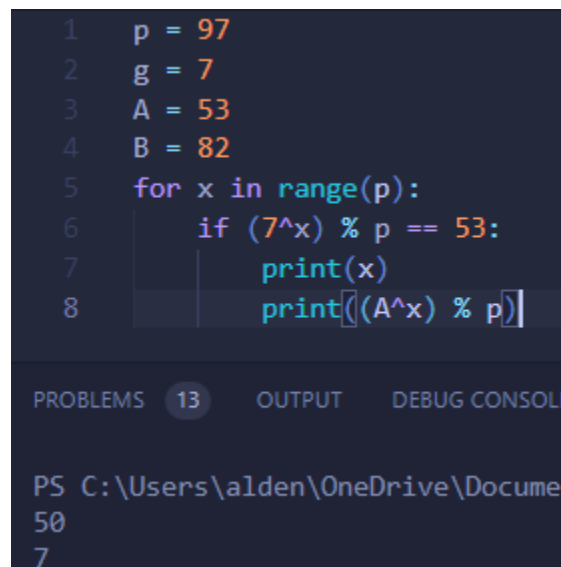


Diffie-Helman

- Figure out the shared secret agreed upon by Alice and Bob. This will be an integer.
 - The shared secret between Alice and Bob is 7
- Show your work. Exactly how did you figure out the shared secret?
 - We took the known variables that were given to us (A, B, g, p), and wrote a short python script to find x and y by brute force. If the integers were of a secure size, this is the step that would fail - as the brute forcing would take an impractical amount of time with a relatively powerful machine.
 - After the script finds x (or y depending on if you are decrypting from bob or alice's side), we can use that number to find their shared secret by plugging it into the formula $B^x \bmod P$ (or $A^x \bmod P$ if decrypting from the other perspective).
 - Alice and Bob's shared secret turned out to be 7, and the python script below is what we used to find it. We verified the message by computing it from the other perspective too - as this took only minor changes to the python code.



```
1 p = 97
2 g = 7
3 A = 53
4 B = 82
5 for x in range(p):
6     if (7^x) % p == 53:
7         print(x)
8         print((A^x) % p)
```

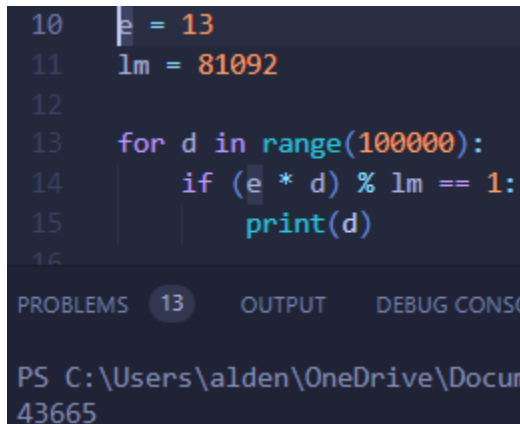
PROBLEMS 13 OUTPUT DEBUG CONSOLE

PS C:\Users\alden\OneDrive\Documents>
50
7

RSA

- Figure out the encrypted message sent from Alice to Bob.
 - Dear Bob, check this out. <https://www.surveillancewatch.io/> See ya, Alice.

- Show your work. Exactly how did you figure out the message?
 - The first thing we did was use an online calculator to find the prime factorization of n . The resulting numbers, 389, and 419 are our p and q respectively. After finding p and q , we used those numbers to find $\lambda(n)$ with the formula $\lambda(n) = \text{LCM}(p-1, q-1)$, where $p = 389$, and $q = 419$. This gives us a $\lambda(n) = 81092$. Once we had $\lambda(n)$, we wrote a short python script to calculate d , shown below.

