Cryptography Homework 2b—Modular Arithmetic with Python

Required Reading

Cryptology2 slides

Cracking Codes with Python by Sweigart

Chapter 13, pages 171 - 183 (or https://inventwithpython.com/cracking/chapter13.html) about modular arithmetic.

Optional Reading

Wikipedia has good articles on modular arithmetic, greatest common divisor (GCD), Least Common Multiple (LCM), Euclid's algorithm and Euclid's extended algorithm.

Video

If you want to understand how Euclid's extended algorithm works, this video by Christoff Paar is superb! https://www.youtube.com/watch?v=fq6SXByItUI.

Python Modules

The ability to add modules that perform functions that occur often is a great thing in any programming language; that way you can easily incorporate code that's already written into your code. In Python, you can take a properly written script that's named with the .py extension and import it into Python.

When you import a module into Python, you do not use the .py extension in your Python statement. You use this statement to import the file cryptomath.py:

```
import cryptomath
```

If you import cryptomath that way, you have to preface every function call with cryptomath, for example:

```
cryptomath.gcd(a, b)
```

To lessen the typing load, you can import the file this way:

```
import cryptomath as cm
```

Then call functions this way:

```
cm.gcd(a, b)
```

To lessen the typing even more you can import the functions you want by name:

```
from cryptomath import gcd, lcm, findInverse
```

Then you can call them without a prefix:

```
gcd(a, b)
```

How Python Finds Modules

The statements above, `import cryptomath` for example, will only work if Python knows where to find the file cryptomath.py. Python looks in two places:

- Python's current working directory; this is often the directory you were in when you executed `python3`
- Python's path. Python has its own path, separate from the $PATH variable that is part of the Linux and Windows operating system.

How to Check Python's Current Working Directory and Path

Use the following to check the current working directory

```
import os
os.getcwd()
```

Use the following to see Python's path.

Linux

```
john@ubuntu:~$ python3

Python 3.6.8 (default, Aug 20 2019, 17:12:48)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.sys.path
['', '/usr/lib/python36.zip', '/usr/lib/python3.6', '/usr/lib/python3.6/lib-
dynload', '/usr/local/lib/python3.6/dist-packages', '/usr/lib/python3/dist-
packages']
>>>
```

Windows

```
PS C:\Users\John> python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.sys.path
['', 'C:\\Users\\John\\AppData\\Local\\Programs\\Python\\Python37\\python37.zip',
'C:\\Users\\John\\AppData\\Local\\Programs\\Python\\Python37\\DLLs',
'C:\\Users\\John\\AppData\\Local\\Programs\\Python\\Python37\\lib',
'C:\\Users\\John\\AppData\\Local\\Programs\\Python\\Python37',
'C:\\Users\\John\\AppData\\Roaming\\Python\\Python37\\site-packages',
'C:\\Users\\John\\AppData\\Local\\Programs\\Python\\Python37\\lib\\site-packages']
>>>
```

Note: The version number in the path (python3.6 in Linux or Python37 in Windows) may change.

Changing the Current Working Directory

Idle often comes up in a strange directory, especially in Windows. Here is how to change directories.

Putting the cryptomath library into your Python

Simple Method

You should have copied the contents of CrackingCodesFiles.zip into a folder called python in your home directory. Change directory to the directory that holds cryptomath.py. It may be either python/CrackingCodesFiles, or python/, depending on how you copied the files. Once you are in the directory that contains cryptomath.py, start Python or Idle.

Permanent Method

Copy cryptomath.py into one of the directories in the Python path. Some possible choices are highlighted above in the section on finding Python's path.

Test the cryptomath library

You should be able to do this:

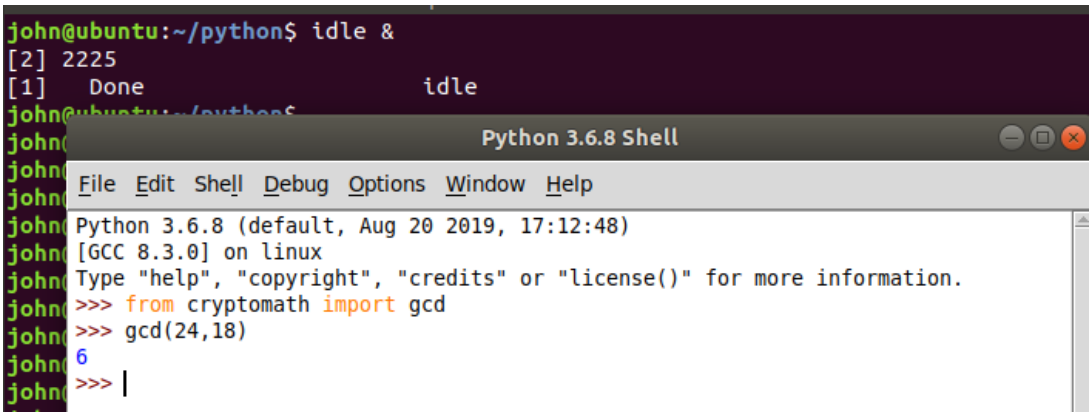
Linux

```
john@ubuntu:~$ cd python/CrackingCodesFiles/
john@ubuntu:~/python/CrackingCodesFiles$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from cryptomath import gcd
>>> gcd(24,18)
6
>>>
```

Windows

```
PS C:\Users\yorks> cd .\python\CrackingCodesFiles\
PS C:\Users\yorks\python\CrackingCodesFiles> python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from cryptomath import gcd
>>> gcd(24,18)
6
>>>
```

IDLE in Linux



Practicing Modular Math from Python

Once you can import cryptomath, you have access to the gcd() and findModInverse() functions. Remember that the way you imported the cryptomath functions (see Python Modules, above) determines how you call the functions. You can run the math commands you need from the Idle shell, or from the terminal.

For Turn In

1) Compute (you can do this easily at an interactive Python prompt. The mod operator in Python is "%")
    a. 34 mod 18
    b. (34 + 97) mod 12 Note: the mod operator has the same priority as multiplication. In a tie, Python executes operators from left to right.
    c. 14 * 71 mod 15
    d. 152 / 71
    e. 152 // 71 (in Python, // is integer division)
    f. 152 – (152 // 71) * 71 (this is the remainder after integer division)

g. 152 mod 71 (you should see that the answer is the remainder when you divide 152//71 (integer division)

2) Compute gcd(36, 45) and gcd(44, 45) using the cryptomath.py module you added above. You should get 9 and 1 as answers, as a check to make sure the code is correct. Now compute gcd(452, 973) and gcd(452,1496). Which one of the pairs of numbers relatively prime, and what is the GCD of the pair that is not relatively prime?

3) For the following numbers, compute the GCD of the number and the modulus. If the number and the modulus are relatively prime, compute the multiplicative inverse. You can use either brute force in Python, or the findModInverse() function from cryptomath.)
   a. 17 mod 26
   b. 16 mod 26
   c. Of the two numbers above, which could be used for the key of an affine cipher and which could not? Why?
   d. Compute the multiplicative inverse of 17 in modulus 27, 28, and 29. Note that the inverse is completely different when the modulus changes.

4) Least Common Multiple. The `lcm(a, b)` is the smallest number that can be divided by both a and b. It is easily computed as `a * b // gcd(a, b)`
   a. Compute the lcm of something simple, like 6 and 9. Check your answer by hand to verify that the equation is correct.
   b. Compute `lcm(252, 196)`