GENERAL

ENCODING

Challenge : ASCII

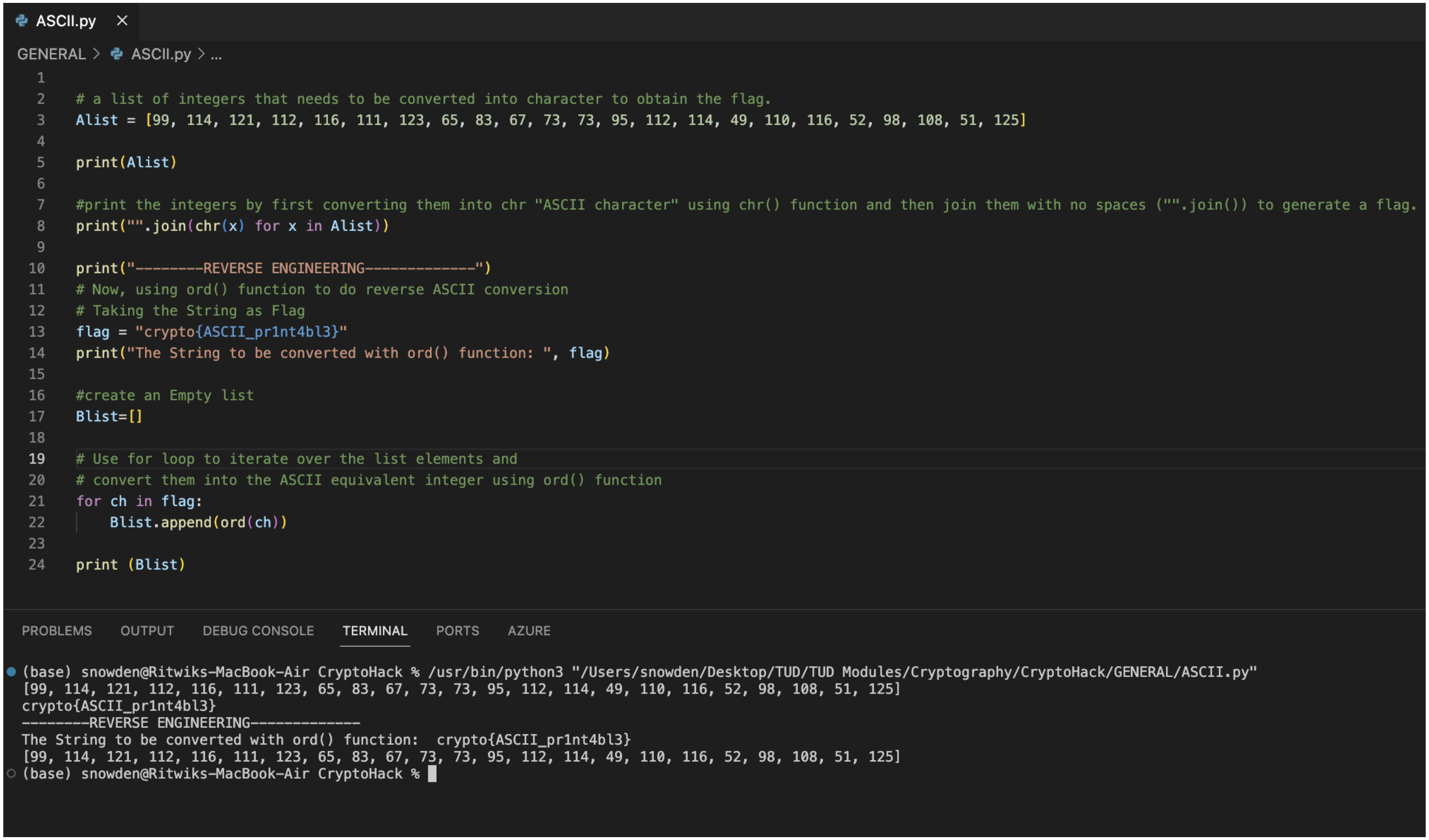
ASCII is a 7-bit encoding standard which allows the representation of text using the integers 0-127.

Using the below integer array, convert the numbers to their corresponding ASCII characters to obtain a flag.

```
[99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49, 110, 116, 52, 98
```

```
# a list of integers that needs to be converted into character to obtain the flag.
Alist = [99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49, 110, 116

#print the integers by first converting them into chr "ASCII character" using chr() funct
print("".join(chr(x) for x in Alist))

# Now, using ord() function to do reverse ASCII conversion
# Taking the String as Flag
flag = "crypto{ASCII_pr1nt4bl3}"
print("The String to be converted with ord() function: ", flag)

#create an Empty list
```

```
Blist=[]

# Use for loop to iterate over the list elements and
# convert them into the ASCII equivalent integer using ord() function
for ch in flag:
    Blist.append(ord(ch))

print (Blist)
```



```python

# a list of integers that needs to be converted into character to obtain the flag.
Alist = [99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49, 110, 116, 52, 98, 108, 51, 125]

print(Alist)

#print the integers by first converting them into chr "ASCII character" using chr() function and then join them with no spaces ("".join()) to generate a flag.
print("".join(chr(x) for x in Alist))

print("--------REVERSE ENGINEERING-------------")
# Now, using ord() function to do reverse ASCII conversion
# Taking the String as Flag
flag = "crypto{ASCII_pr1nt4bl3}"
print("The String to be converted with ord() function: ", flag)

#create an Empty list
Blist=[]

# Use for loop to iterate over the list elements and
# convert them into the ASCII equivalent integer using ord() function
for ch in flag:
    Blist.append(ord(ch))

print (Blist)
```

```
(base) snowden@Ritwiks-MacBook-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/ASCII.py"
[99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49, 110, 116, 52, 98, 108, 51, 125]
crypto{ASCII_pr1nt4bl3}
--------REVERSE ENGINEERING-------------
The String to be converted with ord() function:  crypto{ASCII_pr1nt4bl3}
[99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114, 49, 110, 116, 52, 98, 108, 51, 125]
(base) snowden@Ritwiks-MacBook-Air CryptoHack % 
```

FLAG: **crypto{ASCII_pr1nt4bl3}**

Challenge: Hex

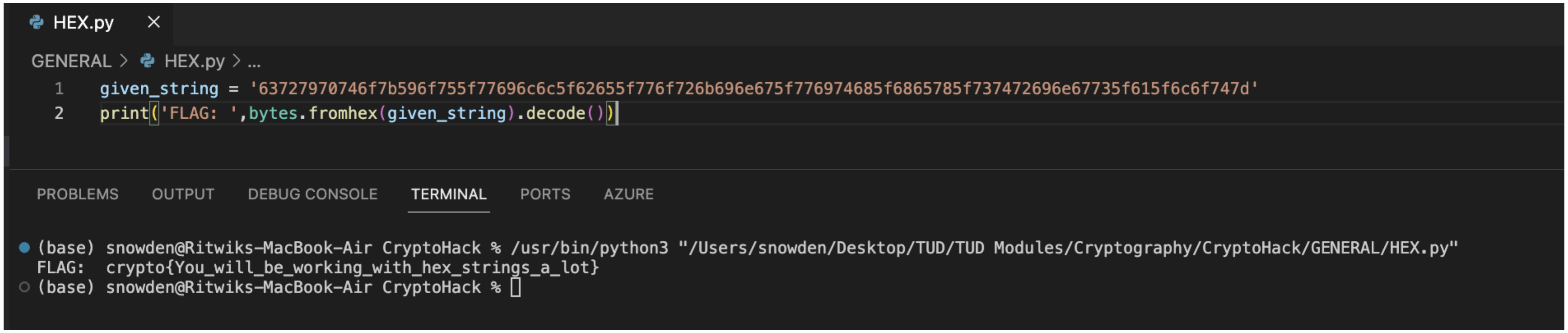
When we encrypt something the resulting ciphertext commonly has bytes which are not printable ASCII characters. If we want to share our encrypted data, it's common to encode it into something more user-friendly and portable across different systems.

