Section 3 - Operators, Type Conversion, and String Methods

Learning Outcomes

This section is an overview of various operators in JavaScript. This section also provides an overview of several global methods to convert string variables to numbers and vice versa. There is also an overview of several string methods to determine the length, sections of a string, as well as replacing text in a string.

- **Operators** - assignment, arithmetic, comparison, and logical
- **Type Conversion** - converting between strings and numbers
- **String Methods** - common methods used when working with strings
- **String Templates** - a powerful alternative to string concatenation

Resources

- Operators - https://www.w3schools.com/js/js_operators.asp
- Type Conversion - https://www.w3schools.com/js/js_type_conversion.asp
- String Methods - https://www.w3schools.com/js/js_string_methods.asp
- String Templates - https://www.w3schools.com/js/js_string_templates.asp

Operators

Assignment Operator

The assignment operator (`=`) assigns a value to a variable.

```
let x = 10;
```

Arithmetic Operators

Arithmetic operators are used to perform arithmetic on numbers.

```
let x = 5;
let y = 2;
let z = x + y;
```

Operator	Description
+	Addition
-	Subtraction
*	Multiplication

Operator	Description
**	Exponentiation (ES2016)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

More Info: https://www.w3schools.com/js/js_arithmetic.asp

Operator Precedence

Operator precedence describes the order in which operations are performed in an arithmetic expression.

```
let x = 100 + 50 * 3;
```

Is the result of example above the same as 150 * 3, or is it the same as 100 + 150?

Is the addition or the multiplication done first?

As in traditional school mathematics, the multiplication is done first.

Multiplication (*) and division (/) have higher precedence than addition (+) and subtraction (-). And (as in school mathematics) the precedence can be changed by using parentheses:

```
let x = (100 + 50) * 3;
```

When using parentheses, the operations inside the parentheses are computed first.

When many operations have the same precedence (like addition and subtraction), they are computed from left to right:

```
let x = 100 + 50 - 3;
```

String Operators

The `+` operator can also be used to `add` (`concatenate`) strings.

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

Output:

```
John Doe
```

Assignment Operators

Assignment operators can be combined with arithmetic operators when assigning values to JavaScript variables.

Operator	Description	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y

Examples

```
let x = 10;
x += 5;    // the value of x is now 15
```

```
let y = 10;
y *= 5;    // the value of y is now 50
```

Comparison Operators

Comparison and operators are used to test for `true` or `false`.

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary

More Info: https://www.w3schools.com/js/js_comparisons.asp

Examples

```
if (age < 18) text = "Too young to buy alcohol";
```

Ternary Operator

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

Syntax

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

Example

```
let voteable = (age < 18) ? "Too young" : "Old enough";
```

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
&&	logical `and`	(x < 10 && y > 1) is true
||	logical `or`	(x == 5 || y == 5) is false
!	logical `not`	!(x == y) is true

More Info: https://www.w3schools.com/js/js_comparisons.asp

Type Conversion

JavaScript variables can be converted to a new variable and another data type:

- By the use of a JavaScript function
- Automatically by JavaScript itself

Converting Strings to Numbers

The global method `Number()` converts a string to a number.

Strings containing numbers (like "3.14") convert to numbers (like 3.14).

Empty strings convert to 0. Anything else converts to `NaN` (Not a Number).

```
Number("3.14")    // returns 3.14
Number(" ")       // returns 0
Number("")        // returns 0
Number("99 88")   // returns NaN
```

In the chapter Number Methods, you will find more methods that can be used to convert strings to numbers:

Method	Description
`Number()`	Returns a number, converted from its argument
`parseFloat()`	Parses a string and returns a floating point number
`parseInt()`	Parses a string and returns an integer

Converting Numbers to Strings

The global method `String()` can convert numbers to strings. It can be used on any type of numbers, literals, variables, or expressions:

```
String(x)         // returns a string from a number variable x
String(123)       // returns a string from a number literal 123
String(100 + 23)  // returns a string from a number from an expression
```

In the chapter Number Methods, you will find more methods that can be used to convert numbers to strings:

Method	Description
`toExponential()`	Returns a string, with a number rounded and written using exponential notation.
`toFixed()`	Returns a string, with a number rounded and written with a specified number of decimals.
`toPrecision()`	Returns a string, with a number written with a specified length

String Methods

String Length

The `length` property returns the `length` of a string.

```
let alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let alphabetLength = alphabet.length;
```

Extracting String Parts

There are 3 methods for extracting a part of a string:

- `slice(start, end)`
- `substring(start, end)`
- `substr(start, length)`

`slice()`: extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters: the start position, and the end position (end not included).

This example slices out a portion of a string from position 7 to position 12 (13-1):

Consider the the characters in a string can be referenced by an index number.

A	p	p	l	e	,		B	a	n	a	n	a	,		K	i	w	i
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

```
let str = "Apple, Banana, Kiwi";
let fruit = str.slice(7, 13);  // the value of part is 'Banana'
```

`substring()`: is similar to `slice()`. The difference is that `substring()` cannot accept negative indexes.

```
let str = "Apple, Banana, Kiwi";
let part = str.substring(7, 13);  // the value of part is 'Banana'
```

`substr()`: is similar to `slice()`. The difference is that the second parameter specifies the `length` of the extracted part.

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(7, 6);  // the value of part is 'Banana'
```

Replacing String Content

The `replace()` method replaces a specified value with another value in a string.

```
let text = "Please visit Microsoft!";
let newText = text.replace("Microsoft", "W3Schools");  // the value of newText is "Please visit W3Schoo
```

Converting to Upper and Lower Case

A string is converted to upper case with `toUpperCase()`

```
let text1 = "Hello World!";
let text2 = text1.toUpperCase();  // the value of text2 is "HELLO WORLD!"
```

A string is converted to lower case with `toLowerCase()`

```
let text1 = "Hello World!";
let text2 = text1.toLowerCase();  // the value of text2 is "hello world!"
```

String `concat()`

`concat()` joins two or more strings

```
let text1 = "Hello";
let text2 = "World";
let text3 = text1.concat(" ", text2);  // the value of text3 is "Hello World"
```

String `trim()`

The `trim()` method removes whitespace from both sides of a string.

```
let text1 = "      Hello World!      ";
let text2 = text1.trim();    // the value of text2 is "Hello World"
```

String Templates

Template Literals use back-ticks (`) rather than the quotes ("") to define a string:

```
let text = `Hello World!`;
```

With template literals, you can use both single and double quotes inside a string:

```
let text = `He's often called "Johnny"`;
```

```
let text =
`The quick
brown fox
jumps over
the lazy dog`;
```

Interpolation

Template literals provide an easy way to interpolate variables and expressions into strings. The method is called `string interpolation`.

The syntax is:

```
${...}
```

Template literals allow variables in strings:

```
let firstName = "John";
let lastName = "Doe";
let text = `Welcome ${firstName}, ${lastName}!`;
```

Template literals allow expressions in strings:

```
let price = 10;
let VAT = 0.25;
let total = `Total: ${(price * (1 + VAT)).toFixed(2)}`;
```