CSE 231 Spring 2018

Programming Project 04

Edit on 2/3/18: clarified in the description of main() that the first prompt is for the rotation, N.

This assignment is worth 40 points (4.0% of the course grade) and must be **completed and turned in before 11:59 PM on Monday, February 12, 2018.**

Assignment Overview
(learning objectives)
This assignment will give you more experience on the use of:

1. integers (int)
2. conditionals
3. iteration
4. string

The goal of this project is to implement a different cryptographic technique by combining two cryptographic techniques. I call it dumbcrypt. Dumbcrypt is a combination of Affine Cipher and Caesar Cipher. The user will enter a sentence, a string composed of letters, numbers, and punctuation. Notice that spaces are not allowed in the input string because spaces provide too many clues to someone trying to crack your coded message. Your job is to encrypt the sentence using a combination of Affine and Caesar Cipher. *The letters and numbers will be encrypted and decrypted using Affine Cipher; the punctuation using Caesar Cipher.*

Assignment Background

Affine Cipher - https://en.wikipedia.org/wiki/Affine_cipher

Affine Cipher is a bit different than Caesar Cipher in that the Affine Cipher uses an encryption function to calculate the integer that corresponds to the cipher text letter. The encryption function for a letter variable named *x* is

$$E(x) = (Ax+N) \bmod M$$

Where A and N are keys of the cipher and M is size of the alphabet (for example, M=26 for English letters; M = 36 for English letters plus digits). The decryption of Affine Cipher only works when A and M are co-primes of each other.

N will be an input to the program and will be fixed for the whole run of the program. N is sometimes referred to as the “rotation” but is effectively a key.

Co-primes are a pair of numbers whose greatest common divisor (GCD) is 1. Only one co-prime is needed (of possibly many). For this project we choose the smallest greater than one so everyone is using the same one—making testing feasible.

Algorithm:

1. Calculate M, the number of characters in the alphabet (Hint: use len())
2. Calculate A using `get_smallest_co_prime(M)`
3. For the character to be encrypted **find** its index in the alphabet: *x* in the formula is the index

4. Apply the formula to get the index of the cipher character: $E(x) = (Ax+N) \bmod M$
5. Using the index get the cipher character from the alphabet
6. Return the character.

Decryption is a little more complicated than encryption. The decryption function is

$$D(x) = A^{-1}(x-N) \bmod M$$

Where A^{-1} is the modular multiplicative inverse (https://en.wikipedia.org/wiki/Modular_multiplicative_inverse). We provide the function `multiplicative_inverse(A,M)` that calculates the multiplicative inverse, A^{-1}, for you. The function checks all the numbers *x* from 1 to M and for every number *x* checks if *(A*x) mod M* is 1. Once you have the A^{-1}, you can use the decryption function to calculate the integer that corresponds to plaintext letter.

The decryption algorithm is the same as encryption with two modifications:

Algorithm:
1. Calculate M, the number of characters in the alphabet (Hint: use len())
2. Calculate A using `get_smallest_co_prime(M)`
3. Calculate A^{-1} using the function `multiplicative_inverse(A,M)`.
4. For the character to be encrypted **find** its index in the alphabet: *x* in the formula is the index
5. Apply the formula to get the index of the cipher character: $D(x) = A^{-1}(x-N) \bmod M$
   Note the formula difference from encryption: the parentheses are different and it has subtraction.
6. Using the index get the cipher character from the alphabet
7. Return the character.

Caesar Cipher - https://en.wikipedia.org/wiki/Caesar_cipher

Caesar Cipher is a simplified form of Affine cipher which follows similar encryption and decryption algorithm. Caesar Cipher uses a single key to calculate the integer that corresponds to the ciphertext letter. The encryption function is

$$E(x) = (x+N) \bmod M$$

The decryption function is

$$D(x) = (x-N) \bmod M$$

The algorithms for Caesar Cipher are similar to the Affine Cipher algorithms

Project Description

Your program must meet the following specifications:

1. You have to use the eight functions in the provided `proj04.py` skeleton. You have to implement and use the following:

a. `check_co_prime(num, M)` : Takes two numbers as parameters. Returns `True` if num and M are co-primes, otherwise returns `False`. (Hint: used GCD)
b. `get_smallest_co_prime(M)` : Takes one number M as a parameter: Returns the smallest coprime of M that is greater than 1. For example, if M = 26, the smallest co-prime greater than one is 3. If M = 210, the smallest co-prime greater than one is 11. (Hint: use `check_co_prime`)
c. `caesar_cipher_encryption(ch,N,alphabet)` : Takes three inputs: a character to encrypt, the rotation N, and the alphabet. Returns the cipher text character using the Caesar Cipher.
d. `caesar_cipher_decryption(ch,N, alphabet)` : Takes three inputs: a character to encrypt, the rotation N, and the alphabet. Returns the plain text character using the Caesar Cipher.
e. `affine_cipher_encryption(ch,N, alphabet)` : Takes three inputs: a character to encrypt, the rotation N, and the alphabet. Returns the cipher text character using the Affine Cipher.
f. `affine_cipher_decryption(ch,N, alphabet)` : Takes three inputs: a character to encrypt, the rotation N, and the alphabet. Returns the plain text character using the Affine Cipher.
g. `main()` : Takes no input. Returns nothing. First prompts for rotation, N. Then prompts for a command (d,e,q), prompts for a string, decrypts or encrypts the string depending on the command using the encryption and decryption functions described above. Remember to use a different encryption/decryption algorithm for punctuation.

