Cryptography

Python

Python is a programming language that is widely used in IT. It is free to use and you can download and install it on your home computer. Today we are going to be using and learning Python 3.

We are going to be using the software IDLE which will use different colours to indicate commands in the python language. For example, when you type `print` in IDLE it will be a pink-purple colour, rather than black.

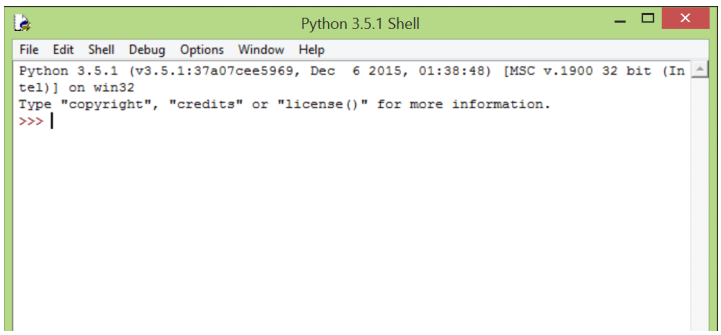
Because IDLE executes the commands you type right away, most of what you will be coding today will not be in IDLE itself. Instead you will type up your code in a second window and then it will be executed, or run, in the IDLE window.

Before we begin

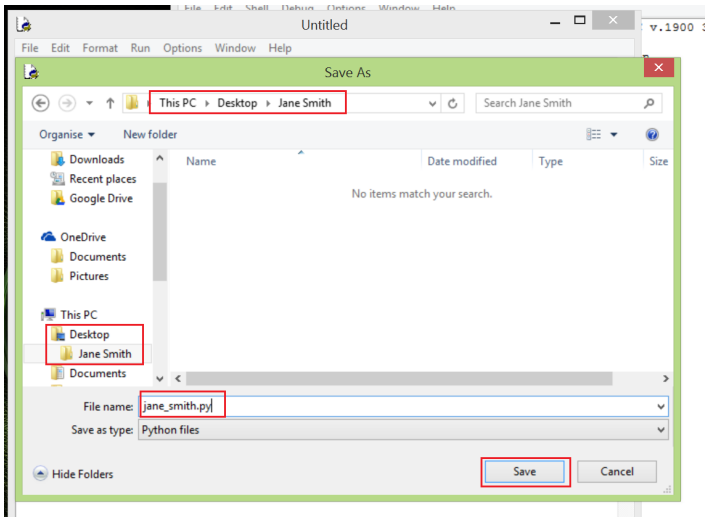
On the desktop of the computer create a folder and name it after yourself, eg Jane Smith. It is in here that you will be saving your code today. To create a folder, right click on the desktop, and then in the menu select ‘New’ -> ‘Folder’. You can then name it after yourself.

Setting up IDLE

In the Start menu, search for the program IDLE and click on it. The program should look something like this:



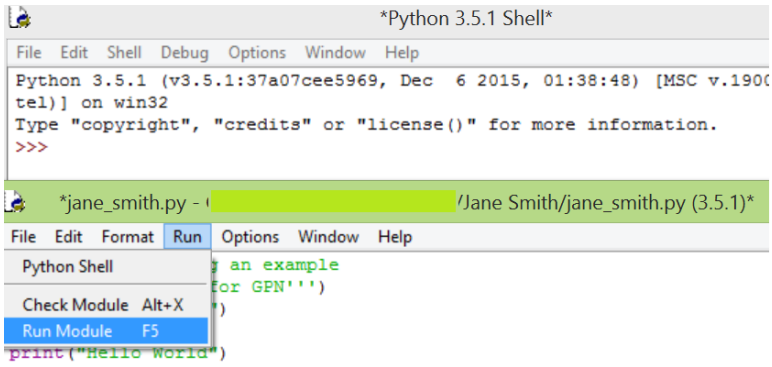
You then need to create a file in which you will be writing your code, so click on File in the top left, and then ‘New File’. You should now have a second window which is blank. In the second window, click File, and then ‘Save As’. You want to save this file in the folder you created on the desktop, and also name it after yourself (with no spaces), for instance jane_smith.py. Remember to add .py to the end so the program knows it is a Python file.



