

PROGRAMMING FOR CRYPTOLOGISTS

Programming for Cryptologists

Part 1

October 2026

Introduction

It is a reasonable assumption, given you're reading this, that you are familiar with the National Cipher Challenge.

The National Cipher Challenge's resources section already contains some great guides to breaking a variety of classical ciphers by hand.

This series of guides focuses on using code to break these ciphers instead.

Part 1 starts with the Caesar cipher and letter frequencies. Future parts will build up towards progressively more complex ciphers.

Goals

By the end of this guide, you should:

- Understand how a Caesar cipher works
- Have a Python script to encrypt and decrypt a Caesar cipher
- Understand how chi squared can be used to compare letter frequencies
- Have a simple score function to score how close to English a decrypt is
- Have put together your own Python script to automatically break an unknown Caesar cipher

Prerequisites

For this guide you will need some basic knowledge of Python. There are many amazing existing guides to help you learn this.

You should also have Python installed (or run it online) if you want to follow along.

Caesar - How does it work?

The Caesar cipher is one of the simplest classical ciphers. It encrypts every letter using a simple rule: shift each letter K places to the right.

What does this mean? If K (the key) is 3, encrypting A (the 1st letter of the alphabet) shifts it to a D (4th letter of the alphabet), and Z (26th) wraps around back to C (3rd).

This is then repeated for each letter.

Here is an example:

```
key: 4
plaintext:  ATTACK AT ONCE
ciphertext: EXXEGO EX SRGI
```

Caesar in code

There are a few things that we need Python to do if we want it to decrypt a Caesar:

- Get the position of a letter in the alphabet, A = 0, Z = 25 (programmers like counting from 0)
- Get the nth letter of the alphabet 0 = A, 25 = Z
- Adding a shift to each letter
- Wrapping around if we reach the end of the alphabet
- Repeating for every letter in the text

Step 1 - nth letters

Before you continue reading, try to find a way to get a letter's position in the alphabet and get the nth letter in Python.

One way to do it:

Python has two useful functions for working with characters:

`ord()`

`chr()`

Try:

```
print(ord("A"))
print(ord("B"))
print(ord("Z"))
```

What do you notice?

How can we make this a value between 0 and 25?

Then try:

```
print(chr(65))
print(chr(66))
print(chr(90))
```

Again, what do you notice?

How can we make this take a value between 0 and 25?

Putting it all together:

```
letter = "C"
# convert to a number 0-25
n = ord(letter) - ord("A")
print(n)
```

```
n = 5
# convert back to a letter
letter = chr(n + ord("A"))
print(letter)
```

Step 2 - adding a shift

Now that we have the letter as a number, this is a lot easier.

Suppose we have the letter D. We can convert it to a number $D = 3$. If the key is 5, we can add a shift of 5, $3 + 5 = 8$. We can then convert it back to a letter $8 = I$.

Try to write the Python to do this.

What happens when you try to shift Z? That's a problem.

Step 3 - wrapping around

$Z + 3$ goes off the end of the alphabet. How can we fix this?

The easiest way to do this is using Python's modulo operator. This gives us the remainder after division.

Try:

```
print(25 % 26)
print(26 % 26)
print(27 % 26)
print(28 % 26)
print(53 % 26)
```

You can probably already see how this can be useful.

All we need to do is to use modulo 26.

```
n = (n + key) % 26
```

Step 4 - repeat

We can now encrypt a single letter. We do not want to have to run our code for every single letter, we want Python to repeat it for every single letter in our text. If you are familiar with Python you will already know the answer: for loops.

If we have a for loop to loop over every letter in the text, we can place our existing code inside and it will run on each letter. The last bit we need is to remember all the encrypted letters at the end.

This is as simple as adding them to a string which we can use later.

Putting it all together:

```
plaintext = "ATTACK AT ONCE"
ciphertext = ""
key = 4
```

```

for letter in plaintext:
    # if it is not a letter it stays the same
    if letter not in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        ciphertext += letter

    else:
        # convert to number
        n = ord(letter) - ord("A")

        # add shift (and wrap)
        n = (n + key) % 26

        # convert back to a letter
        letter = chr(n + ord("A"))

        # Add to ciphertext
        ciphertext += letter

# print the result
print(ciphertext)

```

Step 5 - decrypt

Decrypting is the opposite, we subtract the shift instead of adding it.

```

ciphertext = "EXXEGO EX SRGI"
plaintext = ""
key = 4

for letter in ciphertext:
    # if it is not a letter it stays the same
    if letter not in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        plaintext += letter

    else:
        # convert to number
        n = ord(letter) - ord("A")

        # subtract shift (and wrap)
        n = (n - key) % 26

        # convert back to a letter
        letter = chr(n + ord("A"))

        # Add to plaintext
        plaintext += letter

# print the result
print(plaintext)

```

What now? - Breaking the Caesar

By now you should have a Python script to both encrypt and decrypt a Caesar cipher. However, usually in National Cipher Challenge, you do not have the key. So we need some way of breaking it. Caesar is a great example, there are only 26 possible keys to a Caesar cipher (a shift of 0 to 25). That is not too many, we can try all of them and see which looks the best. Yes, that means another for loop.

