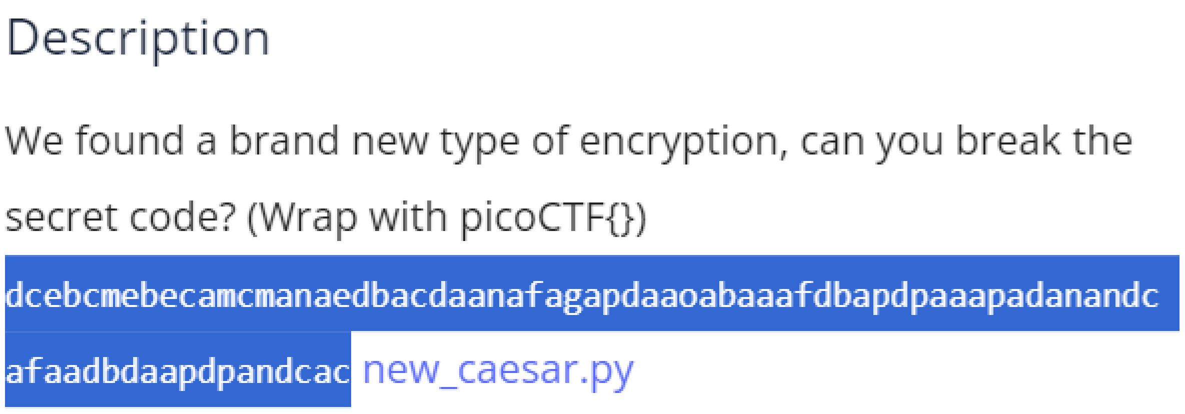
New Caesar

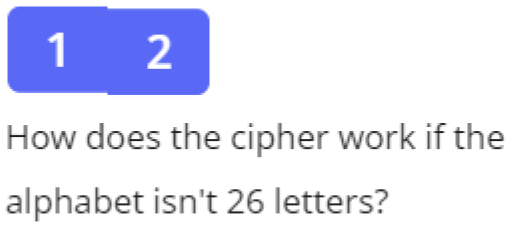

Description

We found a brand new type of encryption, can you break the secret code? (Wrap with picoCTF{})

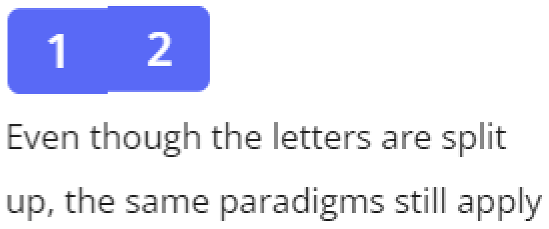
dcebcmebecamcmanaedbacdaanafagapdaaoabaaafdbapdpaaapadanandcafaadbdaapdpandcac new_caesar.py

Toughest one I encountered so far, lotso brainpower and thugging went into this.

I reached the hints pretty quickly on this one, but that was of no use. Hints were pretty useless.

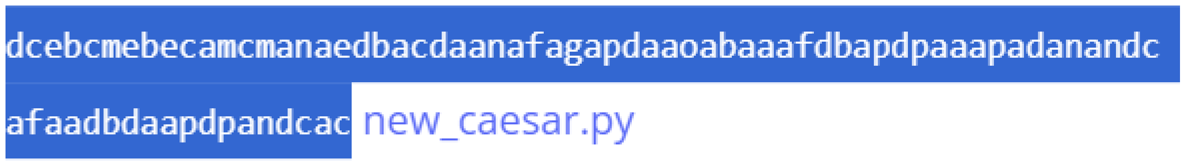


How does the cipher work if the alphabet isn't 26 letters?



