Javascript Basics

If Statements

You should now be familiar with Javascript variables and scope. We'll now move into common logical statements in Javascript, starting with the `if` statement.

Things are now about to get interesting. If you're having trouble following along, the previous videos on variables, functions, and scope will hopefully be able to clarify.

First, the `if` statement. The if statement construct executes a specified section of code if the condition between the parentheses after it is "truthy".

```
if (true) {
  console.log('I show up')
}
if (false) {
  console.log('I do not show up')
}
```

But what does truthy mean?

In Javascript, a "truthy" value is one that, if cast into a Boolean type, would be true. Most defined values that are converted to a Boolean type will become "true", with these exceptions:

- The number 0
- An empty string
- null
- NaN (not a number)

You can experiment by using the Boolean constructor function.

```
Boolean('something')
Boolean('') // empty string
```

Want to see some cool Javascript Ninjutsu? You can also perform the same check with two logical NOT operators, the exclamation mark or bang sign ("!")

```
!!('something')
!!('')
```

And you can get rid of the parentheses since it isn't a function:

```
!!'something'
!!''
```

The NOT operator inverts whatever the boolean representation of the value after it (the operand) is. Inverting the inverted boolean is the same as casting the value to a boolean outright. This is actually a relatively common practice in many Javascript code bases.

When two values are compared with the double or triple equals operator, the result is a boolean value, which is truthy or falsey. Let's create a new if statement that checks to see if a variable is specified:

```
var name = 'Sarah'
if (name) {
  console.log('Name is specified')
}
```

Before going further, I need to mention some things about scoping in if statement blocks. Unlike contexts in functions, variables defined with `var` do not contain their own scope in if statements. So if you do this:

```
var name = 'Sarah'
if (name) {
  var name = 'Different name'
  console.log('Name is specified')
}
console.log(name) // 'Different name'
```

The name variable will change.

ES6 introduces alternative keywords for declaring variables even within the if statement: `let` and `const`:

```
var name = 'Sarah'
if (name) {
  let name = 'Different name'
  console.log('Name is specified')
}
console.log(name) // 'Sarah'
```

But we'll discuss that further once we're done going over Javascript basics.

Now, what if we want to output something else if the name isn't specified? We *could* use the NOT operator we talked about earlier and make another if statement:

```
var name = ''
if (name) {
  console.log('Name is specified')
}
if (!name) {
  console.log('Name is not specified')
}
```

But there's another option: the `else` statement.

The `else` statement construct follows the if statement's end curly bracket, and executes a section of code if the if condition is not met.

```
var name = ''
if (name) {
  console.log('Name is specified')
} else {
  console.log('Name is not specified')
}
```

`if` statements can be nested. Let's say that we're making form validation code that cannot support names that are longer than 10 characters. The string data type has a length property similar to that of the array's length property that specifies how many characters are in the string. We can use the greater than or less than operators to compare length:

```
var name = ''
if (name) {
  if (name.length > 10) {
    console.log('Name is too long')
  } else {
    console.log('Name is specified and is just the right length')
  }
} else {
  console.log('Name is not specified')
}
```

Now let's say that for whatever reason, the name needs to be at least 2 characters long. We can use an `else if` statement to specify a condition and a section of code that will be hit if the if condition above it is not met.

```
var name = ''
if (name) {
  if (name.length > 10) {
    console.log('Name is too long')
  } else if (name.length < 2) {
    console.log('Name is too short')
  } else {
    console.log('Name is specified and is just the right length')
  }
} else {
  console.log('Name is not specified')
}
```

There's nothing magical about the greater or less than symbols. They just compare two values and return a boolean.

```
var name = ''
if (name) {
  var nameLengthGreaterThan10 = name.length > 10
  var nameLengthLessThan2 = name.length < 2
  if (nameLengthGreaterThan10) {
    console.log('Name is too long')
  } else if (nameLengthLessThan2) {
    console.log('Name is too short')
  } else {
    console.log('Name is specified and is just the right length')
  }
} else {
  console.log('Name is not specified')
}
```

You can also use the logical AND and logical OR operators to check multiple conditions at once in one if conditional statement.

The logical AND operator, two ampersands, checks if the former value or expression is truthy. If it can't, it returns the former value. If it can, it returns the second value or expression.

For example:

```
'aaa' && 2
```

Returns 2 because when 'aaa' is converted into a boolean, it is true.