In Python, to try every number within a certain range, you can use 'for num in range(0, end):', end is not inclusive.

```
ciphertext = "EXXEGO EX SRGI"

# For every key
for key in range(0, 26):

    plaintext = ""

    for letter in ciphertext:
        # if it is not a letter it stays the same
        if letter not in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
            plaintext += letter

        else:
            # convert to number
            n = ord(letter) - ord("A")

            # subtract shift (and wrap)
            n = (n - key) % 26

            # convert back to a letter
            letter = chr(n + ord("A"))

            # Add to plaintext
            plaintext += letter

    # print the result
    print("Key:", key)
    print(plaintext)
```

You can now look over the outputs and find which looks like plaintext. Try changing the ciphertext to something from National Cipher Challenge. Congrats, you just broke a Caesar cipher.

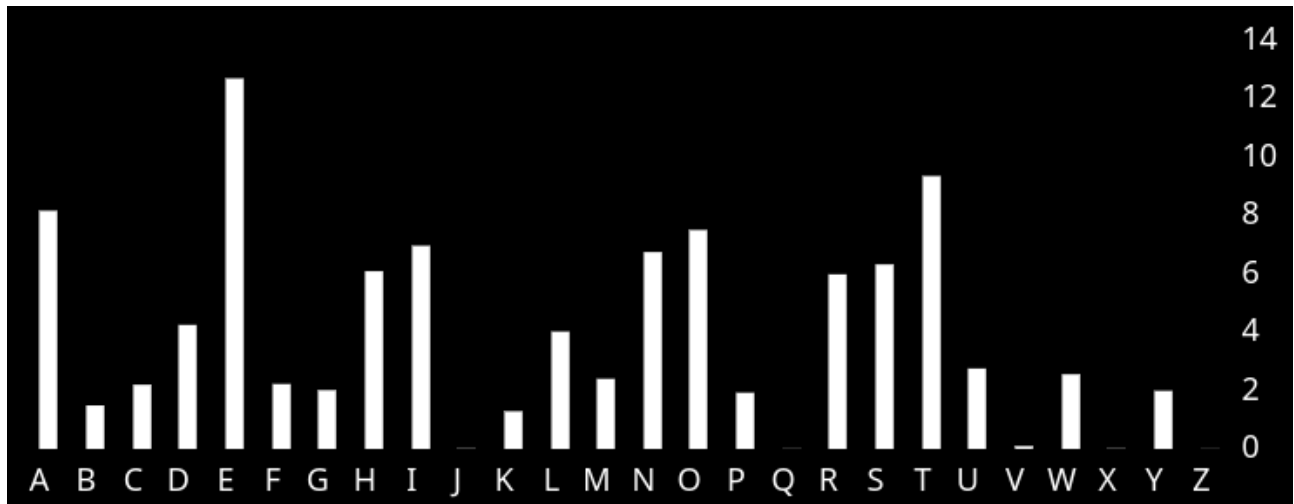
It is still quite a bit of work to look over ALL 26 decrypts. It would be nice if we can ask Python to do that for us.

How to recognise plaintext?

At the moment, our script has no way of knowing which decrypt is correct, it just blindly prints all of them. We need a way to measure how English-like each decrypt is.

