Python Chapter 4: Decision Structures and Boolean Logic

This guide has been written to reinforce your learning with the actual chapter lessons and not to serve as alternative way of learning python. I have tried my best to explain and provide explanations, but if you find anything that needs to be corrected, please inform me.

- Maruthi Basava

BEFORE YOU START WRITING CODE, PLEASE BE AWARE THAT WHITESPACES MATTER IN PYTHON. LEAVING STRAY SPACES IN VARIOUS AREAS COULD CRASH YOUR PROGRAM. PLEASE MAKE SURE THAT THE SYNTAX IS CORRECT.

Relational Operators and the if statements.

Correlates to Assignments (Programs 4.1-4.2)

Programming is basically a set of instructions for computers to understand and to perform. When we want our programs to have a certain level of logic (such as figuring out whether variable a is bigger than variable c), we need to use relational operators and if statements.

If statements are just what you think they are, a statement that just questions things.

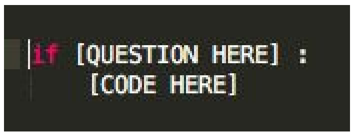
For example:

If a is bigger than c then I want to say “hi”.

Do you see the pattern that follows it?

If … Then…

In english, If you were to ask the question “is A bigger than C?”, you would get a YES or NO answer. Python follows the same pattern.

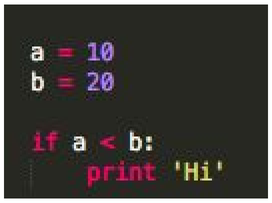


```
if [QUESTION HERE] :
    [CODE HERE]
```

Make sure that you maintain that indent after the colon.

An If statement must answer a yes or no questions, otherwise it wouldn’t work.

Take a look at the example below:



```
a = 10
b = 20

if a < b:
    print 'Hi'
```

We made two variables (a and b) and set them equal to whatever values.
We made an if statement
Then we checked if a is LESS than b, if it is then print out ‘Hi’.

Take a look at this chart.