However:

```
0 && 'something'
```

Returns 0 because when 0 is converted into a boolean it is false.

Remember that Javascript internally compares the boolean representation of whatever is in the conditional's parentheses is, and if it's true, executes the statement. So if we wanted to combine the check to see if the name is the correct length, we could write:

```
var name = ''
if (name) {
  if (name.length > 2 && name.length < 10) {
    console.log('Name is specified and is just the right length')
  } else {
    console.log('The name is not the correct length')
  }
} else {
  console.log('Name is not specified')
}
```

Also, if it makes it easier to read, you can break conditionals up on multiple lines, and even add extra parentheses to organize things.

```
var name = ''
if (name) {
  if (
    (
            name.length > 2 &&
      name.length < 10
    )
  ) {
    console.log('Name is specified and is just the right length')
  } else {
    console.log('The name is not the correct length')
  }
} else {
  console.log('Name is not specified')
}
```

So, under the hood, those checks look a little like this:

```
var name = ''
if (name) {
  if (
    Boolean(
        Boolean(name.length > 2) &&
          Boolean(name.length < 10)
        )
  ) {
    console.log('Name is specified and is just the right length')
  } else {
    console.log('The name is not the correct length')
  }
} else {
  console.log('Name is not specified')
}
```

So, what if instead of checking to see if the length of the name was between 2 and 10, we wanted to ensure it was NOT between 2 and 10 without relying on an else statement? We could use the logical NOT operator again here:

```
var name = ''
if (name) {
  if (
    !(
        name.length > 2 &&
          name.length < 10
        )
  ) {
    console.log('The name is not the correct length')
  } else {
    console.log('Name is specified and is just the right length')
  }
} else {
  console.log('Name is not specified')
}
```

Then if what is between the parentheses following the NOT operator is true, it is converted to false.

But that's a little confusing. We could instead use the logical OR operator.

The OR operator works like this: if the expression or value before it is truthy, it returns that value. If it isn't, ir returns the value or expression after it. So this:

```
'aaa' || 2
```

Returns 'aaa' because 'aaa' is truthy.

Where this:

```
0 || 'something'
```

Returns 'something' because 0 is falsey.

Knowing that, let's change the expression to check if either the length is less than 2 or greater than 10, and if that's the case, show the invalid length warning:

```
var name = ''
if (name) {
  if (name.length < 2 || name.length > 10) {
    console.log('Name is specified and is just the right length')
  } else {
    console.log('The name is not the correct length')
  }
} else {
  console.log('Name is not specified')
}
```

One other note about the logical AND and OR operators. The values on either side can be replaced with expressions. For example, let's say we used a function to get the name's length as opposed to using the string's length property:

```
var name = ''
function getLength (name) {
  return name.length
}
if (name) {
  if (getLength(name) < 2 || getLength(name) > 10) {
    console.log('Name is specified and is just the right length')
  } else {
    console.log('The name is not the correct length')
  }
} else {
  console.log('Name is not specified')
}
```

The AND and the OR operators will evaluate the expression (a function call) only if it is necessary to do so. For example, in this example, the OR operator checks if the result of calling getLength() the first time is less than 2. Only if it is does it invoke the second getLength() call.

So if I made the length one character long, and made getLength output something in the console, we would see that the getLength function is only called once:

```
var name = 'a'
function getLength (name) {
  console.log('(getLength called)')
  return name.length
}
if (name) {
  if (getLength(name) < 2 || getLength(name) > 10) {
    console.log('Name is specified and is just the right length')
  } else {
    console.log('The name is not the correct length')
  }
} else {
  console.log('Name is not specified')
}
```

Where if I made it more than two, it would get called again to ensure it was less than 10:

```
var name = 'abc'
function getLength (name) {
  console.log('(getLength called)')
  return name.length
}
if (name) {
  if (getLength(name) < 2 || getLength(name) > 10) {
    console.log('Name is specified and is just the right length')
  } else {
    console.log('The name is not the correct length')
  }
} else {
  console.log('Name is not specified')
}
```

This is called **short circuiting**.

And again, there's no magic in these if conditions. You can use any of these operators, the greater and less than, the equality operator, and the logical AND or OR operators outside of if conditions.

We went over a lot, so I'll stop here for now. We learned about the if, else if, and else statements and how they check conditions. We also went over how the logical AND, OR, and NOT operators behave.

Using these concepts, we'll move on to for loops next.