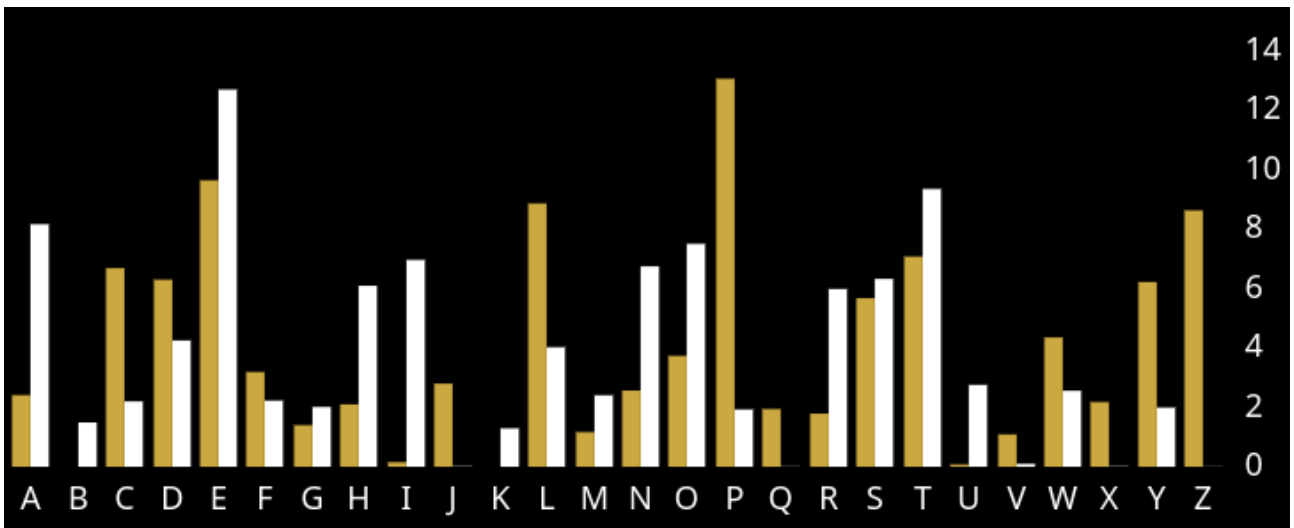
There are several ways to judge how 'good' a decrypt is, each has its advantages and disadvantages. The two that I will bring up are letter frequencies and tetragrams. Tetragrams will have to wait until part 2, so letter frequencies it is.

In English, some letters appear more frequently than others. Courtesy of National Cipher Challenge, here is a nice graph of this.



E and T appear more frequently while X and Z barely appear.

Compare this to text encrypted with a Caesar cipher.



You can see that the bars do not align, some are taller than they should be while others shorter.

It would be nice if we could measure how well it matches the expected frequencies. Hint, there is!

Chi squared

There is a lot more useful maths behind the chi squared statistic. However there are two relevant bits: what it represents and how it can be calculated.

In short, it gives us a measure of how far apart the observed data is from the expected data.

To calculate it, for every data-point (every letter in the alphabet), you first calculate the difference between expected frequency of that letter and what you observed. This might be a negative number, however we only care about how far it is, so we need to turn it into a positive number. One way to do this is to square it, this also has the advantage that larger differences get much larger punishments. Additionally, we can then divide it by the expected frequency to scale it down. Then we just repeat this for all the data points and sum it together.

In essence:

For every data-point:

difference = expected - observed

squared = difference ** 2

score += squared / expected

So now we just need Python to calculate the letter frequencies, run this test on each and pick the lowest score (the closest to correct frequencies).

Letter frequencies in code

Try to have a go at this yourself first. The goal is to calculate how frequently each letter appears in the text, A good start is to count how frequently each letter appears.

Here are some bits that will help make this easier:

Functions: these allow you to write a bit of code and then run it by calling it from elsewhere in the code. You can pass data into it and get data returned from it. I suggest putting all this code into a function called score, it will be useful later.

```
# Define a function
def myFunction(x, y):
    z = x + y
    # Return any values you want to use later
    return z

# Calling the function
a = myFunction(2, 3)
print(a)
b = myFunction(a, 3)
print(b)
```

Dictionaries: you do not need to know how these work but one advantage is that they let us store values like variables, but use another value (a key) to access them. See if the code below can explain it.

```
# Create a dictionary called myDict
myDict = {}

# Set all letters in the dictionary to 0
for l in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
    myDict[l] = 0

print(myDict)

# Access a value (make sure that it exists otherwise you get an error)
print(myDict["A"])
print(myDict["Z"])

# Change a value
for l in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
    myDict[l] = myDict[l] + 1

print(myDict)
```

That should be enough to get started.

Here is my example:

```
def score(text):
    # Create count dictionary and set all to 0
    counts = {}
    total = 0
    for l in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        counts[l] = 0

    for l in text:
        if l in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
            counts[l] += 1
            total += 1

    # Calculate frequencies
    frequencies = {}
    for l in counts:
        frequencies[l] = counts[l] / total

    print(frequencies)

    # We will use this later, currently a placeholder
    return -1

text = "TEST"
print(score(text))
```

Scoring plaintext

Now that you have code to generate letter frequencies for any text, you can combine it with chi squared from earlier to get a complete score function.

