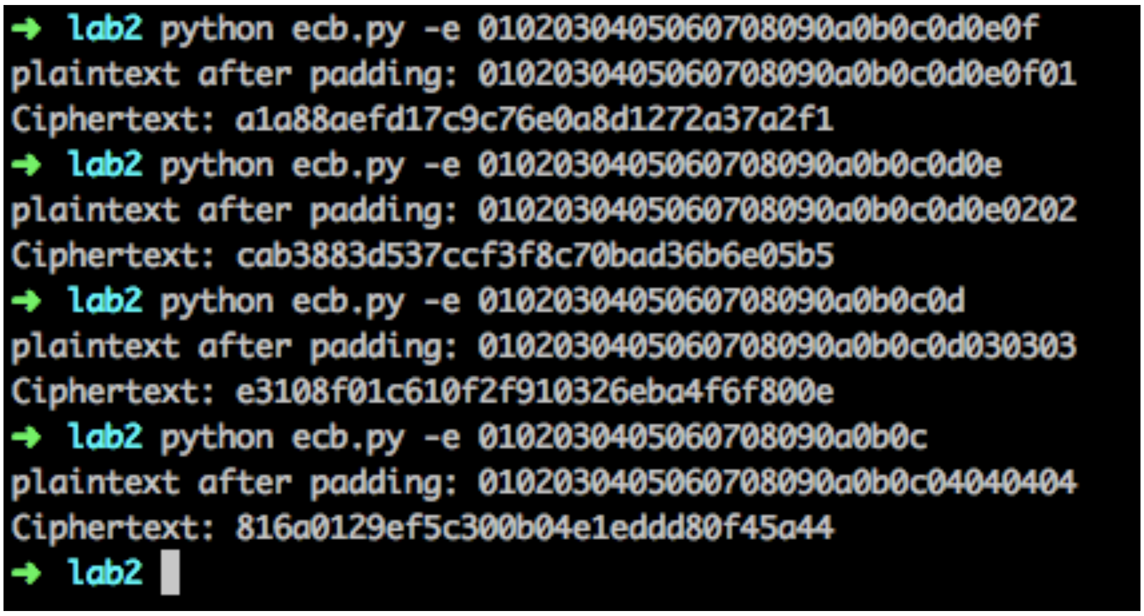
2. (1) I choose the following plaintext in order to have the paddings of 01, 0202, 030303 and 04040404 respectively.

   0102030405060708090a0b0c0d0e0f
   0102030405060708090a0b0c0d0e
   0102030405060708090a0b0c0d
   0102030405060708090a0b0c

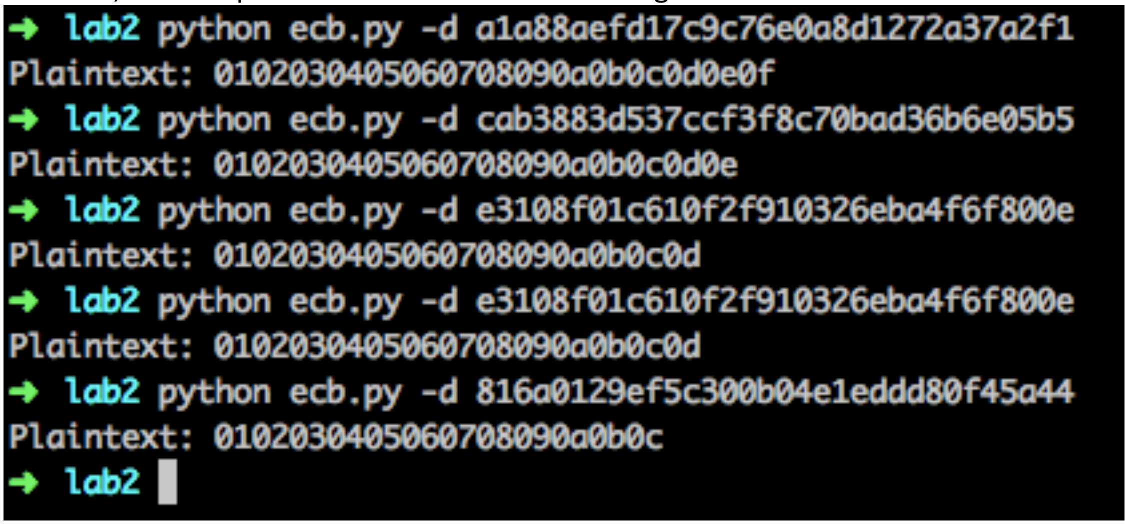
   After using ecb.py to encrypt above plaintexts, we can see that the padding is correct with 01, 0202, 030303 and 0404040404 respectively.

