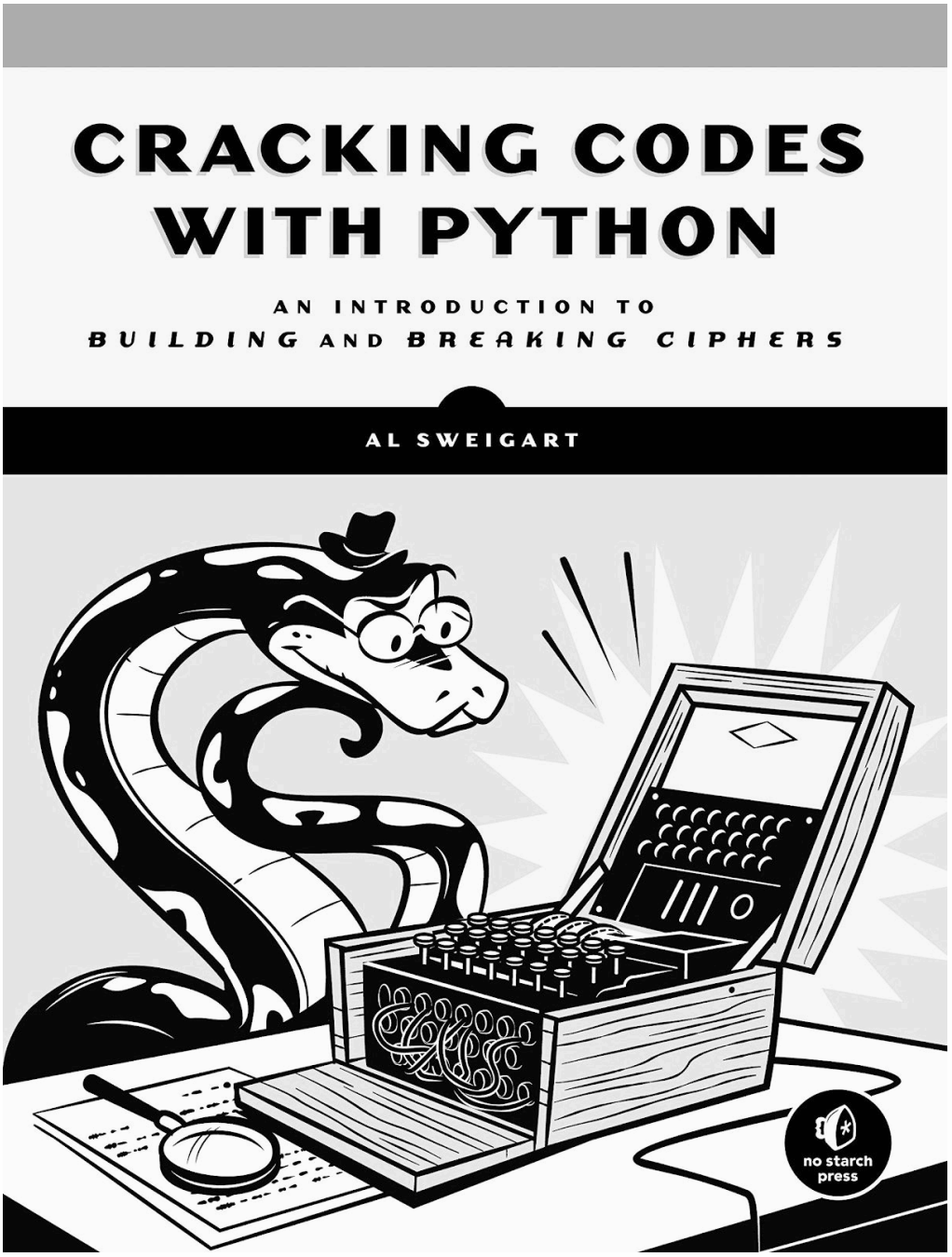


Code Breaking in Python - Cryptography

By Joe Brown 20th January 2020

This workshop is intended for those who have used some block code (eg - scratch / microbit) and/or are just starting out with text based code.

