Arrays and loops

Learn what loops are, and some conventions for writing loops such as using ++ and declaring variables outside the loop

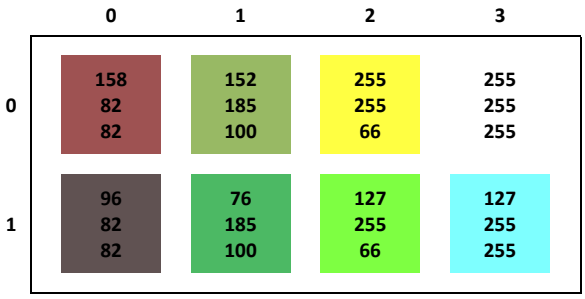
Learn what a while loop is and how to use it

Why repeat things?

Suppose we want to change the color of a picture by reducing the amount of red in it. Recall that each pixel has a red, green, and blue value, which can be manipulated independently. What if we want to decrease the amount of red in each pixel by half?

Example: Decreasing Red

> This image shows four pixels (top row) and the same pixels with less red in them (bottom row). The result might not seem intuitive -- the pixels don't turn from red to pink, but rather from red to greenish blue. This is because, as we decrease the amount of red in a pixel, the overall ratio of green and blue to red goes up (remember, in RGB, all colors are specified using a combination of red, green, and blue values).



> Each column of the 2D array depicts a color and the same color with its red value reduced by half.

There are four steps that we need to take to change one pixel:

- Get the current pixel
- Get the red value of the current pixel
- Change the red value to an appropriate percentage of the original value
- Put the new red value in the current pixel

Note that every color in a pixel has a maximum (255) and a minimum (0). The R(red), G(green), or B(blue) color of a pixel cannot be increased beyond its maximum (255) or decreased below its minimum (0).

Changing one or two pixels should not be too much of a problem. However, if we want to manipulate the *entire* image, it could take some time. Remember how many lines of code it took to draw a line on your image? Now multiply that by four, and by the number of pixels in your image! For example, if there are 172800 pixels in the image, you would have to write 691,200 lines of code to finish this task. If we were to do this for an entire image by hand, it would be a very very long program. We need a smarter way of doing this.

Loops

The solution is a loop. A loop is a way to execute a series of statements that are very similar to each other. Typically, something changes each time the statements are executed (such as the specific pixel being operated on).

It is critical when writing a loop that we have a way to test whether we are done repeating things. In this case, we are done when we have looped through all of the pixels. This test is called the *end condition*, and it takes the form of a test to see if the loop should stop.

Often, a loop is used to do something a fixed number of times. In this case, a loop will use a counter to keep track of when it is finished. Each time the loop repeats, the counter is incremented. When the counter reaches the goal number, the loop stops.

Example: A simple loop in pseudocode

Suppose you are trying to use a robot to beat the sit-up world record, which stands at 133,986 consecutive situps. You may want to use a loop to do this, since you will have the robot complete many repetitions of the *situp* command.

As described above, a loop has three main components: A command that is executed; an end condition which tells when the loop should stop, and (useful in this case but not required) a counter. What we need is:

```
// initialize our counter to 0 since no situps have been done yet
numSitups  is 0;

// repeat this until numSitups > 133,986
      //do a situp
     Display to the screen that The robot sat up!"

      // don't forget to increment the counter after each situp
      // or the loop will go on forever!
      Increase the number of situps by 1
```

Note...

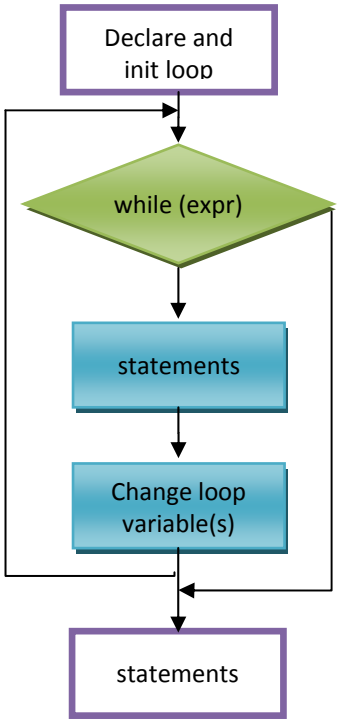
Notice that the code above has the statement **numSitups = numSitups + 1;** to increment the counter variable **numSitups**. Instead of doing this, we can use **numSitups += 1;** , which is an equivalent statement of the code above. What **+=** does is to increment the value of the variable and reassign it to the variable. An even briefer way to say the same thing is: **numSitups++;** Similarly, **numSitups -= 1;** and **numSitpus--;** are equivalent to **numSitups = numSitups - 1;** All three statements decrement the value of the variable. Another name for **++** is the *increment* operator and another name for **--** is the *decrement* operator.

