Javascript Basics

The switch statement

When you need to check a value or result of an expression multiple times and there's potentially overlap between different conditions, you can end up using a lot of `if` statements.

Let's say for example you're writing a function that outputs info about the type of shape passed in:

```
function displayShapeInfo (shape) {
  console.log(shape + ':')

  if (shape === 'circle') {
    console.log('The shape has no sides')
    return
  }

  if (shape === 'triangle') {
    console.log('The shape has three sides')
    return
  }

  if (shape === 'rectangle' || shape === 'square') {
    if (shape === 'square') {
      console.log('The sides are equal length (square)')
    }
    console.log('The shape is rectangular')
    return
  }

  console.log('The shape is perhaps a polygon of some kind')
}

displayShapeInfo('circle')
displayShapeInfo('triangle')
displayShapeInfo('rectangle')
displayShapeInfo('square')
displayShapeInfo('trapezohedron')
```

It's a lot of code considering the if statements are all just checking the value of the shape string. The logic that handles rectangles and squares is a little confusing as well. We shouldn't need to check square twice.

In cases like this, we can use the switch statement to reduce the amount of code.

The switch statement takes a value or result of an expression and compares it with each specified `case` value. If a case matches, it executes all subsequent code (including code under other `case` clause). They typically look something like this:

```
var value = 2
switch (value) {
  case 2:
    console.log('The value is 2')
    break
  case 3:
    console.log('The value is 3')
}
```

The syntax for case statements is a bit exotic. Case clauses are not defined with curly brackets like if conditions and loops are, nor are parentheses required around each case. Instead, a colon specifies that all the code after the case will need to be executed.

The case clause should be considered as the "right side of a strictly equals operand". These code samples are identical:

```
var a = 2
if (a === 2) {
  console.log('a is 2')
}

var a = 2
switch (a) {
  case 2:
    console.log('a is 2')
}
```

Back to the original example: notice the break statement after the first case. As stated, the default behavior of the case clause is to execute all the code after the case, even if it's code that comes after other cases. For example:

```
var value = 2
switch (value) {
  case 2:
    console.log('The value is 2')
  case 3:
    console.log('The value is 3')
}
```

Notice that both console messages appear.

Just like in loops, the break statement exits out of the switch statement immediately. Any code after it, including more code matching the case clause, is skipped.

```
var value = 2
switch (value) {
  case 2:
    console.log('The value is 2')
    break
    console.log('Not called')
  case 3:
    console.log('The value is 3')
}
```

The switch statement can support identical case statements, but bear in mind that any code between the curly brackets in the switch statement stops executing once it hits the break statement. I have never seen code that used switch statements with repeating cases, and linters/static analysis tools like StandardJS actually warn you about that.

Also, code under case clauses do not exist in their own scope, even with ES6 variable constructs.

```
var value = 2
switch (value) {
  case 2:
    var value = 3
}
console.log(value) // 3
```

One way to get around this is to wrap the case in a self-executing function expression, or have the case call a function outside of the scope:

```
var value = 2
switch (value) {
  case 2:
    (function () {
      var value = 3
    })()
}
console.log(value) // 2
```

So what if the value or expression being compared does not match any cases? In that scenario, you can use the `default` keyword.

```
var value = 4
switch (value) {
  case 2:
    console.log('The value is 2')
    break
  case 3:
    console.log('The value is 3')
    break
  default:
    console.log('The value is something else')
}
```

Default matches any case that was not defined. Traditionally it is placed at the bottom of the switch statement, similar to an else statement. Regardless of where it is positioned, the code under it will be executed if no other cases match. If you do add it before other cases, make sure to add a break statement, otherwise the case under it will also get invoked.

For example:

```
var value = 3
switch (value) {
  default:
    console.log('The value is something else')
  case 2:
    console.log('The value is 2?')
    break
  case 3:
    console.log('The value is 3')
    // break not necessary because this is the last clause
}
```

This will output `3` because that case matches.

However:

```
var value = 1
switch (value) {
  default:
    console.log('The value is something else')
  case 2:
    console.log('The value is 2?')
    break
  case 3:
    console.log('The value is 3')
    // break not necessary because this is the last clause
}
```

Will output both the default case and `case 2` because default didn't break out. To fix that, just add the break statement here:

```
var value = 1
switch (value) {
  default:
    console.log('The value is something else')
    break
  case 2:
    console.log('The value is 2?')
    break
  case 3:
    console.log('The value is 3')
    // break not necessary because this is the last clause
}
```

But to avoid this confusion, just get in a habit of putting default at the bottom of the switch statement. That way your coworkers won't hate you.

There can only be one `default` clause defined per switch statement. If you try to add another, an error is thrown.

I'll rewrite the displayShapeInfo function to use switch statements:

```
function displayShapeInfo (shape) {
  console.log(shape + ':')
  switch (shape) {
    case 'circle':
      console.log('The shape has no sides')
      break
```

```
    case 'triangle':
      console.log('The shape has three sides')
      break
    case 'square':
      console.log('The sides are equal length (square)')
    case 'rectangle': // eslint-disable-line no-fallthrough
      console.log('The shape is rectangular')
      break
    default:
      console.log('The shape is a polygon of some kind')
  }
}

displayShapeInfo('circle')
displayShapeInfo('triangle')
displayShapeInfo('rectangle')
displayShapeInfo('square')
displayShapeInfo('trapezohedron')
```

The eslint-disable-line no-fallthrough comment is to tell the linting tool I'm using to not display an error. Many code analysis tools assume that a case statement that doesn't have a break statement is a programmer error.

We can see that this is a lot more concise. In particular, the 'rectangle' and 'square' conditions are easier to read.