Hexadecimal can be used in such a way to represent ASCII strings. First each letter is converted to an ordinal number according to the ASCII table (as in the previous challenge). Then the decimal numbers are converted to base-16 numbers, otherwise known as hexadecimal. The numbers can be combined together, into one long hex string.

Included below is a flag encoded as a hex string. Decode this back into bytes to get the flag.

```
63727970746f7b596f755f77696c6c5f62655f776f726b696e675f776974685f6865785f737472696e67735f
```

```
given_string = '63727970746f7b596f755f77696c6c5f62655f776f726b696e675f776974685f6865785f
print('FLAG: ',bytes.fromhex(given_string).decode())
```



```
given_string = '63727970746f7b596f755f77696c6c5f62655f776f726b696e675f776974685f6865785f737472696e67735f615f6c6f747d'
print('FLAG: ',bytes.fromhex(given_string).decode())
```

```
● (base) snowden@Ritwiks-MacBook-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/HEX.py"
  FLAG:  crypto{You_will_be_working_with_hex_strings_a_lot}
○ (base) snowden@Ritwiks-MacBook-Air CryptoHack %
```

FLAG: **crypto{You_will_be_working_with_hex_strings_a_lot}**

Challenge: Base64

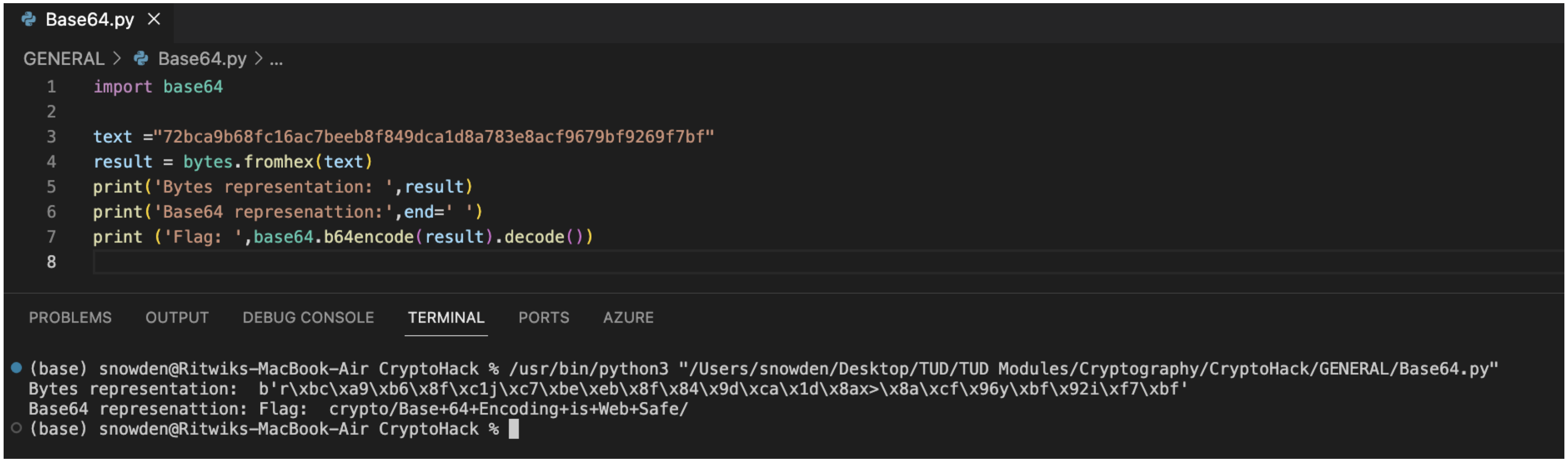
Another common encoding scheme is Base64, which allows us to represent binary data as an ASCII string using an alphabet of 64 characters. One character of a Base64 string encodes 6 binary digits (bits), and so 4 characters of Base64 encode three 8-bit bytes.

Base64 is most commonly used online, so binary data such as images can be easily included into HTML or CSS files.

Take the below hex string, *decode* it into bytes and then *encode* it into Base64.

```
72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf
```

```
import base64

text ="72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf"
result = bytes.fromhex(text)
print('Bytes representation: ',result)
print('Base64 represenattion:',end=' ')
print ('Flag: ',base64.b64encode(result).decode())
```



```python
import base64

text ="72bca9b68fc16ac7beeb8f849dca1d8a783e8acf9679bf9269f7bf"
result = bytes.fromhex(text)
print('Bytes representation: ',result)
print('Base64 represenattion:',end=' ')
print ('Flag: ',base64.b64encode(result).decode())
```

```
(base) snowden@Ritwiks-MacBook-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/Base64.py"
Bytes representation:  b'r\xbc\xa9\xb6\x8f\xc1j\xc7\xbe\xeb\x8f\x84\x9d\xca\x1d\x8ax>\x8a\xcf\x96y\xbf\x92i\xf7\xbf'
Base64 represenattion: Flag:  crypto/Base+64+Encoding+is+Web+Safe/
(base) snowden@Ritwiks-MacBook-Air CryptoHack % 
```

Flag: **crypto/Base+64+Encoding+is+Web+Safe/**

Challenge: Bytes and Big Integers

Cryptosystems like RSA works on numbers, but messages are made up of characters. How should we convert our messages into numbers so that mathematical operations can be applied?

The most common way is to take the ordinal bytes of the message, convert them into hexadecimal, and concatenate. This can be interpreted as a base-16/hexadecimal number, and also represented in base-10/decimal.

To illustrate:

message: HELLO

ascii bytes: [72, 69, 76, 76, 79]

hex bytes: [0x48, 0x45, 0x4c, 0x4c, 0x4f] base-16: 0x48454c4c4f

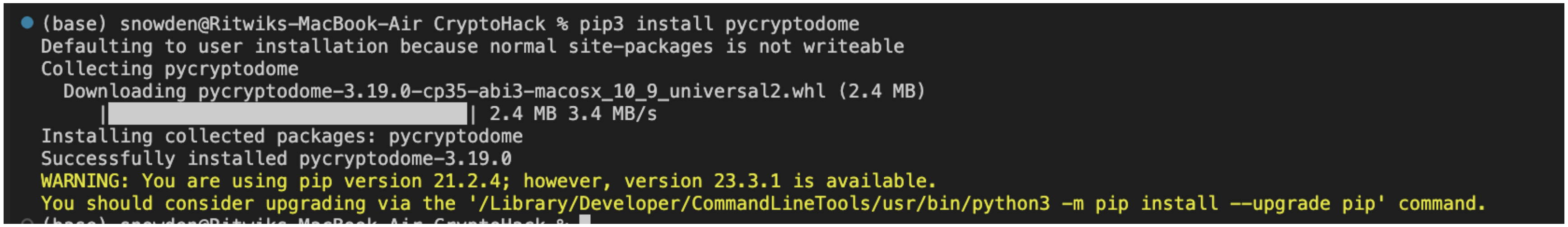
base-10: 310400273487

Convert the following integer back into a message:

```
11515195063862318899931685488813747395775516287289682636499965282714637259206269
```

