Chapter 59: Variable coercion/conversion

Section 59.1: Double Negation (!!x)

The double-negation `!!` is not a distinct JavaScript operator nor a special syntax but rather just a sequence of two negations. It is used to convert the value of any type to its appropriate `true` or `false` Boolean value depending on whether it is *truthy* or *falsy*.

```
!!1             // true
!!0             // false
!!undefined     // false
!!{}            // true
!![]            // true
```

The first negation converts any value to `false` if it is *truthy* and to `true` if is *falsy*. The second negation then operates on a normal Boolean value. Together they convert any *truthy* value to `true` and any *falsy* value to `false`.

However, many professionals consider the practice of using such syntax unacceptable and recommend simpler to read alternatives, even if they're longer to write:

```
x !== 0         // instead of !!x in case x is a number
x != null       // instead of !!x in case x is an object, a string, or an undefined
```

Usage of `!!x` is considered poor practice due to the following reasons:

1. Stylistically it may look like a distinct special syntax whereas in fact it is not doing anything other than two consecutive negations with implicit type conversion.
2. It is better to provide information about types of values stored in variables and properties through the code. For example, `x !== 0` says that `x` is probably a number, whereas `!!x` does not convey any such advantage to readers of the code.
3. Usage of `Boolean(x)` allows for similar functionality, and is a more explicit conversion of type.

Section 59.2: Implicit conversion

JavaScript will try to automatically convert variables to more appropriate types upon use. It's usually advised to do conversions explicitly (see other examples), but it's still worth knowing what conversions take place implicitly.

```
"1" + 5 === "15" // 5 got converted to string.
1 + "5" === "15" // 1 got converted to string.
1 - "5" === -4 // "5" got converted to a number.
alert({}) // alerts "[object Object]", {} got converted to string.
!0 === true // 0 got converted to boolean
if ("hello") {} // runs, "hello" got converted to boolean.
new Array(3) === ",,"; // Return true. The array is converted to string - Array.toString();
```

Some of the trickier parts:

```
!"0" === false // "0" got converted to true, then reversed.
!"false" === false // "false" converted to true, then reversed.
```

Section 59.3: Converting to boolean

`Boolean(...)` will convert any data type into either `true` or `false`.

```
Boolean("true") === true
Boolean("false") === true
Boolean(-1) === true
Boolean(1) === true
Boolean(0) === false
Boolean("") === false
Boolean("1") === true
Boolean("0") === true
Boolean({}) === true
Boolean([]) === true
```

Empty strings and the number 0 will be converted to false, and all others will be converted to true.

A shorter, but less clear, form:

```
!!"true" === true
!!"false" === true
!!-1 === true
!!1 === true
!!0 === false
!!"" === false
!!"1" === true
!!"0" === true
!!{} === true
!![] === true
```

This shorter form takes advantage of implicit type conversion using the logical NOT operator twice, as described in http://stackoverflow.com/documentation/javascript/208/boolean-logic/3047/double-negation-x

Here is the complete list of boolean conversions from the ECMAScript specification

- if `myArg` of type `undefined` or `null` then `Boolean(myArg) === false`
- if `myArg` of type `boolean` then `Boolean(myArg) === myArg`
- if `myArg` of type `number` then `Boolean(myArg) === false` if `myArg` is `+0`, `-0`, or `NaN`; otherwise `true`
- if `myArg` of type `string` then `Boolean(myArg) === false` if `myArg` is the empty String (its length is zero); otherwise `true`
- if `myArg` of type `symbol` or `object` then `Boolean(myArg) === true`

Values that get converted to `false` as booleans are called *falsy* (and all others are called *truthy*). See Comparison Operations.