   The ciphertexts are printed as follows in the terminal:



```
➜  lab2 python ecb.py -e 0102030405060708090a0b0c0d0e0f
plaintext after padding: 0102030405060708090a0b0c0d0e0f01
Ciphertext: a1a88aefd17c9c76e0a8d1272a37a2f1
➜  lab2 python ecb.py -e 0102030405060708090a0b0c0d0e
plaintext after padding: 0102030405060708090a0b0c0d0e0202
Ciphertext: cab3883d537ccf3f8c70bad36b6e05b5
➜  lab2 python ecb.py -e 0102030405060708090a0b0c0d
plaintext after padding: 0102030405060708090a0b0c0d030303
Ciphertext: e3108f01c610f2f910326eba4f6f800e
➜  lab2 python ecb.py -e 0102030405060708090a0b0c
plaintext after padding: 0102030405060708090a0b0c04040404
Ciphertext: 816a0129ef5c300b04e1eddd80f45a44
➜  lab2
```

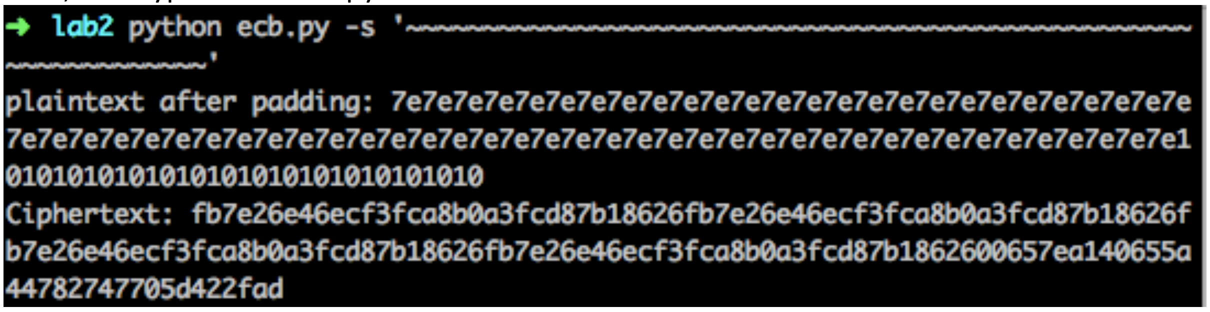
   (2) Then, I decrypt the ciphertexts into plaintexts with the following command in the terminal, and the plaintexts are shown as followings in the terminal



```
➜  lab2 python ecb.py -d a1a88aefd17c9c76e0a8d1272a37a2f1
Plaintext: 0102030405060708090a0b0c0d0e0f
➜  lab2 python ecb.py -d cab3883d537ccf3f8c70bad36b6e05b5
Plaintext: 0102030405060708090a0b0c0d0e
➜  lab2 python ecb.py -d e3108f01c610f2f910326eba4f6f800e
Plaintext: 0102030405060708090a0b0c0d
➜  lab2 python ecb.py -d e3108f01c610f2f910326eba4f6f800e
Plaintext: 0102030405060708090a0b0c0d
➜  lab2 python ecb.py -d 816a0129ef5c300b04e1eddd80f45a44
Plaintext: 0102030405060708090a0b0c
➜  lab2
```

(3) I create the following string '~~~~~~~~~~~~~~~~' as one block , in which has 16 '~' . Then I repeat this block three times to formulate the string '~~~~~~' in which it has 64 '~'

then, I encrypt it with ecb.py in the terminal as follows:



```
➜ lab2 python ecb.py -s '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~'
plaintext after padding: 7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e
7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e1
0101010101010101010101010101010
Ciphertext: fb7e26e46ecf3fca8b0a3fcd87b18626fb7e26e46ecf3fca8b0a3fcd87b18626f
b7e26e46ecf3fca8b0a3fcd87b18626fb7e26e46ecf3fca8b0a3fcd87b1862600657ea140655a
44782747705d422fad
```

The plaintext after padding is
7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e
7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e7e
10101010101010101010101010101010

we have one more padding block 10101010101010101010101010101010

The ciphertext is
fb7e26e46ecf3fca8b0a3fcd87b18626fb7e26e46ecf3fca8b0a3fcd87b18626fb7e26e4
6ecf3fca8b0a3fcd87b18626fb7e26e46ecf3fca8b0a3fcd87b1862600657ea140655a44
782747705d422fad

yes, we have ciphertext fb7e26e46ecf3fca8b0a3fcd87b18626, which is repeated three times, the last block of ciphertext is 00657ea140655a44782747705d422fad which is encrypted by the padding block.

