Cryptography Homework 2—Classic Ciphers

# Shift (Caesar) Cipher

The file, caesarCipher.py, is a modified version of one presented by Al Sweigart at <https://inventwithpython.com/cracking/chapter5.html>. It is available in Canvas, and at the end of this lesson.

1. Copy the file into your development area. Play with it to get used to Python in the development area you have chosen. Use it to encrypt and decrypt messages with different keys. Hand in a screenshot of a successful encryption and decryption attempt.
2. Modify the code to make your own script that can crack a ciphertext encrypted with a shift cipher by brute force. Put the caesarCipher.py code inside a FOR or WHILE loop that will try all possible keys. Make it print the output for each key and look at the messages to decide which one is correct. If you like, have your lab partner encrypt a message and give you the ciphertext but not the key. Use your script to crack a message that you have encrypted. Note: Python uses indentation to create code blocks. You can indent an entire block of text by selecting the block and pressing the tab key. This works in Notepad++ (Windows), gedit (Ubuntu text editor), or Visual Code. In gedit, you must edit your preferences so that the tab key always inserts spaces.  
   Graphical user interface, text, application

   Description automatically generated  
   It does not work in notepad.exe.
3. The caesarCipher.py code, and almost all online decoders only use the upper-case symbols. That seems limiting. Let’s add the lower-case symbols and digits 0 – 9 to our symbol set.  
     
   SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890'  
     
   It turns out that we have introduced a serious vulnerability into our system. What is it? (Hint: Encrypt a message with words in upper and lower-case, with keys 26 and 36.)  
     
   Explain the vulnerability.   
   With keys 26 and 36, upper case letters get mapped to lower case letters, or vice versa. You would have to exclude those keys, or come up with something else entirely.
4. The code that passes characters that are not in SYMBOLS to the ciphertext unencrypted is nice, but it introduces a serious vulnerability. What is it?  
   Things like spaces and punctuation pass straight through the encryption. Then you can tell how long the words are, where sentences are, etc. It makes it easier to break the encryption.

# Affine Cipher, Inverses, and Fields

The file, affineCipher.py, is a modified version of one presented by Al Sweigart at <https://inventwithpython.com/cracking/chapter5.html>. It is available in Canvas, and at the end of this lesson.

1. The code in affineCipher.py also uses an expanded symbol set. It has a different problem than caesarCipher.py had with its expanded symbol set, however.  
     
   SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890 !?.'  
     
   Try encrypting, then decrypting, messages with different keys. Remember that the affine cipher uses a formula of the form, a \* index + b, and the shift (Caesar) cipher uses index + b. What is the problem with this symbol set? Hint: try to find a value for a that causes trouble.  
   The symbol set has 66 characters, so it is modulo 26. Any number that has a factor of 2, 3, or 11 will be unusable in a since there is no multiplicative inverse.
2. Fix the code by adding or removing characters from SYMBOLS so that SYMBOLS becomes a finite field. (Hint: search for “first 100 primes” in your favorite search engine.)  
   If you add one character, for example a comma, the modulo changes from 66 to 67. Since 67 is prime, any number can be used for a.
3. What is necessary to make an integer ring be a field as well as a ring?  
   It has to be based on a prime number. For example, ℤ5, ℤ11 and ℤ29 are fields, while ℤ8 and ℤ9 are just rings.  
     
   1. What characters did you add or remove, and what is the new length of SYMBOLS?
   2. Paste your output from encrypting, and then decrypting, a message of your choice with your choice of key.