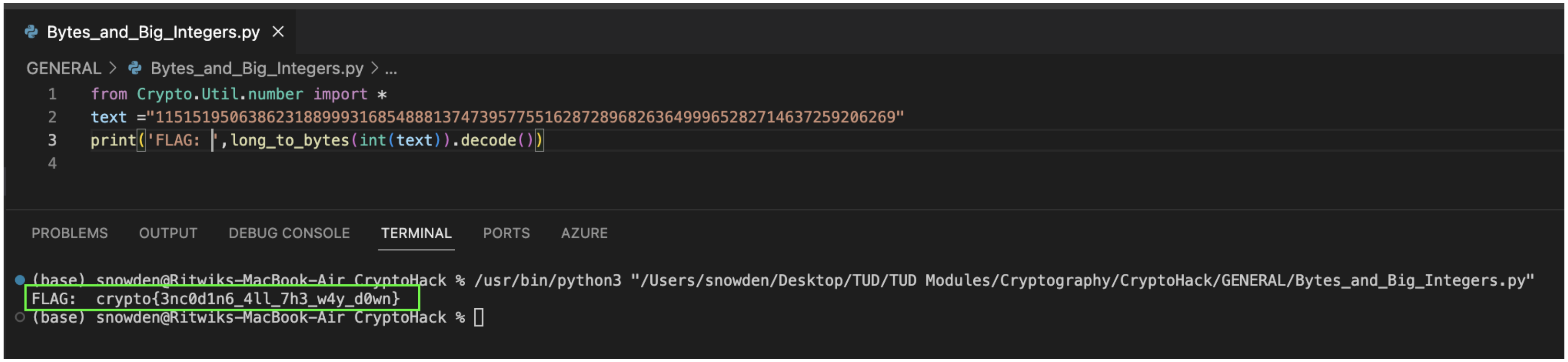
For this one we have to install a python library called PyCryptodome in order to use the `Crypto.Util.number` module.

Installation: `pip3 install pycryptodome`.



```
● (base) snowden@Ritwiks-MacBook-Air CryptoHack % pip3 install pycryptodome
  Defaulting to user installation because normal site-packages is not writeable
  Collecting pycryptodome
    Downloading pycryptodome-3.19.0-cp35-abi3-macosx_10_9_universal2.whl (2.4 MB)
       |████████████████████████████████| 2.4 MB 3.4 MB/s
  Installing collected packages: pycryptodome
  Successfully installed pycryptodome-3.19.0
  WARNING: You are using pip version 21.2.4; however, version 23.3.1 is available.
  You should consider upgrading via the '/Library/Developer/CommandLineTools/usr/bin/python3 -m pip install --upgrade pip' command.
○ (base) snowden@Ritwiks-MacBook-Air CryptoHack % █
```

```
from Crypto.Util.number import *
text ="11515195063862318899931685488813747395775516287289682636499965282714637259206269"
print('FLAG: ',long_to_bytes(int(text)).decode())
```



Bytes_and_Big_Integers.py
GENERAL > Bytes_and_Big_Integers.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
AZURE

```python
from Crypto.Util.number import *
text ="11515195063862318899931685488813747395775516287289682636499965282714637259206269"
print('FLAG: ',long_to_bytes(int(text)).decode())
```

```
(base) snowden@Ritwiks-MacBook-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/Bytes_and_Big_Integers.py"
FLAG:  crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}
(base) snowden@Ritwiks-MacBook-Air CryptoHack % 
```

FLAG: **crypto{3nc0d1n6_4ll_7h3_w4y_d0wn}**

XOR

Challenge: XOR Starter

XOR is a bitwise operator which returns 0 if the bits are the same, and 1 otherwise. In textbooks the XOR operator is denoted by ⊕, but in most challenges and programming languages you will see the caret `^` used instead.

For longer binary numbers we XOR bit by bit: 0110 ^ 1010 = 1100. We can XOR integers by first converting the integer from decimal to binary. We can XOR strings by first converting each character to the integer representing the Unicode character.

Given the string **label**, XOR each character with the integer **13**. Convert these integers back to a string and submit the flag as **crypto{new_string}.**

```
text='label'
result=""

for i in text:
    result += chr(ord(i)^13)

print('FLAG: ',f"crypto{{{result}}}")
```

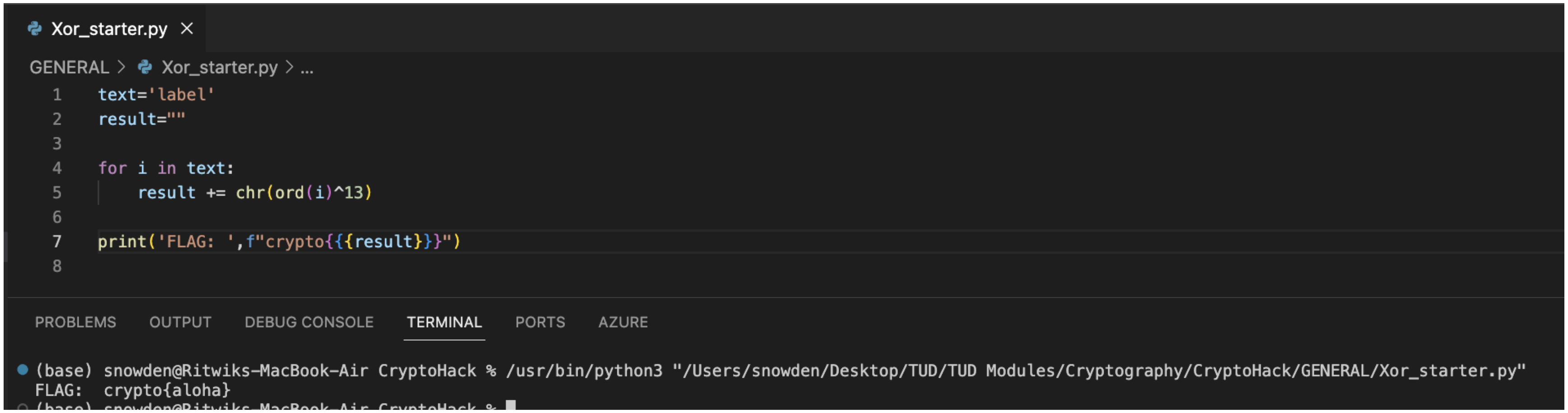
NOTE:

The code `print(f"crypto{{{result}}}")` is a Python f-string (formatted string literal) that incorporates the value of a variable named `result` into a string.

Let's break it down:

1. **Outer curly braces:**
    - The outer double curly braces (`{{` and `}}`) are used to include a literal pair of curly braces in the final string. This is necessary because a single pair of curly braces is the syntax for a placeholder in an f-string.
2. **Inner curly braces and `result`:**
    - Inside the outer curly braces, there is another set of double curly braces `{}` containing the variable `result`. This is the placeholder where the value of the `result` variable will be inserted.
3. **f-string:**

- The `f` at the beginning of the string indicates that this is an f-string. F-strings are a feature introduced in Python 3.6 that allows you to embed expressions inside string literals, using curly braces `{}`.



```python
text='label'
result=""

for i in text:
    result += chr(ord(i)^13)

print('FLAG: ',f"crypto{{{result}}}")
```

```
(base) snowden@Ritwiks-MacBook-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/Xor_starter.py"
FLAG:  crypto{aloha}
(base) snowden@Ritwiks-MacBook-Air CryptoHack %
```

FLAG: **crypto{aloha}**

Challenge: XOR Properties

Below is a series of outputs where three random keys have been XOR'd together and with the flag. Use the above properties to undo the encryption in the final line to **obtain the flag**.

