Introduction to "Switch" Statements

Now that I've taken you through `if` and `else` statements, you can see how conditional logic can be used to branch among different values. JavaScript has another way of controlling flow that's actually designed for that sort of situation - the `switch` statement.

Multiple Possible Outcomes

`switch` statements are designed to handle multiple possible outcomes from the value of a single variable or expression. They can replace a pattern of using `if` and `else` statements repeatedly if you want to test a single expression. We've already seen nested `if` and `else` statements used together to branch across different possible outcomes depending upon the value of a variable, such as color in this case.

```
var color = "red";

if (color == "blue") {
  console.log("Blue is my favorite color");
} else if (color == "red") {
  console.log("That's not as bad as green");
} else {
  conslole.log("You have bad taste");
}
// "That's not as bad as green"
```

`switch` statements are a way to structure code that's trying to accomplish something like this, in a cleaner way. `switch` statements use one statement to handle multiple possible cases for a single value, and this is the way that they look:

```
var color = "red";

switch (color) { // the value to test
  case "blue": // first test case
    console.log("Blue is my favorite color");
    break; // terminate the statement
  case "red": // the second test case
    console.log("That's not as bad as green");
    break; // terminate the statement
  default: // do this if nothing else matches
    console.log("You have bad taste");
}
// "That's not as bad as green"
```

`switch` is passed the variable that we want it to switch on, and this can either be a variable or an expression. The `switch` statement takes a single block that contains a bunch of different cases, and depending on the value of color, one of these cases will be executed.

"Switch" Statement Syntax

One of the advantages of the `switch` statement, is that the syntax looks a little bit less messy than having a bunch of `if`s and `else`s together, but basically we're performing the same operations. The important thing with `switch` statements is to remember not to forget the `break` statements. If there's a `break` missing, the `switch` statement will go on and execute the next case.

```
var color = "red";
```

```
switch (color) {
  case "blue":
    console.log("Blue is my favorite color");
  case "red":
    console.log("That's not as bad as green");
  default:
    console.log("You have bad taste");
}
// "That's not as bad as green"
// "You have bad taste"
```

It may seem inconvenient, having to remember to put in a `break` statement every single time just in order to be able to use the `switch`.

Selective "Break" Use

Using `switch` lets you set up structures that will allow you to execute more than one case if that's what you're going for.

```
var adjectiveCount = 3;
var adjectiveArray = [];
var compliment;

switch (adjectiveCount) {
  case 4:
    adjectiveArray.push("exquisite");
  case 3:
    adjectiveArray.push("fantastic");
  case 2:
    adjectiveArray.push("amazing");
  default:
    adjectiveArray.push("great");
}

compliment = "You have " + adjectiveArray.join(", ") + " taste!";
console.log(compliment);
// You have fantastic, amazing, great taste!
```

So sometimes you'll want to use `break` statements and switches, and sometimes not. Usually you will want a `break` statement between each case and a switch, and some JavaScript compilers will actually give you warnings telling you that you need breaks between your case statements, but the JavaScript will still work without them.