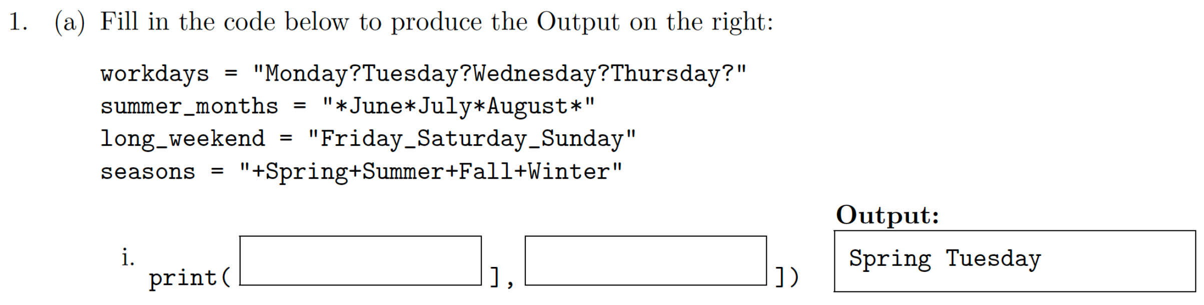
Answer to 1 (a)



1. (a) Fill in the code below to produce the Output on the right:

```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
summer_months = "*June*July*August*"
long_weekend = "Friday_Saturday_Sunday"
seasons = "+Spring+Summer+Fall+Winter"
```

i.
```
print(            ],            ])
```

Output:

```
Spring Tuesday
```

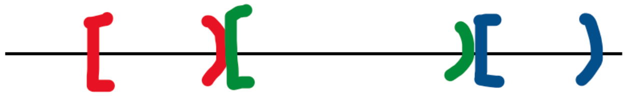
Explanation: This problem tests index and slicing in a string or list.

Word Spring is in string seasons

Letter in left index is included   letters in right index is **not** included

index	0	[1	2	3	4	5	6	7)																		
seasons	+	[S	p	r	i	n	g	+)	S	u	m	m	e	r	+	F	a	l	l	+	W	i	n	t	e	r
Index starting from right	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

To get Spring from string seasons, if we count from left to right, then the leftmost index is 0, and index is increased one a time when moving from left to right. The index of the first letter in Spring is 1, while the last index is 6, to represent in index of Python, where the start index is included **but** the right index is not included – this makes dividing a segment easier, for example,



Also, [start, end) means there are end – start elements in this range.

To get Spring, write seasons[1: 7], you can verify that 7-1 = 6, which are the number of characters in "Spring" without double quotes.

Warning: cannot write seasons[1:7] as seasons[1,7], column symbol between 1 and 7 is like to go from 1 to 7. If you use comma, it is seasons is a two-dimensional array, while 1 is the index of the first dimension and 7 is the index of the second dimension.

If you want to count from right to left, then the rightmost index is -1 and each time when you move to the left, index is increased by 1. It is like when you move from west to east, the coordinate is smaller and smaller.

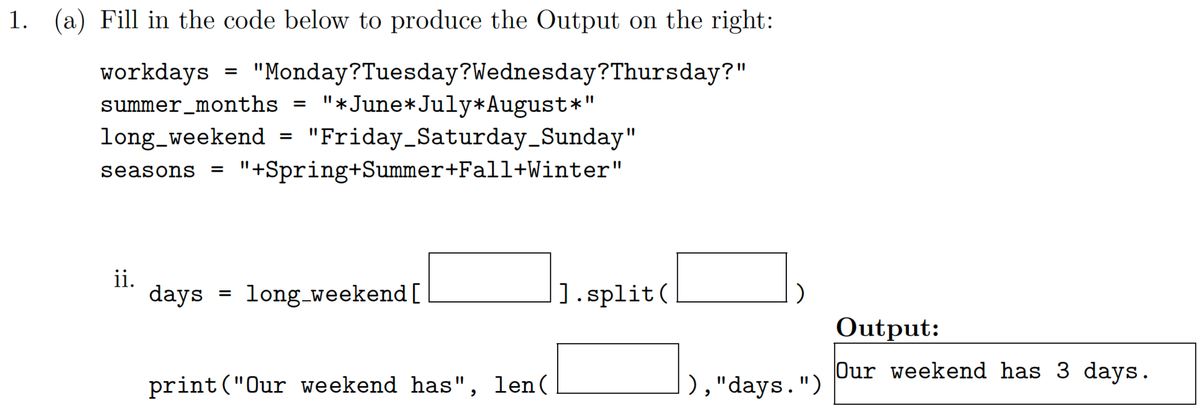
As an alternative, you can write use seasons[-25, -19] to extract “Spring”.

Either seasons[1: 7] or seasons[-25: -19] gives us “Spring”. Note that whichever way you use, the left index is always less than or equal to the right index, also, 7 – 1 = -19 – (-25) = 6, which are the number of letters in the string “Spring” you extract.

Rule of thumb: if the target item is close to the left, then we start to count from left to right, if the target is close to the right, then start to count from right to left, otherwise, the target is close the middle, use whichever counting you feel comfortable.

To get Tuesday, which is in workdays = “Monday?Tuesday?Wednesday?Thursday?”, I will count from left to right since the target is close to the left end. The answer is workdays[7: 14].

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
workdays	M	o	n	d	a	y	?	T	u	e	s	d	a	y	?



1. (a) Fill in the code below to produce the Output on the right:

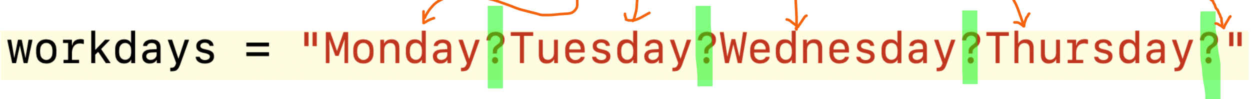
```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
summer_months = "*June*July*August*"
long_weekend = "Friday_Saturday_Sunday"
seasons = "+Spring+Summer+Fall+Winter"
```

ii.
```
days = long_weekend[ □ ].split( □ )

print("Our weekend has", len( □ ),"days.")
```

Output:

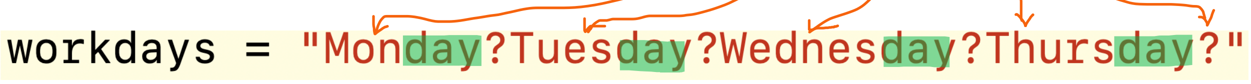
```
Our weekend has 3 days.
```

Method split of a string can divide a string into a list of words. For example, print(workdays.split('?')) returns ['Monday', 'Tuesday', 'Wednesday', 'Thursday', ''], where the last '' is the empty string after removing ?.



```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
```

Similarly, print(workdays.split(“day”)) outputs ['Mon', '?Tues', '?Wednes', '?Thurs', '?']



```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
```

You can think split is to cut the string by pieces.