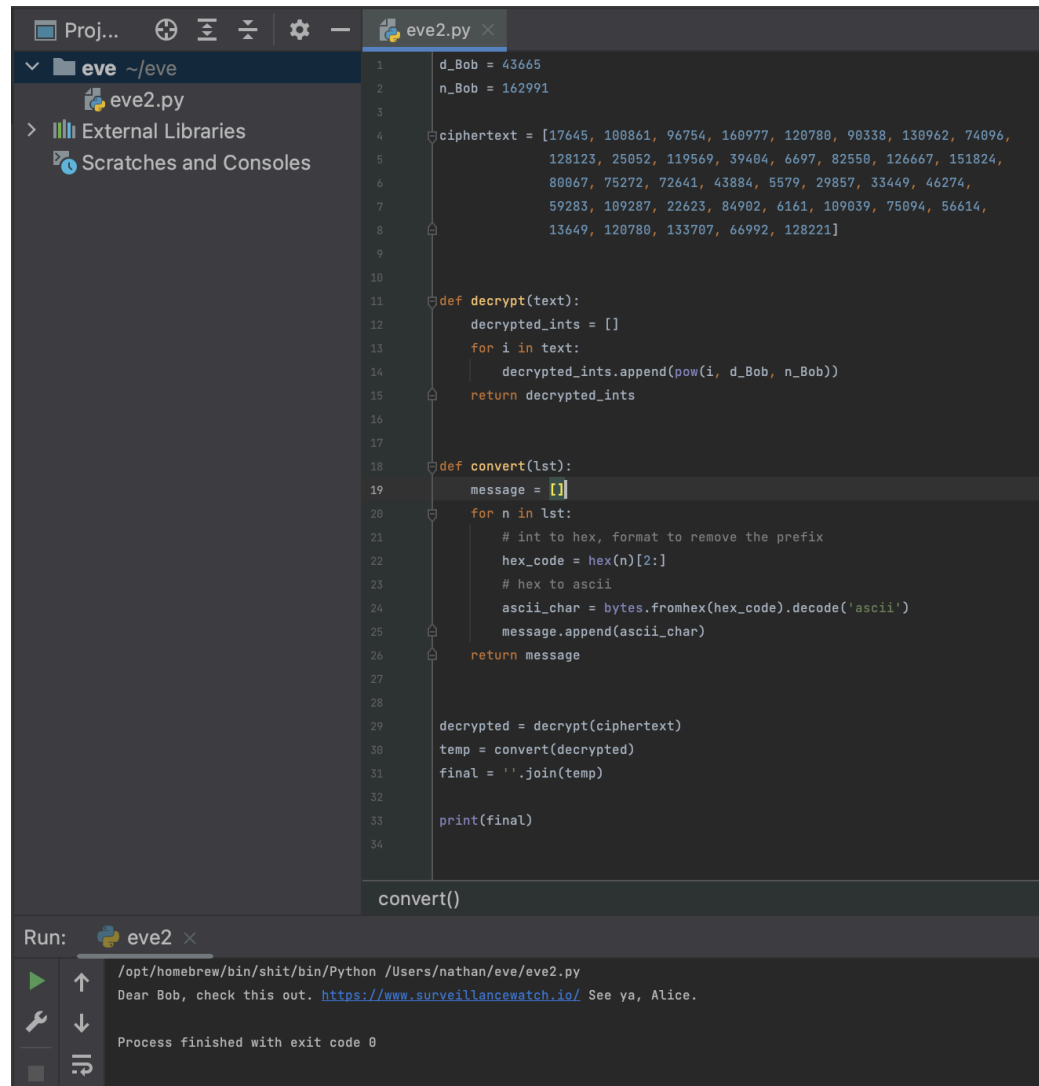


```
10 e = 13
11 lm = 81092
12
13 for d in range(100000):
14     if (e * d) % lm == 1:
15         print(d)
```

PROBLEMS 13 OUTPUT DEBUG CONSOLE

PS C:\Users\alden\OneDrive\Documents> 43665

- After calculating d , it was time to decrypt the message. We wrote another python program (screenshot below) that served two purposes. Firstly it decrypted the ciphertext integers using the standard RSA formula. After initially decrypting the ciphered integers, we found that they did not map correctly to ascii characters, leading us to assume that they had been encoded in another way too. We quickly figured out that they were encoded using hexadecimal, so we wrote another function (had to look up how to do this) to undo that step, and get us the final list of ascii characters. From there, it was as easy as printing the output and seeing what the message was.



```
1 d_Bob = 43665
2 n_Bob = 162991
3
4 ciphertext = [17645, 100861, 96754, 160977, 120780, 90338, 130962, 74096,
5              128123, 25052, 119569, 39404, 6697, 82550, 126667, 151824,
6              88067, 75272, 72641, 43884, 5579, 29857, 33449, 46274,
7              59283, 109287, 22623, 84902, 6161, 109039, 75094, 56614,
8              13649, 120780, 133707, 66992, 128221]
9
10
11 def decrypt(text):
12     decrypted_ints = []
13     for i in text:
14         decrypted_ints.append(pow(i, d_Bob, n_Bob))
15     return decrypted_ints
16
17
18 def convert(lst):
19     message = []
20     for n in lst:
21         # int to hex, format to remove the prefix
22         hex_code = hex(n)[2:]
23         # hex to ascii
24         ascii_char = bytes.fromhex(hex_code).decode('ascii')
25         message.append(ascii_char)
26     return message
27
28
29 decrypted = decrypt(ciphertext)
30 temp = convert(decrypted)
31 final = ''.join(temp)
32
33 print(final)
34
35 convert()
```

Run: eve2 x

```
/opt/homebrew/bin/shit/bin/Python /Users/nathan/eve/eve2.py
Dear Bob, check this out. https://www.surveillancewatch.io/ See ya, Alice.

Process finished with exit code 0
```

- Show precisely where in your process you would have failed if the integers involved were much larger.
 - The process would have failed at our first step, finding the prime factors of n , as with integers of a secure size, any factoring algorithm would take far too long to be practical.
- Explain, briefly, why the message encoding Alice used would be insecure even if Bob's keys involved larger integers.
 - Since the same encoding key is used for every block of text, and the message isn't scrambled in any way, pieces of text that repeat in the original message would repeat in the ciphertext. This could help a malicious actor gain a foothold in deciphering the text even without breaking the cipher directly.