Lesson | Wednesday

Intermediate JavaScript (/intermediate-javascript) / Object-Oriented JavaScript (/intermediate-javascript/object-oriented-javascript) / Switch Cases

Text

In JavaScript and Web Browsers (https://www.learnhowtoprogram.com/introduction-to-programming/javascript-and-web-browsers), we explored branching in depth. In this lesson, we'll learn about **switch statements**, which is another way of writing a conditional. Switch statements are also called **switch cases**. Switch cases do not add any additional functionality that we can't do with the branching we've already learned. However, you will likely see them in code samples on the internet (if you haven't already). They can also be a very convenient way to write conditionals that have many branches.

You won't be expected to use switch statements for this section's independent project — and you may not even get a chance to explore them in this section (depending on whether you incorporate branching in your projects). However, it's important to know what they look like — and this lesson will provide a reference if you want to use them in the future.

Switch Cases

Let's start by looking at the syntax of a switch statement.

```
const color = "red";
switch (color) {
  case ("red"):
    console.log("Red!");
    break;
  case ("green"):
    console.log("Green!");
    break;
  case ("blue"):
    console.log("Blue!");
    break;
  default:
    console.log("It's not red, blue, or green.");
}
```

A switch statement uses the keyword `switch`. We then pass in an expression (in this case, `color`, which contains the string `"red"`). When our code runs, it will execute the code in the *first* `case` clause that *matches* `"red"`. If none of the conditions match, the code in the `default` clause will run. Try the code out in the DevTools console and you'll see that the first `case` is executed.

Note that there is a `break;` statement at the end of each `case`. What happens if you take out each `break;` and run the code? Let's do that — and also change `color` to `"green"`, which matches the second case.

```
const color = "green";
switch (color) {
  case ("red"):
    console.log("Red!");
  case ("green"):
    console.log("Green!");
  case ("blue"):
    console.log("Blue!");
  default:
    console.log("It's not red, blue, or green.");
}
```

The following will be logged to the console:

```
Green!
Blue!
It's not red, blue, or green.
```

Well, that's not what we want here. If we omit `break;`, a switch statement will execute the first clause that matches. Then it will continue to run subsequent cases until it hits a `break;` or runs through every case (including the default).

Take a look at this code. What do you think will be logged to the console?

```
const color = "red";
switch (color) {
  case ("red"):
    console.log("Red!");
  case ("green"):
    console.log("Green!");
    break;
  case ("blue"):
    console.log("Blue!");
    break;
  default:
    console.log("It's not red, blue, or green.");
}
```

The answer is:

```
Red!
Green!
```

This is because the switch statement will evaluate the first case as true and then run that code. Then, because there is no `break;`, it will run the next case even though it doesn't actually match the color `"red"`. Then it will hit a `break;` and complete.

As you can see, omitting a `break;` could lead to some unintended bugs.

Note also that we can't use comparison operators. What do you think will happen when we run this code?

```
// This code will NOT work as you might hope. Don't do this
in your own code!

const number = 3;
switch (number) {
  case (number > 0):
    console.log("Number greater than 0!");
    break;
  case (number < 0):
    console.log("Number less than 0!");
    break;
  default:
    console.log("The number must be 0.");
}
```

You might hope that the following will be returned: `""Number greater than 0!"`. However, that's not the case. Instead, this switch statement will return `"The number must be 0."`. That's because it's looking for an *exact* match with `number` — and `3 > 0` equals `true` and not `3`.

Because of these limitations, you might initially think switch statements aren't very useful. However, as long as you are evaluating exact matches (and don't forget to use `break;` if needed), they can be much easier to read than `if...else` statements — especially when a lot of branches are involved.

There are also situations where we might *want* every case to run if a condition is met. In these situations, we don't need `break;` at all.

Here's an example. We are creating an interactive application about a cat. The cat can have four happiness levels and the effects of happiness are cumulative. If our cat has a `happiness` of `4`, we want every case to run:

```
switch (happiness) {
  case (4):
    console.log("MWAR! I love you!!!");
  case (3):
    console.log("PURR...");
  case (2):
    console.log("Purr...");
  case (1):
    console.log("Meow!");
    break;
  default:
    console.log("grumble grumble...");
}
```

If we run the code above with a `happiness` of `2`, it'll return the following:

```
Purr...
Meow!
```

That sounds like a fairly happy cat.

If we run the code above with a `happiness` of `4`, we'll get this:

```
MWAR! I love you!!!
PURR...
Purr...
Meow!
```

Now that is a *very* happy cat.

We could write the above code using `if` and `else` instead. It looks like this...

```
if (happiness > 3) {
  console.log("MWAR! I love you!!!");
}
if (happiness > 2) {
  console.log("PURR...");
}
if (happiness > 1) {
  console.log("Purr...");
}
if (happiness > 0) {
  console.log("Meow!");
} else {
  console.log("grumble grumble...");
}
```

While it's approximately the same number of lines, there's more to keep track of, including making sure we are using the correct comparison operators.

Ultimately, it comes down to a matter of preference. A switch statement is just syntactic sugar, meaning it uses an `if` statement under the hood. And, we can achieve the same thing with an `if` statement. However, switch statements are a good choice when it makes our code more concise and easier to read.

Previous (/intermediate-javascript/object-oriented-javascript/introduction-to-whiteboarding)
Next (/intermediate-javascript/object-oriented-javascript/further-exploration-local-storage)

Lesson 28 of 33
Last updated March 23, 2023

(http://www.epicodus.com)