

**ASSIGNMENT OVERVIEW**

In this assignment you'll be creating a program called `secret_code.py`, which will allow the user to code and decode messages using a substitution cipher.

This assignment is worth 75 points and is due on the `crashwhite.polytechnic.org` server at 23:59:59 on the date given in class.

**BACKGROUND**

In creating a *ciphertext* (coded) message from a *plaintext* (uncoded, human-readable) message, there are a number of encryption strategies that one can use. One strategy is a simple substitution cipher, in which each character in a plaintext message such as “Too many secrets”, is replaced by a different letter according to a key that, in this case, consists of the 26 letters of the alphabet scrambled.

plaintext	"too many secrets"
alphabet (for plaintext)	abcdefghijklmnopqrstuvwxyz
key (for ciphertext)	morxshpenibqcyvafkgdltzjwu
ciphertext	"dvv cmyw gsrksdg"

Here we can see that any letter in the plaintext that is a “t” (and there are two in “too many secrets”), will be replaced with a “d” in the ciphertext message; likewise, the “o”s in the message will be replaced with “v”s, etc. Exchanging coded messages, then, will consist of three steps:

- **Identify the *key* that will be used.** Both sender and recipient will need to use the same key—this is called a *symmetric* encryption.
- **The sender uses the *key* to convert a *plaintext* message to a *ciphertext* that can be transmitted.**
- **The receiver uses the same *key* to convert the *ciphertext* back to the original *plaintext* message that can be read.**

**PROGRAM SPECIFICATION**

Create a Python program that:

- a. creates a random *key* that can be used to exchange messages
- b. uses that key to convert a *plaintext* message to *ciphertext*
- c. uses the same key to convert the *ciphertext* message back to *plaintext*
- d. provides the user with a menu that allows him/her to perform task *a*, *b*, or *c*.
- e. has thorough comments, written while coding the program, and supplemented with additional comments before submitting.

## ***DELIVERABLES***

### **secret\_code.py**

You should keep a working copy of this file in your home folder on the server and a backup copy of the file elsewhere. To submit your assignment for grading, copy your file to the **forInstructor** directory in your home directory on the **crashwhite.polytechnic.org** before the deadline.

To submit your assignment for grading, copy your file to the **forInstructor** directory in your home directory on the **crashwhite.polytechnic.org** before the deadline.

## ***ASSIGNMENT NOTES***

- This program may be written in a number of ways, but for our purposes, we'll restrict ourselves to using six string operations:
  - the slice `[a:b]` operation, which identifies a subset of a string
  - the `len()` method, which identifies the length of a string
  - the `.find()` method, which finds the position of a string in a larger string
  - the `.replace()` method, which replaces one character in a string with another
  - the `.lower()` method, which converts a string to all lower-case letters
  - the `+` operator, which concatenates two strings

There are a large number of other string operations that are available in Python, and alternate versions of a program written for this assignment might conceivably use some of those.

- This is the first assignment that will provide the user with a menu of alternatives. The general idea is that a user will get to pick one of several different alternatives, and the program then will execute the appropriate section of code. This is easily accomplished with **if-elif-else** statements:

```
user_selection = "0"
print("This program allows you to create keys, encode plaintext, and
decode ciphertext.")
while user_selection != "4":
    print "What would you like to do?"
    user_selection = input("1.create a key 2.encode 3.decode 4. quit > ")
    if user_selection == "1":
        # Selection 1 stuff
        # goes here
    elif user_selection == "2":
        # Selection 2 stuff
    elif user_selection == "3":
        # Selection 3 stuff
    elif user_selection != "4":
        print "Invalid selection")
```

Notice that we've wrapped the **if-elif-else** statements in a **while** loop that lets the program repeat until they specifically choose to quit. You might want to create this skeleton structure first before tackling any of the subsections of the program. Then build and test each of the subsections as you go. This strategy of starting with the "big picture" of your program, and working

down to looking at smaller sections of code is called *top-down design*. Keep in mind that if you pursue this strategy, you'll need to include a **pass** statement for each section of code that hasn't been written yet.

- We are right at the limit of size and complexity in our programs where we'll soon want to think about breaking our code up into smaller sections. For now, writing this assignments in a single **main()** function will help keep you oriented and let you easily see the flow of the instructions.
- On the other hand, if you're interested in trying to break this program up into a number of different functions, you certainly can. Think carefully about what different parts of the program could logically be split into functions. What information, if any, do each of those functions need to have passed to them as parameters? What information, if any, do those functions need to return to the **main()** program?
- To create the key, the program will need to import the **random** module. To create this key, we'll also need to figure out Python code for performing the following steps (written partially in pseudocode)

```
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = ""
# repeat 26 times
#   pick a random letter from the characters in the alphabet string
#   add that character to the list of characters in "key"
#   remove that character from the characters in the alphabet string
#       so we don't use it again
# once we've worked our way through the whole alphabet,
# print out the key so the user knows what it is
```

- To write the “encode” section, needed variables will be **alphabet**, **key** (entered by the user), **plaintext** (entered by the user), and **ciphertext** (produced by the program and printed out for the user). The general strategy will be to go through each letter in the *plaintext*, find the “index” (position) of that letter in the alphabet (where “a” = 0, “b” = 1, etc.), find the corresponding letter at that same position in the key, and add that letter to the ciphertext.

