CISS451: Cryptography and Computer Security
Assignment 1

The goal is to develop a simple library for the shift cipher.

We will be using the Python programming language. The shift cipher is actually simple enough that one can encrypt and decrypt without any writing code. Likewise breaking a shift cipher is easily done because the key space is so small. (Although it's tedious.)

However we will still go ahead with writing this library because it is a step toward more complicated classical ciphers such as the Vigenere and the substitution cipher. And after we have the algorithms (encryption and decryption) of the shift cipher, we can use the brute force approach to break a shift cipher to obtain the key.

On the practical side, of course learning to program is always a good thing and make abstract theory concrete.

There's actually very little programming on your part since this is the first assignment. Most of the code is given to you. Make sure you study and run all the code in this document carefully.

Python programming

Here's a quick-and-dirty tutorial on Python programming. This is not a complete introduction to Python programming. I'll only talk enough to do basic computations, especially relevant to number crunching. (By the way, Python is used in scientific computations in research and in the industry. However I will not go into special libraries which are optimized for speed. If you want to find out more, you can google for scipy and numpy.) I will assume that you're using one of my fedora virtual machines.

Note that I'm using our Fedora 31 virtual machine. The default python version is python 3. If you run python from your bash shell you should see python 3:

```
[student@localhost ~]$ python
Python 3.7.7 (default, Mar 13 2020, 10:23:39)
[GCC 9.2.1 20190827 (Red Hat 9.2.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Open a text editor and edit a file say `x0.py` and write this:

```
# my first program
print("hello world")
```

Now run the program from the shell with this command:

```
[student@localhost ~]$ python x.py
```

and you will get this response:

```
[student@localhost python-examples] python x0.py
hello world
```

Now try this:

```
print("hello world")

x = 1
print("x =", x)

# here's a for-loop
for i in [5,6,7,8,9,10]:
    print(">>>>> .....")
    print("i =", i, i + 2, i - 2, i * 2, i % 2, i / 2)
    print("..... <<<<<")
    print()

print(list(range(5, 11)))
print(list(range(100, 110)))

# here's a for-loop that does the same thing
for i in range(5, 11):
    print(">>>>> .....")
    print("i =", i, i + 2, i - 2, i * 2, i % 2, i / 2)
    print("..... <<<<<")
    print()

# here's a while loop
i = 0
while i < 10:
    print("i =", i)
    print("incrementing i ...")
    i += 1

# you can assign multiple variables
a, b, c = 1, 2, 3
print(a, b, c)
```

and you'll get this:

```
hello world
x = 1
>>>>> .....
i = 5 7 3 10 1 2.5
..... <<<<<

>>>>> .....
i = 6 8 4 12 0 3.0
..... <<<<<

>>>>> .....
i = 7 9 5 14 1 3.5
..... <<<<<

>>>>> .....
i = 8 10 6 16 0 4.0
```

```
..... <<<<<

>>>>> .....
i = 9 11 7 18 1 4.5
..... <<<<<

>>>>> .....
i = 10 12 8 20 0 5.0
..... <<<<<

[5, 6, 7, 8, 9, 10]
[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]
>>>>> .....
i = 5 7 3 10 1 2.5
..... <<<<<

>>>>> .....
i = 6 8 4 12 0 3.0
..... <<<<<

>>>>> .....
i = 7 9 5 14 1 3.5
..... <<<<<

>>>>> .....
i = 8 10 6 16 0 4.0
..... <<<<<

>>>>> .....
i = 9 11 7 18 1 4.5
..... <<<<<

>>>>> .....
i = 10 12 8 20 0 5.0
..... <<<<<

i = 0
incrementing i ...
i = 1
incrementing i ...
i = 2
incrementing i ...
i = 3
incrementing i ...
i = 4
```

```
incrementing i ...
i = 5
incrementing i ...
i = 6
incrementing i ...
i = 7
incrementing i ...
i = 8
incrementing i ...
i = 9
incrementing i ...
1 2 3
```

As you can see most of the operators are what you have seen before in C++/Java/etc. No surprises here. Likewise you also have `<`, `<=`, `>`, `>=`, `==`, `!=`.

Here's how you create and call functions:

```
# here's a function
def f(a, b, c):
    z = a + b + c
    return z

# here's calling the above function
print(f(1, 2, 3))

# you can return multiple values
def g(a, b, c):
    return a + b, a * c, b / c

# ... and of course receive multiple values
i, j, k = g(1, 2, 3)
print(i, j, k)
```

You also have arrays:

```
# you have python lists which are like arrays
x = []
print(x)
x = [42, 43, 44]
print(x)
x.append(100)
```

```
print(x)
del x[1]
print(x)
print(42 in x)
print(45 in x)

x = [5, 6, -5, 8, 2, -10, 11, 9, 13, -1]
for y in x:
    print(y)
print(x[0])
print(x[2])
print(x[-1]) # the last element
print(x[-2]) # the second last element
x.sort()
print(x)
print(x[0])  # smallest
print(x[-1]) # largest

y = x[2:5] # a slice of x
print(y)

# be careful when doing = on lists ...
x = [1,2,3,4,5]
y = x           # y and x references the same value
x[0] = 100
print(x)
print(y) # y references the *same* value as x

# if you want a *copy*, you do this:
import copy
x = [1,2,3,4,5]
y = x[:]
x[0] = 100
print(x)
print(y)
```

You can also do a copy for *any* object like this:

```
import copy
x = [1,2,3,4,5]
y = copy.deepcopy(x)
```

Here's a simple function that computes divisors:

```
def divisors(a):
    xs = []
    for d in range(1, a + 1):
        if a % d == 0:
            xs.append(d)
    return xs

print(divisors(100))
```

Here's one that compute common divisors:

```
def common_divisors(a, b):
    div_a = divisors(a)
    div_b = divisors(b)
    xs = []
    for d in div_a:
        if d in div_b:
            xs.append(d)
    return xs

print(common_divisors(100, 30))
```

Here's one that computes gcd is an extremely inefficient way:

```
def slow_gcd(a, b):
    xs = common_divisors(a, b)
    xs.sort()
    return xs[-1]

print(slow_gcd(100, 30))
```

To time your code you can do this:

```
import time
start = time.clock()
g = slow_gcd(100000000, 200000)
end = time.clock()
print("time taken:", end - start)
```

Here's a self–exercise:

A positive integer n if said to be *perfect* if it is the sum of the divisors of n less than n. For instance 6 is a perfect number since the divisors of 6 which are less than 6 are

$$1, 2, 3$$

and the sum is

$$1 + 2 + 3 = 6$$

Write a program that prompts the user for n and prints all perfect numbers up to n.

(You will discover the first 4 perfect numbers very quickly. The fifth one takes a while.)

```
def isperfect(n):
    sum = 0
    # CODE TO COMPUTE THE SUM OF DIVISORS OF n WHICH ARE < n
    return sum == n

n = input("enter n: ")
for i in range(1, n + 1):
    if i % 100000 == 0:
        print("testing", i)
    if isperfect(n):
        print("***** FOUND", i)
```

The answer is on the next page ...

SPOILERS!!! SPOILERS!!! SPOILERS!!!

```
def isperfect(n):
    sum = 0
    for x in divisors(n):
        if x < n:
            sum += x
    return sum == n
```

Of course building a python list is a waste of resources. You might as well just compute the divisors and add. So here's another exercise on the improved `isperfect` function.