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj04.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via **Mimir** before the project deadline.

Assignment Notes and Other Requirements

1. To clarify the project specifications, sample output is appended to the end of this document.
2. Convert all letters to lower case before encryption or decryption.
3. Items 1-9 of the Coding Standard will be enforced for this project.
4. We provide a `proj04.py` program for you to start with.
5. You may import the GCD function from a module such as math.
6. You are not allowed to use advanced data structures such as lists, sets, dictionaries and classes.
7. You are not allowed to use the `chr()` function.
8. There are two alphabets provided: numbers + letters and string.punctuation
9. The *mod* operator in the encryption formulas is the % operator in Python.
10. The string method `isalnum()` returns `True` if the string is composed of letters and digits.
11. The N in the formulas is often referred to as the "rotation" and is the first value prompted for. It effectively is the key for the encryption and decryption.
12. Error checking (Hint: do error checking after everything else is working.)

    a. Re-prompt if the rotation is not an int.
    b. Print an error message and ignore any commands that are not ‘e’, ‘d’ or ‘q’.
    c. Print an error message if any of the text is not a letter, digit or punctuation. Note that a space will generate an error (spaces provide too many clues for cracking codes).
13. A strings.txt file of strings used is provided to help with testing.
14. I found `else` with the `for` statement to be useful to not print plain and cipher texts if there was an error with a character.

Test Cases
Function Unit Tests

Function 1 test check_co_prime
```
check_co_prime(2,3) == True
check_co_prime(2,8) == False
```

Function 2 test get_smallest_co_prime
```
get_smallest_co_prime(26) == 3
get_smallest_co_prime(210) == 11
get_smallest_co_prime(90) == 7
```

Function 3 test caesar_cipher_encryption
```
caesar_cipher_encryption('a',3,'abcdefgh') == 'd'
caesar_cipher_encryption('e',3,'2cdef78h') == '8'
```

Function 4 test caesar_cipher_decryption
```
caesar_cipher_decryption('d',3,'abcdefgh') == 'a'
caesar_cipher_decryption('8',3,'2cdef78h') == 'e'
```

Function 5 test affine_cipher_encryption
```
affine_cipher_encryption('a',3,'abcdefgh') == 'd'
affine_cipher_encryption('7',3,'2cdef78h') == 'd'
```

Function 5 test affine_cipher_decryption
```
affine_cipher_decryption('d',3,'abcdefgh') == 'a'
affine_cipher_decryption('d',3,'2cdef78h') == '7'
```

Test 1

```
Input a rotation (int): 4
Input a command (e)ncrypt, (d)ecrypt, (q)uit: e
Input a string to encrypt: Hello,World!
Plain text: Hello,World!
Cipher text: dyxxc:gcrxt%
Input a command (e)ncrypt, (d)ecrypt, (q)uit: d
Input a string to decrypt: dyxxc:gcrxt%
Cipher text: dyxxc:gcrxt%
Plain text: hello,world!
Input a command (e)ncrypt, (d)ecrypt, (q)uit: e
Input a string to encrypt: Fahrenheit-451?
Plain text: Fahrenheit-451?
Cipher text: 3edry7dyi1;kp5]
```

```
Input a command (e)ncrypt, (d)ecrypt, (q)uit: d
Input a string to decrypt: 3edry7dyi1;kp5]
Cipher text: 3edry7dyi1;kp5]
Plain text: fahrenheit-451?
Input a command (e)ncrypt, (d)ecrypt, (q)uit: q
```

Test 2

```
Input a rotation (int): x
Error; rotation must be an integer.
Input a rotation (int): 1.2
Error; rotation must be an integer.
Input a rotation (int): 8
Input a command (e)ncrypt, (d)ecrypt, (q)uit: d
Input a string to decrypt: cg)cv22b+
Cipher text: cg)cv22b+
Plain text: go!green#
Input a command (e)ncrypt, (d)ecrypt, (q)uit: e
Input a string to encrypt: hello world
Error with character:
Cannot encrypt this string.
Input a command (e)ncrypt, (d)ecrypt, (q)uit: q
```

Test 3 Blind Test

Grading Rubric

```
5 pts Coding Standard
     (descriptive comments, mnemonic identifiers, format, etc...)

2 pts check_co_prime

3 pts get_smallest_co_prime

4 pts caesar_cipher_encryption

4 pts caesar_cipher_decryption

4 pts affine_cipher_encryption

4 pts affine_cipher_decryption

5 pts Test 1

4 pts Test 2

5 pts Test 3 Blind Test
```