Boolean operators can seem tricky at first, and it takes practice to evaluate them correctly. Write the value (True or False) produced by each expression below, using the assigned values of the variables a, b, and c. Try to do this without using your interpreter, but you should check yourself when you think you've got it. Hint: Work from the inside out, starting with the inner-most expressions, like in arithmetic.

a = False
b = True = 1
c = False = 0

and = multiply
or = add

1. b and c

b = True
c = False
1 · 0 = 0 = False

2. b or c

true + false
1 + 0 = true

3. not a and b

not False
↓
True · True = 1 · 1 = 1 ⇒ True

4. (a and b) or not c

(0 · 1) + (1)
0 + 1 → True = True
False

5. not b and not (a or c)

false · (true + true)
0 · (1 + 1) = 0 False

✓

It is important that we know the type of the values stored in a variable so that we can use the correct operators (as we have already seen!). Python automatically infers the type from the value you assign to the variable. Write down the type of the values stored in each of the variables below. Pay special attention to punctuation: values are not always the type they seem!

1. `a = False` Boolean ✓

2. `b = 3.7` float ✓

3. `c = 'Alex'` String ✓

4. `d = 7` integer ✓

5. `e = 'True'` string ✓

6. `f = 17` integer ✓

7. `g = '17'` String ✓

8. `h = True` Boolean ✓

9. `i = '3.14159'` String ✓

✓ For each of the following fragments of code, write what the output would be. Again, do this without running the code (although feel free to check yourself when you're done).

1.
```
num = 10
while num > 3:
    print num
    num = num - 1
```

✓

starts at 10

satisfies (>3 condition)

prints value

subtracts 1

repeats

2.
```
divisor = 2
for i in range(0, 10, 2):
     print i/divisor
```

✓

start ↓ (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) ↓ ends after 10 integers

skip 2 : (0, 2, 4, 6, 8)

divided by 2

[0, 1, 2, 3, 4] output

=> ~~(1, 3, 5, 7~~, 9)

output ~~(0.5, 1.5, 2.5, 3.5, 4.5)~~

3.
```
num = 10 while True:
    if num < 7:
        break
    print num
    num -= 1
```

✓

10
9
8
7

Starts at 10

if 10 < 7:

stops

if 10 > 7

prints (10)

then go to 10-1 = 9

if 9 < 7

stops

:

4.
```
count = 0
for letter in 'Show!':
    print 'Letter #', count, 'is', letter
    count += 1
```

0

1 2 3 4

✓

letter # 0 is S , letter # 1 is n , letter # 2 is o

letter # 3 is w , letter # 4 is !

Consider the following program that Ben Bitdiddle handed in to the course staff (again, try to do this exercise without running the code!):

```
n = 10
i = 10


while i > 0:
     print i
     if i % 2 == 0:
          i = i / 2
     else:
          i = i + 1
```

What do you think this code is doing? Without comments it is hard to guess what Ben's intention was (*cough* this is why the staff loves to look at commented code!!), so read through it and make a sensible guess as to what it is doing. There's a lot of mistakes in the code so your guess is as good as ours!

1. Draw a table that shows the value of the variables n and i during the execution of the program. Your table should contain two columns (one for each variable) and one row for each iteration. For each row in the table, write down the values of the variables as they would be at the line containing the print statement.

n	i
10	10
10	5
10	6
10	3
10	4
10	2

??

2. Ben made a lot of mistakes. State what you think Ben was trying to do and suggest one or more ways he could fix his code (there's a few good answers for this depending on what you think the code should be doing).

Maybe find even and odds?