3. For cbc.py, I use the following IV: '00000000000000000000000000000000' (In this lab, I use this IV for cbc.py, oracle.py, attack.py)

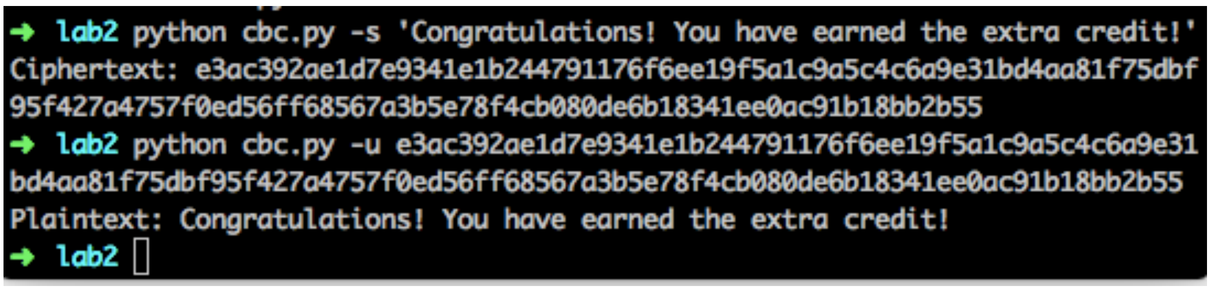
   I use the following command to test cbc.py in the terminal:

   python cbc.py –s 'Congratulations! You have earned the extra credit!'

   the ciphertext is in the following:

   e3ac392ae1d7e9341e1b244791176f6ee19f5a1c9a5c4c6a9e31bd4aa81f75dbf95f427
   a4757f0ed56ff68567a3b5e78f4cb080de6b18341ee0ac91b18bb2b55

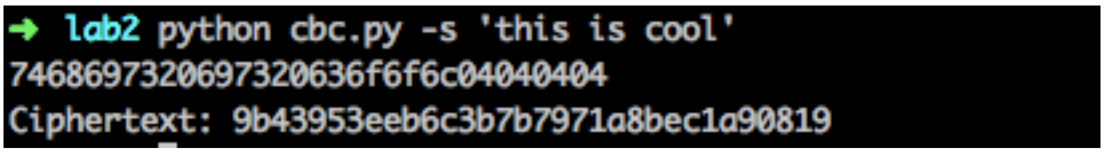
   Then, I decrypt it to get the original plaintext. The result is shown in the following picture.



```
➜ lab2 python cbc.py -s 'Congratulations! You have earned the extra credit!'
Ciphertext: e3ac392ae1d7e9341e1b244791176f6ee19f5a1c9a5c4c6a9e31bd4aa81f75dbf
95f427a4757f0ed56ff68567a3b5e78f4cb080de6b18341ee0ac91b18bb2b55
➜ lab2 python cbc.py -u e3ac392ae1d7e9341e1b244791176f6ee19f5a1c9a5c4c6a9e31
bd4aa81f75dbf95f427a4757f0ed56ff68567a3b5e78f4cb080de6b18341ee0ac91b18bb2b55
Plaintext: Congratulations! You have earned the extra credit!
➜ lab2 
```

4. To test the oracle.py, let’s encrypt a message such as ‘this is cool’ with python cbc.py –s ‘this is cool’

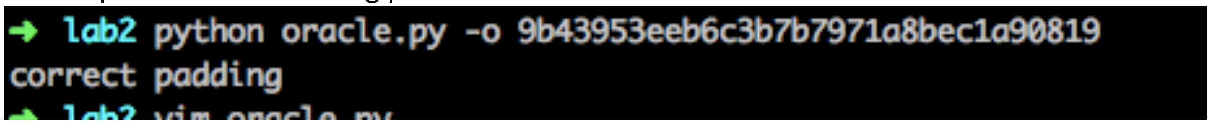
   we will get the ciphertext which has the correct padding in the plaintext.
   And, we get the padded plaintext and ciphertext respectively in the following:
   746869732069732063 6f6f6c04040404
   9b43953eeb6c3b7b7971a8bec1a90819



```
➜ lab2 python cbc.py -s 'this is cool'
74686973206973206f6f6c04040404
Ciphertext: 9b43953eeb6c3b7b7971a8bec1a90819
```

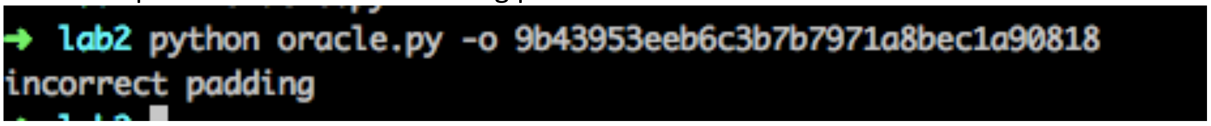
   if we use oracle to test whether it is a correct padding of ciphertext, obviously it is a correct padding.

