Breaking Encryption

Today you are going to write a program to break into an encrypted message.

I have encrypted a message using one of the first known encryption methods, the Caesar Cipher. This cipher rotates the letters of the original message by a fixed number. This number is the **key** to decrypting the message.

For example, if the **key is 3** and I want to encrypt the letter ‘a’, I count forwards 3 letters from ‘a’ to get to b -> c -> **d**. If I want to encrypt the letter ‘x’ with a key of 5, the count will wrap around. So ‘x’ would become y -> z -> a -> b -> **c**.

I have written a library that you can use to decrypt and encrypt your own messages. A library is code that someone else has written, that we can use in our programs. Using libraries saves a lot of work!

The message I would like you to decrypt is

xubbe, mubsecu je setu sbkr

For simplicity, any spaces or punctuation are passed straight through my cipher library, so commas and spaces on the input will become commas and spaces on the output.

Now let’s write some code!

Getting Started

The first task is to write some code that can encrypt a message typed in by the user. The Code Club tutor will have started you off with a file that looks like this:

```
from CaesarCipher import encryptMessage, decryptMessage
```

The first line of code is the import statement that gives you access to the functions in the Caesar Cipher library.

First you should create a message to encrypt, and a key to encrypt it with. We will be using variables called message and key. Add the following code:

```
message = 'This is a secret message'
key = 16
```

You will now use the **encryptMessage** function to turn this secret message into something that can only be read by someone who knows the key. Add the following code:

```
cipherText = encryptMessage(message, key)

print('Encrypted Message: ' + cipherText)
```

You can run this code by opening a terminal (the Code Club Teacher will show you how) and typing the following:

```
python3 ./BreakEncryption.py
```

From the message and key values that I used above you should get the following output:

```
jxyi yi q iushuj cuiiqwu
```

Congratulations, you have just used the Caesar Cipher to encrypt a message. Experiment with different messages and different values for key to see what you get.

Decrypting Messages

Decrypting messages is very similar to encrypting them. But this time we will be passing in the cipherText to the **decryptMessage** function.

You can just add the following code below your existing stuff.

```
# should decrypt to 'hello' with a key of 7
cipherText = 'olssv'
key = 7
```

This is re-assigning the variables that were originally created before. Now to decrypt this message.

```
plainText = decryptMessage(cipherText, key)

print('Decrypted Message: ' + plainText)
```

Now run this code using the same command as before, and it should look like this:

```
$ python3 ./BreakEncryption.py
Encrypted Message: olssv
Decrypted Message: hello
```

Brute Force Decryption

Now for the tricky bit, how do you decrypt a message if you don't have the key?

Ask yourself this question, how many values can the key take? The key is used to rotate letters through the alphabet.

- A key of 0 would turn a into a
- A key of 1 would turn a into b
- A key of 2 would turn a into c
- all the way up to a key of 26 which would turn **a** back into **a** because it would go round the whole alphabet. So a key of 26 is equivalent to a key of 0 (zero)

This means there are only 26 valid values for key from 0 to 25. Any other value will act just like one of the values between 0 and 25 (for example a key of 30 will act the same as a key of 4).

So can we just call our **decryptMessage** function 26 times and see if the message jumps out as us? For this we need to use a loop.

Here is a simple loop that prints out the numbers 0 to 25 in python. Note that we specify 26 as the upper bound because range considers the upper bound exclusive:

```
for x in range(0, 26):
   print(x)
```

Create a variable to contain the encrypted message from our challenge above, feel free to copy and paste this bit, put it above the loop as shown:

```
cipherText = 'xubbe, mubsecu je setu sbkr'
for x in range(0, 26):
   print(x)
```

Now instead of just printing out the value of x, let us use it as the key in calls to decryptMessage as shown:

```
for x in range(0, 26):
   plainText = decryptMessage(cipherText, x)
   print('Key: {} - {}'.format(x, plainText))
```

The print statement now uses string formatting to put variables into a printable string.

Run this code and you should end up with something like this:

```
Key: 0 - xubbe, mubsecu je setu sbkr
Key: 1 - wtaad, ltardbt id rdst rajq
Key: 2 - vszzc, kszqcas hc qcrs qzip
Key: 3 - uryyb, jrypbzr gb pbqr pyho
Key: 4 - tqxxa, iqxoayq fa oapq oxgn
Key: 5 - spwwz, hpwnzxp ez nzop nwfm
Key: 6 - rovvy, govmywo dy myno mvel
Key: 7 - qnuux, fnulxvn cx lxmn ludk
Key: 8 - pmttw, emtkwum bw kwlm ktcj
Key: 9 - olssv, dlsjvtl av jvkl jsbi
Key: 10 - nkrru, ckriusk zu iujk irah
Key: 11 - mjqqt, bjqhtrj yt htij hqzg
Key: 12 - lipps, aipgsqi xs gshi gpyf
Key: 13 - khoor, zhofrph wr frgh foxe
Key: 14 - jgnnq, ygneqog vq eqfg enwd
Key: 15 - ifmmp, xfmdpnf up dpef dmvc
Key: 16 - hello, welcome to code club
Key: 17 - gdkkn, vdkbnld sn bncd bkta
Key: 18 - fcjjm, ucjamkc rm ambc ajsz
Key: 19 - ebiil, tbizljb ql zlab ziry
Key: 20 - dahhk, sahykia pk ykza yhqx
Key: 21 - czggj, rzgxjhz oj xjyz xgpw
Key: 22 - byffi, qyfwigy ni wixy wfov
Key: 23 - axeeh, pxevhfx mh vhwx venu
Key: 24 - zwddg, owdugew lg ugvw udmt
Key: 25 - yvccf, nvctfdv kf tfuv tcls
```

Now scan down this by eye, can you find the message? Most of the decryption attempts show nonsense, but one of them should read clearly.

When you find the answer, tell the Code Club Teacher you have completed the challenge!

Summary

In this project you have learnt how to call functions, write loops and print values in Python code. You have seen how computers can be used to break encryption using brute force methods.

If computers can break encryption so easily, how does it work in real life? Modern encryption standards have many more potential keys, billions upon billions of them. The algorithms are far more complex and so it takes more time to try out every single key.