

COMP30680

Web Application Development

JavaScript part 3 – Conditional statements and loops

David Coyle

d.coyle@ucd.ie

Booleans

The ability to test the Boolean value of an expression is fundamental for JavaScript comparisons and conditionals.

Most programming languages have a data type that can only have one of two values, e.g

YES / NO

ON / OFF

TRUE / FALSE

JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.

It also has a Boolean() function, which returns whether an expression is true or false:

```
Boolean(10 > 9)           // returns true
```

This can be shortened to :

```
(10 > 9)                   // also returns true  
10 > 9                     // also returns true
```

Booleans

The ability to test the Boolean value of an expression is fundamental for JavaScript comparisons and conditionals.

Most programming languages have a data type that can only have one of two values, e.g

YES / NO

ON / OFF

TRUE / FALSE

JavaScript has a **Boolean** data type. It can only take the values **true** or **false**.

It also have a Boolean() function, which returns whether an expression is true or false:

```
Boolean(10 > 9) // returns true
```

This is a comparison operator.
In this case greater than

This can be shortened to :

```
(10 > 9) // also returns true  
10 > 9 // also returns true
```

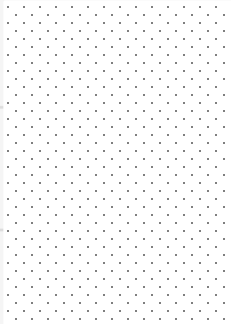
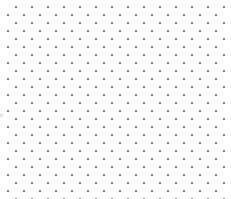


Comparison Operators

Alongside Logical operators, which we will see shortly, Comparison operators are used to test for *true* or *false*.

Comparison operators determine the equality or difference between variables or values.

Assume `x = 5`;

Operator	Description	Comparing	Returns
==	equal to	<code>x == 8</code>	
		<code>x == 5</code>	
		<code>x == "5"</code>	
===	equal value and equal type	<code>x === 5</code>	
		<code>x === "5"</code>	

Comparison Operators

Again assume `x = 5`;

<code>!=</code>	not equal	<code>x != 8</code>	true
<code>!==</code>	not equal value or not equal type	<code>x !== 5</code>	false
		<code>x !== "5"</code>	true
		<code>x !== 8</code>	true
<code>></code>	greater than	<code>x > 8</code>	false
<code><</code>	less than	<code>x < 8</code>	true
<code>>=</code>	greater than or equal to	<code>x >= 8</code>	false
<code><=</code>	less than or equal to	<code>x <= 8</code>	true

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that **x = 6** and **y = 3**, the table below explains the logical operators:

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
	or	(x == 5 y == 5) is false
!	not	!(x == y) is true

Conditional (Ternary) Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

```
variablename = (condition) ? value1:value2
```

Conditional Statements

In JavaScript we have the following conditional statements:

- Use **if** to specify a block of code to be executed, if a specified condition is true
 - Use **else** to specify a block of code to be executed, if the same condition is false
 - Use **else if** to specify a new condition to test, if the first condition is false
-
- Use **switch** to specify many alternative blocks of code to be executed

These are
often used
together

http://www.w3schools.com/js/js_if_else.asp

If ... else statements

An **if statement** specifies a block of code to execute if a condition is true:

```
if (condition) {  
    block of code to be executed if the condition is true  
}
```

An **else statement** is optional. It specifies a block of code to execute if the condition is false:

```
if (condition) {  
    block of code to be executed if the condition is true  
} else {  
    block of code to be executed if the condition is false  
}
```


If ... else example

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to display a time-based greeting:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
    var hour = new Date().getHours();
    var greeting;
    if (hour < 18) {
        greeting = "Good day";
    } else {
        greeting = "Good evening";
    }
    document.getElementById("demo").innerHTML = greeting;
}
</script>

</body>
</html>
```

If ... else statements

The **else if** statement allows you to specify more than one test condition. These additional conditions are checked if the first condition is not true (and so on for further if .. else statements):

```
if (condition1) {  
    block of code to be executed if condition1 is true  
} else if (condition2) {  
    block of code to be executed if the condition1 is false and condition2 is true  
} else {  
    block of code to be executed if the condition1 is false and condition2 is false  
}
```

Long lists of if ... else if ... statements can become a bit cumbersome. The **Switch statement** provides an alternative.

Switch statements

The switch statement allows you to perform a range of different actions based on a range of different conditions.

```
switch(expression) {  
    case n:  
        code block  
        break;  
    case n:  
        code block  
        break;  
    default:  
        default code block  
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

When execution reaches a **break** keyword, it breaks out of the switch block.

The **default** keyword specifies the code to run if there is no case match:

Loops

Loops allow you to execute the same block of code multiple times, but with a different value each time. They save you having to write the same code over and over again!

JavaScript supports different kinds of loops:

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

For loop

Loops through a block of code a number of times. It has the following syntax:

```
for (statement 1; statement 2; statement 3) {  
    code block to be executed  
}
```

Statement 1 is executed before the loop (the code block) starts.

Normally you will use statement 1 to initialize the variable used in the loop ($i = 0$).

Statement 2 defines the condition for running the loop (the code block).

Often statement 2 is used to evaluate the condition of the initial variable.

Statement 3 is executed each time after the loop (the code block) has been executed.

Often statement 3 increments the value of the initial variable.

For loop

```
text += cars[0] + "<br>";  
text += cars[1] + "<br>";  
text += cars[2] + "<br>";  
text += cars[3] + "<br>";  
text += cars[4] + "<br>";  
text += cars[5] + "<br>";
```



```
for (i = 0; i < cars.length; i++) {  
    text += cars[i] + "<br>";  
}
```

This is just six cars, it could be thousands or millions.

While loop

The while loop loops through a block of code as long as a specified condition is true.

```
while (condition) {  
    code block to be executed  
}
```

```
<p id="demo"></p>  
  
<script>  
var text = "";  
var i = 0;  
while (i < 10) {  
    text += "<br>The number is " + i;  
    i++;  
}  
document.getElementById("demo").innerHTML = text;  
</script>
```

The Do/While Loop

The do/while loop is a variant of the while loop.

```
do {  
    code block to be executed  
}  
while (condition);
```

The difference: This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
<script>  
var text = ""  
var i = 0;  
  
do {  
    text += "<br>The number is " + i;  
    i++;  
}  
while (i < 10);  
  
document.getElementById("demo").innerHTML = text;  
</script>
```

```
The number is 0  
The number is 1  
The number is 2  
The number is 3  
The number is 4  
The number is 5  
The number is 6  
The number is 7  
The number is 8  
The number is 9
```


Questions, Suggestions?

Next class:

JavaScript part 4 - Objects