Boolean Practice

The logic combinations you learned from the last exercise are called *Boolean logic* expressions. Boolean logic is used *everywhere* in programming. It is a fundamental part of computation, and knowing the expressions is akin to knowing your scales in music.

In this exercise you will take the logic exercises you memorized and start trying them out in Python. 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, start Python in your Terminal and type each logic problem in to confirm your answers.

1. `True and True`
2. `False and True`
3. `1 == 1 and 2 == 1`
4. `"test" == "test"`
5. `1 == 1 or 2 != 1`
6. `True and 1 == 1`
7. `False and 0 != 0`
8. `True or 1 == 1`
9. `"test" == "testing"`
10. `1 != 0 and 2 == 1`
11. `"test" != "testing"`
12. `"test" == 1`
13. `not (True and False)`
14. `not (1 == 1 and 0 != 1)`
15. `not (10 == 1 or 1000 == 1000)`
16. `not (1 != 10 or 3 == 4)`
17. `not ("testing" == "testing" and "Zed" == "Cool Guy")`
18. `1 == 1 and (not ("testing" == 1 or 1 == 0))`
19. `"chunky" == "bacon" and (not (3 == 4 or 3 == 3))`
20. `3 == 3 and (not ("testing" == "testing" or "Python" == "Fun"))`

I will also give you a trick to help you figure out the more complicated ones toward the end.

Whenever you see these Boolean logic statements, you can solve them easily by this simple process:

1. Find an equality test (== or !=) and replace it with its truth.
2. Find each `and/or` inside parentheses and solve those first.
3. Find each `not` and invert it.
4. Find any remaining `and/or` and solve it.
5. When you are done you should have True or False.

I will demonstrate with a variation on item 20:

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

Here's me going through each of the steps and showing you the translation until I've boiled it down to a single result:

1. Solve each equality test:

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

   `"testing" != "test"` is `True`: `True and not (True or "Python" == "Python")`

   `"Python" == "Python"`: `True and not (True or True)`

2. Find each `and/or` in parentheses ():

   `(True or True)` is True: `True and not (True)`

3. Find each `not` and invert it:

   `not (True)` is False: `True and False`

4. Find any remaining `and/or`s and solve them:

   `True and False` is False

With that we're done and know the result is False.

WARNING! The more complicated ones may seem *very* hard at first. You should be able to take a good first stab at solving them, but do not get discouraged. I'm just getting you primed for more of these "logic gymnastics" so that later cool stuff is much easier. Just stick with it, and keep track of what you get wrong, but do not worry that it's not getting in your head quite yet. It'll come.

What You Should See

After you have tried to guess at these, this is what your session with Python might look like:

```
$ python3.6
Python 2.5.1 (r251:54863, Feb  6 2009, 19:02:12)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> True and True
True
>>> 1 == 1 and 2 == 2
True
```

Study Drills

1. There are a lot of operators in Python similar to `!=` and `==`. Try to find as many "equality operators" as you can. They should be like < or <=.
2. Write out the names of each of these equality operators. For example, I call `!=` "not equal."
3. Play with Python by typing out new Boolean operators, and before you press Enter try to shout out what it is. Do not think about it. Shout the first thing that comes to mind. Write it down, then press Enter, and keep track of how many you get right and wrong.
4. Throw away the piece of paper from 3 so you do not accidentally try to use it later.

Common Student Questions

Why does `"test" and "test"` return `"test"` or `1 and 1` return `1` instead of `True`? Python and many languages like to return one of the operands to their Boolean expressions rather than just `True` or `False`. This means that if you did `False and 1` you get the first operand (`False`), but if you do `True and 1` you get the second (`1`). Play with this a bit.

Is there any difference between `!=` and <>? Python has deprecated <> in favor of `!=`, so use `!=`. Other than that there should be no difference.

Isn't there a shortcut? Yes. Any `and` expression that has a `False` is immediately `False`, so you can stop there. Any `or` expression that has a `True` is immediately `True`, so you can stop there. But make sure that you can process the whole expression because later it becomes helpful.