Sources:
“Cryptography with Python” by Tutorials Point tutorialspoint.com/cryptography_with_python
“Cracking Codes with Python” by Al Sweigart inventwithpython.com/cracking

We're making and breaking secret codes or secret messages using 'encryption' to create them and 'decryption' to translate them. We'll be writing in the language of python.
`As we add new code, previously written code will be in grey.`
To code in python on your web browser, follow this link: jumpto.cc/python-new

1. Reverse Cypher

This takes your message and reverses it. For example:
'Hello There!' becomes '!erehT olleH' Let's code it!

1.1 Input
Firstly we need to take in the secret message by using **raw_input**.
Below is the way to code it. Type it in the left pane of the web page.

```
message = raw_input("What is your message? ")
```

1.2 Variable
We're storing the input that is typed as a **variable** called "message"
A **variable** is a container for storing data values - in this case the variable "message" will store your secret message.

1.3 Print
We can test this by showing the message on screen with **print**:

```
message = raw_input("What is your message? ")
print(message)
```

Click the run ▷ button , and look on the right hand panel on the screen.
You should see "What is your message?" appear in this panel. Click on this right panel and type an answer and press enter. Then you should see your answer displayed back to you.

The **input** (what you've typed) is being stored in the **variable** "message" and displayed back to you using **print.**

Now we need to reverse this message to create a reverse cypher.

1.4 Length
Delete the print(message) line.
We need to measure the length of the input so we know how many letters to we need to modify. We can do this with **len**, and we'll use the **variable** "letter":

```
message = raw_input("What is your message? ")
letter = len(message) - 1
```

We need to minus 1 off this as computers count from 0, so as we work through the letters if there are 5 letters in the message, we'll count them 0,1,2,3,4.

1.5 While Loop

The way we'll work through the message letter by letter is with a **while** loop.
Let's say the message is "hello" then letter = len("hello") - 1 which means letter = 4.
We'd want to through letter by letter in reverse: o l l e h storing them into "translated" to give us our secret message "olleh"
We do this with a while loop:

```
message = raw_input("What is your message? ")
letter = len(message) - 1
while letter >= 0:
      letter = letter - 1
```

The above code basically says while letter is more than or equal to 0, take 1 off letter.
Code that is "indented" - moved across from the margin - is within the while loop.
Make sure you don't miss this! Press the tab key to indent / move across.
In the example "hello", letter = 4 so in the while loop: letter will count down 4, 3, 2, 1, 0.
We can use this to reverse our letters of our message. We can use **print** to test this:

```
message = raw_input("What is your message? ")
letter = len(message) - 1
while letter >= 0:
      print(message[letter])
      letter = letter - 1
```

So when you run your code and type in a message, you'll see the reverse of your message printed out letter by letter on a different line. That's what the `message[letter]` is doing - the square brackets are for looking in the **variable** "message" for each letter in the whole message. We know the reverse cypher is working but now we want to display the encoded message properly!

1.6 Rewriting a variable

Remove the `print(message[letter])` line. Let's add another **variable** called "translated" and we'll start with it empty like so: `translated = ""`.
As we go through letter by letter, we'll add the letters into this variable:
`translated = translated + message[letter]`
This is saying "translated" is the previous letters in the while loop and the current letter.

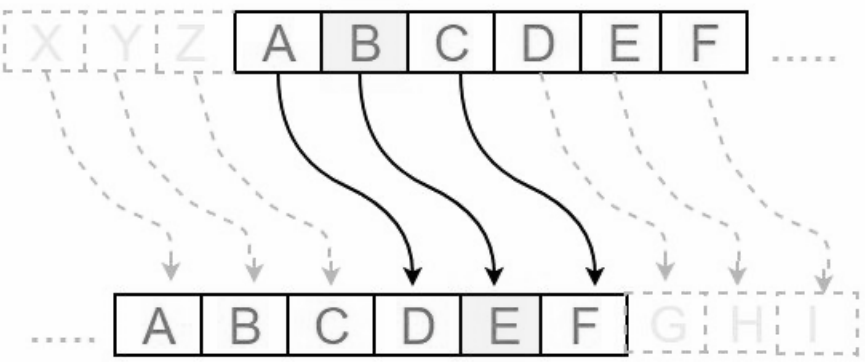
```
message = raw_input("What is your message? ")
letter = len(message) - 1
translated = ""
while letter >= 0:
      translated = translated + message[letter]
      letter = letter - 1
print("Your code is " + translated)
```