KEY1 = a6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313

KEY2 ^ KEY1 = 37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e

KEY2 ^ KEY3 = c1545756687e7573db23aa1c3452a098b71a7fbf0fddddde5fc1

FLAG ^ KEY1 ^ KEY3 ^ KEY2 = 04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf

KEY2 = KEY1 ^ (KEY2 ^ KEY1)

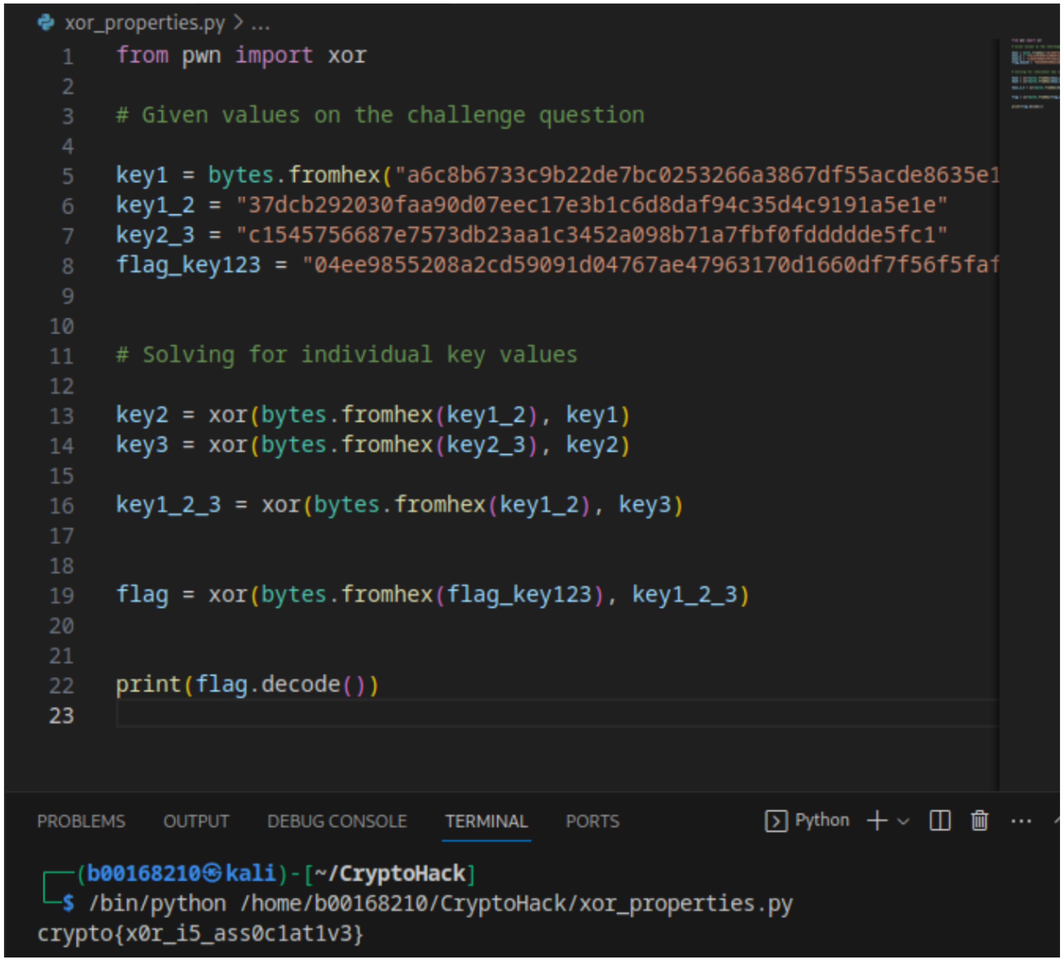
KEY3 = {KEY2} ^ (KEY2 ^ KEY3)

= {~~KEY1~~ ^ (~~KEY2~~ ^ ~~KEY1~~)} ^ (~~KEY2~~ ^ KEY3)

FLAG = KEY1 ^ KEY2 ^ KEY3 ^ (FLAG ^ KEY1 ^ KEY3 ^ KEY2)

```
from pwn import xor

# Given values on the challenge question

key1 = bytes.fromhex("a6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313")
key1_2 = "37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e"
key2_3 = "c1545756687e7573db23aa1c3452a098b71a7fbf0fddddde5fc1"
flag_key123 = "04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf"



# Solving for individual key values

key2 = xor(bytes.fromhex(key1_2), key1)
key3 = xor(bytes.fromhex(key2_3), key2)

key1_2_3 = xor(bytes.fromhex(key1_2), key3)
```

```
flag = xor(bytes.fromhex(flag_key123), key1_2_3)


print(flag.decode())
```



```python
from pwn import xor

# Given values on the challenge question

key1 = bytes.fromhex("a6c8b6733c9b22de7bc0253266a3867df55acde8635e1
key1_2 = "37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e"
key2_3 = "c1545756687e7573db23aa1c3452a098b71a7fbf0fddddde5fc1"
flag_key123 = "04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf


# Solving for individual key values

key2 = xor(bytes.fromhex(key1_2), key1)
key3 = xor(bytes.fromhex(key2_3), key2)

key1_2_3 = xor(bytes.fromhex(key1_2), key3)


flag = xor(bytes.fromhex(flag_key123), key1_2_3)


print(flag.decode())

```

```
┌──(b00168210㉿kali)-[~/CryptoHack]
└─$ /bin/python /home/b00168210/CryptoHack/xor_properties.py
crypto{x0r_i5_ass0c1at1v3}
```

FLAG: **crypto{x0r_i5_ass0c1at1v3}**

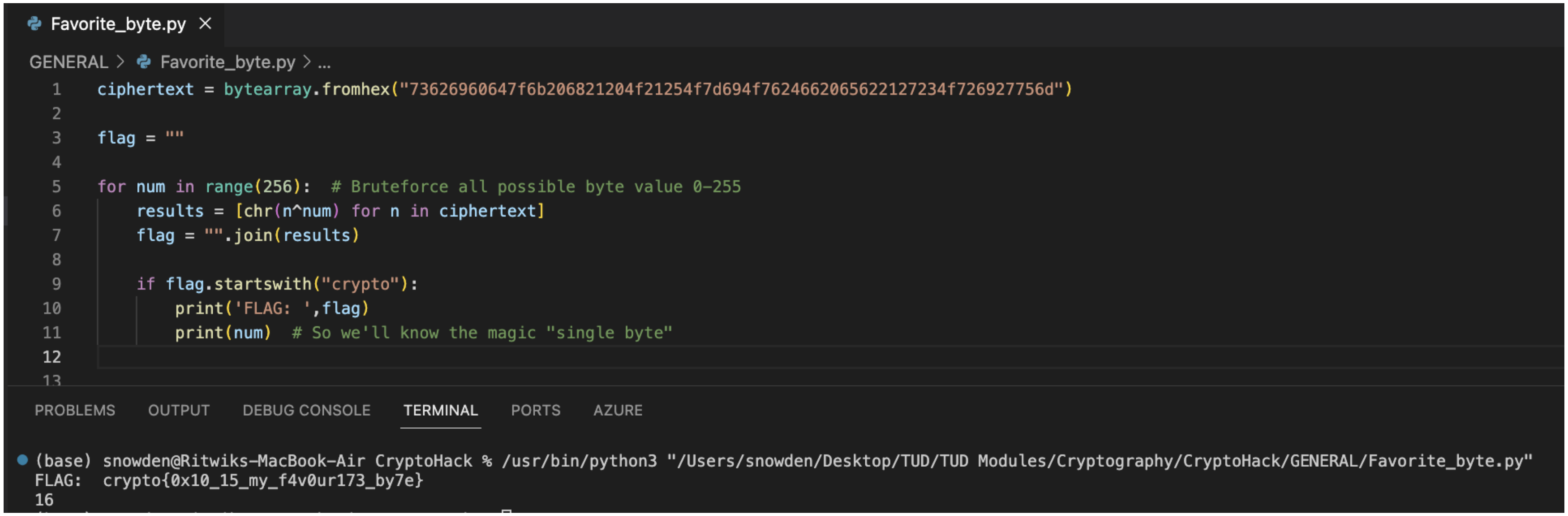
Challenge: Favourite byte

For the next few challenges, you'll use what you've just learned to solve some more XOR puzzles.