Conditionals

The purpose of this exercise is to understand conditionals. Tiberius is looking for his dream job, but has some restrictions. He loves California and would take a job there if it paid over 40,000 a year. He hates Massachusetts and demands at least 100,000 to work there. Any other place he's content to work for 60,000 a year, unless he can work in space in which case he would work for free. The following code shows his basic strategy for evaluating a job offer.

```
pay = _____
location = _____
```

← that's where you put values that you wanna test.

```
if location == "U.S.S. Enterprise":"
    print("So long, suckers! I’ll take it!")
elif location == "Massachusetts":
    if pay < 100000:
       print("No way")
    else:
       print("I’ll take it!")
elif location == "California" and pay > 40000:
    print("I’ll take it!")
elif pay > 60000:
    print("I’ll take it!")
else:
    print("No thanks, I can find something better.")
```

For each of the following job offers, write down the output that would be generated. Do this without running the code. It is an important skill to be able to understand what a piece of code does without running it.

1.
```
location = "Massachusetts"
pay = 50000
```

No way

2.
```
location = "Iowa"
pay = 50000
```

No thanks, I can find something better.

3.
```
location = "California"
pay = 50000
```

I'll take it!

4.
```
location = "U.S.S. Enterprise"
pay = 1
```

So long, suckers! I'll take it!

5.
```
location = "California"
pay = 25000
```

No thanks, I can find something better.

Finding Bugs

The following set of instructions were given to Bonnie Bitdiddle, and she produced the code below. Find at least four bugs she made, and say how to fix them.

Instructions: Write a `quotient` function that inputs two integers, and then returns a list with the integer division answer and remainder. Also write a `large_num` function that takes a number, and returns `True` if that number is bigger than 10000, and `False` otherwise. Additionally, write some code to test your functions.

```
def quotient(int_1, int2):
    return (int_1/int_2, int_1*int_2)

def large_num(num):
    res = (num > 10000);

quotient(a, b)
quotient_ab = num
print('a:', a, 'b:', b, 'quotient_ab:', quotient_ab)
big = large_num(b)
print 'b is big:', big
```

% → For remainde

double for quotient

do an if statement.

~~return (True)~~

5 2

5/2

Bugs:

1. Remainder equationn

2. define num / a, b

3. return ~~[illegible]~~ missing.

$((-\sqrt{x})^n$

4. paranthasies print ( )

Enter the code into a python notebook and "complete the assignment"

Mystery Program

Bonnie next turned in the following uncommented code. Help us figure out what it does!

```
1 print("Think of a number between 1 and 100, but don’t tell me what you choose.")
2 min_n=1
3 max_n = 100
4 right_answer = False
5
6  while not right_answer:
7        mid_n = (max_n + min_n + 1)//2
8        answer = ~~raw_~~input(’Is it ’ + str(mid_n) + ’? ’)
9        if answer[0] == ’y’:
10           right_answer = True
11       elif answer.startswith(’higher’):
12           min_n = mid_n + 1
13       elif answer.startswith(’lower’):
14           max_n = mid_n - 1
15       else:
16       print("Sorry, I don’t understand your answer.")
17
18 print(’Woohoo! I got it!’)
```

Starts at false, till proves oposite

→integer with no decimal 5/2=2 not 2,5

1. The while loop exits when the variable right_answer is True. What will cause right_answer to be true?

   answer[0] == 'y' → right answer = True

2. How many times will the program print out ’Woohoo! I got it!’?

   ~~[illegible]~~ ① Once when answer is 'y'

   ~~[illegible]~~ ② once when it finds right answer

3. What are we using the variable answer for?

   to make the first guess ('is it' + equation + '?')

4. The program makes a guess in line 8. What user responses will be understood by the program after it makes its guess?

   ① y, ② higher, ③ lower

   anything that starts with the code will understand

5. If the program gets the response ’higher’, what does that tell it about its guess?

   it's a wrong guess, set a new min → max and take mid

6. What are the variables min_n, max_n and mid_n used for?

   limit the possibilites for the answer ~~and~~ make a guess.

This is an example of **binary search**, a simple but important algorithm in computer science. If you’re curious, or confused, read the Wikipedia article on binary search to find out more and get a good explanation of what’s going on here.