Test out your code - encrypt some messages. You can also put reversed messages in and it will decrypt and give you back the original message! Give this a test. Maybe give a reversed secret code to someone else in the workshop to translate back.

This is the simplest but also the weakest form of encryption as others can easily work out your secret message! - you can almost tell it just by looking at it. So let's try a more secure cypher.

2. Caeser Cypher

In the Caeser cypher, each letter is replaced by a letter a fixed number of positions down the alphabet, below is the example for key of 4 positions switched:



2.1 Alphabet & Key
Start a new file, by either clearing out previous code or refreshing the page.
We'll need the alphabet to cycle through to let's add that in.
We also need a 'key' which is how many letters we'll be shifting the message by.
For this example we're using 6 but you could use any number:

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = 6
translated = ""
```

2.2 For Loop
This time instead of a while loop, we'll use a **for loop**:

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = 6
translated = ""
for letter in message:
```

Where in this case it will go through each letter in the message and we can use letter as a variable. Which is great as we'll need to modify each letter.
Like the **while loop** you need to indent (press the tab key) to put code within the loop.

2.3 If Statements

If statements are very useful as checks. In this case we can use it to check **if** the letter in the message is in our alphabet, then we know to shift it. We're only working with lower case letters with this Caeser cypher, so no capitals, numbers or symbols.

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = 6
translated = ""
for letter in message:
      if letter in alphabet:
```

2.4 Else Statements

If the letter in the message isn't part of our alphabet (for example a space), we want to pass it straight into the translated output, by using an **else statement**.
Don't forget code that's applied during a statement is indented (tab key)!

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = 6
translated = ""
for letter in message:
      if letter in alphabet:

      else:
            translated = translated + letter
```

2.5 Find

When we know the letter is in our alphabet using that **if statement**, we then need to find out what position it is in the alphabet. The python function **find()** is really useful for this. So let's find the position of the letter and then modify it by our key:

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = 6
translated = ""
for letter in message:
      if letter in alphabet:
            number = alphabet.find(letter)
            number = number + key

      else:
            translated = translated + letter
```

So we're finding the position of the letter in the alphabet then shifting it by our key.

For example for the letter “c”:
`number = alphabet.find(letter)`
“c” is position 2 in alphabet, so number = 2 (remember computers count starting from 0)
`number = number + key`
then as our key = 6, number = 2 + 6 = 8.

We can then turn that number back in to a letter with alphabet[number]
alphabet[8] = g

We’ll then write that letter to our translated message, like we did with the reverse cypher.
translated = translated + alphabet[number] or in our example case:
translated = (previous letters of translated message) + g

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = 6
translated = ""
for letter in message:
      if letter in alphabet:
            number = alphabet.find(letter)
            number = number + key
            translated = translated + alphabet[number]
      else:
            translated = translated + letter
print("Your code is " + translated)
```

Why not test the code now? You will find some messages will work (for example “hello”) and some give the error “index out of range” and these are when your message contains later letters in the alphabet like x, y and z. Why do you think that is?

2.6 Length & wrap around
There’s a slight issue here:
With our key=6, “z” needs to become “f”, so wrap back to the start of the alphabet.
Let’s run through the code:

`number = alphabet.find(letter)` = find(z) = 25
`number = number + key` = 25 + 6 = 31.
`translated = translated + alphabet[number]`
alphabet[31] would return an error “out of range”.

