Loops

What if I had a table of data that we needed to print out in a pattern, like a Sudoku puzzle. Or maybe I wanted scan through the numbers 1 through 100. It would be difficult to do it with just using if statements and functions. Processes like repetition are tough. However we have loops: for loops and while loops.

While loops

```
while (condition):
  do task
```

If we were to look at this after our lecture on `if` statements, we would see a bit of a similarity. There is. The two lines above mean the following: `While` this `condition` is `True`, do task. Think about it for a moment. ### True loops What happens when we do this code:

```
while(True):
  print "Hello!"
```

Think about it.

This is called an *infinite loop* or a *forever loop*. It will go on forever until your computer's battery runs out or it dies. If this happens in your Python shell, don't worry. Restart the shell or kill Python. To restart the Python shell if you are using Wing 101, go to the window in Wing 101 that has the Python Shell. In the top right corn there is a tab that has the word "options". Click it and the first option is "restart Python". Click it and it will fix your problem. To kill Python, launch your Operating System's task manager and end the task.

To avoid a infinite loop, you must have an *end condition*. That is to say, a condition that is met that causes the loop to end. A while loop is like an if statement: it does not do the task inside of it if the condition is `False`.

Examples

The best way of explaining the while loop is to do some examples.

```
x = 0
while (x <10):
  print x
  x+=1
```

This will print out a number going from 0 to 9, one on each line:

```
0
1
2
```

```
3
4
5
6
7
8
9
```

We could make a menu:

```
from math import pi
user = 0
while (user != '-1'):
  print "Enter a number from below:"
  print "\t1) print \"Hello, world!\""
  print "\t2) print the number 2"
  print "\t3) print pi"
  print "Enter -1 to quit"
  user = raw_input("Enter: ")
  if (user == "1"):
    print "Hello, world!"
  elif (user == "2"):
    print 2
  elif (user == "2"):
    print "%.4f" % pi
  print
print "Bye!"
```

Here is the result with me enter numbers into it. Watch how it handles things that are not options:

```
Enter a number from below:
  1) print "Hello, world!"
  2) print the number 2
  3) print pi
Enter -1 to quit
Enter: 1
Hello, world!

Enter a number from below:
  1) print "Hello, world!"
  2) print the number 2
  3) print pi
Enter -1 to quit
Enter: 2
2

Enter a number from below:
```

```
  1) print "Hello, world!"
  2) print the number 2
  3) print pi
Enter -1 to quit
Enter: 3
3.1416

Enter a number from below:
  1) print "Hello, world!"
  2) print the number 2
  3) print pi
Enter -1 to quit
Enter: pineapple

Enter a number from below:
  1) print "Hello, world!"
  2) print the number 2
  3) print pi
Enter -1 to quit
Enter: quit

Enter a number from below:
  1) print "Hello, world!"
  2) print the number 2
  3) print pi
Enter -1 to quit
Enter: -1

Bye!
```

`While` loops are powerful and very useful.

For loops

There are two types of loops. `While` loops and `for` loops. When you are beginning to program, some ask "which loop should I use?" For most cases, it does not matter. If you are doing something until a certain condition is met, you need a `while` loop. If you are doing something a certain number of times, a `for` loop is needed. After a certain point, most things you program can do either of this.

```
for item in group:
  do_task_with(item)
```

A for loop means this: `for` every `item` in a `group` of items, do a task.

```
x=0
```

```
while x < 5:
  print x
  x+=1

for x in range(5):
  print x
```

Both loops do the same thing:

```
0
1
2
3
4
0
1
2
3
4
```

range(int)

The function `range(int)` is a function that gives you a group (a list) of numbers from `0` up until the `number` you entered. If you say `print range(4)`, it will output `[0,1,2,3]`. The reason it starts at zero is because computer count from zero, so when it stops at `3` you have `4` numbers in your list.

Examples

For loops are good with creating numbers. Here is nested `for` loops. You can do this with `while` loops but programmers tend to write lines like this:

```
for i in range(5):
  for j in range(5):
    print "("+str(j)+","+str(i)+")", #I want the format (j,i)
  print
```

This will create a grid of numbers. Remember that they are being printed in a pattern of (i,j). Once the `for` loop with the `j` has finished, a newline is entered and the `for` loop starts again.

```
(0,0) (1,0) (2,0) (3,0) (4,0)
(0,1) (1,1) (2,1) (3,1) (4,1)
(0,2) (1,2) (2,2) (3,2) (4,2)
(0,3) (1,3) (2,3) (3,3) (4,3)
(0,4) (1,4) (2,4) (3,4) (4,4)
```

This could be used for coordinates in math, places in a game, or comparing values. Here is a something with lists:

```
colors = list()
flag = True;
colors.append("blue")
colors.append("green")
colors.append("red") # l0 = ["blue", "green", "red"]
user = raw_input("What is your favorite color? ")
user = user.lower()
for color in colors:
  if (user == color):
    print "I like that color as well!"
    flag = False;
print "That's a good color."
if flag:
  colors.append(user)
```

This code might be a lot, so bare with me. This is a different way of doing a `for` loop that I talked about at the beginning of this section on `for` loops. `for` item `in` group is what it means. So `for` every `color` in `colors`, we are going to see if it matches the one the user put in. If it does, we print `I like that color as well!`

At the end, if the color is not in the list of colors, then we add it to that list.

You might be wondering what a list is. That is in the next lecture about data structures. What I say is that it contains information to make groups of info easy to add, edit, access, and remove.f