Exercise 33: While Loops

Now to totally blow your mind with a new loop, the `while-loop`. A `while-loop` will keep executing the code block under it as long as a boolean expression is `True`.

Wait, you have been keeping up with the terminology, right? That if we write a line and end it with a `:` (colon) then that tells Python to start a new block of code? Then we indent and that's the new code. This is all about structuring your programs so that Python knows what you mean. If you do not get that idea then go back and do some more work with `if-statements`, functions, and the `for-loop` until you get it.

Later on we'll have some exercises that will train your brain to read these structures, similar to how we burned boolean expressions into your brain.

Back to `while-loops`. What they do is simply do a test like an `if-statement`, but instead of running the code block *once*, they jump back to the "top" where the `while` is, and repeat. A `while-loop` runs until the expression is `False`.

Here's the problem with `while-loops`: Sometimes they do not stop. This is great if your intention is to just keep looping until the end of the universe. Otherwise you almost always want your loops to end eventually.

To avoid these problems, there are some rules to follow:

1. Make sure that you use `while-loops` sparingly. Usually a `for-loop` is better.
2. Review your while statements and make sure that the boolean test will become `False` at some point.
3. When in doubt, print out your test variable at the top and bottom of the `while-loop` to see what it's doing.

In this exercise, you will learn the `while-loop` while doing these three checks:

```
1  i = 0
2  numbers = []
3
4  while i < 6:
5      print "At the top i is %d" % i
6      numbers.append(i)
7
8      i = i + 1
9      print "Numbers now: ", numbers
10     print "At the bottom i is %d" % i
11
12
13 print "The numbers: "
14
15 for num in numbers:
16     print num
```

What You Should See

```
$ python ex33.py
At the top i is 0
Numbers now:  [0]
At the bottom i is 1
At the top i is 1
Numbers now:  [0, 1]
At the bottom i is 2
At the top i is 2
Numbers now:  [0, 1, 2]
At the bottom i is 3
At the top i is 3
Numbers now:  [0, 1, 2, 3]
At the bottom i is 4
At the top i is 4
Numbers now:  [0, 1, 2, 3, 4]
At the bottom i is 5
At the top i is 5
Numbers now:  [0, 1, 2, 3, 4, 5]
At the bottom i is 6
The numbers:
0
1
2
3
4
5
```

Study Drills

1. Convert this `while-loop` to a function that you can call, and replace `6` in the test (`i < 6`) with a variable.
2. Use this function to rewrite the script to try different numbers.
3. Add another variable to the function arguments that you can pass in that lets you change the `+ 1` on line 8 so you can change how much it increments by.
4. Rewrite the script again to use this function to see what effect that has.
5. Write it to use `for-loops` and `range`. Do you need the incrementor in the middle anymore? What happens if you do not get rid of it?

If at any time that you are doing this it goes crazy (it probably will), just hold down CTRL and press `c` (CTRL-`c`) and the program will abort.

Common Student Questions

Q: What's the difference between a `for-loop` and a `while-loop`?

A `for-loop` can only iterate (loop) "over" collections of things. A `while-loop` can do any kind of iteration (looping) you want. However, `while-loops` are harder to get right and you normally can get many things done with `for-loops`.

Q: Loops are hard. How do I figure them out?

The main reason people don't understand loops is because they can't follow the "jumping" that the code does. When a loop runs, it goes through its block of code, and at the end it jumps back to the top. To visualize this, put print statements all over the loop printing out where in the loop Python is running and what the variables are set to at those points. Write `print` lines before the

loop, at the top of the loop, in the middle, and at the bottom. Study the output and try to understand the jumping that's going on.

Video