Section 59.4: Converting a string to a number

```
Number('0') === 0
```

`Number('0')` will convert the string (`'0'`) into a number (`0`)

A shorter, but less clear, form:

```
+'0' === 0
```

The unary `+` operator does nothing to numbers, but converts anything else to a number.
Interestingly, `+(-12) === -12`.

```
parseInt('0', 10) === 0
```

`parseInt('0', 10)` will convert the string (`'0'`) into a number (`0`), don't forget the second argument, which is radix. If not given, parseInt could convert string to wrong number.

Section 59.5: Converting a number to a string

```
String(0) === '0'
```

`String(0)` will convert the number (`0`) into a string (`'0'`).

A shorter, but less clear, form:

```
'' + 0 === '0'
```

Section 59.6: Primitive to Primitive conversion table

Value	Converted To String	Converted To Number	Converted To Boolean
undefinded	"undefined"	NaN	false
null	"null"	0	false
true	"true"	1	
false	"false"	0	
NaN	"NaN"		**false**
"" empty string		0	**false**
" "		0	**true**
"2.4" (numeric)		2.4	true
"test" (non numeric		NaN	true
"0"		0	**true**
"1"		1	true
-0	"0"		**false**
0	"0"		false
1	"1"		true
Infinity	"Infinity"		true
-Infinity	"-Infinity"		true
[]	""	0	true
[3]	"3"	3	true
['a']	"a"	NaN	true
['a','b']	"a,b"	NaN	true
{ }	"[object Object]"	NaN	true
function(){}	"function(){}"	NaN	true

Bold values highlight conversion that programmers may find surprising

To convert explicitly values you can use String() Number() Boolean()

Section 59.7: Convert an array to a string

`Array.join(separator)` can be used to output an array as a string, with a configurable separator.

Default (separator = ","):

```
["a", "b", "c"].join() === "a,b,c"
```

With a string separator:

```
[1, 2, 3, 4].join(" + ") === "1 + 2 + 3 + 4"
```

With a blank separator:

```
["B", "o", "b"].join("") === "Bob"
```

Section 59.8: Array to String using array methods

This way may seem to be useless because you are using anonymous function to accomplish something that you can do it with join(); But if you need to make something to the strings while you are converting the Array to String, this can be useful.

```
var  arr = ['a', 'á', 'b', 'c']

function upper_lower (a, b, i) {
  //...do something here
  b = i & 1 ? b.toUpperCase() : b.toLowerCase();
  return a + ',' + b
}
arr = arr.reduce(upper_lower); // "a,Á,b,C"
```

Section 59.9: Converting a number to a boolean

```
Boolean(0) === false
```

`Boolean(0)` will convert the number `0` into a boolean **`false`**.

A shorter, but less clear, form:

```
!!0 === false
```

Section 59.10: Converting a string to a boolean

To convert a string to boolean use

```
Boolean(myString)
```

or the shorter but less clear form

```
!!myString
```

All strings except the empty string (of length zero) are evaluated to **`true`** as booleans.

```
Boolean('') === false   // is true
Boolean("") === false   // is true
Boolean('0') === false  // is false
Boolean('any_nonempty_string') === true // is true
```

Section 59.11: Integer to Float

In JavaScript, all numbers are internally represented as floats. This means that simply using your integer as a float is all that must be done to convert it.

Section 59.12: Float to Integer

To convert a float to an integer, JavaScript provides multiple methods.

The `floor` function returns the first integer less than or equal to the float.

```
Math.floor(5.7); // 5
```

The `ceil` function returns the first integer greater than or equal to the float.

```
Math.ceil(5.3); // 6
```

The `round` function rounds the float.

```
Math.round(3.2); // 3
Math.round(3.6); // 4
```

`Version ≥ 6`

Truncation (`trunc`) removes the decimals from the float.

```
Math.trunc(3.7); // 3
```

Notice the difference between truncation (`trunc`) and `floor`:

```
Math.floor(-3.1); // -4
Math.trunc(-3.1); // -3
```

Section 59.13: Convert string to float

`parseFloat` accepts a string as an argument which it converts to a float/

```
parseFloat("10.01") // = 10.01
```