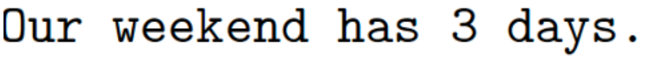
```
long_weekend = "Friday_Saturday_Sunday"
```

print(long_weekend.split('_'))
# output ['Friday', 'Saturday', 'Sunday']

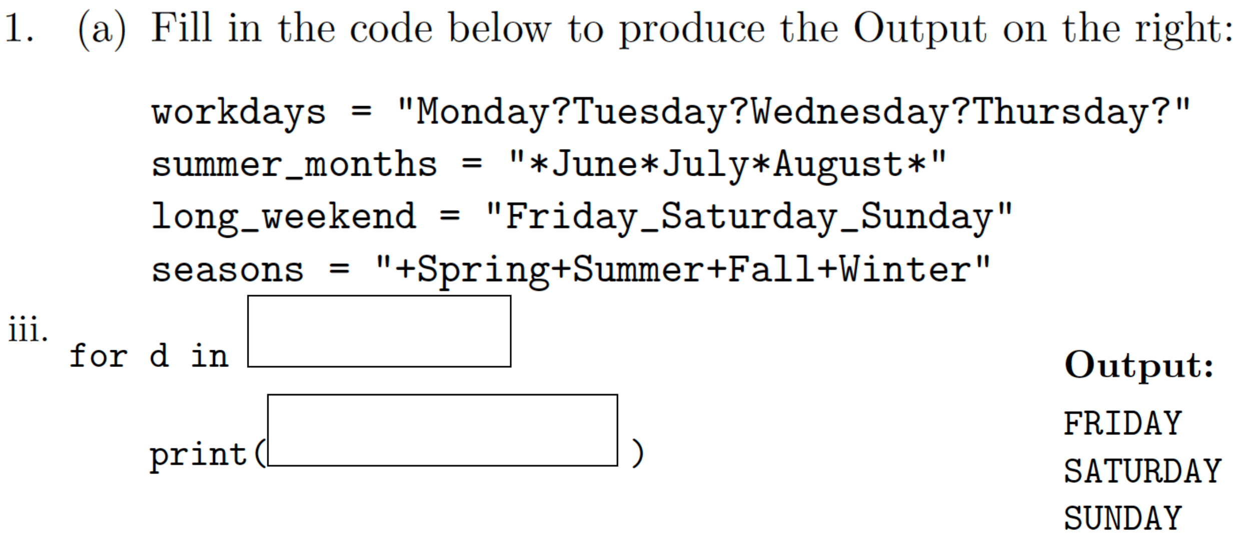
print(long_weekend[:].split('_')) # long_weekend[:] means all the letters in long_weekend.
# output ['Friday', 'Saturday', 'Sunday']

```
days = long_weekend[:].split('_') #save ['Friday', 'Saturday', 'Sunday'] to days
print("Our weekend has", len(days), "days")
```

The above two statements produce



```
Our weekend has 3 days.
```



1. (a) Fill in the code below to produce the Output on the right:

```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
summer_months = "*June*July*August*"
long_weekend = "Friday_Saturday_Sunday"
seasons = "+Spring+Summer+Fall+Winter"
```

iii.
```
for d in □
    print(□)
```

Output:

```
FRIDAY
SATURDAY
SUNDAY
```

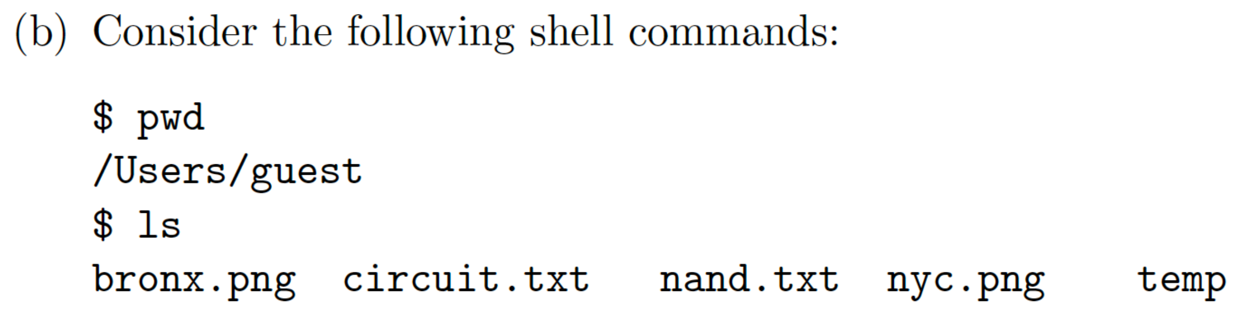
```
for d in days:
  print(d)
```

Code to test the above problem is as follows.

```
workdays = "Monday?Tuesday?Wednesday?Thursday?"
summer_months = "*June*July*August*"
long_weekend = "Friday_Saturday_Sunday"
seasons = "+Spring+Summer+Fall+Winter"

print(seasons[1:7], workdays[7:14])
#cannot replace : by , ie, seasons[1,7] is wrong.
#print(seasons[-25:-19], workdays[7:14]) #also work
```

```
print(workdays.split('?'))
#output ['Monday', 'Tuesday', 'Wednesday', 'Thursday', '']

print(workdays.split('day'))
#output ['Mon', '?Tues', '?Wednes', '?Thurs', '?']

print(long_weekend.split('_'))
# output ['Friday', 'Saturday', 'Sunday']

print(long_weekend[:].split('_'))
# output ['Friday', 'Saturday', 'Sunday']

days = long_weekend[:].split('_')
print("Our weekend has", len(days), "days")

for d in days:
    print(d)
```

Answer to 1 (b)



