**Loops**

You’ll often have parts of code that you have to execute over and over again, and loops are a good way to do this efficiently.

**For-loops**

For-loops are a good way to get a segment of code to execute a fixed number of times. The basic structure of a for-loop is:

for(initialization; condition; iteration)

{

//code

}

Initialization is when you create a variable. In most for loops, it is the creation of an int. Condition is what the for-loop checks every time to make sure that it’s true before continuing. Iteration is what the for-loop does every time it is executed. Usually, it is just making the variable you initialized go up.

In this example, the computer will print out “Hello” ten times:

for(int i = 0; i < 10; i++)

{

System.out.println(“Hello”);

}

Every time the for-loop executes, it checks to make sure that i is less than 10, then increases the value by one. Here’s another example of when a for-loop would be useful:

System.out.println(“Here are the two times tables up to 10”);

for(int i = 0; i < 11; i++)

{

System.out.println(“Two times” + i + “is” + i\*2);

}

We can make the for-loop a little more complicated – i doesn’t just have to be going up by one each time:

System.out.println(“Powers of two!”);

for(int k = 1; k < 100; k\*=2)

{

System.out.println(k);

}

One last thing to consider about for-loops: The condition statement doesn’t always have to be < or >. It just has to be something that is true or false that the for-loop can check.

**While-loops**

Even though for-loops can have a condition other than > or <, it is much easier to do that with a while loop. In a while loop, a segment of code will continuously execute as a long as a condition is true. This is the basic structure of a while loop.

while(condition)

{

//code

}

Here’s an example:

while(!isEmpty())

{

remove();

}

This is a very standard example of a while-loop. It is basically saying to continue to remove objects as long as whatever is not empty.

The while-loop can also mimic a for-loop:

int num = 0;

while(num < 7)

{

System.out.println(num);

num++

}

This is just like saying:

for(int num = 0; num < 7; num++)

{

System.out.println(num);

}

WARNING: Infinite Loops

Infinite loops are very easy to get into with a while-loop. In order to get into an infinite loop, the condition that you are checking never evaluates to false. For example:

int a = 2;

while(a > 0)

{

System.out.println(a);

a++;

}

This will cause an infinite loop because a will never be less than zero, so the condition will never be false. Infinite loops used to be very dangerous for the computer, but modern computers have a built in protection. However, infinite loops still have the potential to crash the computer, so be careful 

**Do-while-loops**

There is another slight variation of a while-loop that is very uncommon. It’s called the do-while-loop. It is basically the same thing as a while-loop, except you are guaranteed that the code will execute at least once. In a while loop, if the condition isn’t true, the code won’t execute. It will always execute at least once in a do while loop. Here’s the basic structure:

do

{

//code

}

while(condition);

Do-while loops are rarely used because they aren’t often necessary. Most times, whatever you need to do can be solved with a while-loop.

**Projects**

Here are some projects that will make sure that you know everything you need to from this lesson:

Don’t forget, you can always ask and answer questions on the website, and if that doesn’t help, bring your questions to our next meeting.