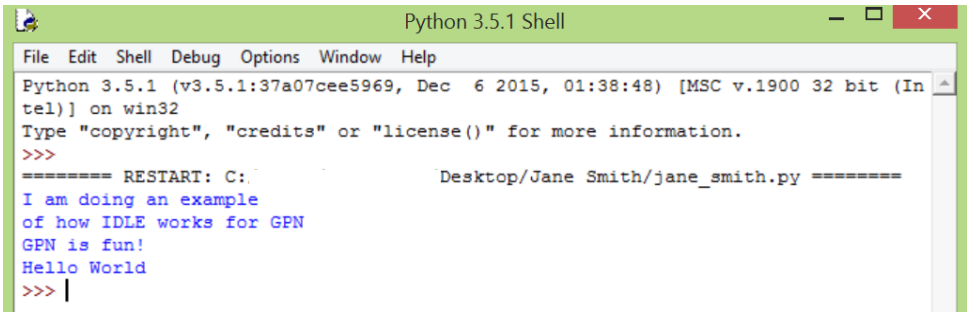
It is in this file that is named after you that you will be doing most of the exercises today.

Running IDLE

As was mentioned earlier, IDLE runs the commands you have typed immediately. As such, for a lot of the work today, you are going to be typing it into the file you just created and saved with your name. After you have typed your code in this file you can then run the file from the 'Run' menu.



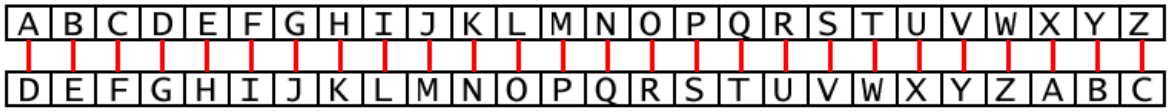
The code will then run in the IDLE window



The code that you write for Cryptography workbook should all be in the .py file and not in IDLE.

The Caesar Cipher

The Caesar Cipher is named after Julius Caesar of Ancient Rome who was the first person to use this style of cryptography. He used it to protect the messages he sent to his army. The Caesar Cipher is a substitution cipher where letters of the alphabet are shifted by a fixed amount. For example, Julius Caesar moved the alphabet 3 places when writing his messages to his generals.



Alphabet shifted by 3 spaces.

As a result, the information in the messages would go from looking like:

Generals, we attack the Gauls at dawn.

to

Jhqhudov, zh dwwdfn wkh Jdxov dw gdzq.

While Julius Caesar shifted the alphabet by 3 places, you can shift the alphabet by a different amount to make it more secure. Today you are going to learn how to write a Python program that will shift the alphabet for you and encrypt your message using the Caesar Cipher.

Step 1: Input your secret message

To start we'll be asking users for input and creating variables to store important information we need the computer to remember.

1. Make a variable to store the secret message the user will input. Variable names should be something that will make sense when you (or someone else) reads your code, such as `secret_message`.
2. Make a variable to store the amount the alphabet will be shifted by. This shift is called the key.

Since key and `secret_message` are user inputs we can use the `input(...)` function to ask people to type in their input.

The `input()` function is an inbuilt function in Python 3. It prompts users to supply information to the computer program. For example if I wanted a user to input their age, I would type:

```
user_age = input("What is your age? ")
```

This line takes the user input and assigns it to the variable `user_age`. The input function stores all inputs as a string. If I wanted to check the value for user_age, I would use the print function:

```
print("The age of the user is: ", user_age)
```

3. Use the `input()` function to prompt users for their secret message.
4. Use the `input()` function to prompt users for their key value. Your output should look similar to Illustration 1.

```
Type your message (in lower case letters, write spaces as '_') to be encoded: i_love_gpn
Enter the key value (0-26): 0
```

Illustration 1: Your code should now prompt the user to input their secret message and enter the key value. You'll see in the next step why we require messages to be written in lowercase letters and we use "_" for spaces.

5. Because the key input from the user is a string, you need to convert the key value into a integer, by using "`int()`" e.g.

```
user_age = input("What is your age? ")
user_age_integer = int(user_age)
```

Hint! **Remember** to type all of this code into the text .py file, not directly into IDLE.