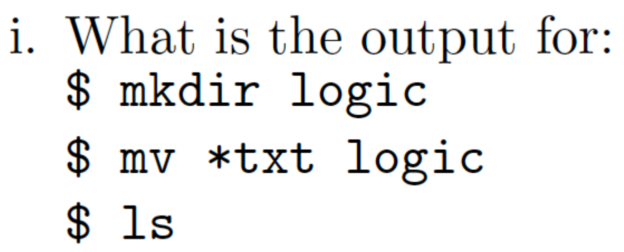
(b) Consider the following shell commands:

```
$ pwd
/Users/guest
$ ls
bronx.png   circuit.txt    nand.txt   nyc.png     temp
```

So here is the structure. The current folder is guest.

```
/ -----
    |__ Users
            |___ guest
                     |____ bronx.png
                     |____ circuit.txt
                     |____ nand.txt
                     |____ nyc.png
                     |____ temp
```

Root directory / has a subdirectory (folder) named Users and Users in turn has a subdirectory guest. Under guest, we have files Bronx.png, circuit.txt, nand.txt, and nyc.png and subdirectory temp. Normally, it is a file name is followed by a suffix, like .png, .txt, .py, and so on, while a folder name does not has a suffix.



i. What is the output for:

```
$ mkdir logic
$ mv *txt logic
$ ls
```

After command **mkdir logic**, we get

```
/ -----
    |__ Users
            |___ guest
                     |____ bronx.png
                     |____ circuit.txt
                     |____ nand.txt
                     |____ nyc.png
                     |____ temp
                     |____ logic
```

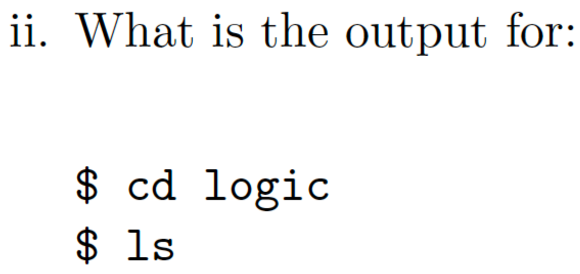
After **mv *.txt logic**, which moves all the files ended with .txt to folder logic, we get

```
/ -----
    |__ Users
            |___ guest
                    |____ bronx.png
                    |____ nyc.png
                    |____ temp
                    |____ logic
                              |____ circuit.txt
                              |____ nand.txt
```

Run command **ls**, which list the contents of the current directory, we get

```
bronx.png  nyc.png  temp  logic
```

Problem 1 (b) ii



ii. What is the output for:

```
$ cd logic
$ ls
```

After **cd logic**, the current directory is logic.

```
/ -----
    |__ Users
            |___ guest
                    |____ bronx.png
                    |____ nyc.png
                    |____ temp
                    |____ logic
                              |____ circuit.txt
                              |____ nand.txt
```

After **ls**, the output is

```
circuit.txt   nand.txt
```

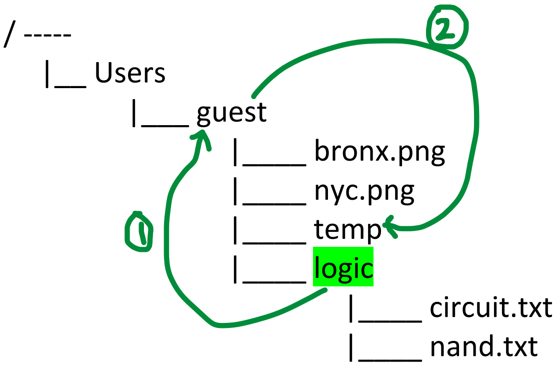
Problem 1 (b) iii

iii. What is the output for:

```
$ cd ../temp
$ pwd
```

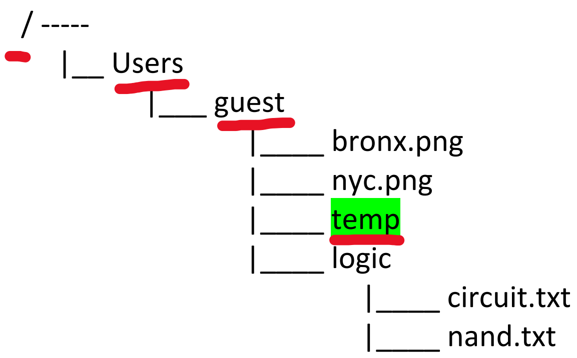
After cd **../temp**, where .. is the parent directory guest of current directory logic. Then move to temp

Before:



```
/ -----
   |__ Users
          |___ guest
                 |____ bronx.png
                 |____ nyc.png
                 |____ temp
                 |____ logic
                         |____ circuit.txt
                         |____ nand.txt
```

After:



```
/ -----
   |__ Users
          |___ guest
                 |____ bronx.png
                 |____ nyc.png
                 |____ temp
                 |____ logic
                         |____ circuit.txt
                         |____ nand.txt
```

Run command **pwd**, which return the path of current directory. That is,

/Users/guest/temp

In terminal of Mac or WSL.

(1) Type in the commands, that is, **contents after %** with return key. Suppose I am in Downloads directory. Type in cd ~ and press enter key. See current directory changes from Downloads to ~.

```
laptopuser@LaptopUsersMBP2 Downloads % cd ~
laptopuser@LaptopUsersMBP2 ~ %
```

(2) Enter pwd and enter return key.

```
laptopuser@LaptopUsersMBP2 ~ % pwd
```

(3) Output the current directory. My user name is laptopuser, your username can be different.

```
/Users/laptopuser
```

(4) Create a guest_dir and move to it. Enter mkdir guest_dir && cd $_ with return key. See the current directory changes from ~ (home directory) to guest_dir.

```
laptopuser@LaptopUsersMBP2 ~ % mkdir guest_dir && cd $_
laptopuser@LaptopUsersMBP2 guest_dir %
```

(5) Command touch is to create an empty file.

```
laptopuser@LaptopUsersMBP2 guest_dir % touch bronx.png
laptopuser@LaptopUsersMBP2 guest_dir % touch circuit.txt
laptopuser@LaptopUsersMBP2 guest_dir % touch nand.txt
laptopuser@LaptopUsersMBP2 guest_dir % touch nyc.png
laptopuser@LaptopUsersMBP2 guest_dir % mkdir temp
laptopuser@LaptopUsersMBP2 guest_dir % ls
bronx.png     nand.txt     temp
circuit.txt   nyc.png
```