Our alphabet doesn’t go beyond “z” or alphabet[25] so that’s why it’s “out of range”
With our key of 6, we need “z” to return “f” which is alphabet[5].
The way to return number = 5 is to wrap around to the beginning of the alphabet by taking away 26 or the length of the alphabet using **len()** which is the python function for length.

We can check if the number is more than the maximum size of our alphabet (which is 25) by doing length of alphabet - 1. “>” is the sign for more than

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = 6
translated = ""
for letter in message:
      if letter in alphabet:
            number = alphabet.find(letter)
            number = number + key
            if number > (len(alphabet) - 1):
                  number = number - len(alphabet)
            translated = translated + alphabet[number]
      else:
            translated = translated + letter
print("Your code is " + translated)
```

So now we’ve made any letter being shifted beyond the alphabet wrap back round!
Here’s the example for “z” working through the code, remember we want it to become “f”

`number = alphabet.find(letter)` = find(z) = 25
`number = number + key` = 25 + 6 = 31
`if number > (len(alphabet) - 1):` if 31 > (26 - 1) then:
`number = number - len(alphabet)` = 31 - 26 = 5
`translated = translated + alphabet[number]` = previous letters + f

There we go! A fully working caeser cypher for non capital letter messages. Test it out and make sure you aren’t seeing any errors for any message made of non capital letters.

2.7 Decrypting Caeser
The easiest way to decrypt our message is to subtract the key!
So with our current encrypter code with key=6, “hello” is outputted as “nkrru”
This is how to decrypt, modify your code with the following:

- subtract the key from the number rather than add it.
- Change the **if** number bigger than alphabet to **if** number less than 0
- And add the alphabet to number rather than subtract it
  (This is to fix the wrap around of the alphabet in reverse)

The solution is on the next page, but see if you can code without it!
Run the code and enter “nkrru” and you should get “hello” back.
Let’s try another one, decrypt “qnuux qxf jan hxd cxmjh” with key = 9!
If you’ve managed this, why not send a secret message to someone else in the workshop?
Choose a key between 2 and 25 and encrypt a message with this key.
Give them your encrypted message and your key value, they should be able to decrypt it.

solution:

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = 6
translated = ""
for letter in message:
    if letter in alphabet:
        number = alphabet.find(letter)
        number = number - key
        if number < 0:
            number = number + len(alphabet)
        translated = translated + alphabet[number]
    else:
        translated = translated + letter
print("Your code is " + translated)
```

2.8 Brute Forcing Caeser

What if you don't know the key for someone's Caeser code? A common hacking term is "brute forcing" in which we try lots of different options to work out the answer.
We can modify the decrypter to try all the keys possible in a Caeser cypher!

Edit your decrypting code so the whole of the **for loop** "for letter in message" is inside another **for loop** testing key in the range 0 - 25, and print out each one.
You'll need to clear "translated" for each time you test the key.
To do for loops for a number you'd typically do `for [variable] in range(number):`
The solution is below, but see if you can code without it!

Run your bruteforcing code for this message:
"wdetujaan jhxcv qgjit udgrt ndj wpkt udjcs dji bn ztn xh uxuittc"
Can you decrypt it? What is the key value? Can you print out the key values too?

solution:

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
for key in range(len(alphabet)):
    translated = ""
    for letter in message:
        if letter in alphabet:
            number = alphabet.find(letter)
            number = number - key
            if number < 0:
                number = number + len(alphabet)
            translated = translated + alphabet[number]
        else:
            translated = translated + letter
    print(key, translated)
```

Another **for loop** solution is "for key in range(26)" - this will try key = 0 to 25.

3. Vignere Cypher

3.1 Encrypting Vignere

While the Caeser cypher is more secure than the reverse cypher, we can do better!
The Vignere cypher is like a layered Caeser Cypher - instead of the key being a number it's a word. For example, if the key is "pizza" and the message is "hello good day":

message	**h**:7	**e**:4	**l**:11	**l**:11	**o**:14	**g**:6	**o**:14	**o**:14	**d**:3	**d**:3	**a**:0	**y**:24
key	**p**:15	**i**:8	**z**:25	**z**:25	**a**:0	**p**:15	**i**:8	**z**:25	**z**:25	**a**:0	**p**:15	**i**:8
translated	**w**:22	**m**:12	**k**:10	**k**:10	**o**:14	**v**:21	**w:**22	**n**:13	**c**:2	**d**:3	**p**:15	**g**:6

As you can probably imagine this is much more difficult to brute force as the key is constantly shifting.
So let's take our Caeser cypher encrypting solution to start:

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = 6
translated = ""
for letter in message:
        if letter in alphabet:
                number = alphabet.find(letter)
                number = number + key
                if number > (len(alphabet) - 1):
                     number = number - len(alphabet)
                translated = translated + alphabet[number]
        else:
                translated = translated + letter
print("Your code is " + translated)
```

Here's the steps you need to follow to turn your Caeser encrypter into a Vignere encrypter:

- Set the key to "pizza"
- Create a **variable** called "key_index" and set to 0 before the **for loop**. This is what letter position in the key we are - so we can loop the key: p i z z a p i z etc.
    - If key_index = 2 and key = "pizza"
    - Then key[key_index] = key[2] = z (remember computers count from 0!)
- Create a **variable** called "shift". This is the numerical value of the letter in the key.
    - So if the current letter in the key is p, shift = 15.
    - Use alphabet.find to change the letter to the number value.
- Add "shift" to "number" instead of adding "key" to "number"
- Then add one to key_index
- if key_index is equal the length of the key, then set it to 0. (how we're looping round pizzapizzapiz..)

The solution on the next page, try not to look. You can try testing your code with "hello good day" with key "pizza" should translate to "wmkko vwnc dpg"

Solution:

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = "pizza"
translated = ""
key_index = 0
for letter in message:
  if letter in alphabet:
    number = alphabet.find(letter)
    shift =  alphabet.find(key[key_index])
    number = number + shift
    key_index = key_index + 1
    if key_index == len(key):
      key_index = 0
    if number > (len(alphabet) - 1):
      number = number - len(alphabet)
    translated = translated + alphabet[number]
  else:
    translated = translated + letter
print("Your code is " + translated)
```

3.2 Decrypting Vignere

It's exactly the same as decrypting Caeser! As we've already created the code for the shifting key, we just need to modify our encrypter to decrypt with the following three steps:

- subtract the shift from the number rather than add it.
- Change the **if** number bigger than alphabet to **if** number less than 0
- And add the alphabet to number rather than subtract it
  (This is to fix the wrap around of the alphabet in reverse)

The code solution is on the next page.
Test this with "wmkko vwnc dpg" and key "pizza" you should get back "hello good day"
How about "b vlxkvs cj msfe qzzl wscvy" with key "burger"

Try sending secret codes with Vignere to other people in the workshop - give them your key and encrypted message, see if they can decrypt it!

You've probably realised that creating a way to brute force a Vignere cypher is much more difficult than a Caeser one as the key isn't just a fixed number between 0-25. This is particularly the case if your Vignere key is a long word and not even one in the dictionary… so it's harder to predict!

If you're interested in trying to break more complicated cyphers, you should have a look at frequency analysis, there's a great chapter on it in the Cracking Codes With Python book: inventwithpython.com/cracking/chapter19.html

There's plenty of advanced python code breaking in that book as well as many other cypher examples in the Tutorials Point's Cryptography With Python: tutorialspoint.com/cryptography_with_python

Solution

```
message = raw_input("What is your message? ")
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = "pizza"
translated = ""
key_index = 0
for letter in message:
  if letter in alphabet:
    number = alphabet.find(letter)
    shift = alphabet.find(key[key_index])
    number = number - shift
    key_index = key_index + 1
    if key_index == len(key):
      key_index = 0
    if number < 0:
      number = number + len(alphabet)
    translated = translated + alphabet[number]
  else:
    translated = translated + letter
print("Your code is " + translated)
```