This section contains the most difficult part of the assignment. It's just a few lines of code, but understanding them may be a challenge initially. Use the diagram on the first page of this document as an example to trace through these lines of code, and see if it makes some sense.

```
plaintext = plaintext.lower()
ciphertext = ""
for i in range(len(plaintext)):
    alphabet_position = alphabet.find(plaintext[i])

    if alphabet_position >= 0:
        ciphertext = ciphertext + key[alphabet_position]
    else:
        # if position = -1, character wasn't found so
        # just add regular character unencrypted
        ciphertext = ciphertext + plaintext[i]
```

- The “decode” section will reverse this process, with the same variables needed: **alphabet**, **key** (entered by the user), **ciphertext** (entered by the user), and **plaintext** (produced by the program and printed out for the user). Once you understand the principles of the “encode” segment, writing the “decode” segment will be much easier.

## ***GETTING STARTED***

1. With paper and pencil, and perhaps in collaboration with a partner, identify what the main components are that you’ll need to include in your program.
2. Sketch out the basic flow of your program using a flowchart, and write some pseudocode that you can use to begin implementing those main components.
3. Either on your personal computer or on the *crashwhite.polytechnic.org* server, open up two windows: a text editor to write the program (on the left), and a shell to perform test runs of your program on the right.
4. Enter pseudocode in text editor, and then fill in more details for various parts of the code.
5. Save your program with the required name.
6. Copy this initial version of the program to the *crashwhite.polytechnic.org* server (to make sure that the upload/copy process works).

```
$ scp secret_code.py studentID@crashwhite.polytechnic.org:
~/forInstructor
```

7. Switching to the shell, run your program as new features are added, making sure to fix old problems before adding new components. You’ll repeatedly run through this edit-run, edit-run process to find bugs and fix them.
8. When your program is completed (but before the deadline), copy it to the server as indicated above. The newer version of your program will replace the old one there.

## ***QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)***

1. Given that for a given key, any plaintext letter will always be encoded to the identical cipher letter, how might a “frequency analysis” of a coded message allow someone to determine the plaintext? (Google “frequency analysis” in quotes if you’re not sure what this means.)
2. Why would transmitting a credit card number using our current version of the program be a bad idea? How would you alter the program to make it better?
3. Substitution ciphers have been around a very long time; one version is called the Caesar cipher because it was used by Augustus Caesar to encrypt his correspondence. With current analysis techniques, substitution ciphers aren’t considered secure (see 1. above). But what about SHA-512? MD5? Blowfish? Take a look at some of these algorithms and find out what distinguishes them.

## ***SAMPLE INTERACTIONS***

This program allows you to create keys, encode plaintext, and decode ciphertext.

What would you like to do?

1.create a key 2.encode 3.decode 4. quit > 1

The key is: ecqzvhsxgrdpiabftoyljuwmnk

You might wish to copy this for future use.

What would you like to do?

1.create a key 2.encode 3.decode 4. quit > 2

Please enter the key to use: ecqzvhsxgrdpiabftoyljuwmnk

Please enter the message to be encoded: This is AWESOME!

Your ciphertext is:

lxgy gy ewvybiv!

What would you like to do?

1.create a key 2.encode 3.decode 4. quit > 3

Please enter the key to use: ecqzvhsxgrdpiabftoyljuwmnk

Please enter the message to be decoded: lxgy gy ewvybiv!

Your plaintext is:

this is awesome!

What would you like to do?

1.create a key 2.encode 3.decode 4. quit > 5

Error. Invalid selection.

What would you like to do?

1.create a key 2.encode 3.decode 4. quit > 4

PhileasFogg:~ rwhite\$

## ***ADDITIONAL REFERENCES***

Singh, S. *The Code Book: The science of secrecy from ancient Egypt to quantum cryptography*. (2000).

Poe, E.A. *The Gold Bug*. (1843).

Stephenson, N. *Cryptonomicon*. (1999).

[http://en.wikipedia.org/wiki/Pretty\\_Good\\_Privacy](http://en.wikipedia.org/wiki/Pretty_Good_Privacy)

[http://en.wikipedia.org/wiki/GNU\\_Privacy\\_Guard](http://en.wikipedia.org/wiki/GNU_Privacy_Guard)