below by typing it into a file named *numberProperties.py*. Actually typing it will help you understand how a program's *main* function works.

```
def main():

    playing = True
    while playing == True:

        num_input = input('Give me a number from 1 to
10000.  Type -1 to exit. ')

        try:
            num = int(num_input)

            if (num == -1):
                playing = False
                continue

            if isPrime(num):
                print(str(num) + ' is prime')
            if isComposite(num):
                print(str(num) + ' is composite')
            if isPerfect(num):
                print(str(num) + ' is perfect')
            if isAbundant(num):
                print(str(num) + ' is abundant')
            if isTriangular(num):
                print(str(num) + ' is triangular')
```

```
            if isPentagonal(num):
                print(str(num) + ' is pentagonal')
            if isHexagonal(num):
                print(str(num) + ' is hexagonal')
            if isNarcissistic(num):
                print(str(num) + ' is narcissistic')

        except:
            print('Sorry, the input is not an int.  Please
try again.')

#Insert these lines at the end of your program (after all
your function definitions)
#They will automatically run your main function and program
if __name__ == '__main__':
    main()
```

Submission

Submit *numberProperties.py*.

Testing Your Program

An excellent resource for numbers that have these interesting properties is the online encyclopedia of integer sequences (OEIS). Go to oeis.org and type in the property you are looking for and you will get a number of test cases for each. For example, to see examples of a perfect number, search for "perfect": http://oeis.org/search?q=perfect&sort=&language=english&go=Search

Evaluation

Correctness - 10 pts
Does the code work as expected?

Code reuse – 5 pts
This is the tricky part of the assignment. Are there common pieces of code that you can separate out into helper functions instead of having them copied in multiple (two ore more) places? If there are, we want you to create these helper functions.

Docstrings/Comments - 5 pts
Does every function have a docstring defined? Does it appropriately describe what the function does? Are all non trivial lines of code commented?

Style – 5 pts
Style includes things like variable names, comments, function names, and general readability of your code.