I've hidden some data using XOR with a single byte, but that byte is a secret. Don't forget to decode from hex first.

```
73626960647f6b206821204f21254f7d694f7624662065622127234f726927756d
```

```
ciphertext = bytearray.fromhex("73626960647f6b206821204f21254f7d694f7624662065622127234f7

flag = ""

for num in range(256):  # Bruteforce all possible byte value 0-255
    results = [chr(n^num) for n in ciphertext]
    flag = "".join(results)

    if flag.startswith("crypto"):
        print('FLAG: ',flag)
        print(num)  # So we'll know the magic "single byte"
```



```python
ciphertext = bytearray.fromhex("73626960647f6b206821204f21254f7d694f7624662065622127234f726927756d")

flag = ""

for num in range(256):  # Bruteforce all possible byte value 0-255
    results = [chr(n^num) for n in ciphertext]
    flag = "".join(results)

    if flag.startswith("crypto"):
        print('FLAG: ',flag)
        print(num)  # So we'll know the magic "single byte"
```

```
(base) snowden@Ritwiks-MacBook-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/Favorite_byte.py"
FLAG:  crypto{0x10_15_my_f4v0ur173_by7e}
16
```

FLAG: **crypto{0x10_15_my_f4v0ur173_by7e}**

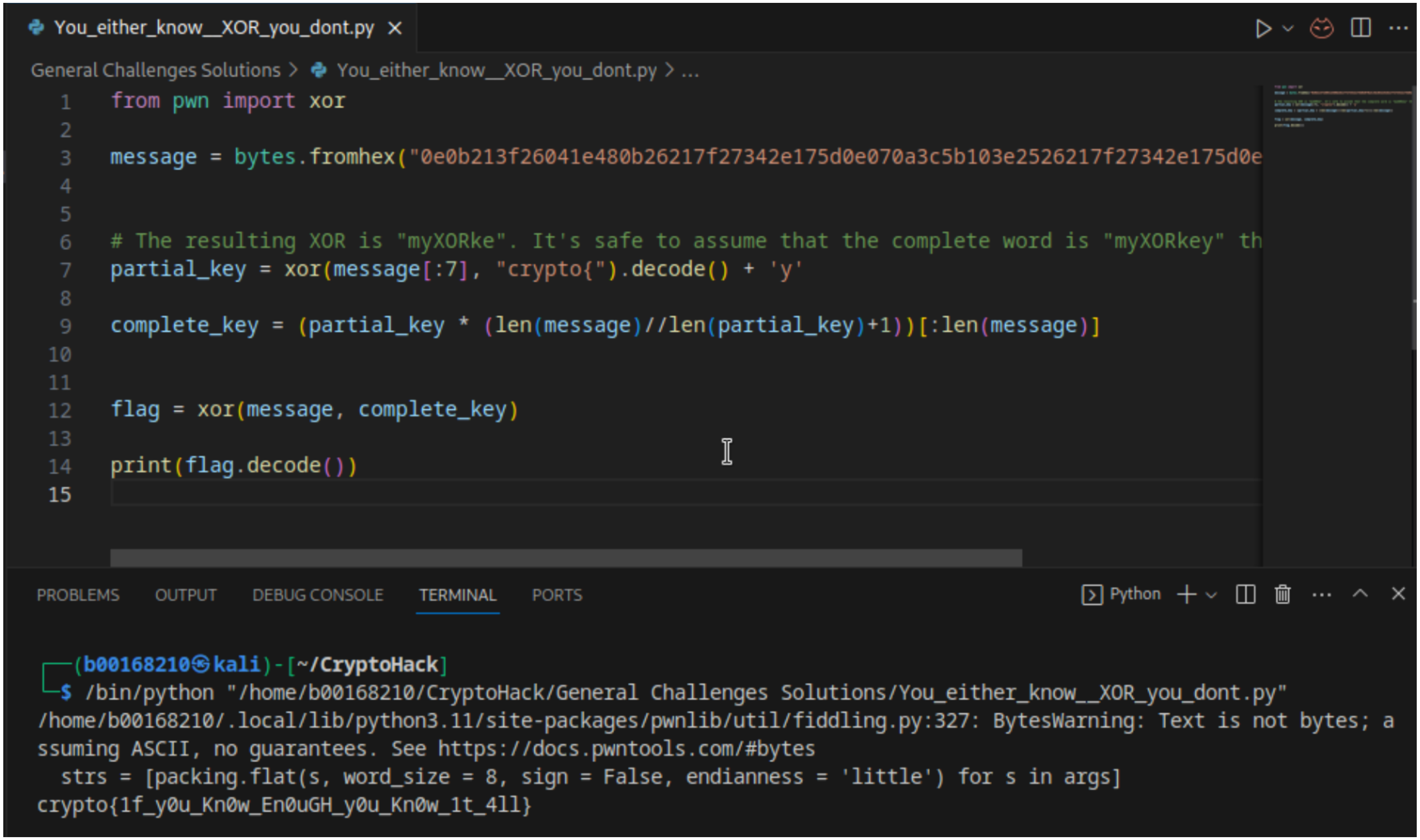
Challenge: You either know, XOR you don't

I've encrypted the flag with my secret key, you'll never be able to guess it.

`0e0b213f26041e480b26217f27342e175d0e070a3c5b103e2526217f27342e175d0e077e263451150104`

```
from pwn import xor

message = bytes.fromhex("0e0b213f26041e480b26217f27342e175d0e070a3c5b103e2526217f27342e1


# The resulting XOR is "myXORke". It's safe to assume that the complete word is "myXORkey
```

```
partial_key = xor(message[:7], "crypto{").decode() + 'y'

complete_key = (partial_key * (len(message)//len(partial_key)+1))[:len(message)]


flag = xor(message, complete_key)

print(flag.decode())
```


You_either_know__XOR_you_dont.py
General Challenges Solutions > You_either_know__XOR_you_dont.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python

```python
from pwn import xor

message = bytes.fromhex("0e0b213f26041e480b26217f27342e175d0e070a3c5b103e2526217f27342e175d0e


# The resulting XOR is "myXORke". It's safe to assume that the complete word is "myXORkey" th
partial_key = xor(message[:7], "crypto{").decode() + 'y'

complete_key = (partial_key * (len(message)//len(partial_key)+1))[:len(message)]


flag = xor(message, complete_key)

print(flag.decode())

```

```
┌──(b00168210㉿kali)-[~/CryptoHack]
└─$ /bin/python "/home/b00168210/CryptoHack/General Challenges Solutions/You_either_know__XOR_you_dont.py"
/home/b00168210/.local/lib/python3.11/site-packages/pwnlib/util/fiddling.py:327: BytesWarning: Text is not bytes; a
ssuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  strs = [packing.flat(s, word_size = 8, sign = False, endianness = 'little') for s in args]
crypto{1f_y0u_Kn0w_En0uGH_y0u_Kn0w_1t_4ll}
```

FLAG: **crypto{1f_y0u_Kn0w_En0uGH_y0u_Kn0w_1t_4ll}**

Challenge: Lemur XOR

I've hidden two cool images by XOR with the same secret key so you can't see them!