(6) Create a directory called logic under current directory. Move all the files ended with .txt to logic.

```
laptopuser@LaptopUsersMBP2 guest_dir % mkdir logic
laptopuser@LaptopUsersMBP2 guest_dir % mv *.txt logic
laptopuser@LaptopUsersMBP2 guest_dir % ls
bronx.png     logic        nyc.png              temp
```

(7) Move to logic and display its contents.

```
laptopuser@LaptopUsersMBP2 guest_dir % cd logic
```

```
laptopuser@LaptopUsersMBP2 logic % ls
circuit.txt         nand.txt
```

(8) Move to temp directory of parent directory. Display path information of temp.

```
laptopuser@LaptopUsersMBP2 logic % cd ../temp
laptopuser@LaptopUsersMBP2 temp % pwd
/Users/laptopuser/guest_dir/temp
```

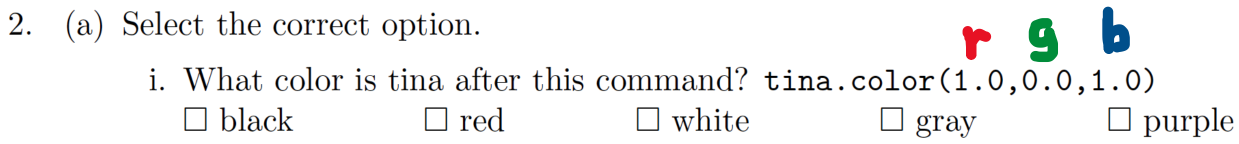
(9) Remove guest_dir if we no longer need it or after we finish the purpose of testing.
We were in temp directory when we type in cd ../.., where .. means parent directory. So we move from temp to its parent directory guest_dir, then move to the parent directory of guest_dir, which is ~ (home directory).

```
laptopuser@LaptopUsersMBP2 temp % cd ../..
laptopuser@LaptopUsersMBP2 ~ % rm -r guest_dir
```

Be very careful of rm command, after running it, the contents cannot be restored. Unlike move to trash, you can still have a chance to recover the contents. After rm command runs successfully, the things removed are gone. Option -r means recursion. This is useful when you want to remove a non-empty directory, but again, **double check before you press the return key after a rm command.**

Problem 2

Problem 2 (a) i



2. (a) Select the correct option.

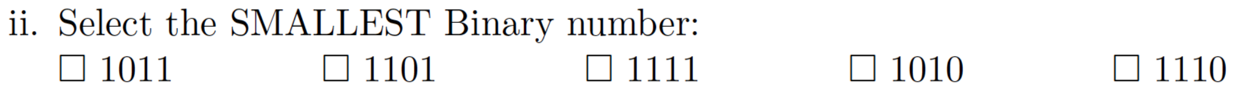
r g b

i. What color is tina after this command? `tina.color(1.0,0.0,1.0)`

□ black □ red □ white □ gray □ purple

(1.0, 0.0, 1.0) means red component is 1, green component is 0, and blue component is 1. Red and blue together makes purple.

Problem 2 (a) ii



ii. Select the SMALLEST Binary number:

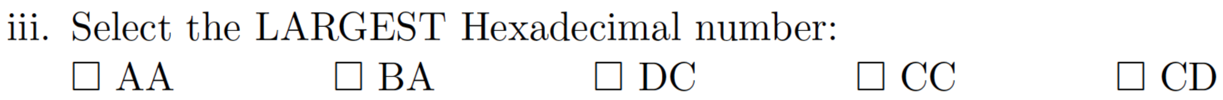
□ 1011 □ 1101 □ 1111 □ 1010 □ 1110

(1) We are talking about unsigned number. Compare the leftmost (most significant) bit, every number has that bit as 1.
(2) Move to the second most significant digit, ie, 2nd digit from left. Since we are looking for the smallest number, only the one with 0 are possible candidates. We have 1011 and 1010 left.
(3) Compare 1011 and 1011. The third bit (the third most significant bit) from left is the same.
(4) Move to the least significant bit, ie, the rightmost bit. The smallest number is 1010.

It is like to compare number in decimal system, compare the digits in the most significant digit to the least significant one. For example,

(a) 123 is smaller than 200, since the hundred digit 1 in 123 is smaller than hundred digit 2 in 200.
(b) 123 is smaller than 139. With the same hundreds digit, tens digit of 123 is smaller than the tens digit 139.
(c) 123 is smaller than 125. With the same hundreds and tens digits, compare ones digit.

Problem 2 (a) iii



iii. Select the LARGEST Hexadecimal number:

□ AA □ BA □ DC □ CC □ CD

Hexadecimal number is similar to decimal number, the only difference hexadecimal digit is 0, 1, …, 9, A (equivalent to 10), B (equivalent to 11), …, F (equivalent to 15).

Compare the most significant digit, The largest one is D. The answer is DC. It is like decimal number 70 is larger than 69 since its larger most significant digit.

Problem 2 (a) iv

iv. What is the binary number equivalent to decimal 14?

☐ 1011 ☐ 1101 ☐ 1111 ☐ 1010 ☐ 1110

(1) Divide 14 by 2, the quotient is 7 and the remainder is 0.

```
2  |  14
  +-------
      7       0
```

(2) Divide 7 by 2, the quotient is 3 and the remainder is 1. It is like to divide 7 pens among 2 students, each student get 3 pens, and there is one pen left.

```
2  |  14
  +-------
  2 | 7       0
    +---
      3       1
```

(3) Divide 3 by 2, the quotient is 1 and the remainder is 1. In fact, if the number is odd, when it is divided by 2, the remainder is 1, otherwise, the remainder is 0.

```
2  |  14
  +-------
  2 | 7       0
    +---
  2 | 3       1
    +----
      1       1
```

(4) Divide 1 by 2, the quotient is 0 and the remainder is 1.

```
 2  |  14
  +-------
  2 | 7       0   ↑
    +---          |
  2 | 3       1   |
    +----         |
   2 | 1      1   |
     +----        |
      0       1   |
```

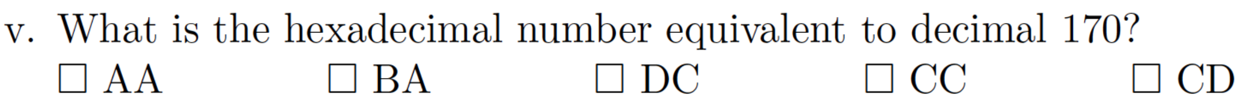
(5) When the quotient is 0, we can stop. And string the remainders backwards. The answer is 1110.