Hint! **Remember** that the value for the `key variable` needs to be stored as a number rather than a string. You can do this by using the `int() function.`

Step 1. Achievements:

- ❏ Create a variable to store the secret message.
- ❏ Create a variable to store the key value.
- ❏ Ask the user for their secret message using `input().`
- ❏ Ask the user for a key value using `input().`
- ❏ `Convert the key value into an integer (number) using int().`

Step 2: Converting letters to numbers

Here we're going to take the first steps to encode the user's `secret_message`! We need to tell the computer the alphabet we're using so it can shift the alphabet to encode our message. To start we'll select the first letter in `secret_message, then we'll` find the position of the first letter in our alphabet.

1. Create a variable to store your alphabet.
2. Write down the "allowed" alphabet for your code as one long string, with no spaces in between the letters. We're going to include spaces in our alphabet, we will use "_" to represent a space. We're using "_" because later, if you need to copy a message it is easy to miss a space, but easy to see "_".

```
alphabet = "abc...xyz_"
```

Hint! Lowercase and uppercase letters are different in Python. For example, 'a' is not the same as 'A'. Also remember to include '_' in your alphabet string.

Now we're going to encode the first letter in the `secret_message`. We're only doing the first letter because it's good to start simple!

3. Create a variable to store the first letter of `secret_message`.
4. Select the first letter in the secret message and store it in your variable. To select a letter in a string we use square brackets, for example to select the 5th letter in `secret_message` we would write:

```
test_letter = secret_message[4]
```

Hint! Python starts counting from 0 not 1. What number should you use to select the 1st letter in secret_message?

5. Now we need to find the position of `test_letter` in the alphabet. First create a variable `position`, to store this value.
6. Next use the function `<<some string>>.find()` to find the position of `test_letter` in the alphabet. For example if I have `alphabet = "abc...xyz_"` and I want to find the position of the letter "b" I'll type:

```
position_of_letter_b = alphabet.find("b")
print(position_of_letter_b)
```

What will you type to find the position of `test_letter` in the alphabet?

7. You can use the `print()` function to check your values as go. Use the `print()` function to check your value for position. You can comment out this print statement when it's not needed. Your result should look similar to Illustration 2.

Hint! The `print()` function helps you debug your code. Later you will be able to comment these statements out with the # symbol at the start of the line, e.g:

```
# print("this line won't print")
```

```
Type your message (in lower case letters, write spaces as '_') to be encoded: i_love_gpn

Enter the key value (0-26): 0
test: letter =  i
test: position of the letter in the alphabet is =  8
```

Illustration 2: Using the print command to check your code, you should see the code selects

the first letter of your message and prints out its position in the alphabet.

Step 2. Achievements:

- ❏ Create a variable to store the first letter of your secret message: `letter`.
- ❏ Select the first letter of your secret message and store it in `letter`.
- ❏ Using `print()`, you test the result of your program by printing `letter`.
- ❏ You create a variable, `position`, to store the position of `letter` in the alphabet.
- ❏ Using the function `find()` your program returns the position of `letter` in the alphabet and stores it in the variable `position`.
- ❏ Using `print()`, you test the result of your program by printing `position`.

Step 3: Shift the position of your letter by the key value

Now you know the position of the first letter, we're going to shift the position by the key value to get a new position. Because we have only 27 characters, it only makes sense to have a key value in the range [0-26]. let's get started,

1. Create a variable to store your new position: `new_position`
2. `Add the key value to position and store in the variable new_position`
3. `Test your output by by printing out your result using print()`. Try different key values, you should see something similar to Illustration 3.

```
Type your message (in lower case letters, write spaces as '_') to be encoded: i_love_gpn

Enter the key value (0-26): 0
test: letter =  i
test: position of the letter in the alphabet is =  8
test: new position is =  8


Type your message (in lower case letters, write spaces as '_') to be encoded: i_love_gpn

Enter the key value (0-26): 25
test: letter =  i
test: position of the letter in the alphabet is =  8
test: new position is =  33
```

Illustration 3.a: Here we show two examples adding key to position. In Example 1 key = 0, position = 8, and new_position = 8 (= 0 + 8). In Example 2 key = 25, position = 8, and new_position = 33 (=25 + 8). However now new_position falls outside of our key range.