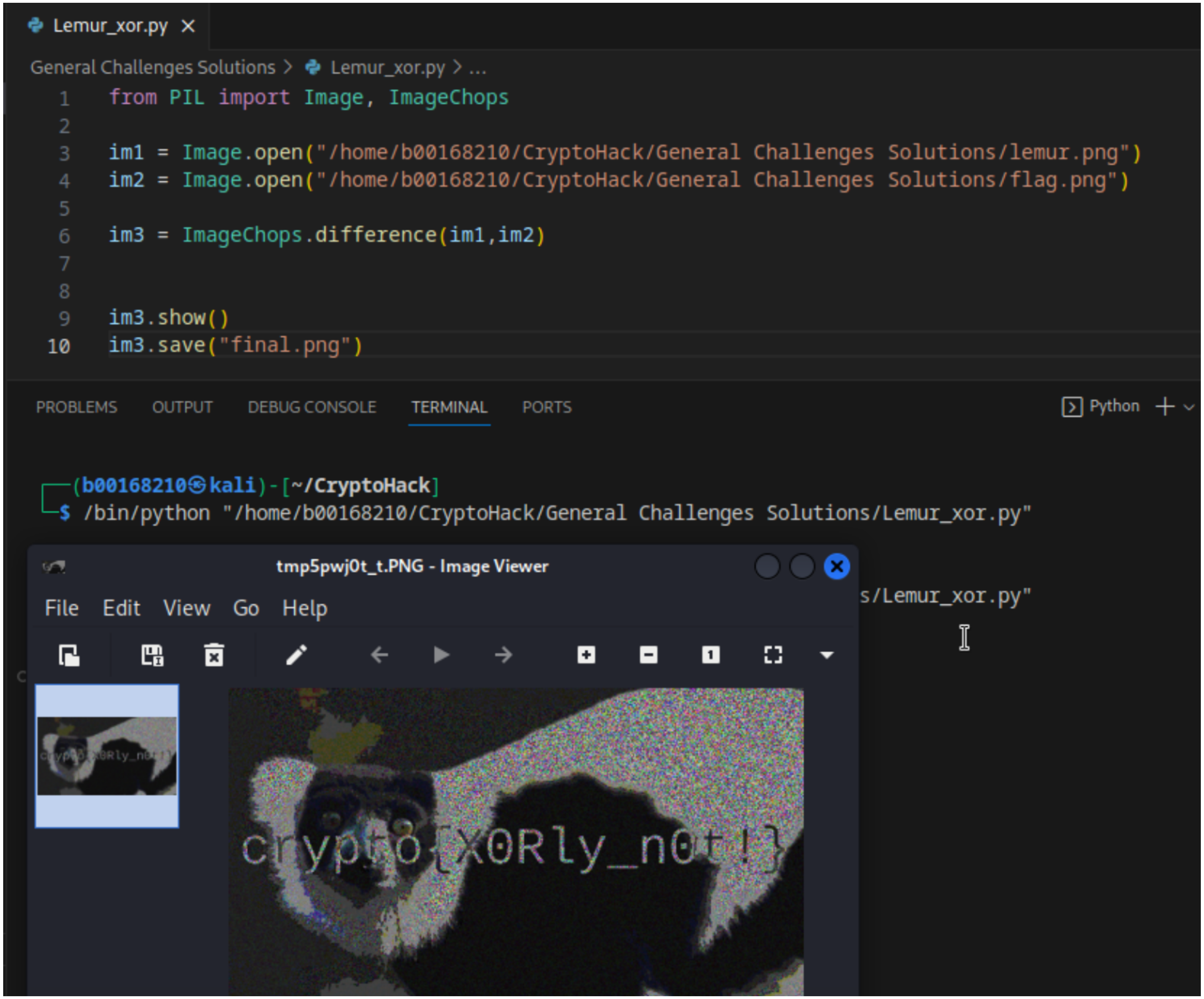
This challenge requires performing a visual XOR between the RGB bytes of the two images - not an XOR of all the data bytes of the files.

Challenge files:

-lemur.png

-flag.png

```
from PIL import Image, ImageChops

im1 = Image.open("/home/b00168210/CryptoHack/General Challenges Solutions/lemur.png")
im2 = Image.open("/home/b00168210/CryptoHack/General Challenges Solutions/flag.png")

im3 = ImageChops.difference(im1,im2)

im3.show()
im3.save("final.png")
```



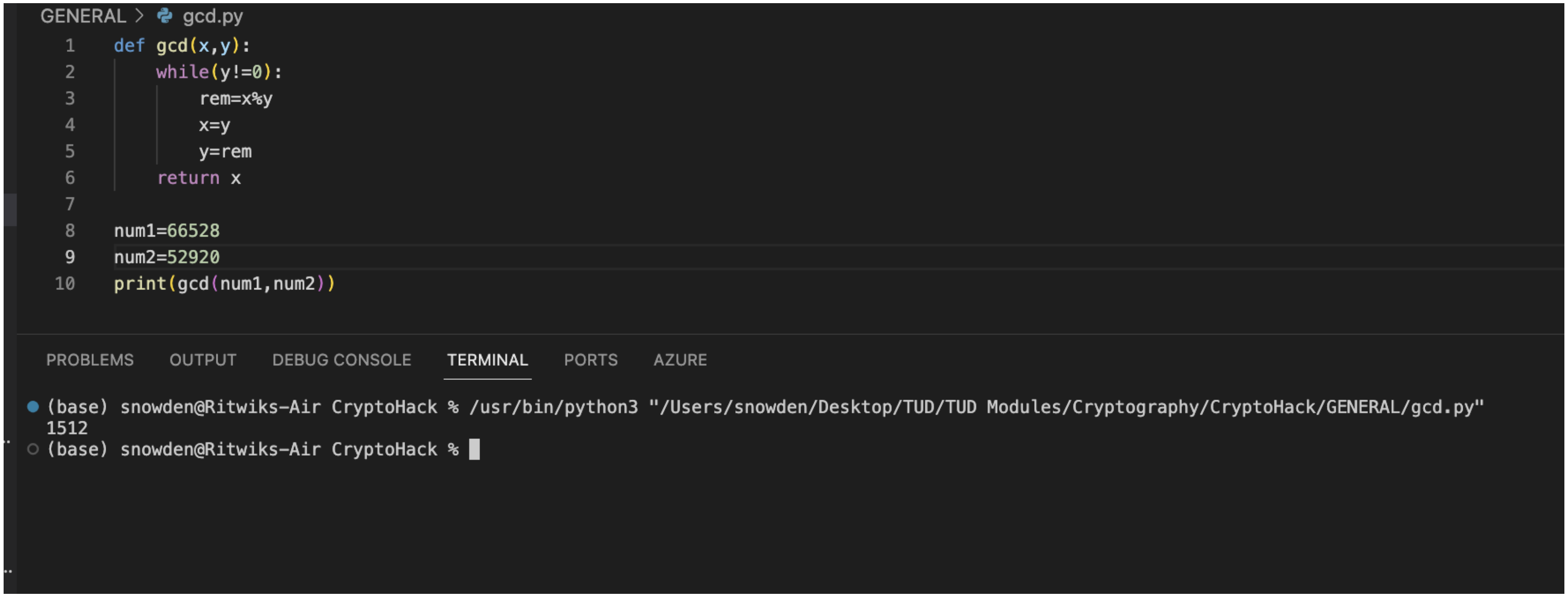
```python
from PIL import Image, ImageChops

im1 = Image.open("/home/b00168210/CryptoHack/General Challenges Solutions/lemur.png")
im2 = Image.open("/home/b00168210/CryptoHack/General Challenges Solutions/flag.png")

im3 = ImageChops.difference(im1,im2)


im3.show()
im3.save("final.png")
```

```
┌──(b00168210㉿kali)-[~/CryptoHack]
└─$ /bin/python "/home/b00168210/CryptoHack/General Challenges Solutions/Lemur_xor.py"

                                                              s/Lemur_xor.py"
```

