Lab Programming Assignment #4 (Extra Credit)

Points Possible: 20

Due Date: Tuesday, November 26 by 11:59 pm

This extra credit assignment will go towards the Lab grade category.

Instructions:

You can work in teams of up to three or individually, up to you. Submit one file: **Programming Assignment 4-<Name of Submitter>.py**. Example: Programming Assignment 4-John Doe.py assuming John Doe is the person submitting the file on BeachBoard. Only submission via BeachBoard will be accepted.

That file should contain all of your answers (which should be in Python code).

That file should also be executable and display all of your answers via the following command in the Python command prompt:
exec(open("<Your Filename>.py").read())

Example:
exec(open("Programming Assignment 4-John Doe.py").read())

A part of my testing will be executing that command, so it is important that that command work with your Python file and display the answers.

Be sure to also type in your name and the names of your team members within the file, at the top. When typing your Python code, clearly indicate the question number above it so I know which question you're answering. You should write them in your file as Python comments.

Also, ensure the function names are EXACTLY that of what is listed in the problems below.

Do not use any print statements. Instead, it should use the **return** function.

Failure to do any of the above will result in point deductions. No exceptions.

RSA Encryption:
Write a function that will RSA encrypt a string of text using the same method as mentioned in the Rosen 7th edition book (the "RSA Encryption" sub-section in section 4.6, page 299).

Your function should be named "RSAencrypt" and should take four inputs: p, q, e, and message, where n = p • q.
p = an integer, the p value in the RSA encryption.
q = an integer, the q value in the RSA encryption.
e = an integer, a number that is relatively prime to (*p*-1)(*q*-1). In other words gcd(e, (*p*-1)(*q*-1)) = 1.
message = a string, this is the message that is to be encrypted

Your function should output the following if the message value is "":
"Message length needs to be greater than 0."

Your function should output the following if p or q is not prime:
"Both p and q need to be prime."

Your function should output the following if e is not relatively prime to (*p*-1)(*q*-1), in other words, if gcd(e, (*p*-1)(*q*-1)) ≠ 1:
"e needs to be relatively prime to (p-1)(q-1). "

Additional Items to Note:

1. Per the book, if the size of the last block is not equal to the proper block size, you must pad it with trailing X's.
2. The output of the function should be a list containing the encrypted blocks.
3. Each element (encrypted block) in the outputted list needs to be in the proper block size. Thus, the elements should be strings. This is because some elements should have leading zeros for padding in case their integer values are smaller than the block size. To see an example, see the "Examples" section below.
4. Since modern computers can handle large exponents, no need to use modular exponentiation.
5. You can only import the following: from math import gcd. You cannot import any other library or function (you can use built-in Python functions that don't require an import though).
6. Your solution should output each encrypted block in the correct block size of 2N digits. Note that N is not the same as n. See page 299 of the Rosen book for further details.
7. I will only input upper case letters as the **message** value when I test your code.
8. I will only input positive integers for **p**, **q**, and **e** when I test your code.
9. You can use helper functions (e.g. to check if p and q are prime) and name them anything you want as I will only be testing using the RSAencrypt function name. You must use the exact name of "RSAencrypt" though.
10. In your code, the check to see if p and q are both prime should occur before the check of e being relatively prime to (p-1)(q-1). So if there's a scenario where both exceptions exist, only the prime exception should be outputted ("Both p and q need to be prime.").

Examples:

```
>>> RSAencrypt(43,59,13,"STOP")
['2081', '2182']
>>> RSAencrypt(43,59,13,"ABC")
['0001', '0027']
>>> RSAencrypt(43,59,13,"")
'Message length needs to be greater than 0.'
>>> RSAencrypt(503,509,21,"HELLO")
['019565', '161371']
>>> RSAencrypt(43,14,13,"ABC")
'Both p and q need to be prime.'
>>> RSAencrypt(11,13,10,"EXCELLENT")
'e needs to be relatively prime to (p-1)(q-1).'
```