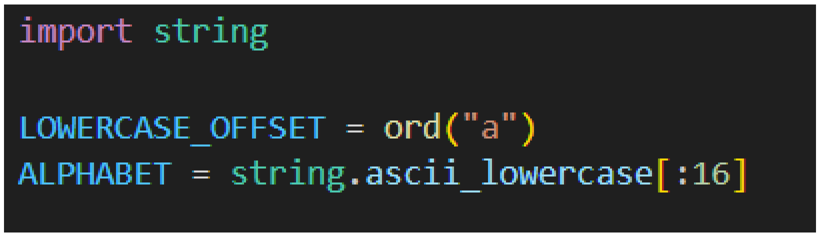
Even though the letters are split up, the same paradigms still apply

After a while, I started to properly make sense of what was happening in the code.
So I assumed the code I was looking at was what was used to encrypt the flag to give out whatever bullcrap this is :


dcebcmebecamcmanaedbacdaanafagapdaaoabaaafdbapdpaaapadanandcafaadbdaapdpandcac new_caesar.py

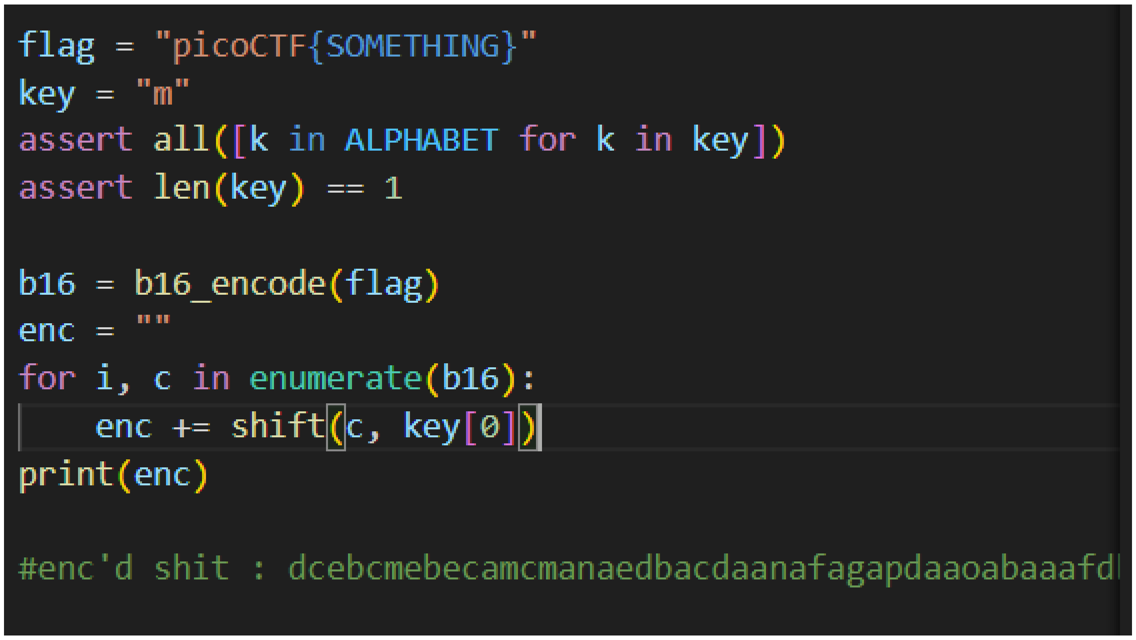
And I needed to reverse engineer this encryption algorithm to find the actual flag.

Looking through the code :



```python
import string

LOWERCASE_OFFSET = ord("a")
ALPHABET = string.ascii_lowercase[:16]
```

Some basic universal stuff for the code was being declared.
The ascii value of ‘a’ was being taken. (97)
And a list of the first 16 alphabets was being taken. (abcdefghijklmnop)



```python
flag = "picoCTF{SOMETHING}"
key = "m"
assert all([k in ALPHABET for k in key])
assert len(key) == 1

b16 = b16_encode(flag)
enc = ""
for i, c in enumerate(b16):
    enc += shift(c, key[0])
print(enc)

#enc'd shit : dcebcmebecamcmanaedbacdaanafagapdaaoabaaafdl
```