Can you see a problem here? The number 34 is not in the range [0 - 26]. Will it still correspond to a letter, even though it is outside the range [0 - 26]?

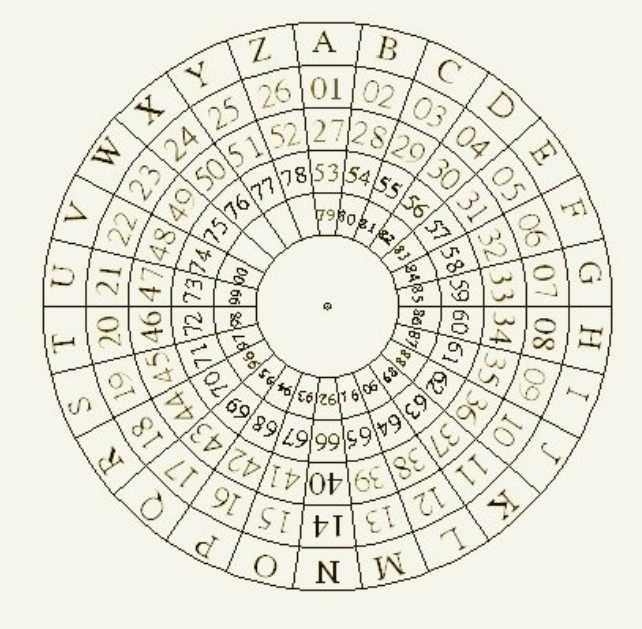


Illustration 3.b: This illustration shows us how numbers outside the range of 01-26 continue to correspond to letters of the alphabet provided (note this illustration does not include '_' or space). In the most outer ring of numbers you can see 01-26, but the next ring has 27-52, showing these numbers continuing to correspond to letters of the alphabet.

Clearly we have two situations to deal with:

- `When new_position` is less than or equal to 26
- `When the new_position` is more than 26 we need to map it back into the range [0-26] by subtracting the value 27

In these situations we can use an `if` statement. If statements allow your code to make a decision for you. For example:

```
if age > 15:
    print("You are old enough to learn to drive")
else:
    print("You are too young to learn to drive")
```

This section of code uses an if statement to check the value of the variable age and if it is greater than 15 it will print out "You are old enough to learn to drive", if the variable 15 or lower, it will instead print out "You are too young to learn to drive".

Hint! `if` statements have the general form:

```
if <condition 1>:
      <do something>
elif <condition 2>:        # This is optional.
      <do something else>
```

```
    else <condition 3>:        # This is optional
          <do something else>
```

4. Use an `if` statement to check `new_position` in the range [0-26]. Use the above structure to create your own `if` statement and replace `<condition>` and `<do something>`.
5. Test your output by by printing out your result, you should get something similar to Illustration

```
Type your message (in lower case letters, write spaces as '_') to be encoded: i_love_gpn

Enter the key value (0-26): 25
test: letter =  i
test: position of the letter in the alphabet is =  8
test: new position is =  6
```

Illustration 3.c: Previously this would have resulted in new_position = 33, but because 33 is greater than 26, the if statement applied to the result and 33 - 27 = 6

Step 3. Achievements:

- ❏ Create a variable to store the new position of your letter: `new_position`.
- ❏ Add `key_integer` to position and store the answer in `new_position`.
- ❏ Using an `if` statement you keep the range of `new_position` between [0,26].
- ❏ Using `print()`, you test the result of your program by printing `new_position`.

Step 4: Encrypting your the first letter

Now you'll use all the tricks you've learned so far to encrypt your first letter!

1. Create a variable for your encrypted letter: `encrypted_letter`.
2. Using the same trick used in Step 2, part 4, select the letter at position `new_position` from `alphabet`
3. Test your output by by printing out your result using `print()`

Step 5. Achievements:

- ❏ Create a variable to store the encrypted letter: `enctrypted_letter`.
- ❏ Select the letter in the alphabet corresponding to `new_position`.
- ❏ Using `print()`, test the result of your program by printing `encrypted_letter`.