For a bit of help, you can use these as expected frequencies:

```
expected = {
    'E': 0.120,
    'T': 0.091,
    'A': 0.081,
    'O': 0.077,
    'I': 0.073,
    'N': 0.070,
    'S': 0.063,
    'R': 0.060,
    'H': 0.059,
    'D': 0.043,
    'L': 0.040,
    'U': 0.029,
    'C': 0.027,
    'M': 0.026,
    'F': 0.023,
    'Y': 0.021,
    'W': 0.021,
    'G': 0.020,
    'P': 0.018,
    'B': 0.015,
    'V': 0.011,
    'K': 0.007,
    'X': 0.002,
    'Q': 0.001,
    'J': 0.001,
    'Z': 0.001,
}
```

And here is my example:

```
def score(text):
    # Create count dictionary and set all to 0
    counts = {}
    total = 0
    for l in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        counts[l] = 0

    for l in text:
        if l in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
            counts[l] += 1
            total += 1

    # Calculate frequencies
    frequencies = {}
```

```

for l in counts:
    frequencies[l] = counts[l] / total

expected = {
    'E': 0.120,
    'T': 0.091,
    'A': 0.081,
    'O': 0.077,
    'I': 0.073,
    'N': 0.070,
    'S': 0.063,
    'R': 0.060,
    'H': 0.059,
    'D': 0.043,
    'L': 0.040,
    'U': 0.029,
    'C': 0.027,
    'M': 0.026,
    'F': 0.023,
    'Y': 0.021,
    'W': 0.021,
    'G': 0.020,
    'P': 0.018,
    'B': 0.015,
    'V': 0.011,
    'K': 0.007,
    'X': 0.002,
    'Q': 0.001,
    'J': 0.001,
    'Z': 0.001,
}

score = 0

for letter in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
    difference = frequencies[letter] - expected[letter]
    squared = difference ** 2
    score += squared / expected[letter]

return score

text_1 = "PUT SOME CIPHERTEXT HERE"
print(score(text_1))

text_2 = "PUT SOME PLAINTEXT HERE"
print(score(text_2))

```

Now, try calling `score` for something that you know is not plaintext and try it on plaintext, you should notice a difference. Crucially, this is a difference that the computer can easily notice, just tell Python to pick the key with the lowest score.

Putting it all together

We have everything we need to break a Caesar cipher automatically, now to put it all together. Try every possible key, then score it. Keep track of the lowest score and which key gave it. Then, once every key has been tested, you should be able to output the best key and the best plaintext.

My example:

```
ciphertext = "EXXEGO EX SRGI"

def score(text):
    # Create count dictionary and set all to 0
    counts = {}
    total = 0
    for l in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        counts[l] = 0

    for l in text:
        if l in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
            counts[l] += 1
            total += 1

    # Calculate frequencies
    frequencies = {}
    for l in counts:
        frequencies[l] = counts[l] / total

    expected = {
        'E': 0.120,
        'T': 0.091,
        'A': 0.081,
        'O': 0.077,
        'I': 0.073,
        'N': 0.070,
        'S': 0.063,
        'R': 0.060,
        'H': 0.059,
        'D': 0.043,
        'L': 0.040,
        'U': 0.029,
        'C': 0.027,
        'M': 0.026,
        'F': 0.023,
        'Y': 0.021,
        'W': 0.021,
        'G': 0.020,
        'P': 0.018,
        'B': 0.015,
        'V': 0.011,
        'K': 0.007,
        'X': 0.002,
        'Q': 0.001,
        'J': 0.001,
```

```

        'Z': 0.001,
    }

    score = 0

    for letter in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        difference = frequencies[letter] - expected[letter]
        squared = difference ** 2
        score += squared / expected[letter]

    return score

lowest_score = float('inf')
best_key = 0
best_plaintext = ""

# For every key
for key in range(0, 26):

    plaintext = ""

    for letter in ciphertext:
        # if it is not a letter it stays the same
        if letter not in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
            plaintext += letter

        else:
            # convert to number
            n = ord(letter) - ord("A")

            # subtract shift (and wrap)
            n = (n - key) % 26

            # convert back to a letter
            letter = chr(n + ord("A"))

            # Add to plaintext
            plaintext += letter

    # score the plaintext
    s = score(plaintext)

    # if it is a better score, save it
    if s < lowest_score:
        lowest_score = s
        best_key = key
        best_plaintext = plaintext

print("Best key:", best_key)
print(best_plaintext)

```

Congratulations

Congratulations, you have just put together a Python script that can automatically break Caesar ciphers. This is just the beginning, over time your code will be able to break more and more ciphers.

See you in part 2!

Extensions:

Research the Affine cipher. From there, try to change a copy of your Caesar code to be able to break an Affine cipher.

Get ahead before part 2 and do some research on tetragram scoring.