Our flag was then being weird-caeser shifted using some key
There were 2 conditions on the key :

1.It’s a letter in the list ALPHABET i.e. it’s a letter between a and p

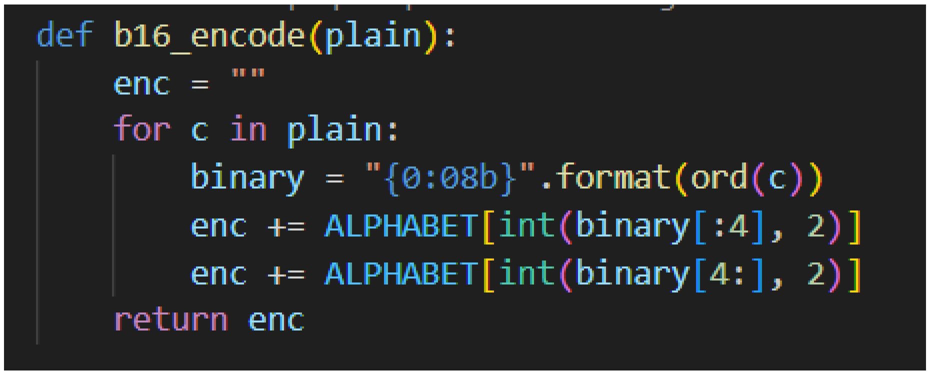
2.the length of the key is 1

(So the key is somewhere b/w a-p)
I supposed that meant I might have to attempt a trial and error decoding at all of these letters

(i later found out this trial and error method of cracking is called *brute forcing*, which seemed like an incredibly common terminology amongst hacky people as I found out from several yt videos.)

now , our flag was being encoded using this function, base16
And this base16 encoded stuff was weird-caesar shifted with the key.

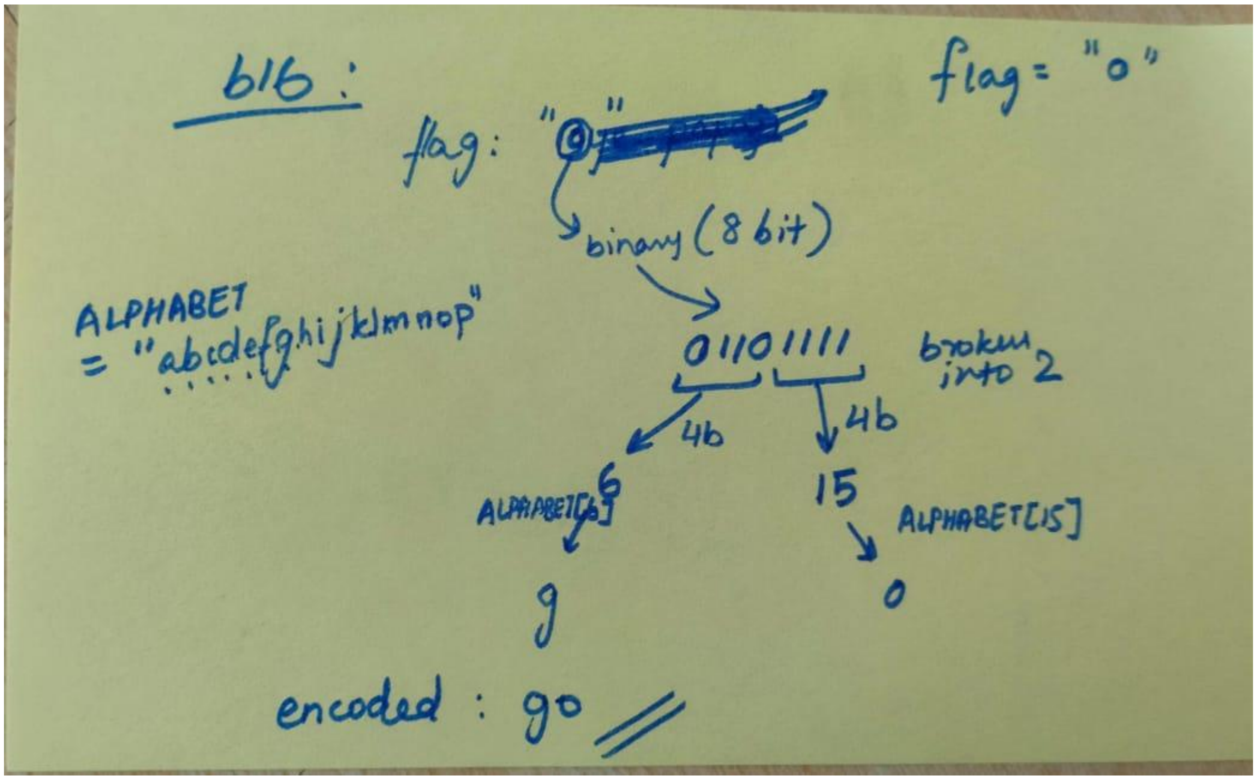
So I decided to take a look at how and what base16 was doing its thing on the flag:



```python
def b16_encode(plain):
    enc = ""
    for c in plain:
        binary = "{0:08b}".format(ord(c))
        enc += ALPHABET[int(binary[:4], 2)]
        enc += ALPHABET[int(binary[4:], 2)]
    return enc
```

