Handout 1 – Loops

[…]

- Don't understand loops (or at least *while*, or *for*)

I'll review this at the beginning of the class, but quick reference: we use loops when we want to keep repeating a set of statements.

If we know exactly how many times we'll be repeating it, we could just copy/paste the code that many times. We could even put the code into a function and just call that function that many times. Even in this case, though, this is not good practice, and it's even impractical -- what if you have to call the function 100 times? 1000? etc.

So that's why there is the concept of looping. There are two ways to loop that we've taught you: **while** and **for** loops (although there was no need to use **for** in either of the lab problems yesterday).

The syntax is:

```
while [expression that evaluates True or False]:
    [statements to be executed]

for [variable] in range([start], stop, [step] ):
    [statements to be executed]
```

For **while**, the computer first evaluates the expression. If it's true, it executes the statements. Then it returns to the expression and evaluates it again -- if it's true again, it executes the statements again. And it keeps checking like this, and repeating as long as the expression is true.

I mentioned this in office hours, it might be useful to think like this: if you think in terms of "until", e.g. "play the game until there are no stones", then we can always write that in terms of "while" by just negating our expression, e.g. "play the game WHILE there ARE stones".

The **for** and **range** statements are confusing to a lot of people. Play around with range in the shell to get an idea for what start, stop and step mean, e.g. range(1, 11, 2) returns the numbers 1, 3, 5, 7, 9. Remember that you never include the stop number. Once you understand **range**, then **for** just means that through each iteration of the loop ("iteration" means a time that you're executing the statements inside), the variable after **for** is assigned to the next number in that range.

Keep playing around with these, it'll start to make sense.

[…]