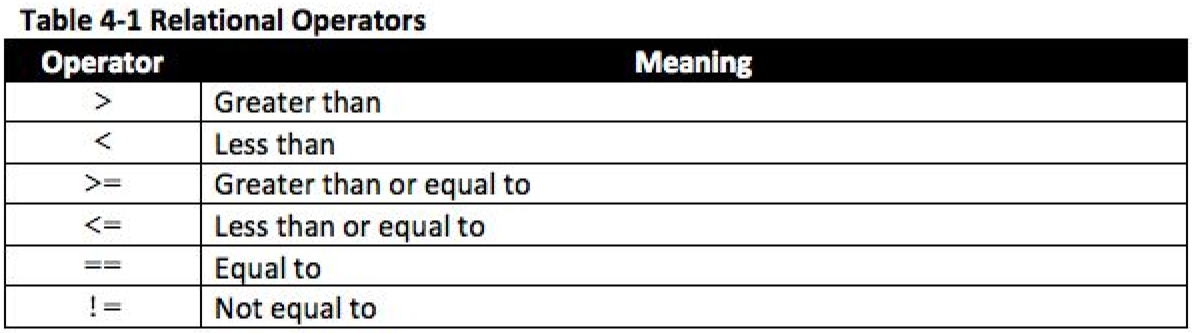
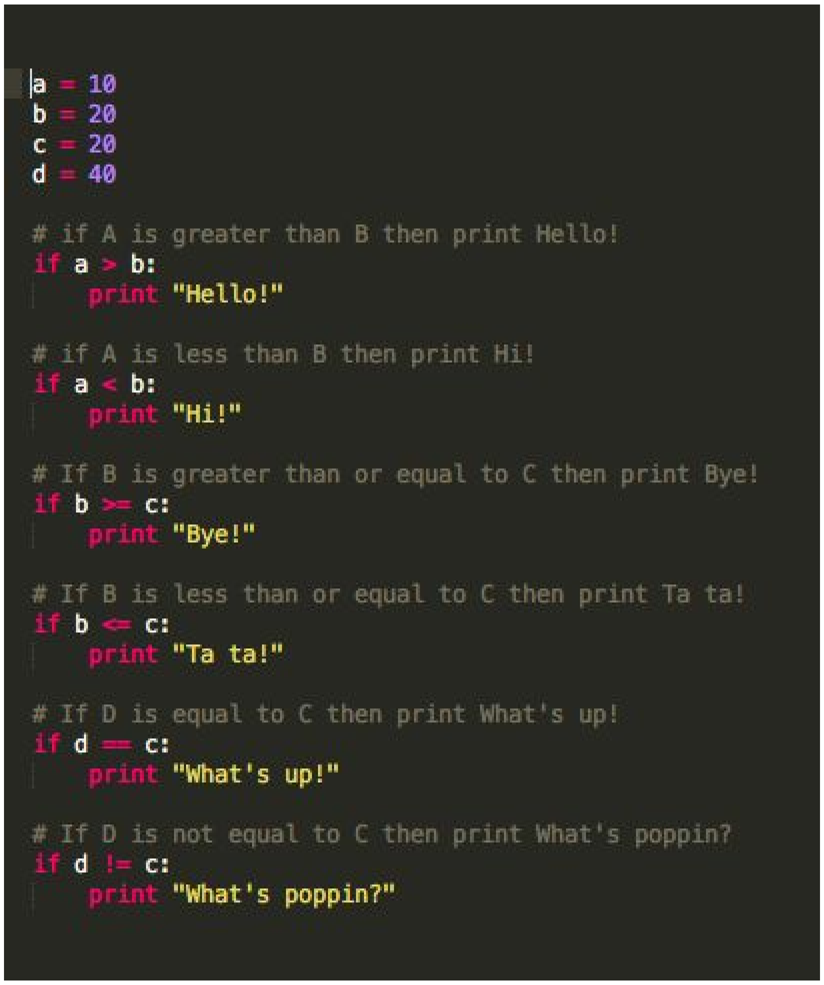


Table 4-1 Relational Operators

Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

It lists every single relational operator.
These operators just compare the values and return a true or false value.

This code below shows how you can use them.



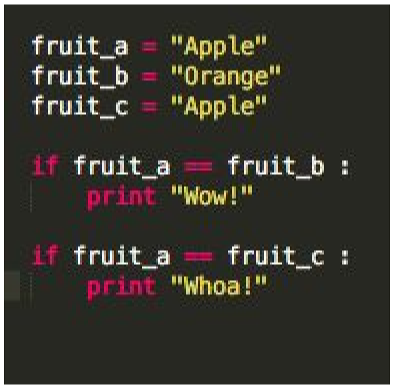
```
a = 10
b = 20
c = 20
d = 40

# if A is greater than B then print Hello!
if a > b:
    print "Hello!"

# if A is less than B then print Hi!
if a < b:
    print "Hi!"

# If B is greater than or equal to C then print Bye!
if b >= c:
    print "Bye!"

# If B is less than or equal to C then print Ta ta!
if b <= c:
    print "Ta ta!"

# If D is equal to C then print What's up!
if d == c:
    print "What's up!"

# If D is not equal to C then print What's poppin?
if d != c:
    print "What's poppin?"
```

Ok now we learned how to compare numbers, what do you do when you want to compare strings?

Correlates to Assignments (Programs 4.3)

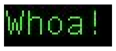
You can use == to compare two strings just like numbers.