So, this guy was taking each letter in the flag,
converting it to ascii, then into 8 bit binary
This 8 bit binary was broken down into 2 halves, first 4 bits and second 4 bits,
and these 4 bit binaries corresponded to some number
which was taken into index of ALPHABET and that element was added to enc

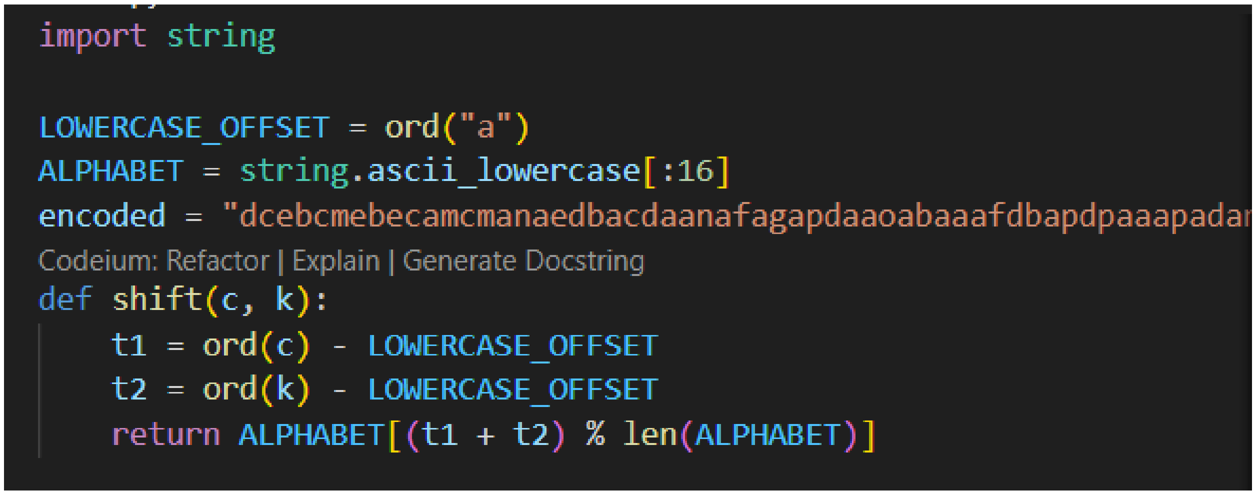
Sumthing like this ig -



this was what my poor braincells managed to gather. Can't explain it any better than this.