(6) Quick verify (optional): binary number 1110 is decimal number 14.

exponent	3	2	1	0
rank	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Binary number	1	1	1	0
rank * bit when bit is not zero	1 * 8	1*4	1*2	

Add 1*8 + 1*4 + 1*2 = 14.

Problem 2 (a) v



v. What is the hexadecimal number equivalent to decimal 170?

□ AA □ BA □ DC □ CC □ CD

Answer: same as above, the only different is that the base of hexadecimal number is 16.

(1) Divide 170 by 16, the quotient is 10, the remainder is 10, which is A in hexadecimal system.

```
16 | 170
   +--------
        10          10 = A16
```

(2) Divide 10 by 16, the quotient is 0, the remainder is 10, which is A.

```
16 | 170
   +--------
   16 | 10          10 = A16       ↑
      +------
          0          10  = A16
```

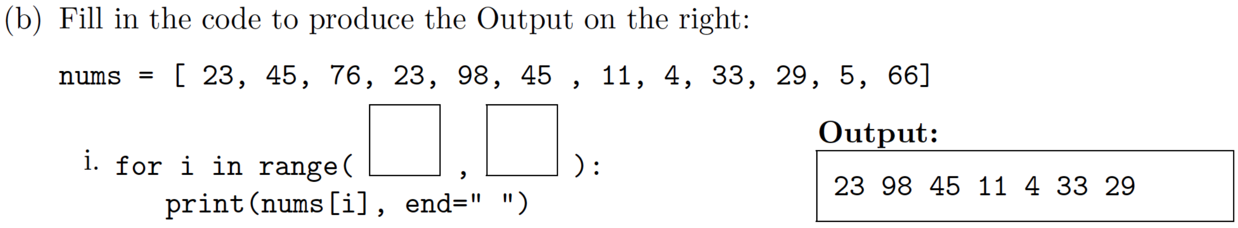
(3) Stop when quotient is 0. String remainders backwards. We get AA_{16}, which is equivalent to 170 in decimal system.

(4) Verify (optional): hexadecimal number AA is equivalent to 170 in decimal system.

exponent	1	0
rank	$16^1 = 16$	$16^0 = 1$
Hexadecimal number	A	A
Multiple rank and digit	Hexadecimal A is same as 10 in decimal system, 10 * 16	10 * 1

Add 10 * 16 + 10 * 1 = 170.

Problem 2 b (i)



(b) Fill in the code to produce the Output on the right:

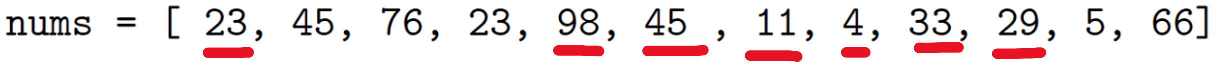
```
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]

i. for i in range( □ , □ ):
       print(nums[i], end=" ")
```

Output:

```
23 98 45 11 4 33 29
```

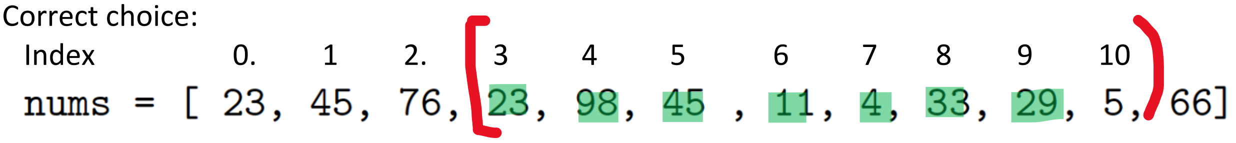
Answer:
There are two occurrences of 23. If we choose the first one, the indices are not evenly spaces so we cannot use range function.

Wrong choice:



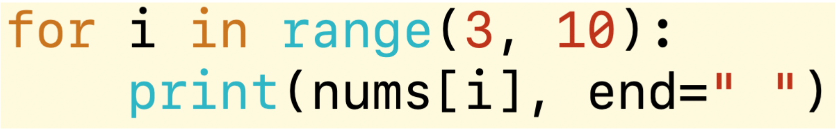
```
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]
```

Correct choice:



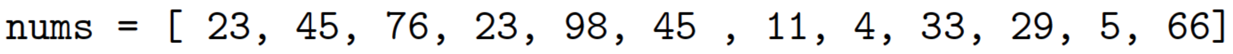
```
Index         0.    1    2.    3    4    5     6    7   8    9   10
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]
```

The answer is as follows. Note that element at index 3 is included, but element at index 10 is not. Also 10 – 3 = 7, which means this range includes 7 elements. Is that pretty?

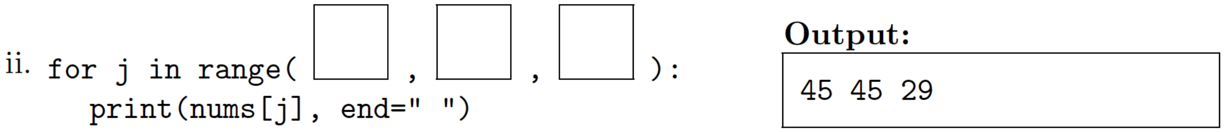


```
for i in range(3, 10):
    print(nums[i], end=" ")
```

Problem 2 (b) ii



```
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]
```

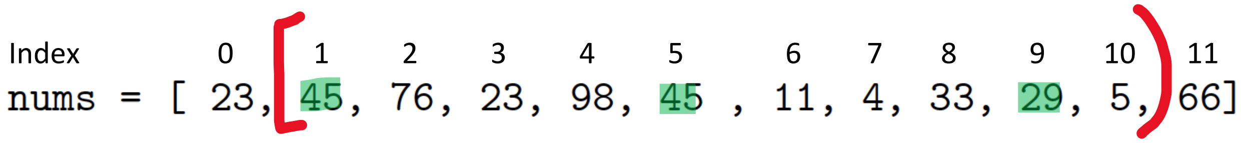


```
ii. for j in range( □ , □ , □ ):
        print(nums[j], end=" ")
```

