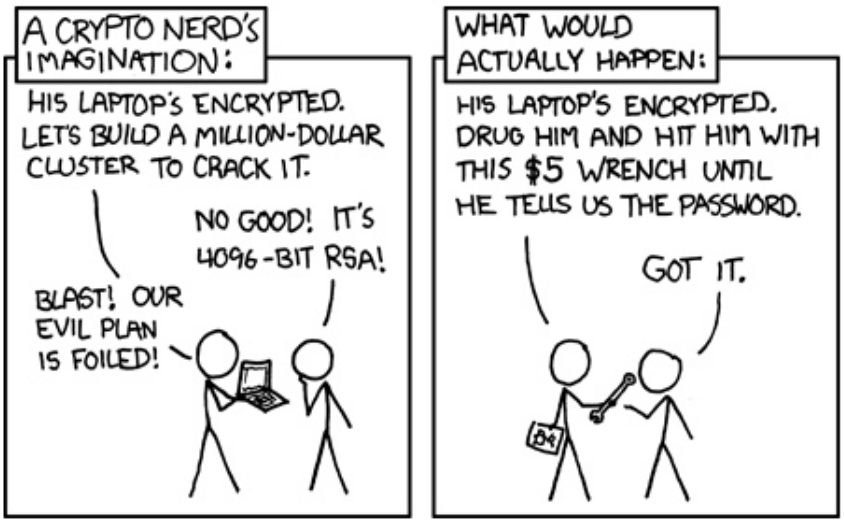
CS150 - Assignment 3
Due: Wednesday Oct. 5, at the beginning of class

For this assignment, we're going to be playing with basic cryptography. Cryptography is a field at the intersection of math and computer science that studies ways of securely communicating information. Throughout history, it has played an important roll in a variety of cultures in a variety of contexts. We'll play with some basic encryption techniques, but there is a wealth of information on modern encryption out there (Wikipedia is a good place to start if you're interested).

In cryptography, the two key functions are encryption, where we take a message and encrypt it, and decryption, where we take an encrypted message and present the original message. In this assignment, we'll play with a few different ways of doing this.



http://xkcd.com/538/

1 Starter

An important skill for any programmer is the ability to utilize and extend previously written code. For this assignment, I'm going to give you some code to get you started. This code has some variables and functions written for you already and you will be filling in some of the details.

To download the starter code go to:

http://www.cs.middlebury.edu/~dkauchak/classes/cs150/assignments/assign3/

You should see a single file called `encryption-starter.py`. Right-click on this file and select "`Save as...`" and then save it somewhere (for example, create a directory for this assignment and save it there). Then open WingIDE and open the saved file. This file does not have any syntax errors, however, until you fill in some of the details not all of the functions will run.

2 Warming up

To get you warmed up we're going to try a very simple method for encryption. You have a friend who says she has a great idea for encrypting a message. To encrypt it, you repeat each letter 3 times. To decrypt it, you reverse the process. For example:

"secret message" *would become* "sssseeecccrrreeettt mmmeeesssssaaagggeee"

- Write a function called `fools_encrypt` that takes the message as a parameter and returns the message encrypted using this scheme (Hint: the '*' is an operation for strings that replicates a string some number of times, for example try typing `"run "*10`).

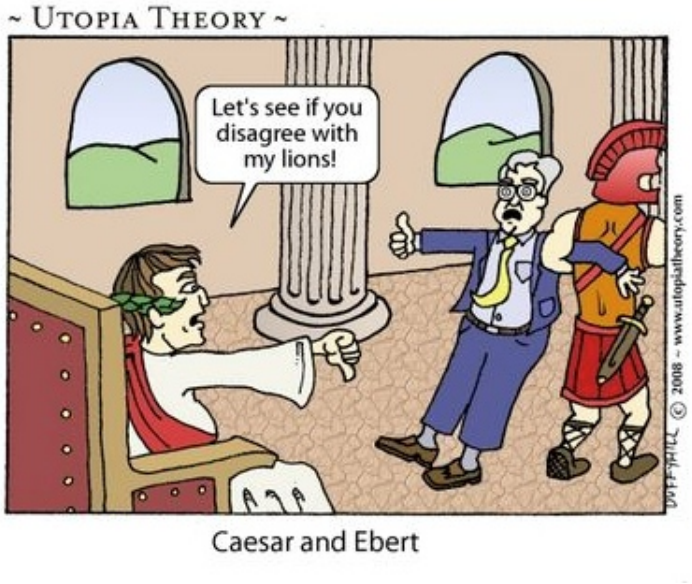
If you want a challenge, try and write a function called `fools_decrypt` that decrypts this. If you just want to try out decrypting it, copy the working function below and try decrypting your message to make sure it works:

```
def fools_decrypt(message):
    """Decrypts the encrypted message which was encrypted using fools_encrypt"""
    decrypted = ""

    for i in range(len(message)/3):
        decrypted = decrypted + message[i*3]
```

```
return decrypted
```

3 Caesar's method



Caesar and Ebert

http://www.utopiatheory.com/tag/caesar-and-ebert/

Now, let's try and implement Caesar's encryption scheme. If you need a review on how it works, take a look at the lab prep handout again. In the starter, I've given you a function called `shift_letter`. Play with this function and try some different letters (lowercase letters only) and different number offsets to make sure you understand how it works. Notice that you can also pass it negative numbers to shift the other direction.