So the key to decodingding is to reverse engineer this function, the shift would cancel out itself (caesar shift apparently does that, some guy on yt said this, i didn't ss)

So to decode, I started writing a new python script :



```
import string

LOWERCASE_OFFSET = ord("a")
ALPHABET = string.ascii_lowercase[:16]
encoded = "dcebcmebecamcmanaedbacdaanafagapdaaoabaaafdbapdpaaapadar
Codeium: Refactor | Explain | Generate Docstring
def shift(c, k):
    t1 = ord(c) - LOWERCASE_OFFSET
    t2 = ord(k) - LOWERCASE_OFFSET
    return ALPHABET[(t1 + t2) % len(ALPHABET)]
```

Put the universally needed shit in.
Also put in the shift function exactly the same without tweaks.
So to get my flag, I needed to basically :

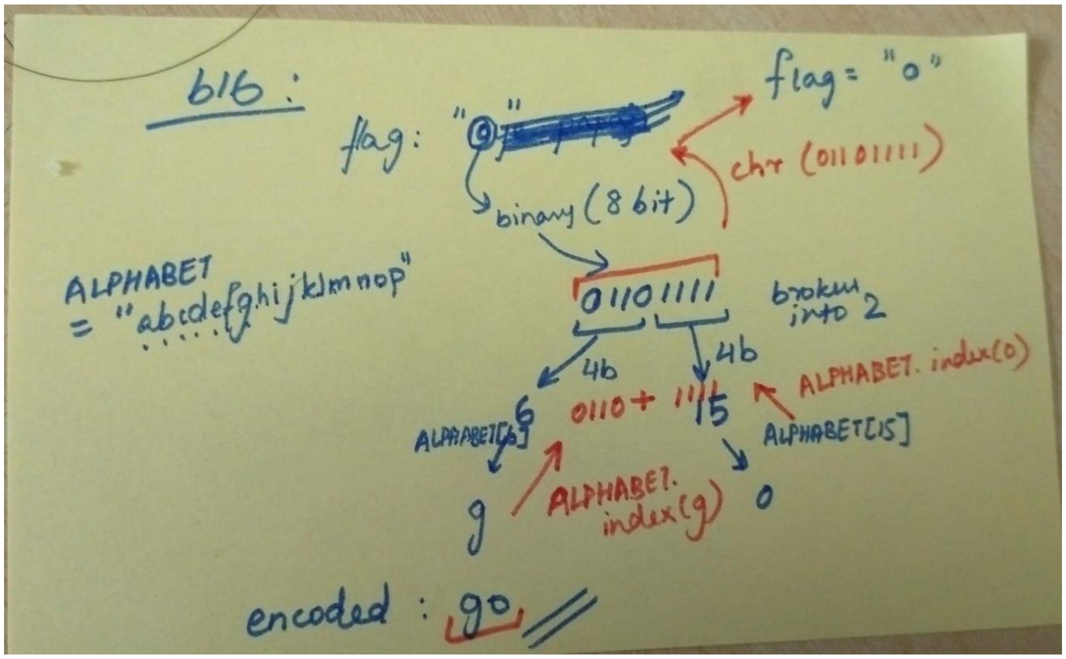
1. caeser shift the encoded thing first
2. Reverse base16 the thing

