Python for Scientific Data Analysis

Basic Python

Section 4: If-Then Statements, Logic/Truth Values, Boolean Values

If-Then Statements

Basic Examples

If-Then statements perform an operation if a statement is True. "If [x is true], then do [y]". The if-then statements are ended by a `:`.

The program `ex29.py` is an example:

```
def execute(people=20,cats=30,dogs=15):
 if people < cats:
    print("Too many cats! The world is doomed!")

 if people > cats:
    print("Not many cats! The world is saved!")

 if people < dogs:
    print("The world is drooled on!")

 if people > dogs:
    print("The world is dry!")


 dogs += 5

 if people >= dogs:
    print("People are greater than or equal to dogs.")

 if people <= dogs:
    print("People are less than or equal to dogs.")


 if people == dogs:
    print("People are dogs.")
```

Before we proceed, note one thing about the values for variables "people", "cats", and "dogs" in the function call. They seem to be assigned specific values. And indeed the values listed are exactly what are used if you do *not* override them at the function call.

So what happens in this program? Depending on the value of "people", "dogs", and "cats" different things are printed. Here are some examples:

```
from ex29 import execute

execute()
```

prints out ...

```
Too many cats! The world is doomed!
The world is dry!
People are greater than or equal to dogs.
People are less than or equal to dogs.
People are dogs.
```

But

```
from ex29 import execute

execute(people=50,dogs=30)
```

prints out ...

```
Not many cats! The world is saved!
The world is dry!
People are greater than or equal to dogs.
```

and

```
execute(people=50,dogs=30,cats=70)
```

prints out ...

```
Too many cats! The world is doomed!
The world is dry!
People are greater than or equal to dogs.
```

If/El-If/Else Statement Structures

Now we are going to add another wrinkle: If/El-If/Else structures. Before, with an if-then statement, we execute a block of code if a given condition is met. The If/El-If/Else is like that but allows for a branching of code.

See *ex30.py* below

```
def execute(people=30,cars=40,trucks=15):
 if cars > people:
    print("We should take the cars.")
 elif cars < people:
    print("We should not take the cars.")
 else:
    print("We can't decide.")

 if trucks > cars:
    print("That's too many trucks.")
 elif trucks < cars:
    print("Maybe we could take the trucks.")
 else:
    print("We still can't decide.")

 if people > trucks:
    print("Alright, let's just take the trucks.")
 else:
    print("Fine, let's stay home then.")
```

With no changes to the variables, this executes as ...

```
>> from ex30 import execute
>> execute()

We should take the cars.
Maybe we could take the trucks.
Alright, let's just take the trucks.
```

or, setting cars =0, we get ...

```
>>> execute(cars=0)
We should not take the cars.
That's too many trucks.
Alright, let's just take the trucks.
```

Logic and Truth Values

[shamelessly copied from Learning Python the Hard Way]

Logic describes whether things are or are-not, true or not-true (false), etc. They form the backbone of decision trees in code.

Python uses different key *terms* to decide whether or not something is True or False. Below is a list of truth *terms*. Logic on a computer is about seeing whether some combination of these *terms* is True at a given point in the program:

- `and`
- `or`
- `not`
- `!=` (not equal)
- `==` (equal)
- `>=` (greater than or equal to)
- `<=` (less than or equal to)
- `True`
- `False`

Truth Tables

We will now use the above characters to make truth tables.

NOT	True?
not False	True
not True	False

OR	True?
True or False	True
True or True	True
False or True	True
False or False	False

AND	True?
True and False	False
True and True	True
False and True	False
False and False	False

NOT OR	True?
not (True or False)	False
not (True or True)	True
not (False or True)	False
not (False or False)	True

NOT AND	True?
not (True and False)	True
not (True and True)	False
not (False and True)	True
not (False and False)	True

!=	True?
1!=0	True
1!=1	False
0!=1	True
0!=0	False

==	True?
1==0	False
1==1	True
0==1	False
0==0	True

Boolean Operations

Take each of these logic problems and write what you think the answer will be. In each case it will be either True or False. Once you have the answers written down, you will start Python in

your Terminal and type each logic problem in to confirm your answers.

```
True and True
False and True
1 == 1 and 2 == 1
"test" == "test"
1 == 1 or 2 != 1
True and 1 == 1
False and 0 != 0
True or 1 == 1
"test" == "testing"
1 != 0 and 2 == 1
"test" != "testing"
"test" == 1
not (True and False)
not (1 == 1 and 0 != 1)
not (10 == 1 or 1000 == 1000)
not (1 != 10 or 3 == 4)
not ("testing" == "testing" and "Zed" == "Cool Guy")
1 == 1 and (not ("testing" == 1 or 1 == 0))
"chunky" == "bacon" and (not (3 == 4 or 3 == 3))
3 == 3 and (not ("testing" == "testing" or "Python" == "Fun"))
```

What about this one?

```
3 != 4 and not ("testing" != "test" or "Python" == "Python")
```

We can break down this complex boolean operation as follows. First, what about `3 != 4`? That's a TRUE statement. So we have TRUE AND [something]. Let's deal with the latter now. `"testing" != "test" or "Python" == "Python"`? That's TRUE. So TRUE AND not(TRUE): in other words, TRUE AND FALSE which equals FALSE.