```
def isperfect(n):
    sum = 0
    # YOUR CODE TO COMPUTE THE SUM OF DIVISORS DIRECTLY WITH
    # USING THE div FUNCTION.
    return sum == n
```

Time it and see if you can improve the performance.

Can you further improve it? [HINT: if $1 < d < n$ is a divisor of n, then you have already found another which might be distinct from d.]

Also, since you're scanning a range of values for perfect numbers, some computations actually repeat. For instance if you already know all divisors of 10, how would you compute the divisors of 30?

With some improvements, you should be able to compute the 5th perfect number and beat the ancient greek mathematicians.

Large numbers and large loops

Try this:

```
print(2**1000)
```

`**` is the exponentiation operator. As you can see, Python can handle extremely huge numbers. This is helpful extremely in Cryptography since in the real world, Cryptography usually involves extremely huge numbers.

Recall that you can do for-loops:

```
for i in range(1, 100):
    print(i)
```

Now, in the case of for instance looping over a huge range of numbers, you can do this:

```
for i range(1, 1000000):
    print(i)
```

or this:

```
i = 1
while i < 1000000:
    print(i)
    i += 1
```

However for python2 `range(1, 1000000)` creates a list before you can even run the for-loop. In fact, although the following works for python3, they might not work for python2:

```
for i in range(1, 1000000000):
    print(i)

for i in range(1, 1000000000000000):
    print(i)
```

Python programming: Characters and mod 26

For classical ciphers and also many modern applications, you need to convert data from one format to another. For classical ciphers, you need to convert between a-z and mod 26 integers. Modern computer systems have to translate between different data and binary numbers.

To convert from characters to integer ASCII code in Python try this

```
c = 'm'
print(ord(c))
```

To convert from ASCII code back to characters try this:

```
print(chr(97))
```

Of course if you want `a` to correspond to 0, to convert from character to mod 26 integer, you do this:

```
c = 'm'
print(ord(c) - ord('a'))
```

and to convert from mod 26 integer to character you do this:

```
print(chr(2 + ord('a')))
```

i.e., 2 in $\mathbb{Z}/26$ corresponds to `c`.

So here's a simple function to convert from characters to mod 26:

```
def to_int(c):
    return ord(c) - ord('a')
```

It's a good idea to check that c is a string of length 1. (Python does not have the concept of characters like those in C++, but only of strings. A character in Python is just a string of length 1.) To enforce that c is a string of length 1 we do this:

```
def to_int(c):
    if not isinstance(c, str): raise ValueError("%s is not str" % c)
    if len(c) != 1: raise ValueError("length of %s is not 1" % c)
    if not('a' <= c <= 'z'): raise ValueError("char %s not in a..z" % c)
    return ord(c) - ord('a')
```

You don't have to know how those checks work. You can just use the code above.

Here's the code to convert from integer mod 26 to character:

```
def to_chr(i):
    if not isinstance(i, int): raise ValueError("%s is not int" % i)
    if i < 0 or i > 25: raise ValueError("int %s not in 0..25" % i)
    return chr(i + ord('a'))
```

Altogether we have these functions to convert between a-z and $\mathbb{Z}/26$:

```
def to_int(c):
    if not isinstance(c, str): raise ValueError("%s is not str" % c)
    if len(c) != 1: raise ValueError("length of %s is not 1" % c)
    if not('a' <= c <= 'z'): raise ValueError("char %s not in a..z" % c)
    return ord(c) - ord('a')

def to_chr(i):
    if not isinstance(i, int): raise ValueError("%s is not int" % i)
    if i < 0 or i > 25: raise ValueError("int %s not in 0..25" % i)
    return chr(i + ord('a'))
```

It's also a good idea to have a similar function to convert a string (i.e. length not necessarily 1) to a list of integers. So we do this:

```
def to_ints(cs):
    ints = []
    for c in cs:
        ints.append(to_int(c))
    return ints

def to_chrs(xs):
    chrs = []
    for x in xs:
        chrs.append(to_chr(x))
    return chrs
```

Altogether we have