…with each letter from a-p

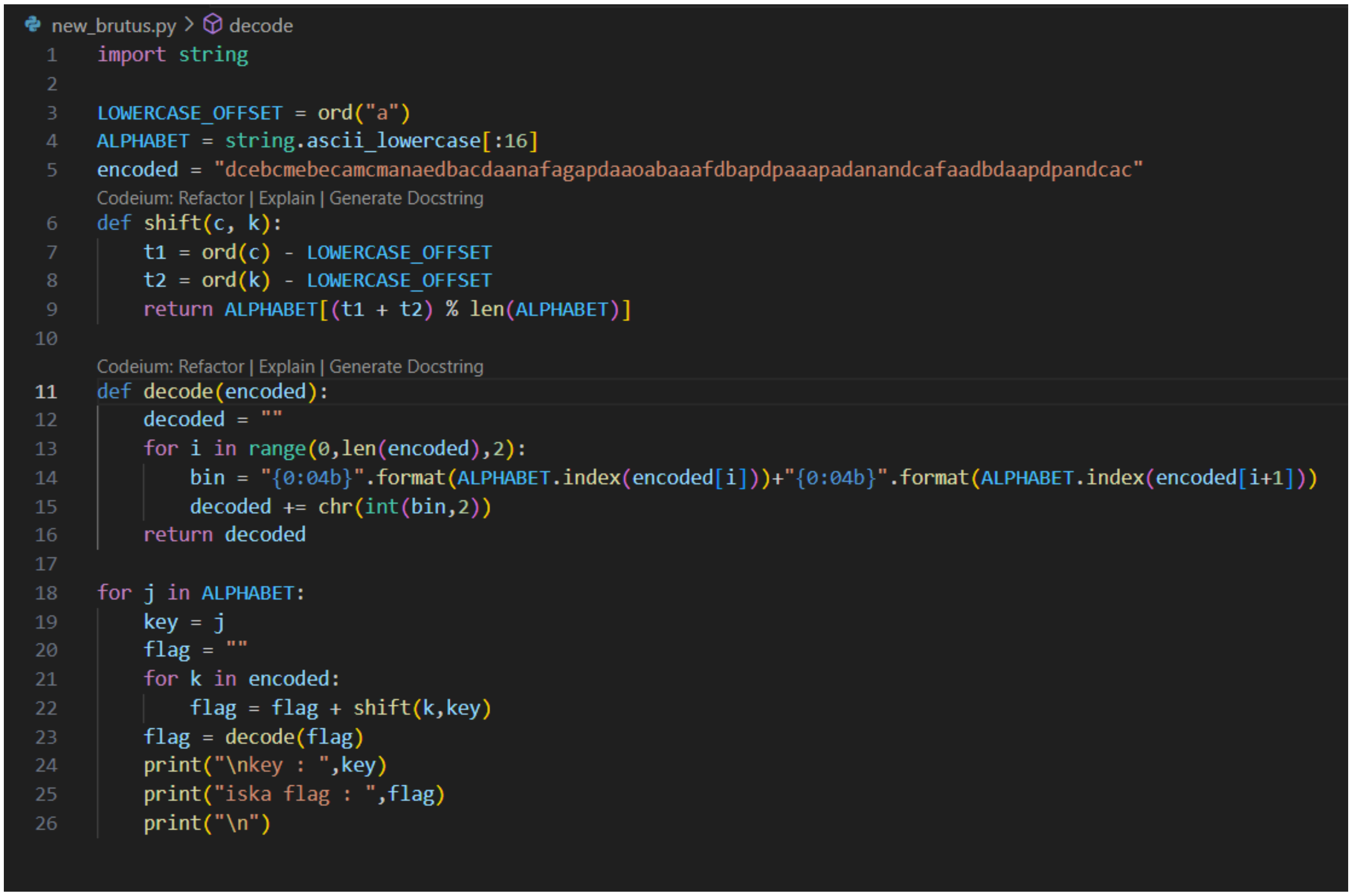
Now to reverse my base16 boi :
My approach to decoding was this :

- I would take every two elements in the encoded string
  (as it was splitting each element of the original plaintext into two 4-bit parts)
- Get the index of these elements in the ALPHABET list, and convert this index to 4 bit
  And add the 4 bit parts.

