Block Statements

This is lesson four in the Introduction to Programming in JavaScript, and I'm going to be going over **block statements** which we covered briefly in an earlier lesson.

What is a Statement?

Statements are sets of instructions that tell the computer how to perform an action. A typical statement in JavaScript usually begins with a keyword or a variable. A statement always ends with a semicolon. Statements in JavaScript are executed sequentially, for the most part. And as I've mentioned, statements may be grouped in a block with curly braces. But why would you want group statements?

Why Group Statements?

A program may need to repeat the same set of steps over and over again. Each one of those steps could be a separate statement, and by grouping those statements together, you can tell the program to consider that particular group of statements as a single block and treat them consistently. So, when would a program need to repeat a set of statements? Well, let me give you an example of a situation you might encounter.

```
var colors = ["red", "green", "blue"];
var question;

question = "Do you like " + colors[0] + "?";
console.log(question);
// "Do you like red?"

question = "Do you like " + colors[1] + "?";
console.log(question);
// "Do you like green?"

question = "Do you like " + colors[2] + "?";
console.log(question);
// "Do you like blue?"
```

There's a lot of repeated code in here. We've had to redefine question three times. Each time, we had to write out the whole string, we had to add the quotes, we had to add the concatenation operators, we had to use the question mark, we had to do the `console.log`. It would be so much more convenient, if we could get JavaScript to do that for us. This is a computer - it's supposed to be good at doing repetitive things for us so that we don't have to do them ourselves. And that brings us to one of our first convenient uses for block statements, iteration.

Block Statements for Iteration

You can group statements together so that they can be repeated within a block, and then you can use control flow statements to execute them. And control flow statements tell JavaScript when to execute a block of statements, how often, how many times, etc. Let me show you right here in pseudocode how that would work:

```
var colors = ["red", "green", "blue"];
var question;

// PSEUDO CODE (Not Real JavaScript)
```

```
FOR EACH COLOR IN COLORS {
  question = "Do you like " + colors[THE CURRENT COLOR] + "?";
  console.log(question);
  // "Do you like red?" (first time)
  // "Do you like green?" (second time)
  // "Do you like blue?" (third time)
}
```

Block Statements for Conditions

Another way to use block statements is to execute a block conditionally based on the value of a boolean. What we can tell JavaScript to do is execute a block only if a condition is evaluated as `true`. Remember, we've discussed before how to evaluate an expression to see whether it returns a value of `true`. Let me show you some pseudocode to explain where we're going with this.

```
var colors = ["red", "green", "blue"];
var selection = 0;

console.log("You chose " + colors[selection]);
// "You chose red"

// PSEUDO CODE (Not real JavaScript)
IF THE SELECTION IS RED {
  console.log("Just like Red Riding Hood")
} OTHERWISE, IF THE SELECTION IS GREEN {
  console.log("Just like Peter Pan")
} OTHERWISE, IF THE SELECTION IS BLUE {
  console.log("Just like Little Boy Blue")
} OTHERWISE {
  console.log("That's nice")
}
// "Just like Red Riding Hood"
```

Coming Up: Control Flow and Iteration

In the next few lessons, I'm going to take you through some examples of Control Flow and Iteration because there are some key words that will tell JavaScript to do exactly what we've just tried to do in the pseudo code in the previous examples: execute a block of statements. Those keywords that I'm going to go over are `if`, which does control flow based on a boolean, `for`, which can iterate over the elements in an array, `switch`, which can do control flow based on a value, and `while`, which could iterate, based on a boolean value.