Step 5: Encrypting a whole messages

Let's encrypt a whole message! To start:

1. Define an empty string variable to store the final encrypted message. An empty string is just a string with nothing in it:

```
empty_string = “ ”
```

Now you can use a for loop to iterate through all the letters of secret_message. Let’s look at an example which takes every letter in a string and adds the phrase “is my favourite letter” (try it!):

```
word = “elephant”
for letter in word:   # start of for loop
      print(letter+“ is my favourite letter”)   # body of for loop.
```

Hint! To concatenate (add) strings, use +

2. Wrap code in a for loop to loop over all characters in the secret message. Use + to build your encrypted message - see Illustration 4 for an example.

Hint! Experiencing errors? Check:

- where should you put the for loop in your code?
- where have you defined encrypted_message? Inside or outside of the for loop?
- have you indented your code inside the for loop?
- check all terms with square brackets [ ], do they have numbers in them? Is this correct?

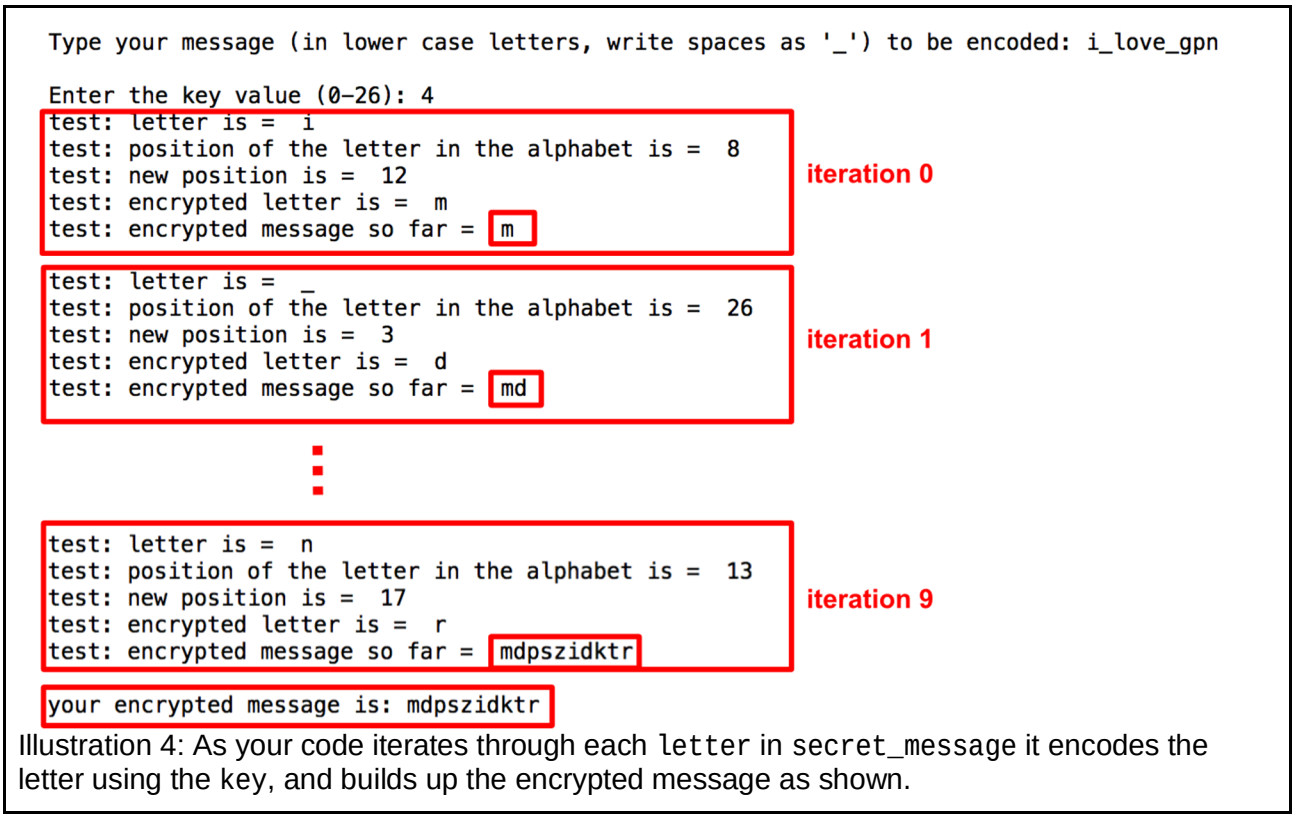


Illustration 4: As your code iterates through each `letter` in `secret_message` it encodes the letter using the key, and builds up the encrypted message as shown.