   the output is in the following picture:



```
➜ lab2 python oracle.py -o 9b43953eeb6c3b7b7971a8bec1a90819
correct padding
```

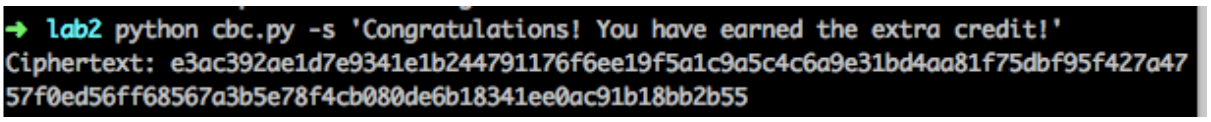
   Then, let’s modify the ciphertext to 9b43953eeb6c3b7b7971a8bec1a90818, which will lead to uncorrect padding because we change the last bit of ciphertext from 9 to 8. The output is shown in the following picture.



```
➜ lab2 python oracle.py -o 9b43953eeb6c3b7b7971a8bec1a90818
incorrect padding
```

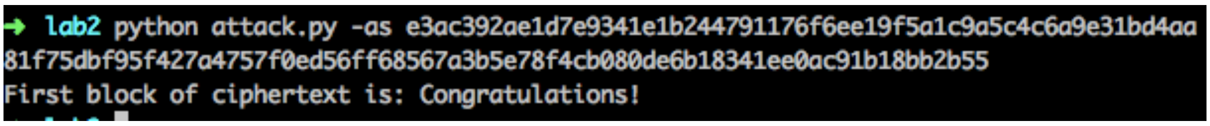
5. We first encrypt the plaintext ‘Congratulations! You have earned the extra credit!’ in question 3 with the following command in the terminal.

   python cbc.py -s 'Congratulations! You have earned the extra credit!'



```
➜ lab2 python cbc.py -s 'Congratulations! You have earned the extra credit!'
Ciphertext: e3ac392ae1d7e9341e1b244791176f6ee19f5a1c9a5c4c6a9e31bd4aa81f75dbf95f427a47
57f0ed56ff68567a3b5e78f4cb080de6b18341ee0ac91b18bb2b55
```

after we get the ciphertext, we can use attack.py to decrypt the first block of ciphertext. The result is shown in the following picture.



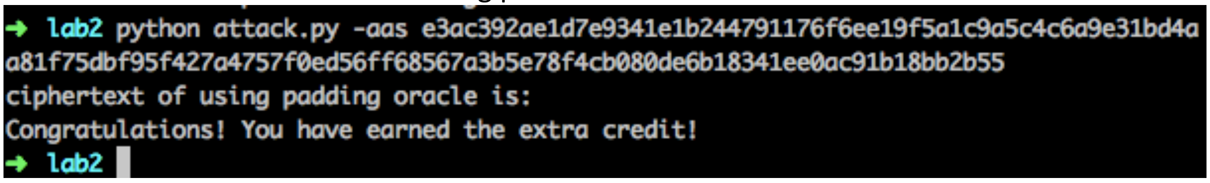
```
➜ lab2 python attack.py -as e3ac392ae1d7e9341e1b244791176f6ee19f5a1c9a5c4c6a9e31bd4aa
81f75dbf95f427a4757f0ed56ff68567a3b5e78f4cb080de6b18341ee0ac91b18bb2b55
First block of ciphertext is: Congratulations!
```

6. I think it is easy to decrypt the remaining blocks after you successfully decrypt the first block. We just repeat the process of decrypting the first block for the remaining blocks.

   In order to decrypt all the blocks of ciphertext in question 3, we can use the same ciphertext in question 5, and decrypt it with padding oracle attack.
   We use the following command in the terminal:

   python attack.py -aas
   e3ac392ae1d7e9341e1b244791176f6ee19f5a1c9a5c4c6a9e31bd4aa81f75dbf95f427
   a4757f0ed56ff68567a3b5e78f4cb080de6b18341ee0ac91b18bb2b55

   the result of is shown in the following picture.



```
➜ lab2 python attack.py -aas e3ac392ae1d7e9341e1b244791176f6ee19f5a1c9a5c4c6a9e31bd4a
a81f75dbf95f427a4757f0ed56ff68567a3b5e78f4cb080de6b18341ee0ac91b18bb2b55
ciphertext of using padding oracle is:
Congratulations! You have earned the extra credit!
➜ lab2
```