My literal thought process tryna figure this out :



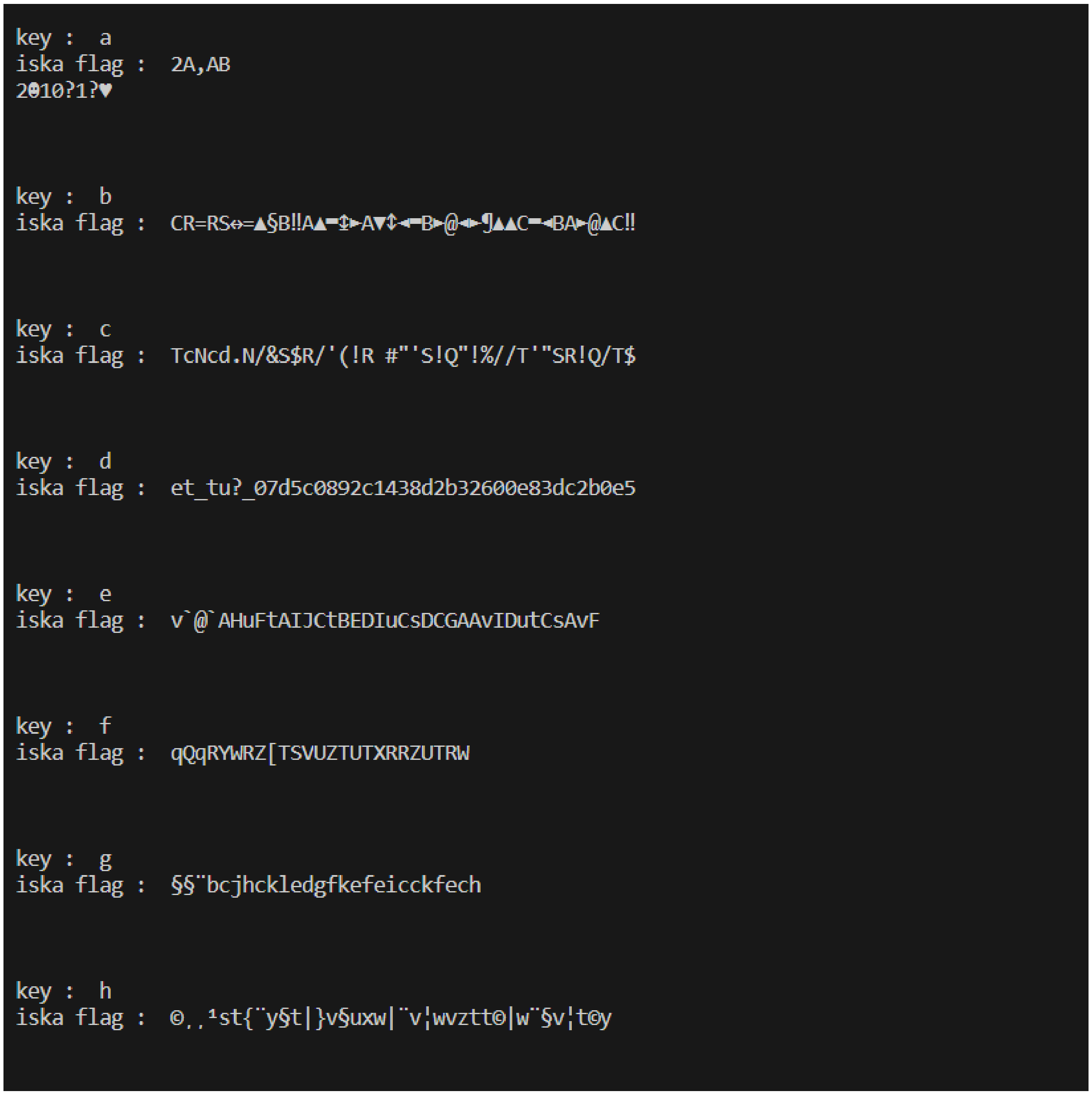
After 2 days of debugging and learning how to work with binary on python I created this :



