Exercises:

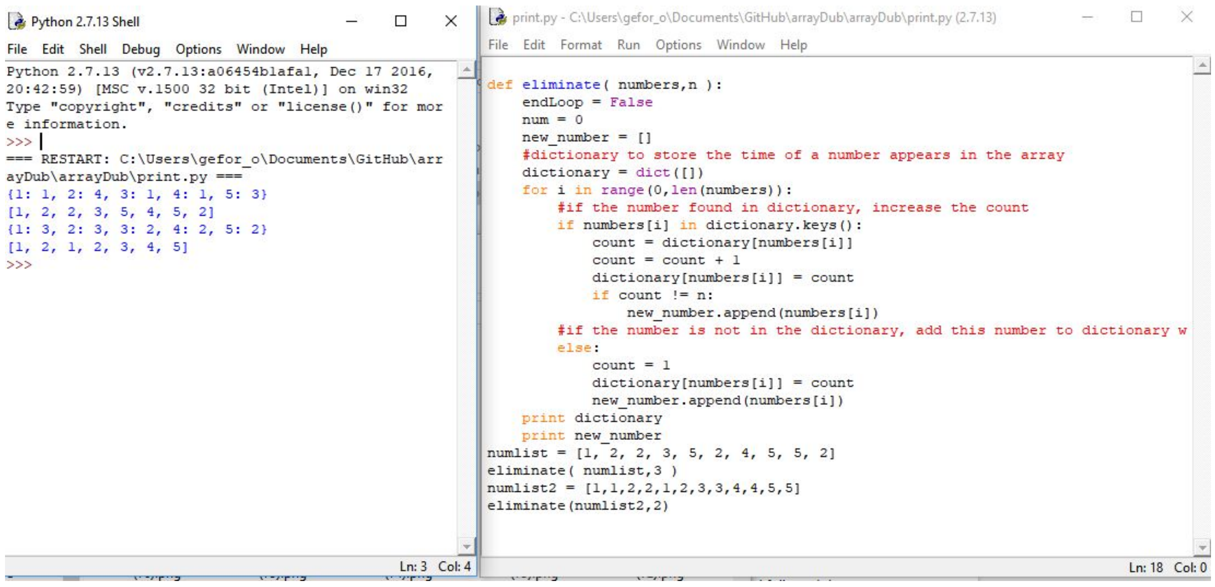
First exercise:

Git: https://github.com/geforcexfx/arrayDub/blob/master/print.py

Write a function that remove the n-th duplicate in the array. I use Python to write this program. The big O notation of this program is O(n) because it grows linearly and in direct proportion to the size of the array of numbers. In other word, it is proportion to the loop that iterates to each element of the array:

```
for i in range(0,len(numbers)):
```

Test:
Open the program in python compiler, I use IDLE so I compile the program by pressing F5(Figure 1). The program should remove the third instance of the number. The function: eliminate(numlist,3) with the array of numbers: numlist = [1, 2, 2, 3, 5, 2, 4, 5, 5, 2].
The first print in the IDLE shell is the dictionary, it shows numbers in the array and their time of appearance: {1: 1, 2: 4, 3: 1, 4: 1, 5: 3}, the second print is the result after it remove the third instance of the number, as expected, it removes the number 2 of index 5 in the array and number 5 of index 8 in the array. The second print is the result: [1, 2, 2, 3, 5, 4, 5, 2]
To run the second test, with the array is: numlist2 = [1,1,2,2,1,2,3,3,4,4,5,5] and remove the 2nd instance of the array : eliminate(numlist2,2)
The result is as expected: [1, 2, 1, 2, 3, 4, 5]


Python 2.7.13 Shell
File Edit Shell Debug Options Window Help
Ln: 3 Col: 4
print.py - C:\Users\gefor_o\Documents\GitHub\arrayDub\arrayDub\print.py (2.7.13)
File Edit Format Run Options Window Help
Ln: 18 Col: 0

```
Python 2.7.13 (v2.7.13:a06454b1a1a1, Dec 17 2016,
20:42:59) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for mor
e information.
>>> 
=== RESTART: C:\Users\gefor_o\Documents\GitHub\arr
ayDub\arrayDub\print.py ===
{1: 1, 2: 4, 3: 1, 4: 1, 5: 3}
[1, 2, 2, 3, 5, 4, 5, 2]
{1: 3, 2: 3, 3: 2, 4: 2, 5: 2}
[1, 2, 1, 2, 3, 4, 5]
>>>
```

```
def eliminate( numbers,n ):
    endLoop = False
    num = 0
    new_number = []
    #dictionary to store the time of a number appears in the array
    dictionary = dict([])
    for i in range(0,len(numbers)):
        #if the number found in dictionary, increase the count
        if numbers[i] in dictionary.keys():
            count = dictionary[numbers[i]]
            count = count + 1
            dictionary[numbers[i]] = count
            if count != n:
                new_number.append(numbers[i])
        #if the number is not in the dictionary, add this number to dictionary w
        else:
            count = 1
            dictionary[numbers[i]] = count
            new_number.append(numbers[i])
    print dictionary
    print new_number
numlist = [1, 2, 2, 3, 5, 2, 4, 5, 5, 2]
eliminate( numlist,3 )
numlist2 = [1,1,2,2,1,2,3,3,4,4,5,5]
eliminate(numlist2,2)
```