FLAG: **crypto{X0Rly_n0t!}**

MATHEMATICS

Challenge: Greatest Common Divisor

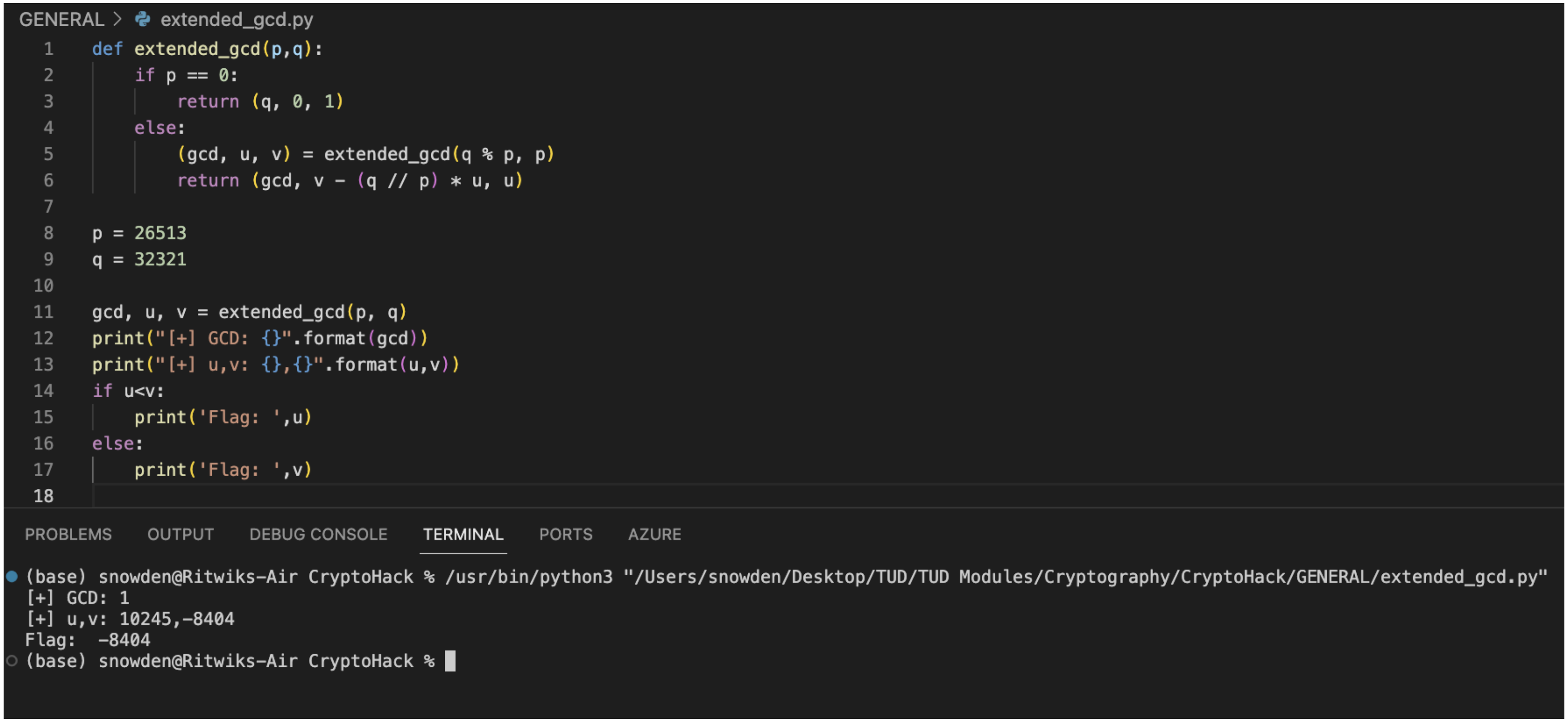


```python
def gcd(x,y):
    while(y!=0):
        rem=x%y
        x=y
        y=rem
    return x

num1=66528
num2=52920
print(gcd(num1,num2))
```

```
● (base) snowden@Ritwiks-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/gcd.py"
  1512
○ (base) snowden@Ritwiks-Air CryptoHack % █
```

1512

Challenge: Extended GCD

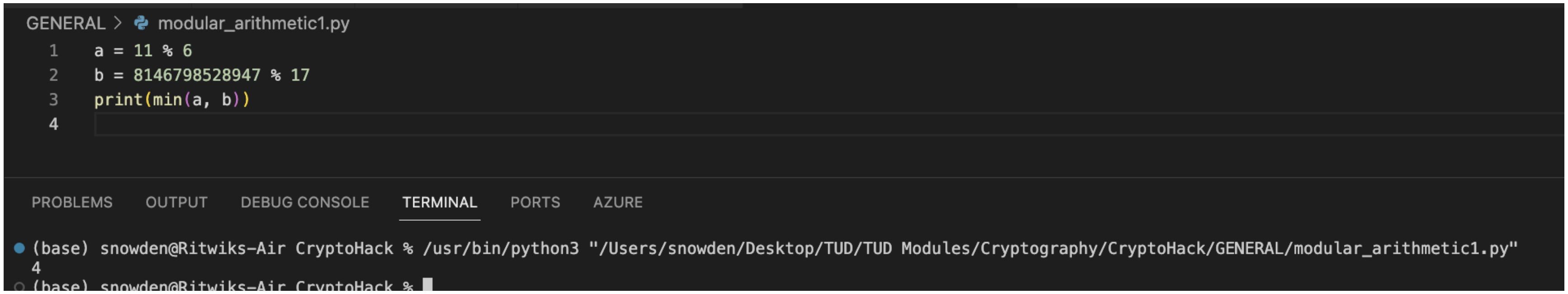


```python
def extended_gcd(p,q):
    if p == 0:
        return (q, 0, 1)
    else:
        (gcd, u, v) = extended_gcd(q % p, p)
        return (gcd, v - (q // p) * u, u)

p = 26513
q = 32321

gcd, u, v = extended_gcd(p, q)
print("[+] GCD: {}".format(gcd))
print("[+] u,v: {},{}".format(u,v))
if u<v:
    print('Flag: ',u)
else:
    print('Flag: ',v)
```

```
(base) snowden@Ritwiks-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/extended_gcd.py"
[+] GCD: 1
[+] u,v: 10245,-8404
Flag:  -8404
(base) snowden@Ritwiks-Air CryptoHack %
```

FLAG: **-8404**

Challenge: Modular Arithmetic 1



```python
a = 11 % 6
b = 8146798528947 % 17
print(min(a, b))

```

```
(base) snowden@Ritwiks-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/modular_arithmetic1.py"
4
(base) snowden@Ritwiks-Air CryptoHack %
```

4

Challenge: Modular Arithmetic 2

This problem right here deals with Fermat's Little Theorem.