While Loops

In Java, one way to repeat a block of statements while some expression is true is to use a while loop. First, we create a counter and set it to the start value. Then we check that the counter is less than the stop value. If it is less, then execute the statements in the loop. We next add one to the counter and go back to check that the counter is less than the stop value. The syntax for a while loop is very straightforward:

```
while(condition) {
    statements to execute
}
```

However, the details of what happens in this code are more complex. They are best explained using a flowchart. A flowchart tells you what order things happen in when code is being executed. When reading it, start at the top (declare and init loop variable(s) in the example below). Then just follow the arrows. If there is more than one arrow coming out of the same box or diamond, that means that more than one thing might happen.



This is a *flow chart* for a while loop. It starts with the top box (declare and init loop variable(s)). Next it checks whether the loop should proceed (represented with the yellow diamond). If while(expression) is true (that is, if the loop should keep going), then the blue box labeled *Statement(s)* will be executed, the loop variables will be changed, and execution will return to the top of the loop. If while(expression) is false, then the loop will end, and the rest of the code in your program (represented by the white box labeled *Statement(s)* will be executed.

Example: Translating Pseudocode to a While Loop

We can take the pseudocode from before and add the code necessary to create a while loop for sitting up. Since 133,986 is enough situps to fill many many pages of your output panel, let's try doing this for 133 situps. If you want to experiment with a larger number, feel free. You can always kill the process if you get tired of watching "The robot sat up!" scroll by.

```
// initialize our counter to 0 since no situps have been done yet
int numSitups = 0;

// repeat this until numSitups > 133 (or more). But be sure not to use
// a comma if you want to do 133,986 situps, it will cause an error. Just
// type 133986
while (numSitups <= 133) {
   //do a situp
   System.out.println(numSitups + ". The robot sat up!");
   // don't forget to increment the counter after each situp
   // or the loop will go on forever!
   numSitups = numSitups + 1;
}
```

Notice that we have only added two lines of code to our pseudocode from above: the while loop, and where it ends

What's going on at each step? Java steps through the code evaluating each line as it goes. First it initializes numSitups to 0. Next, it starts the loop. The very first thing it does is check the end condition (numSitups <= 133). Is it true? If so, Java will enter the loop and execute the code inside it (between the { and }).

On line 8, it prints out "The robot sat up!" and on line 11 it increments the counter variable, numSitups. Then execution returns to the top of the loop (line 6) and checks if numSitups is less than 133 yet. You can try this code out yourself.

did I get this

Using a while loop and the following line:
System.out.println("*****");
Print out 40 lines with 5 stars.

Our Solution:

```
int counter = 0;
while (counter < 40) {
   System.out.println(“*****”);
   counter = counter + 1;
}
```

The counter starts from 0, and each iteration through the loop, 1 is added to the counter. The test expression is (counter < 40) because on the 40th iteration, the counter loop to stop.

If the test expression was (counter <= 40), then we would have printed 41 lines of stars

Practice: Adding numbers

Now suppose you want to add all the numbers from 1 to 100. You will need something to hold the total so far. You will also need something that counts from 1 to 100 (the *counter* variable). Finally, you need to stop counting when you get to 100 (the *end condition*).

As with the situps, we will start by writing pseudocode. Some questions you need to ask yourself include: What type of variable should hold the total? What value should it start out with? What type of variable should the counter variable be? What value should it start with?

learn by doing

Write some pseudocode for adding up the numbers from 1 to 100.

Our Solution:

Both variables should be of type int. The total should start with 0 and the counter should start with 1. Based on this, we can write something very similar to the previous pseudocode. Try it yourself and paste the answer in below.

```
// initialize counter to 1
int counter = 1;
int total = 0;

// repeat until counter > 100 and add the value to total
total = total + counter;

//increment counter to prevent getting an infinite loop
counter++;
```

learn by doing

Now that you've written the pseudocode, see if you can insert the correct Java statements to total 100 numbers.

Our Solution:

```
// initialize counter to 1
int counter = 1;
int total = 0;

// repeat until counter > 100 and add the value to total
while (counter <= 100)  {
   total = total + counter;

   //increment counter to prevent getting an infinite loop
   counter++;
}
```

How does this loop work? At the end of the first step inside the loop, **total = total + counter = 0 + 1 = 1**. At the end of the next step, **counter = counter + 1 = 1 + 1 = 2**. Now the code inside the loop is complete, so Java moves *back up to the beginning of the loop.*

Once again, Java checks the end condition. In this case, **counter == 2** so **counter <= 100** is definitely true. Once again, Java enters the loop and executes the code inside it. This time, **total = 1 + 2 = 3** and **counter = 2 + 1 = 3**.