Output:

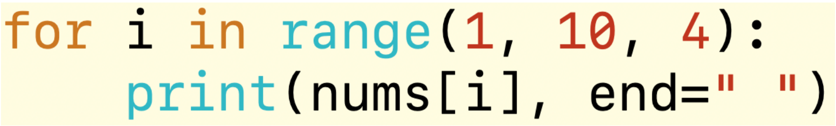
```
45 45 29
```

Answer:



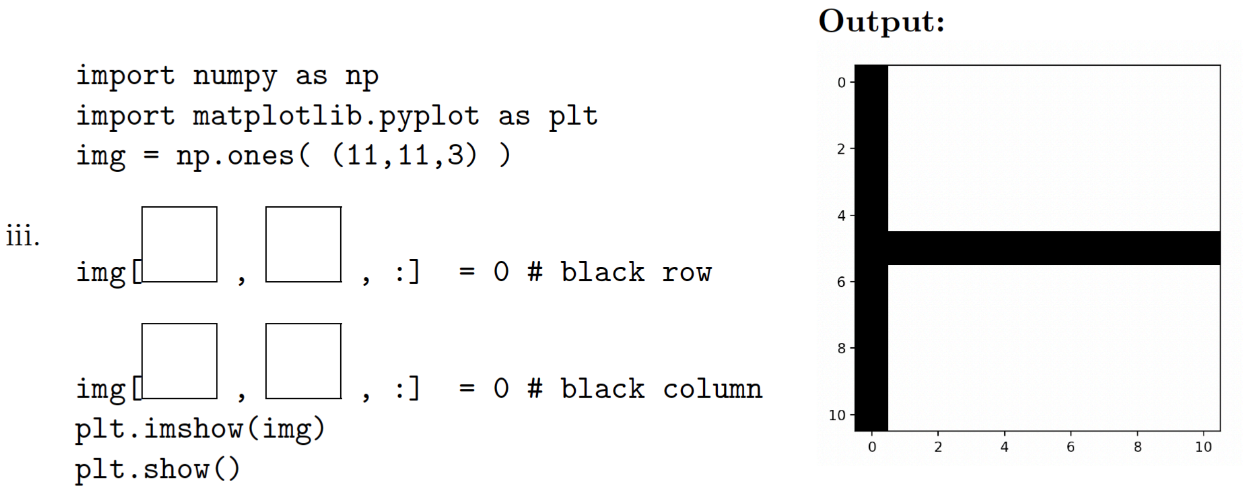
```
Index      0    1    2    3    4    5     6    7   8    9   10   11
nums = [ 23, 45, 76, 23, 98, 45 , 11, 4, 33, 29, 5, 66]
```

Use (start, end, step) version of range. Note that the items we selected are indexed at 1, 5, 9, respectively. Said differently, start at 1, end at 10, increase by 4 each time.



```
for i in range(1, 10, 4):
    print(nums[i], end=" ")
```

Problem 2 b (iii)



```
import numpy as np
import matplotlib.pyplot as plt
img = np.ones( (11,11,3) )
```

iii.

```
img[□ , □ , :]  = 0 # black row

img[□ , □ , :]  = 0 # black column
plt.imshow(img)
plt.show()
```

Output:

Explanation:

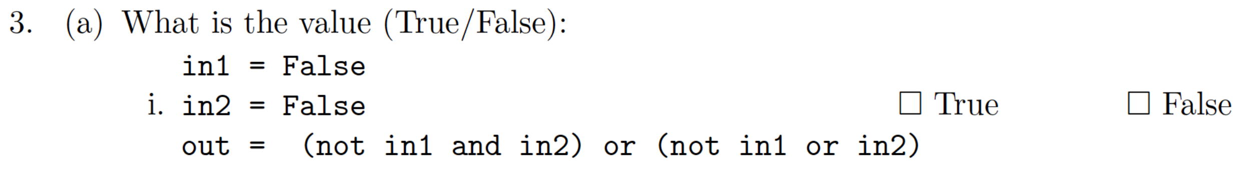
(1) For the horizontal line, row is indexed at 5, column index is all, so use img[5, :, :]. The third dimension is rgb (red, green, blue) channels, use : means choose all of them. When r, g, b are all zeros, the color is black.

(2) For the vertical line, row index is all and column index is 0, so use img[:, 0, :].

Complete code is as follows.

```
import matplotlib.pyplot as plt
import numpy as np

img = np.ones((11, 11, 3))
img[5, :, :] = 0 #set row
img[:, 0, :] = 0 #set column

plt.imshow(img)
plt.show()
```

Problem 3 (a) i



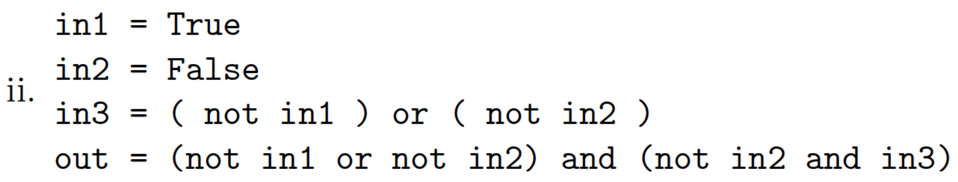
3. (a) What is the value (True/False):

i.
```
in1 = False
in2 = False
out =  (not in1 and in2) or (not in1 or in2)
```
□ True □ False

When in1 is False and in2 is False, we have

(1) not in1 as True
(2) in2 as False
(3) Then (not in1 and in2) is (True and False), which is False.
(4) Also (not in1 or in2) is (True or False), which is True.
(5) (not in1 and in2) or (not in1 or in2) is False or True, which is True.