```
def to_int(c):
    if not isinstance(c, str): raise ValueError("%s is not str" % c)
    if len(c) != 1: raise ValueError("length of %s is not 1" % c)
    if not('a' <= c <= 'z'): raise ValueError("char %s not in a..z" % c)
    return ord(c) - ord('a')

def to_chr(i):
    if not isinstance(i, int): raise ValueError("%s is not int" % i)
    if i < 0 or i > 25: raise ValueError("int %s not in 0..25" % i)
    return chr(i + ord('a'))

def to_ints(cs):
    ints = []
    for c in cs:
        ints.append(to_int(c))
    return ints

def to_chrs(xs):
    chrs = []
    for x in xs:
        chrs.append(to_chr(x))
    return chrs

print("test ...")
print(to_chrs([0,1,2,3,4,5]))
print(to_ints(['a', 'c', 'e', 'z']))
```

Make sure you study and run the above code.

Basically, a statement like `isinstance(i, int)` returns `True` iff `i` has an `int` value and `isinstance(c, str)` returns `True` iff `c` has a string value. The `raise` is the same as the C++ `throw` (as in throwing an exception); `ValueError` is a pre-defined exception class in Python.

Converting between list of characters and strings

Note that in the above, for the `to_chrs`, you get a list of characters. Sometimes it's more readable to print a string rather than the characters. So it might be a good idea to know how to convert between strings and lists of characters.

You can easily get a list of characters from a string like this:

```
s = "jump"
cs = list(s)
print(cs)
```

And from list of characters to string you do this:

```
cs = ['j', 'u', 'm', 'p']
s = ''.join(cs)
print(s)
```

Python Programming: Cleaning up a string

For classical ciphers, you usually convert uppercase to lowercase and throw away the non a-z. That's easy to do in Python. First in Python run

```
import string
print(string.ascii_lowercase)
print('A' in string.ascii_lowercase)
print('!' in string.ascii_lowercase)
print('b' in string.ascii_lowercase)
```

and then this:

```
import string
def cleanup(cs):
    cs = cs.lower()
    xs = []
    for c in cs:
        if c in string.ascii_lowercase:
            xs.append(c)
    return ''.join(xs)

print(cleanup("Hello, world!"))
```

Python Programming: The code

```
import string, sys

def to_int(c):
    if not isinstance(c, str): raise ValueError("%s is not str" % c)
    if len(c) != 1: raise ValueError("length of %s is not 1" % c)
    if not('a' <= c <= 'z'): raise ValueError("char %s not in a..z" % c)
    return ord(c) - ord('a')

def to_chr(i):
    if not isinstance(i, int): raise ValueError("%s is not int" % i)
    if i < 0 or i > 25: raise ValueError("int %s not in 0..25" % i)
    return chr(i + ord('a'))

def to_ints(cs):
    ints = []
    for c in cs:
        ints.append(to_int(c))
    return ints

def to_chrs(xs):
    chrs = []
    for x in xs:
        chrs.append(to_chr(x))
    return chrs

def cleanup(cs):
    cs = cs.lower()
    xs = []
    for c in cs:
        if c in string.ascii_lowercase:
            xs.append(c)
    return ''.join(xs)
```

Make sure you understand the above code and try out the test code to understand how to use the functions.

By the way, in the above, we're not using any long integers in Python. So the above can be easily translated into for instance C++ if you wish.

Q1. The goal is to count the number of solutions, in $\mathbb{Z}/135246$, to the following equation:

$$x^3 + 425x^2 + 79x + 42 \equiv 0 \pmod{135246}$$

Write a program that prints the number of solutions. The program must perform a brute force search in $\mathbb{Z}/135246$ and count all the solutions. When I run your program, it just prints an integer. Do *not* so anything else.

Q2. The goal is to find all solutions, in $\mathbb{Z}/135246$, to the following equation:

$$x^3 + 425x^2 + x + 42 \equiv 0 \pmod{135246}$$

Write a program that prints the number of solutions. The program must perform a brute force search in $\mathbb{Z}/135246$ and count all the solutions.

Q3. Using brute force, write a python function that compute the inverse of a mod n if the inverse exists. c is an inverse of a mod n if $0 \leq c < n$ and

$$ca \equiv 1 \pmod{n}$$

Your code should look like this:

```
def inv(a, n):
    # loop i from 1 to n - 1 and test if i is the
    # inverse of a mod n




