Homework 2: Number Properties

This assignment is designed to give you practice writing code and applying lessons and topics for the current module.

This homework deals with the following topics:

- Loops
- Functions

About the Assignment

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 of number properties.

We also want you to learn how to reuse code in this assignment, by writing and using functions. 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.

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.

Factor - A factor is a number that divides evenly into another number. For example, 2 is a factor of 4 because 2 divides evenly into 4. 3 and 4 are both factors of 12 because they both divide evenly into 12.

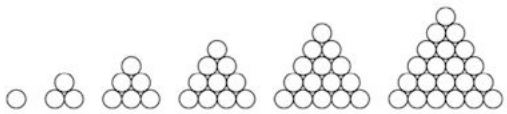
Prime number - A number with exactly 2 factors -- 1 and itself. For example, 2 is a prime number because its only 2 factors are 1 and 2. 41 is also a prime number because it can only be divided by 1 and 41.

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.

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.

getFactors(x) - function that returns a list of factors of a given number x. Basically, finds the numbers between 1 and the given integer that divide the number evenly.

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.

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

Each function has been defined for you, but without the code. See the docstring in each function (in the starter code) for more details on what the function is supposed to do and how to write the code. It should be clear enough. In some cases, we have provided hints and examples to get you started.

For example, we have defined an "isPrime" function for you (see below) which returns whether or not a given number is prime. Read the docstring, which explains what the function is supposed to do. Then write your code where it says "# TODO" to implement the function. ("# TODO" is a special kind of highlighted comment in Python which indicates there is a task to complete or issue that requires attention.) You'll do this for each function in the program.

```
def isPrime(x):
    """Returns whether or not the given number x is prime.

   A prime number is a natural number greater than 1 that cannot be
formed by multiplying two smaller natural numbers.

    For example:
    - Calling isPrime(11) will return True
    - Calling isPrime(71) will return True
```

```
    - Calling isPrime(12) will return False
    - Calling isPrime(76) will return False
    """

    # TODO
```

Please do not change the name of any of the functions. We are going to use automated tests for checking them and any incorrect spelling will result in a loss of points.

The main function has been completely implemented for you (see example snippet below). Use it to run and interact with your program, and to see if your functions are working as expected. Spend some time on testing.

```
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 (num <= 0 or num > 10000):
                continue

            factors = getFactors(num)
            print("The factors of", num, "are", factors)

            if isPrime(num):
                print(str(num) + ' is prime')
```

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

Submission

Open the Jupyter Notebook directly in Coursera (you will find it in the item following this reading). To complete the assignment, complete the provided Jupyter Notebook file, following the detailed instructions in each cell. Test your submission before submitting by following the instructions on the assignment page in Coursera. When you're happy with your solutions, click the 'Submit Assignment' button in the top right.

Evaluation

Points: Each function is worth 1 point.