Once again, Java checks the end condition. In this case, **counter == 3** so **counter <= 100** is definitely true. Once again, Java enters the loop and executes the code inside it. This time, **total = 3 + 3 = 6** and **counter = 3 + 1 = 4**.

Things are starting to get repetitive, and that's why a loop is so useful here. When the loop exits, it will be because **counter <= 100** is false.

learn by doing

Open up the Netbeans project MediaFiveEx. You will find a class in mediafiveex package called Looping. This has the code that we just wrote.

The method addTo100() contains the code we just wrote. To run it, you'll need to add to the main method in MediaFiveEx2:

```
Looping looper = new Looping();
looper.addTo100();
```

What value does total have at the end? _____

Modify the code to print out the value of counter after the loop ends?

What value does counter have at the end? _____

Our Solution:

What value does total have at the end? _____**5050**_____

```
Looping looper = new Looping();
int sum = 0;
sum = looper.addTo100();
System.out.println( sum );
```

Modify the code to print out the value of counter after the loop ends?

```
public int addTo100() {
    // initialize our counter to 1
    int counter = 1;
    int total = 0;
    //repeat this until counter > 100
    while(counter<=100) {
        //add in the new value
       total = total + counter;
      //don't forget to increment the counter or the loop will go on forever!
      counter = counter + 1;
   }

   System.out.println (counter);
   return total;
}
```

What value does counter have at the end? _____**101**_____

System.out.println(counter) will cause the value of counter to be printed to the screen. To get the final value of counter, put System.out.println(counter) right before line 19 of the program (return total;). If you want to see all the intermediate values, put it before line 18 (}). This will place it inside the loop so that it will get run on every iteration.

There are an endless number of ways that you can use loops to accomplish your goals. For example, you don't always have to increment the counter by 1. If you increment it by 2, you should be able to loop through only odd numbers.

did I get this

Now let's write a loop, with a slightly different spin..
Add a method to the Looping class that adds all the ODD numbers from 1 to 100.

Our Solution:

```
int total = 0;
int num = 1;
while (num <=100) {
    total = total + num;
    num = num +2;
}
```

Instead of incrementing by 1 each time we go through the loop (which would give us ALL numbers from 1 to 100), we increment by 2, so that the program adds 1, 3, 5, 7, etc... The total should be 2500 if you implement this correctly.

Now that we've gotten a handle on while loops, let's see if we can use a while loop to accomplish our original goal of changing the red values of all the pixels in our picture. We'll do this by adding a method to the Picture object (available as one of the classes in the media package).

By now, you should have MediaFiveEx open. In the media folder open up Picture.java and add the following code to the class.

```
public void decreaseRed() {
   Pixel[] pixelArray = this.getPixels();
   int value = 0;

   // this will be our counter
   int count = 0;

   // loop through all the pixels in the array
   while (count < pixelArray.length) {
     // get the red value of the current pixel
     value = pixelArray[count].getRed();
     // decrease the red value by 50%
     value = value / 2;
     // set the red value of the current pixel to the new value
     pixelArray[count].setRed(value);
     count = count + 1;
   }
}
```

Don't forget to try it out! Recompile your Picture class, add your code to MediaFiveEx3, load in a new picture, call decreaseRed() on it, and repaint!.

```
Picture pic = new Picture(FileChooser.pickAFile());
pic.show();
pic.decreaseRed();
pic.repaint();
```

did I get this

Look at the code below and answer the following questions:

```
int a = 0;
int b = 0;
int c = 2;
int count = 0;

while( count < 5 ) {
   a = count + 1;
   b = a * 2;
   c = 5;
   count = count + 1:
}
```

What about he value of c at the end of the loop? _____

At the end of the loop, what is the value of variable a? _____

Our solution

What about he value of c at the end of the loop?

c is modified by each iteration of the loop, and its value is always the same: 5.

At the end of the loop, what is the value of variable a?

The correct answer is 5. a is one larger than the value of counter at the end of the loop. When the counter reaches a value of 5, the loop code will not execute. During the last execution of the loop code, the counter is 4.

Summary

In summary, you can use a loop to repeat a series of Java statements while some test is true. On this page we learned about while loops. We will learn about other loops in the next few pages.

A while loop requires you to more explicitly define what it does. Often, you will use a counter to keep track of how many times the loop has executed. You need to declare the counter before the loop and increment it at the end of the loop.

The syntax for a **while** loop is:

```
while(condition) {
   statements to execute
}
```

You can easily increment (or decrement) a counter using the syntax *variableName++* or *variableName--*