Figure 1: Exercise 1 test

Second exercise

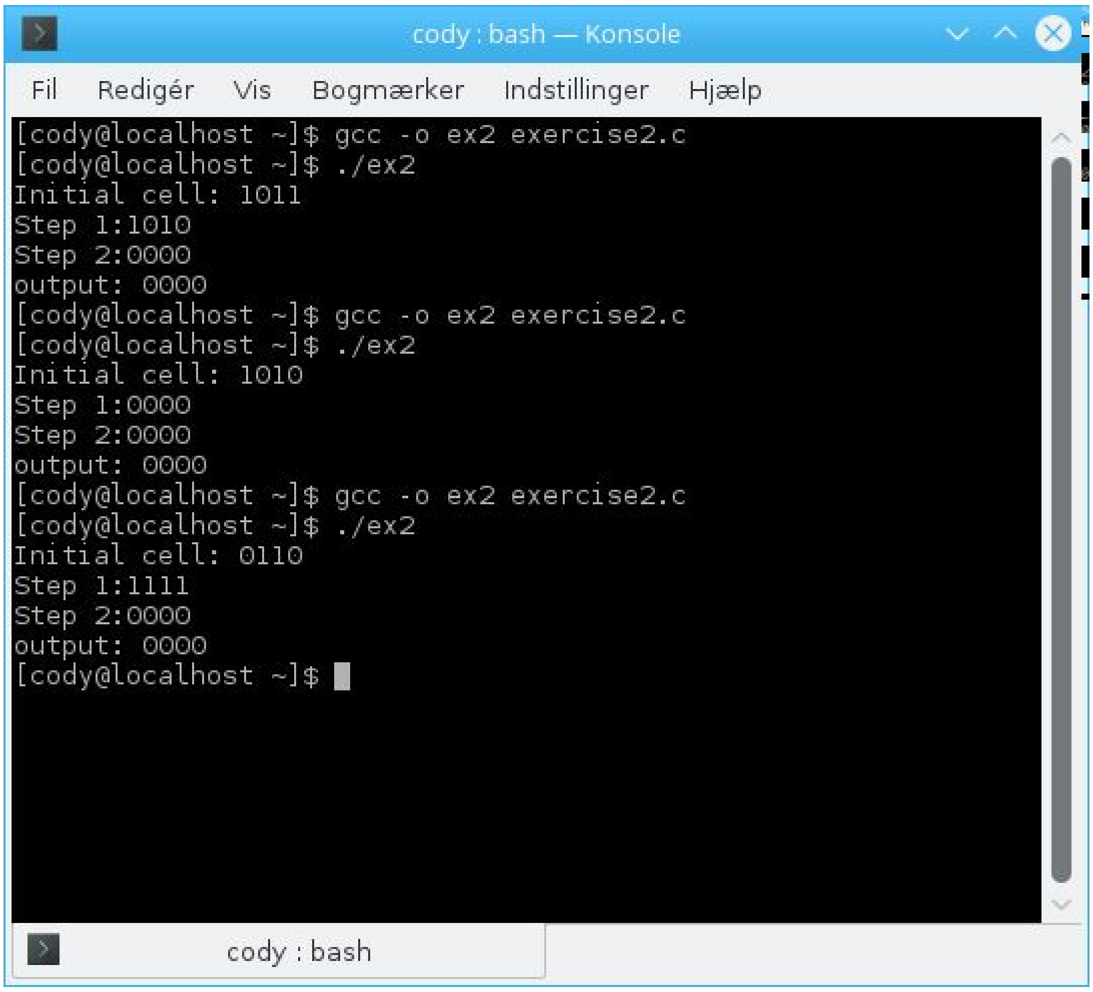
Git: https://github.com/geforcexfx/arrayDub/blob/master/exercise2.c

A number of sensitive cells with value 0 or 1 which I created as an array of 0 and 1. I followed the example so I created 4 elements in the array, I called cells. The cells in the array are inter-connected like a circle, the beginning and the last cell in the array are also connected like a circle. The change of the cell after one step depends on its adjacent cells.

The cell gets excited state 1 if one of its adjacent cells is 1, the cell turns to quiet state 0 when both of its adjacent cells are either 0 or 1.
Testing:
We compile the C program, browse to the folder you store the exercise2.c file. In linux we use command *gcc -o ex2 exercise2.c*
After that we execute the program by command *./ex2*
Requirement as indicated in the example:
- initial configuration is 1,0,1,1
- the configuration after 1 step will be: 1,0,1,0
- the configuration after 2 steps and the output will be: 0,0,0,0

The result of the program is the output, it is show as the same as the example(Figure 2)
I change 2 other configurations to make sure it works as expected.
Initial configurations: 1010
Step 1: 0000
Step 2: 0000
Output: 0000
The program starts at first cell, its left and right neighbor are 0, so it turns to quiet state. The program keep irterate second cell, both of its neighbors are 1, it stay quiet. On the third cell, both of its neighbors are 0, it turn to 0. The forth cell keeps quiet state because both of its neighbor are 1. (Figure 2)

I enter another initial condition which is 0,1,1,0 and it follows the same rules, it gives the expected result output as well.



```
[cody@localhost ~]$ gcc -o ex2 exercise2.c
[cody@localhost ~]$ ./ex2
Initial cell: 1011
Step 1:1010
Step 2:0000
output: 0000
[cody@localhost ~]$ gcc -o ex2 exercise2.c
[cody@localhost ~]$ ./ex2
Initial cell: 1010
Step 1:0000
Step 2:0000
output: 0000
[cody@localhost ~]$ gcc -o ex2 exercise2.c
[cody@localhost ~]$ ./ex2
Initial cell: 0110
Step 1:1111
Step 2:0000
output: 0000
[cody@localhost ~]$
```

Figure 2: Exercise 2 test