Problem 3 (a) ii



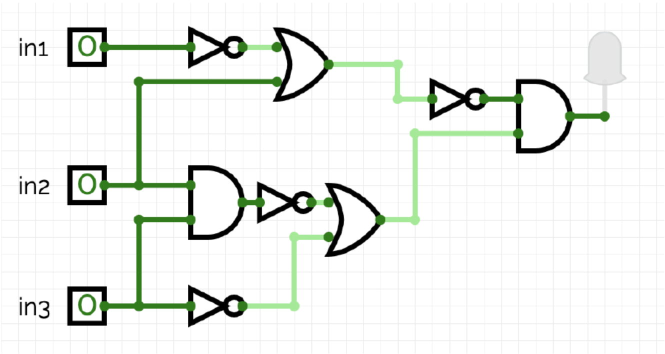
ii.
```
in1 = True
in2 = False
in3 = ( not in1 ) or ( not in2 )
out = (not in1 or not in2) and (not in2 and in3)
```

When in1 is True, in2 is False

(1) not in1 is False
(2) not in2 is True
(3) in3 = (not in1) or (not in2) is False or True, which is True.
(4) out = (not in1 or not in2) and (not in2 and in3)
    (4a) (not in1 or not in2) is (False or True), which is True
    (4b) (not in2 and in3) is (True and True) which is True.
    Remember, NOT has higher precedence than AND, which in turn has higher precedence than OR. So we will run not in2 first, before we use the result of not in2 to and with in3.

    so True and True is True.

Problem 3 (a) iii

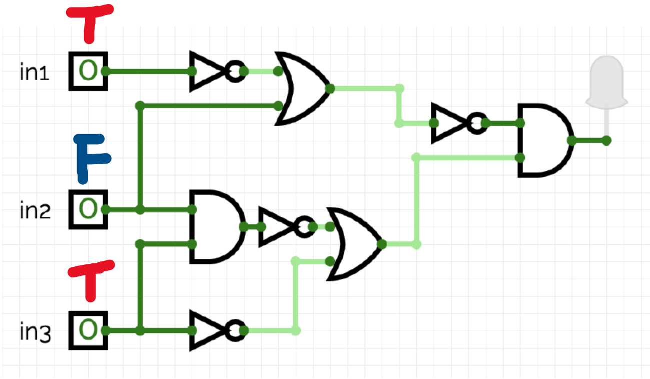


iii.

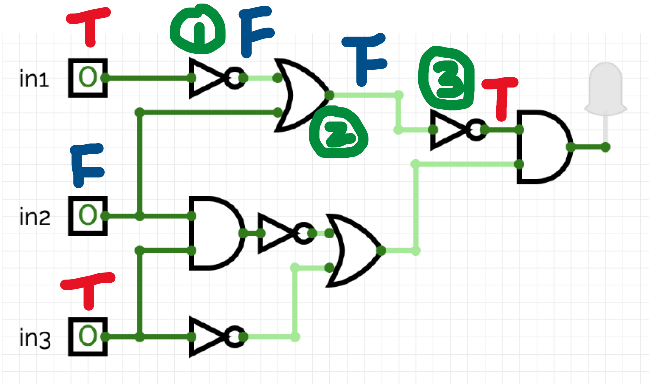
```
in1 = True
in2 = False
in3 = True
```

□ True □ False

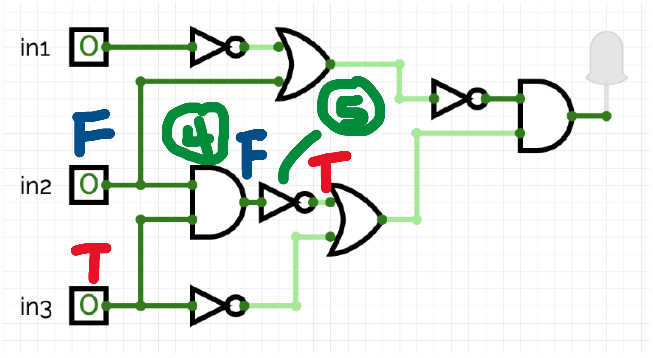
Answer:



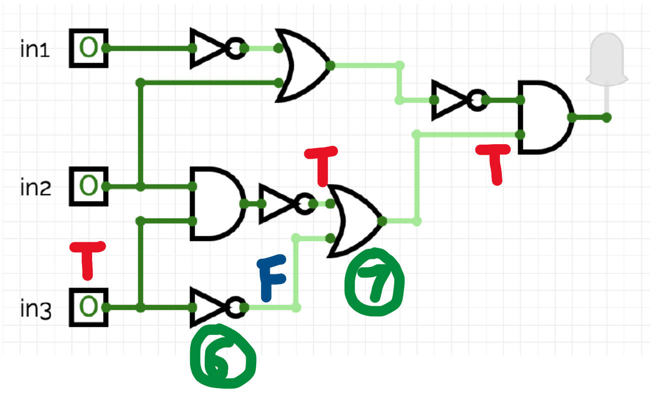
(1) Then not in1 is F, and F or F (for the top OR door) is F, and not F (for the rightmost NOT door) is T.



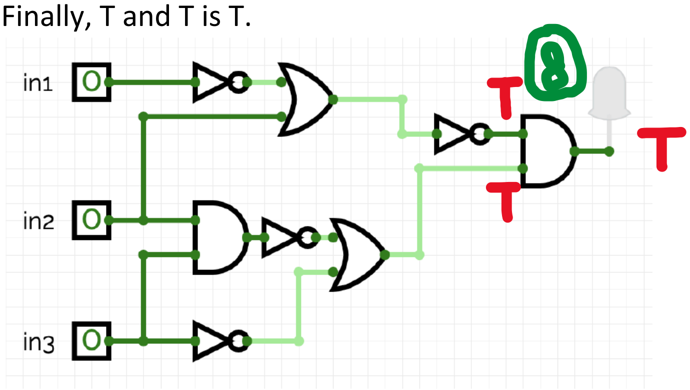
(2) Next, in the left AND door, F and T is F. In the middle row NOT door, not F is T.



(3) Third, in the bottom NOT door, not T is F. In the bottom OR door, T or F is T.



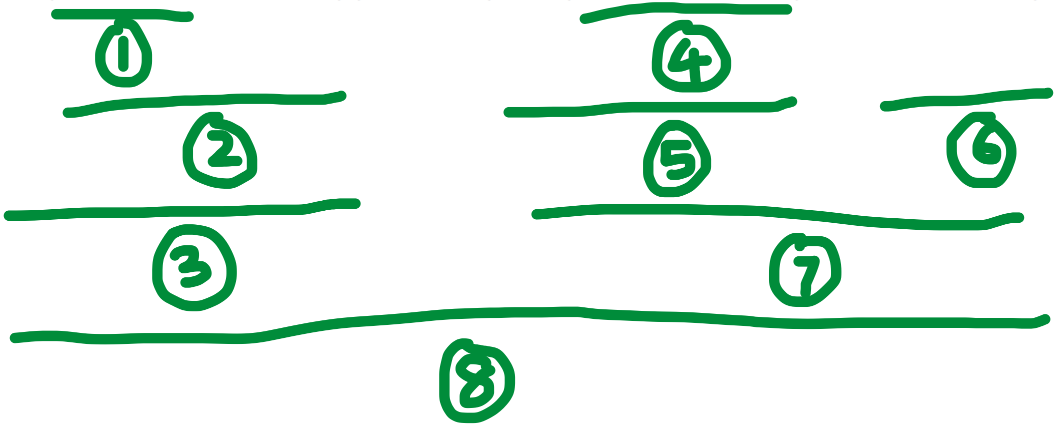
(4) Finally, T and T is T.