1. **Encryption**

   Once you're comfortable with how `shift_letter` works

   - write a function called `caesar_encrypt` that takes two parameters: the message to be encrypted and the number offset for the encryption. You will very likely want to utilize the `shift_letter` function.

2. **Decryption**

   One of the nice things about this encryption scheme is that we can use the same function for encrypting and decrypting a message. Before reading on, think about how we could do this... I really mean it, think about it... To decrypt a message, we can use our `ceasar_encrypt` function, but now, give it a negative number. Encrypt a message or two and make sure that you can decrypt them.

Decrypt the following message with number offset 5:

'htruzyjwexhnjshjenxestertwjefgtzyehtruzyjwxeymfsefxywtstrcenxefgtzyeyjqjxhtujx'

It's a quote from a famous computer scientist (Dijsktra). You don't have to write it down, but it's a good check to make sure you've got things working.

Warning: our encryption schemes will only work for lowercase letters (specifically the letters a-z and space), so make sure that you're message doesn't have any other characters otherwise, it won't work correctly.

4 Substitution ciphers

Ceasar's method is just an example of a more general type of encryption schemes called substitution ciphers. For a general substitution cipher a *key* is used. The key has all of the same letters as the alphabet, but the order has been shuffled.

When encrypting, you encrypt a letter at a time. For each letter in the message you replace it with the corresponding letter in the key. For example, say you have following alphabet and key (I've put the indices in to make it easier to understand):

```
alphabet: a b c d e f g h i j k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z  ' '
          0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
key:      h v i e k s y r b d a  j  q  w  n  c  x  m  g  u  f  l  t  p ' ' o  z
```

and you see the character 'k', you look that up in the alphabet and notice that it is at index 10. You then use this index in the key to get 'a' as the letter. This process is repeated for each letter in the message until leaving an encrypted message. Make sure you understand this process before moving on to the coding.

1. **Key generation**

   Before we can actually encrypt anything, we first need to generate a key. We could do this by hand, but it's a pain. In the starter code, I've included a function called `keygen`. This function takes any string as a parameter and generates a random key based on that input string (think of it like a password). I've given you all the code for this function except it relies on the `splice` function, which needs to be implemented.

   - Fill in the details of the `splice` function.

   When you're done, you should be able to generate random keys using the `keygen` function. Try out a few examples and make sure that it is a permutation of all of the letters in the alphabet and that you get the same thing back if you give it the same password. For example:

   ```
   >>> keygen("doglover")
   ```

```
'teso flcdgwhnvrxizajpyqbmuk'
>>> keygen("doglover")
'teso flcdgwhnvrxizajpyqbmuk'
```

2. **Encryption**

   Now that you can generate random keys, you're ready to encrypt something.

   - Write a function called `subst_encrypt` that takes two parameters: the message to be encrypted and the key (not the password). The function should return the message encrypted using the key. Notice that there is a variable called `ALPHABET` which has all of the letters and space in order. You'll likely need to use this.

3. **Decryption**

   For a general substitution cipher we also need to write a decryption method.

   - Write a function called `subst_decrypt` that takes two parameters: the encrypted message and the key (not the password). The function should return the message decrypted using the key (i.e. the original message before encryption).

   This function should be *very* similar to the encryption function with just one or two minor changes (it's the same basic idea, but in the other direction).

   When you're done, you can generate a new key then encrypt and decrypt messages. For example:

```
>>> key = keygen("mypassword")
>>> encrypted = subst_encrypt("this is a secret message", key)
>>> encrypted
' krmqrmqiqmwyxw qhwmmijw'
>>> subst_decrypt(encrypted, key)
'this is a secret message'
```

   Decrypt the following message using the password "i like cs":

```
"hvwbuwihbhvacsbruenhbrbueedwrcbaibwowcbazbgenbrfwblfecsbgenbrfwbecdgbezzbugbrbuah"
```

   Again, you dont have to write it down, but its a good check to make sure you've got things working.

5 Extra credit

If you do extra credit, please include a comment at the top stating what you did.

- **[1 point]** Caesar's method can actually be written using our code for the general substitution cypher. Write a function called `shift` that takes as input a number. The function should return a key that is the inorder alphabet shifted up by the number input. For example:

  ```
  >>> shift(3)
  'defghijklmnopqrstuvwxyz abc'
  ```

  If you then use this key with your `subst_encrypt` function you should get the same answer as if you'd used your `caesar_encrypt` method with the number offset

- **[1 point]** Write a pair of functions `my_encrypt` and `my_decrypt` that implement your own encryption algorithm. You may rely on whatever input you would like, but make it clear in the documentation how they should be run. Points will be awarded based on difficulty and creativity.

6 When you're done

When you're all done you should have three different encryption/decryption methods and a method for generating random keys. Make sure that your program is properly commented:

- You should have comments at the very beginning of the file stating your name, course (including section number), assignment number and the date.
- Each function should have an appropriate *docstring*
- Other miscellaneous comments to make things clear

In addition, make sure that you've used good ***style***.

What to hand in:

Your program printed out. You should have implemented:

- `fools_encrypt`
- `caesar_encrypt`
- `splice`
- `subst_encrypt`
- `subst_decrypt`

Grading

	points
`fools_encrypt`	3
`caesar_encrypt`	3
`splice`	2
`subst_encrypt`	3
`subst_decrypt`	3
lab prep	3
comments, style	3
extra credit	2
total	20 (+2)