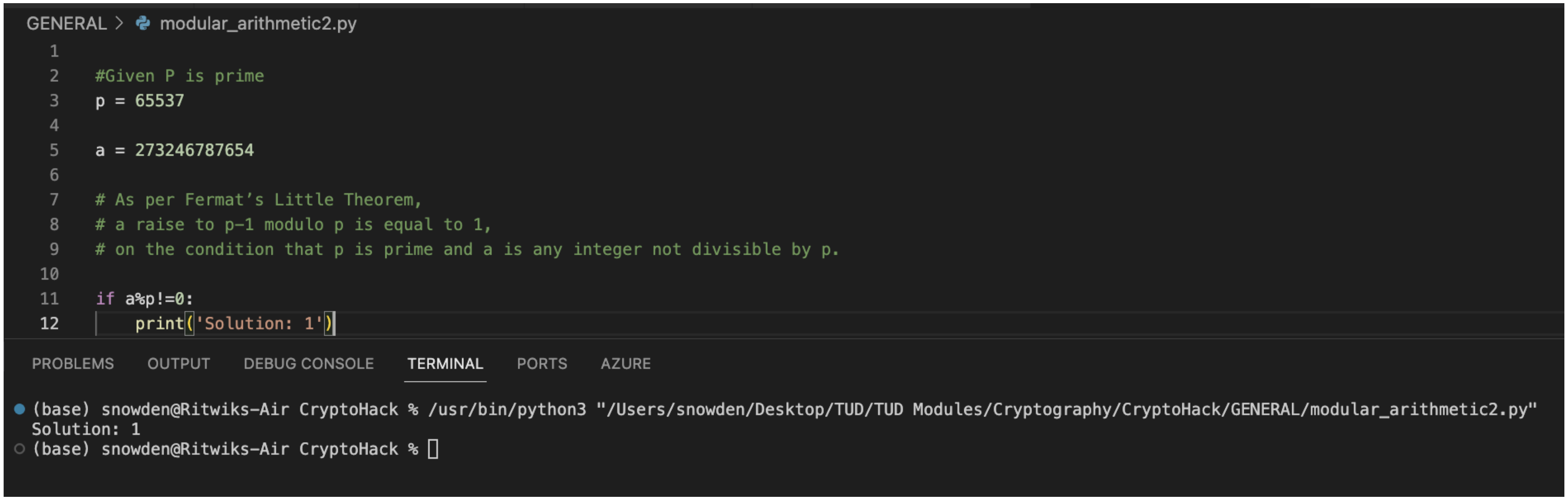
$$a^{p-1} \equiv 1 \pmod{p}$$

Fermat's Little Theorem (FLT)

You could say that *a* raise to *p-1* modulo *p* is equal to 1, on the **condition that p is prime** and ***a* is any integer not divisible by *p*.**

```
#Given P is prime
p = 65537

```

```
a = 273246787654

# As per Fermat’s Little Theorem,
# a raise to p-1 modulo p is equal to 1,
# on the condition that p is prime and a is any integer not divisible by p.

if a%p!=0:
    print('Solution: 1')
```



```

#Given P is prime
p = 65537

a = 273246787654

# As per Fermat’s Little Theorem,
# a raise to p–1 modulo p is equal to 1,
# on the condition that p is prime and a is any integer not divisible by p.

if a%p!=0:
    print('Solution: 1')
```

```
(base) snowden@Ritwiks-Air CryptoHack % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/modular_arithmetic2.py"
Solution: 1
(base) snowden@Ritwiks-Air CryptoHack %
```

1

Challenge: Modular Inverting

What is the inverse element: `3 * d ≡ 1 mod 13`?

Find ‘d’ such that,

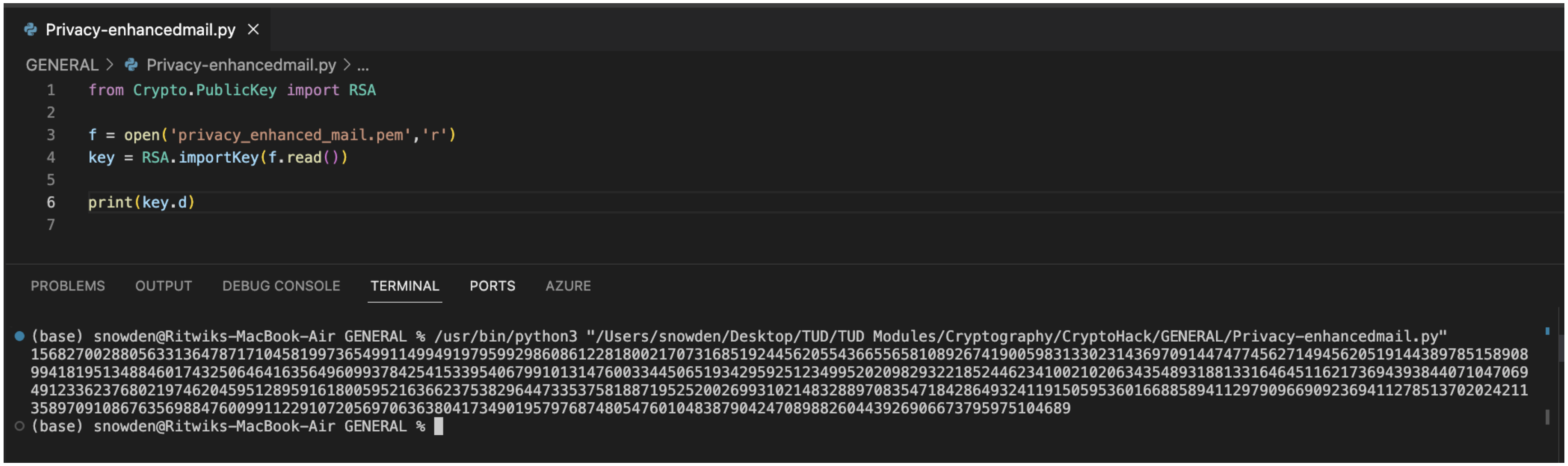
3*d -1 = multiple of 13

3***9**-1 = 26 = 13*2

Solution: **9**

DATA FORMATS

Privacy-Enhanced Mail?


Privacy-enhancedmail.py ×
GENERAL > Privacy-enhancedmail.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
AZURE

```python
from Crypto.PublicKey import RSA

f = open('privacy_enhanced_mail.pem','r')
key = RSA.importKey(f.read())

print(key.d)

```

```
● (base) snowden@Ritwiks-MacBook-Air GENERAL % /usr/bin/python3 "/Users/snowden/Desktop/TUD/TUD Modules/Cryptography/CryptoHack/GENERAL/Privacy-enhancedmail.py"
  15682700288056331364787171045819973654991149949197959929860861228180021707316851924456205543665565810892674190059831330231436970914474774562714945620519144389785158908
  99418195134884601743250646416356496099378425415339540679910131476003344506519342959251234995202098293221852446234100210206343548931881331646451162173694393844071047069
  49123362376802197462045951289591618005952163662375382964473353758188719525200269931021483288970835471842864932411915059536016688589411297909669092369411278513702024211
  35897091086763569884760099112291072056970636380417349019579768748054760104838790424708988260443926906673795975104689
○ (base) snowden@Ritwiks-MacBook-Air GENERAL % █
```