    # if you exit the loop without finding the inverse,
    # return None. I have already done it for you.
    return None

a = int(input()) # get integer value for a from user
n = int(input()) # get integer value for n from user
print(inv(a, n))
```

If the inverse cannot be found, you must execute `return None`.

Q4. Consider the following equation:

$$27109x^3 \equiv 5316 \pmod{135246}$$

Suppose you want to find all solution (in $\mathbb{Z}/135246$). Instead of doing a brute force search, I want you to rewrite the equation to this form:

$$x^3 \equiv c \pmod{135246}$$

(Of course the solutions to this equation must be the same as the original.) The reason for using this instead is of course that during a brute force search using the original equation, I would have to waste time multiplying x^3 with 27109 for each candidate in $\mathbb{Z}/135246$. However after changing the equation to the second version, the left-hand-side of the equation does not need to multiply x^3 with a coefficient.

Write a program that prints `c`. Note that `c` must be an integer in $\mathbb{Z}/135246$, i.e., $0 \leq c < 135246$.

Q5. Write a function `Shift_E` that accepts a key `k` and a string `s` so that calling `Shift_E(k, s)` returns the encrypted string using the shift cipher with key `k`. Note that `s` is assumed to be a string that is already "cleaned up", i.e., `s` containing only lowercase letters. For instance executing

```
print(Shift_E(3, 'abc'))
```

give us the output

```
def
```

(Of course if you have a text with punctuations and spaces, you would have to execute

```
s = cleanup(s)
```

before running `s` through the `Shift_E` function for encryption.)

Here's the skeleton code that you must use:

```
def Shift_E(k, m):
    #
    # TODO
    #

k = int(input()) # get integer value for k from user
m = ~~raw_~~input() # get string value for m from user
print(Shift_E(k, cleanup(m)))
```

Q6. Write a function `Shift_D` that accepts a key `k` and a string `s` so that calling `Shift_D(k, s)` returns the decryption of `s` with key `k`. For instance executing

```
print(Shift_D(3, 'def'))
```

give us the output

```
abc
```

You can now perform launch a brute force attack on a shift cipher. For instance if `s` is a ciphertext, you can do this:

```
def Shift_bruteforce(s):
    for k in range(0, 26):
        print(k, Shift_D(s, k))
```

and quickly find the decrypted string that makes most sense. Your source file must contain the following code at the bottom:

```
...

k = int(input())     # get integer value for k from user
c = input() # get string value for c from user
c = cleanup(c)
print(Shift_D(k, c))
Shift_bruteforce(k, c)
```

That's it!

C++

If you're learning Python for the first time, it might help if you look at the following C++ code and compare against the Python version. Note for instance that since C++ has static type checking, some of the parameter checks in Python are not neceesary in C++. And because of types and automatic type conversion, some typecasting operations in Python won't appear below.

```
#include <vector>

class ValueError
{};

int to_int(char c)
{
    return c - 'a';
}

char to_chr(int i)
{
    if (i < 0 || i > 25) throw ValueError();
    return i + 'a';
}

std::vector< int > to_ints(const std::vector< char > & cs)
{
    std::vector< int > ints;
    for (unsigned int i = 0; i < cs.size(); ++i)
    {
        ints.push_back(to_int(cs[i]));
    }
    return ints;
}

std::vector< char > to_chrs(const std::vector< int > & xs)
{
    std::vector< char > chrs;
    for (unsigned int i = 0; i < xs.size(); ++i)
    {
        chrs.push_back(to_chr(xs[i]));
    }
    return chrs
}

std::string cleanup(const std::string & cs)
```

```
{
    std::string ds;
    for (unsigned int i = 0; i < cs.size(); ++i)
    {
        char c = cs[i];
        if ('a' <= c && c <= 'z')
        {
            ds += c;
        }
        else if ('A' <= c && c <= 'Z')
        {
            ds += (c - 'A') + 'a';
        }
    }
    return ds;
}
```