3. Print out the encrypted message
4. Tidy up your code by commenting out all "test" print statements.

Step 5. Achievements:

- ❏ Use a `for` loop to encrypt all `letters` in the `secret_message`.
- ❏ Using `print()`, print out the encrypted message.
- ❏ Remove extra print statements.

Step 6 extension: Using your code to decode a Caesar Cipher.

This section is optional - do it if you have time!

If you have an encoded message and the key, how would you decode the message? Hint: you need to do the opposite of adding the key - you don't need to change much! To get started you need to let the user input a mode: encrypt or decrypt.

1. Make a variable to store the mode (encrypt or decrypt).
2. Use the `input()` function to prompt user to input one of the two modes.
3. Use an `if` statement to check the mode and modify the key if necessary. Test your code as shown in Illustration 5.

Hint! Uncomment the test-print statements to troubleshoot any issues!

Example 1: encryption

```
Do you want to encrypt (type e) or decrypt (type d)?e

Type your message (in lower case letters, write spaces as '_') to be encoded: i_love_gpn

Enter the key value (0-26): 4
your encrypted message is: mdpszidktr
```

Example 2: decryption

```
Do you want to encrypt (type e) or decrypt (type d)?d

Type your message (in lower case letters, write spaces as '_') to be encoded: mdpszidktr

Enter the key value (0-26): 4
your encrypted message is: i_love_gpn
```

Illustration 5: An example of encoding and decoding a message. Try it yourself!

4. Try to decode the message pyeetxhgytrhntkhwd wth key = 7

Step 6. Achievements:

- ❏ Create a variable to hold onto the mode.
- ❏ Prompt the user in input a mode.
- ❏ Use an `if` loop to detect the mode and modify key if necessary.

Step 7 extension: Make your code more robust!

Can you think of different ways to break your code? What happens if users can input the wrong input - try it!

1. First try encoding and decoding some incorrect inputs to see what happens. Examples include:
    a. `Entering key` values which are less than zero or greater than 26.
    b. Typing in a `secret_message` which has characters not contained in the alphabet, e.g. `i_love_gpn!!!!!`

There are several ways to make your code more robust to handle user input errors. We're going to look at one solution, but you can use your own solution! It's going to be tricky because we need to make sure the `key` is in range ***and*** the letters in the `secret_message` are contained in the `alphabet`.

We are going to use the `if` statement to check both the `key` is in range and the `secret_message` is in `alphabet`. It will look like:

```
if <check key in range> and <check secret_message in alphabet>:
    <your code here>
else:
```

<print error message>

2. Using what we've done so far, you're able to fill in the parts:
    - <check key in range>
    - <print error message>

However <print error message> is trickier. This is because we need to take each letter in the secret_message and check it's in the alphabet. What we need to check is:

- is letter in the secret_message ***and***
- is the same letter in alphabet.

Luckily Python has a built in function to take care of this: it's called the `all` function. It looks like:

```
all(<thing> in <group 1> for <thing> in <group 2>)
```

The function all will return true when <thing> is in <group 1> and <group 2>.

3. Your job now is to replace <check secret_message in alphabet> with the all function. You need to decide what will replace <thing>, <group 1>, <group 2> in your code.

Can you think of any other places there may be user input errors? How would you fix it?

Step 7. Achievements:

- ❏ Compare user input to `alphabet`.
- ❏ Report an error if user input is not in `alphabet`.
- ❏ Compare key to the accepted range [0,26].
- ❏ Report an error if key is not in the range [0,26].
- ❏ Find other errors and fix them!

Step 8 extension: Crack the Caesar Cipher with brute force!

Did you know you can break your Caesar Cipher with brute force? It's easy if you know the length of the alphabet being used. I'm going to assume our alphabet is the same as we've used previously: "`abcd...xyz_`"

We know there are 27 different characters, which means the key to break the code must be in the range (0,27). Thus if we take any encrypted message and try every value of the key from 0 to 27, and print out all 27 combinations, one message should make sense!