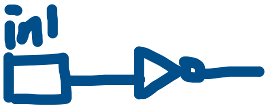
Problem 3 (b)

(b) Draw a circuit that implements the logical expression:

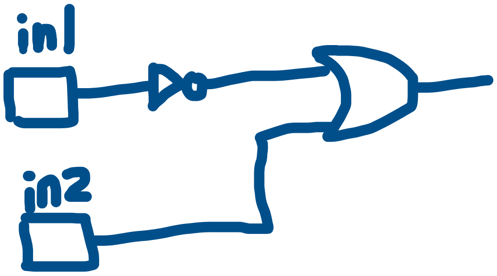
```
(not(not in1 or in2)) and (not(in2 and in3) or not in3)
```



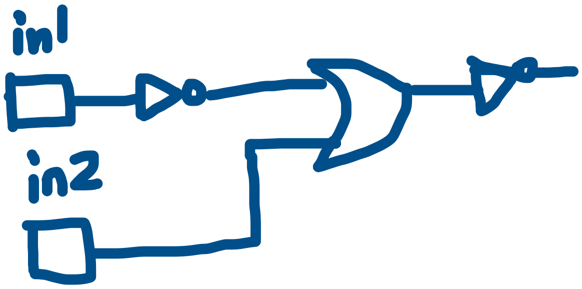
(1) Step 1: not in1



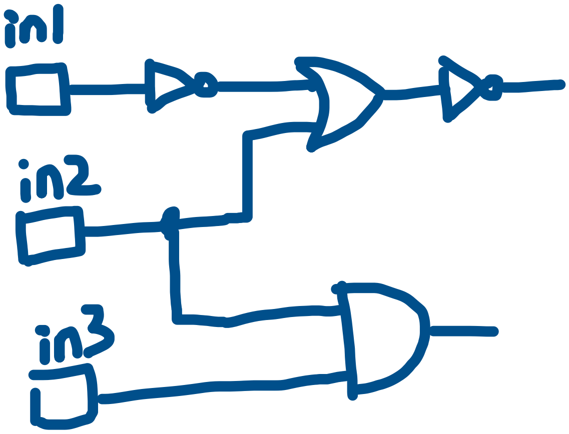
(2) Step 2: not in1 or in2



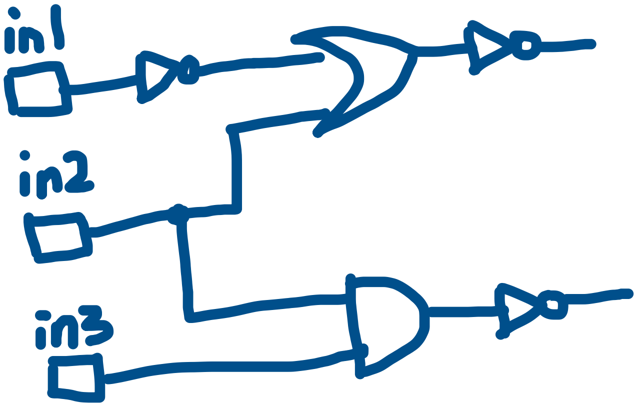
(3) Step 3: (not (not in1 or in2))



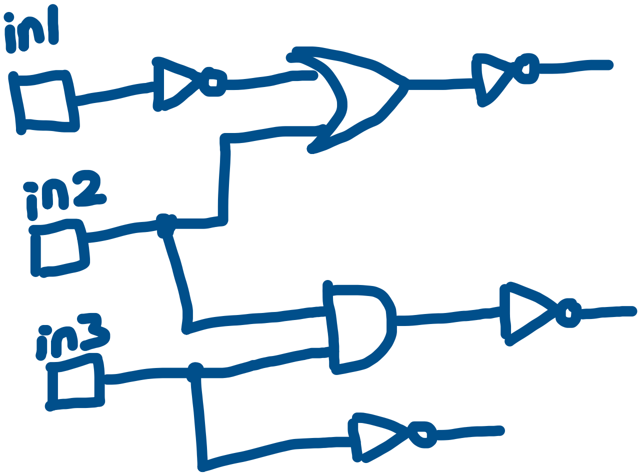
(4) Step 4: in2 and in3



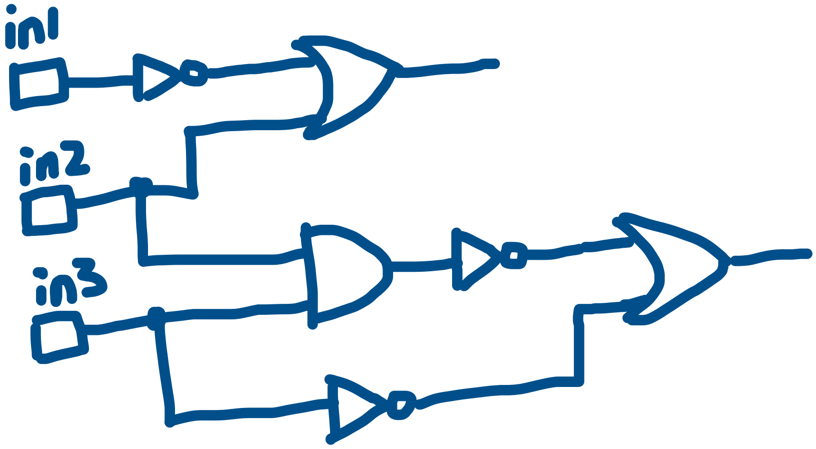
(5) Step 5: not (in2 and in3)



(6) Step 6: not in3



(7) Step 7: not (in2 and in3) or not in3



(8) Step 8: (not(not in1 or in2)) and (not(in2 and in3) or not in3)