For example:



```
fruit_a = "Apple"
fruit_b = "Orange"
fruit_c = "Apple"

if fruit_a == fruit_b :
    print "Wow!"

if fruit_a == fruit_c :
    print "Whoa!"
```

Terminal:

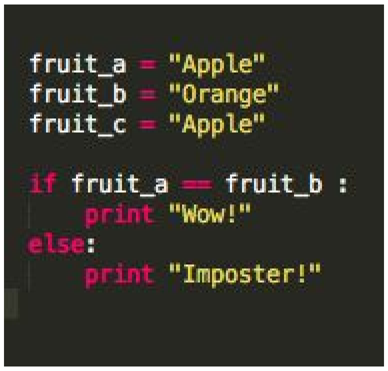


```
Whoa!
```

When we asked whether fruit_a is equal to fruit_b, nothing happened
But when asked whether fruit_a is equal to fruit_c, then “Whoa!” appeared in the terminal.

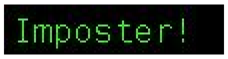
The If statement also has another part, the else.

It would be written like this:



```
fruit_a = "Apple"
fruit_b = "Orange"
fruit_c = "Apple"

if fruit_a == fruit_b :
    print "Wow!"
else:
    print "Imposter!"
```

And in the terminal:



```
Imposter!
```

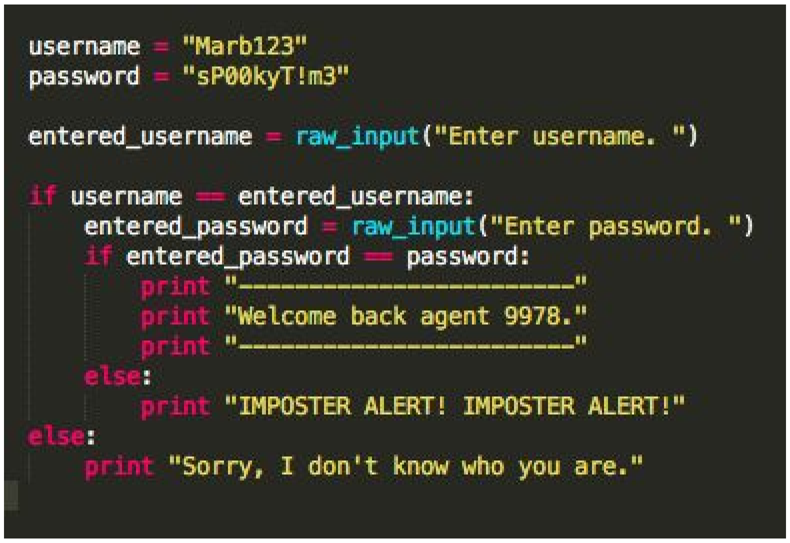
Now you can tell python what to do if the question returns a false.

Nested If and else Statements:

Correlates to Assignments (Programs 4.4 - 4.5)

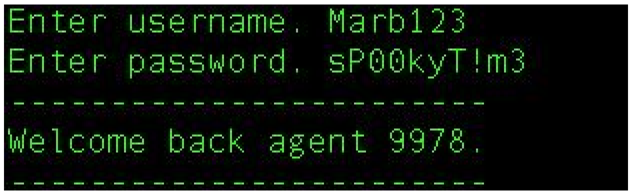
You can put if statements inside of if statements or else statements.

Take a look at this program here.



```
username = "Marb123"
password = "sP00kyT!m3"

entered_username = raw_input("Enter username. ")

if username == entered_username:
    entered_password = raw_input("Enter password. ")
    if entered_password == password:
        print "-------------------------"
        print "Welcome back agent 9978."
        print "-------------------------"
    else:
        print "IMPOSTER ALERT! IMPOSTER ALERT!"
else:
    print "Sorry, I don't know who you are."
```

Terminal:



```
Enter username. Marb123
Enter password. sP00kyT!m3
-------------------------
Welcome back agent 9978.
-------------------------
```

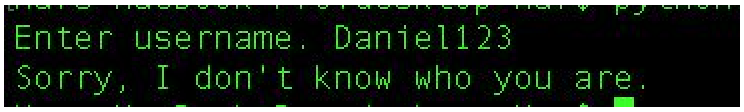
Program Explanation:

1. So first we create two variables, username and password, with their respective values.
2. Then we create another another variable, entered_username, which will be collected from raw_input function.
3. We create an if statement to see if the entered username and the actual username match, if they do then the code inside runs, if it doesn't match, then it prints "Sorry, I don't know who you are. "
4. When the username matches, it goes inside.
5. Once inside, we create a entered_password variable that stores the value from the raw_input

6. Then we create an if statement to compare the two password values, if it works, then it proceeds to the code inside, otherwise it will print out “IMPOSTER ALERT! IMPOSTER ALERT!”.
7. Once inside, It nicely prints out “Welcome back agent 9978.”

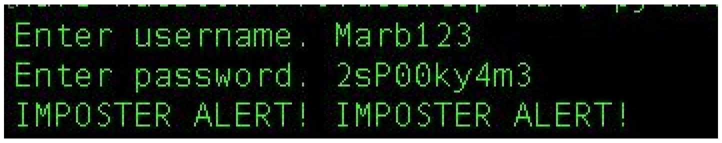
What happens when we enter in the wrong information?

Wrong Username:



```
Enter username. Daniel123
Sorry, I don't know who you are.
```

Right Username, Wrong Password:

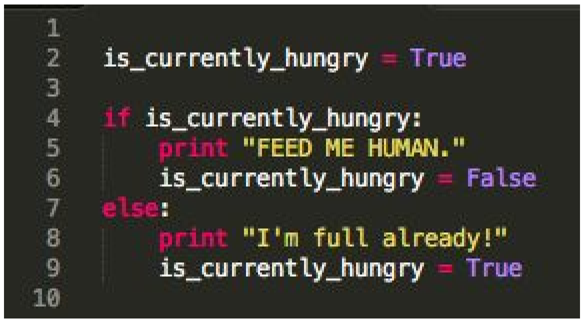


```
Enter username. Marb123
Enter password. 2sP00ky4m3
IMPOSTER ALERT! IMPOSTER ALERT!
```

Booleans:

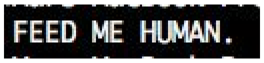
Booleans are another type of variables that can store TRUE or FALSE.

Here we have a program that tells us whether our pet needs to be fed:



```python

is_currently_hungry = True

if is_currently_hungry:
    print "FEED ME HUMAN."
    is_currently_hungry = False
else:
    print "I'm full already!"
    is_currently_hungry = True

```

Terminal:



```
FEED ME HUMAN.
```

So what did we do here?

1. First we created a variable is_currently_hungry and set it to True (meaning that our pet is hungry)
2. Then we made a **If** and **else** statement that checked if **is_currently_hungry** is **True**, and if it is, then it would **print** out **“FEED ME HUMAN.”** and then set **is_currently_hungry** to

false. If **is_currently_hungry** is **False**, then it would move into the **else** code block then **print** out **"I'm full already!"** and **set is_currently_hungry** to **True**.

Logical Operators:

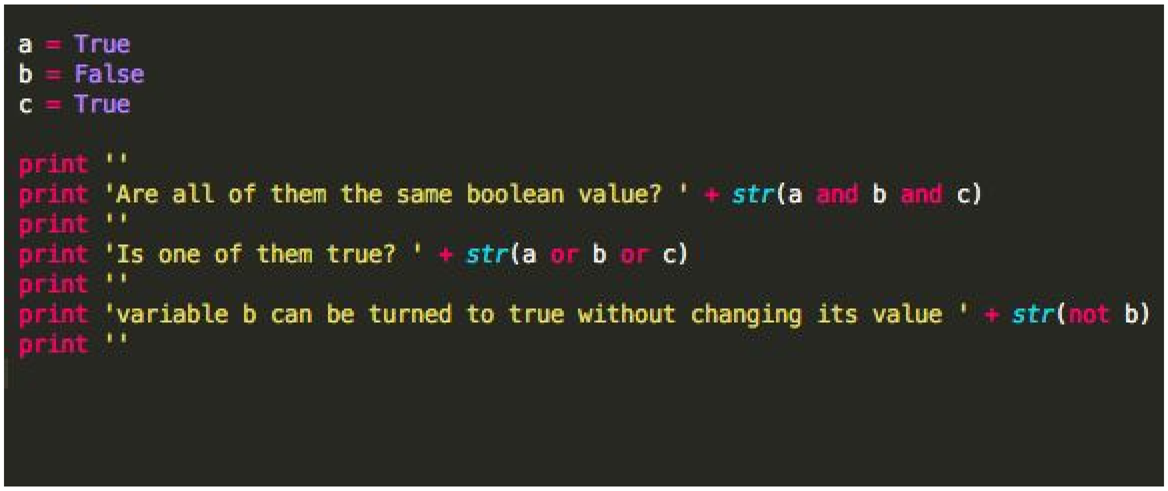
There are three logical operators:

and - connects multiple boolean statements and checks if **ALL** are true or false.

or - ** connects multiple boolean statements and checks for the **FIRST true

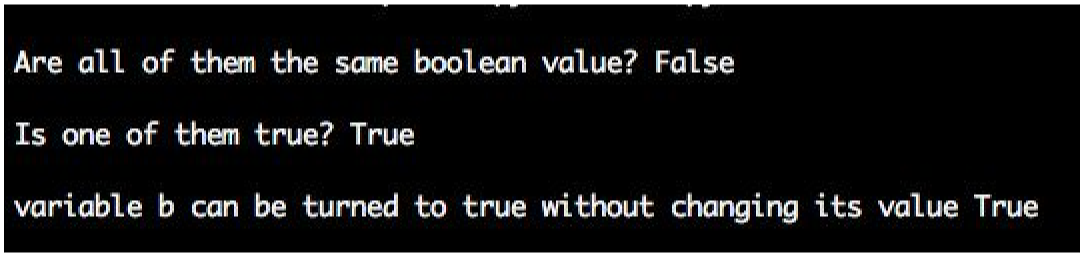
not - turns a true into a false, a false into a true. It's the opposite.

Example:



```
a = True
b = False
c = True

print ''
print 'Are all of them the same boolean value? ' + str(a and b and c)
print ''
print 'Is one of them true? ' + str(a or b or c)
print ''
print 'variable b can be turned to true without changing its value ' + str(not b)
print ''
```

Terminal:



```
Are all of them the same boolean value? False

Is one of them true? True

variable b can be turned to true without changing its value True
```

Let’s reflect on what we have done here.

1. We made three variables (**a**, **b**, and **c**) and set them to whatever boolean values.
2. Then for aesthetic purposes (for the sake of the readability in the terminal), we printed blank strings to space out the results.
3. The second print statement printed the string along with the boolean (REMEMBER: boolean is a different type of variable than strings, so you have to use the **str** function to print it out.)
    a. (**a and b and c**) can be read as (**True and False and True**). For and operator, all values must be the same. Since (**True and False and True**) gave us a **False**, to make it give us a **True**, we must make into (**True and True and True**).
    b. Also (**False and False and False**) gives us True because all of them are equal in values.
4. The fourth print statement checks if at least one of them are **True**
    a. (**a or b or c**) can be read as (**True or False or True**). The computer always picks the True no matter what. Like you can have a million Falses but only one True, and the computer always gives you True.
        i. (**True or False or False or False or False or False or False or False**) = **True**
        ii. (**False or False or False or False or False or False or False or False**) = **False**
5. The sixth print statement returns the opposite boolean value.
    a. (**not b**) can be read as (**not False**) which means True. I mean, think about it, what is the opposite of False? True right?
    b. Now to make (**a and b and c**) give us True, we can add a not right next to b.
        i. (**a and not b and c**) would then give us **True**.