1. You need to modify the `input()` function in Step 6 and add an additional mode for brute-forcing the Caesar Cipher. Users should now have the option to encode, decode and crack the Caesar Cipher.

2. Modify the `if` statement in Step 6 used to check the mode. It should now be able to handle the brute-force option. Think carefully about your structure. Encode and decode are very similar steps. A simple idea is presented below, but if you can think of a better way: do it!

```
If mode == <brute force>:
      <brute force crack!>
else:
      elif mode == <encode>:
            <set key value>
      else mode == <decode>:
            <set key value to something else>

      <encode/decode code>
```

3. Now you need to force the key to increment through the range (0,27). You can use a `for` loop, a `while` loop, or you can choose your own method!

   If you choose to try a while loop, they look like this:

```
While <condition>:
      <do stuff>
```

   This loop will continue to run until the condition is false. Lets look at an easy example you can start with.

```
x = 0                          # initialise loop variable
While x < 27:                  # start of while loop: is x less than 27?
                               # if true, continue. If false, stop.
      print ("I love cheese!") # print "I love cheese"
      x = x + 1                # increase x by one
      print (x)                # print x and go back to the start.
```

   See what happens? At first `x = 0` the while loop checks the condition: is `0 < 27`? The answer is yes, so the loop prints the statement "I love cheese"; then it adds 1 to x and prints out the value of x and returns to the start of the loop. The while loop will stop when x reaches the value 27.

Hint! If your `while` loop keeps running and doesn't stop, this is called an infinite loop. You can stop an infinite loop by pressing CONTROL-C.

If you see an infinite loop, your `while` loop is probably defined in the wrong place. Try defining it somewhere else!

Hint! Don't forget to initialise your variable (which variable?) used in the `<condition>`. Don't forget to update your variable in the loop or your while loop won't stop. Your tutors can help you plan your code if you need it.

4. The body of your while loop needs to be decoding the secret message for each value of the key. You could write some new code or you may be able to copy-and-paste some code you've already written. Think about it! Ask your tutors if this is tricky!
5. Make sure your code prints out the value of the key value tried and the message so you can read all the outputs and pick out the decoded message!
6. `Test your code then try to decode this: rhntxylykoytutwahwheumytzhktrhnktphkd`

Step 8. Achievements:

- ❏ Modify `input()` to include a "break code" option (used in Step 6).
- ❏ Modify `the if statement` used in Step 6, to accept `mode == <break code>`
- ❏ Use a loop to try all key values when decoding the message.
- ❏ `Print out all key values and corresponding messages`

Step 9: advanced challenge! Check your decoded messages against a dictionary.

In the previous step we took an encrypted message, tried all possible key values and then printed out all of our attempts to decode the message. Then we had to read all 27 messages and pick which one made sense - in this challenge you're going to compare the output message to a dictionary file and when English is detected in the message, the code will terminate.

We'll give you some hints but we'll leave it up to you to figure this one out!

Hint! You can find a dictionary file here http://www.math.sjsu.edu/~foster/dictionary.txt

Hint! You'll need to save the dictionary file into the same directory as your python code.

Hint! You'll need to read in the dictionary file, look at the code below and see what it does!

```
    english_dictionary_file = "dictionary.txt"

    with open(english_dictionary_file) as file:
    english_dictionary = file.read().splitlines()
```

`Hint!` `You can use the all function to compare words in the dictionary to words in the output message.`

Step 9. Achievements:

- ❏ Read in a dictionary file.

- ❏ Compare words in the dictionary to words in message.
- ❏ Print out message when English is detected.

Step 10: advanced challenge! Change your alphabet

Do you want to include more letters and symbols in your secret message? If so you'll need to update the alphabet! See if you can figure it out! Remember, this step is optional and you can complete this step at any time!

Hint! If your alphabet is longer how will this impact your code? Check:

- Your print statements correctly reflect the range that can be input by users
- Your maths is correct in your code (see Step 3 in particular!)

Step 8. Achievements:

- ❏ Change the alphabet
- ❏ Make sure your code is adjusted for the new length of the alphabet.