new_brutus.py > decode

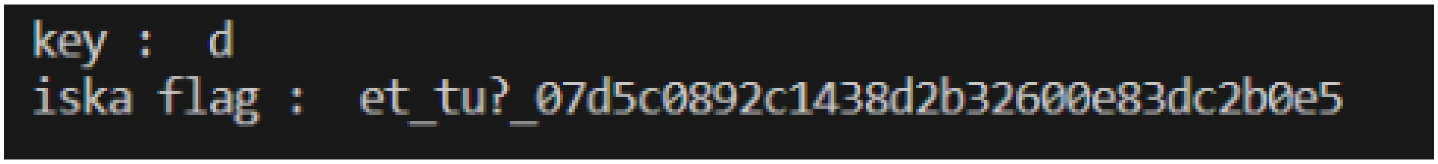
```python
import string

LOWERCASE_OFFSET = ord("a")
ALPHABET = string.ascii_lowercase[:16]
encoded = "dcebcmebecamcmanaedbacdaanafagapdaaoabaaafdbapdpaaapadanandcafaadbdaapdpandcac"
Codeium: Refactor | Explain | Generate Docstring
def shift(c, k):
    t1 = ord(c) - LOWERCASE_OFFSET
    t2 = ord(k) - LOWERCASE_OFFSET
    return ALPHABET[(t1 + t2) % len(ALPHABET)]

Codeium: Refactor | Explain | Generate Docstring
def decode(encoded):
    decoded = ""
    for i in range(0,len(encoded),2):
        bin = "{0:04b}".format(ALPHABET.index(encoded[i]))+"{0:04b}".format(ALPHABET.index(encoded[i+1]))
        decoded += chr(int(bin,2))
    return decoded

for j in ALPHABET:
    key = j
    flag = ""
    for k in encoded:
        flag = flag + shift(k,key)
    flag = decode(flag)
    print("\nkey : ",key)
    print("iska flag : ",flag)
    print("\n")
```

I ran this poo and :



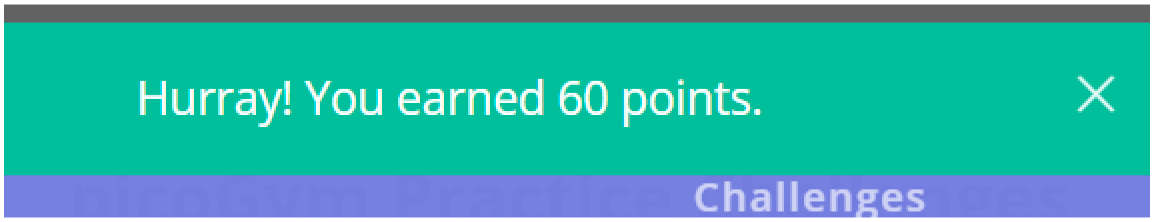
```
key :  a
iska flag :  2A,AB
2☻10?1?♥



key :  b
iska flag :  CR=RS↔=▲§B‼A▲▬↨►A▼↕◄▬B►@◄►¶▲▲C▬◄BA►@▲C‼



key :  c
iska flag :  TcNcd.N/&S$R/'(!R #"'S!Q"!%//T'"SR!Q/T$



key :  d
iska flag :  et_tu?_07d5c0892c1438d2b32600e83dc2b0e5



key :  e
iska flag :  v`@`AHuFtAIJCtBEDIuCsDCGAAvIDutCsAvF



key :  f
iska flag :  qQqRYWRZ[TSVUZTUTXRRZUTRW



key :  g
iska flag :  §§¨bcjhckledgfkefeicckfech



key :  h
iska flag :  ©¸¸¹st{¨y§t|}v§uxw|¨v¦wvztt©|w¨§v¦t©y
```

I started trying each of these into pico one by one and of course:



```
key :  d
iska flag :  et_tu?_07d5c0892c1438d2b32600e83dc2b0e5
```

witty.



Hurray! You earned 60 points.