Programming Foundations - Module 1

Noroff Guide

Noroff Education

Table of Contents

Module overview

Introduction

Welcome to Module 1 of Programming Foundations.

If anything is unclear, check your progression plan and/or contact a tutor on Discord.

Module structure	**Estimated workload**
1.1. Lesson and task Your first application	8 hours
1.2. Lesson and task Strings	8 hours
1.3. Lesson and task Alerts and debugging	8 hours
1.4. Lesson and task Loops	8 hours
1.5. Lesson and task Self-study	8 hours

Learning outcomes

In this module, we are covering the following knowledge learning outcomes:

→ The candidate has knowledge of primary concepts, core ideas and general basic methods within programming.

→ The candidate has knowledge of
  program syntax, program structure, control structures, data types, a
  nd variables as used in the JavaScript language.
→ The candidate has knowledge of development and debugging within a browser.
→ The candidate has knowledge of industry relevant software used for writing JavaScript code.
→ The candidate can
  update their knowledge of basic programming concepts.

In this module, we are covering the following skill learning outcomes:

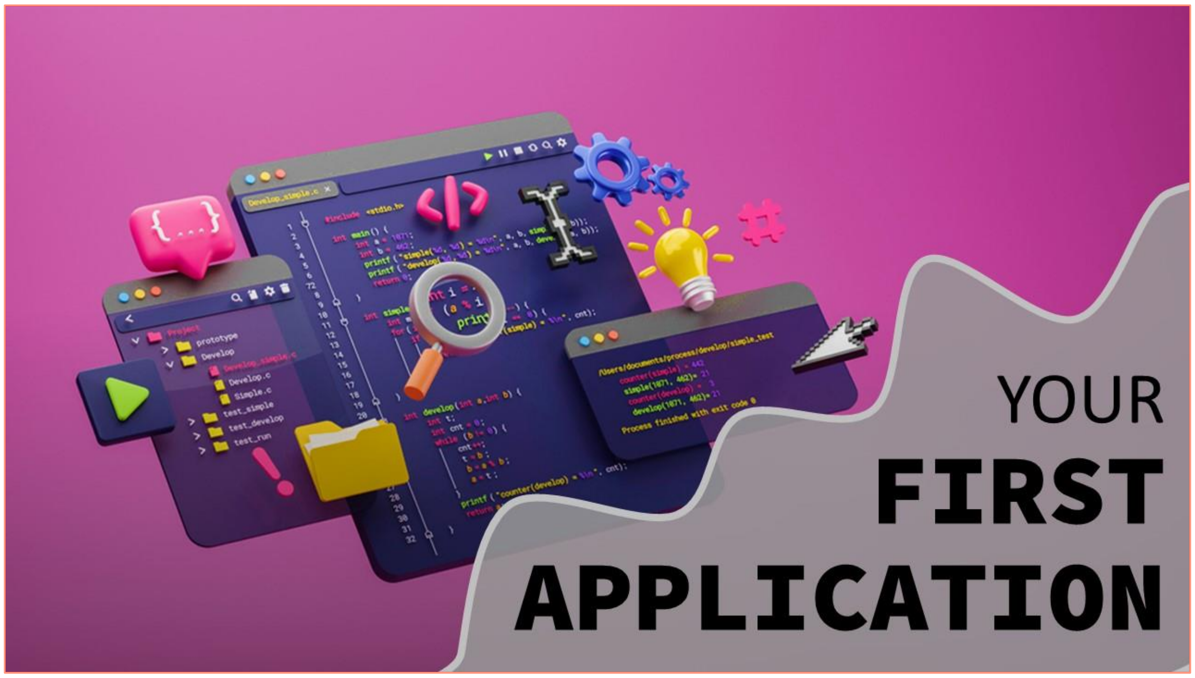
→ The candidate can apply knowledge of program syntax, program structure, control structures, data types, and variables to complete simple tasks using the JavaScript language.
→ The candidate can apply knowledge of browsers and development tools to set up a computer to develop and debug small JavaScript programs.
→ The candidate masters the use of relevant tools for writing, editing and debugging JavaScript source code.

In this module, we are covering the following general competence learning outcome:

→ The candidate can carry out simple programming tasks with the JavaScript language.

1.1. Lesson - Your first application

Introduction



The standard website consists of HTML templates, CSS styles, and JavaScript scripts.

HTML presents static content. CSS is responsible for the modern and smooth view of our HTML elements, while JavaScript (JS) is a programming language enabling us to create dynamic content on websites.

Visual Studio Code is the most popular code editor for web application development (and for many other applications like data science applications written in Python). It's available to download for free under a standard MIT license. It contains numerous extensions which heavily improve the coding experience.

This lesson guides through the installation process of Visual Studio Code and presents a basic website structure. Examples of a standard web application are presented and explained.

The lesson also provides an introduction to computer programming. We'll learn about variables and the basic operations we can exert on them.

Variables are containers for data. Before a value is assigned to a variable, it's like an empty whiteboard. However, the caveat is that we have different types of whiteboards for different kinds of data. Imagine we have whiteboards on which only numbers can be written and those on which only character strings can be written.

The same applies to variables. Hence, with each variable, a type is associated. We are free to choose this type when we construct a variable, and we can change it after that, but if a variable has a given value assigned, its type is uniquely determined.

Assigning a value to a variable is like writing on the whiteboard. When we want to change a variable's value, we simply assign a new value to it. It's like overwriting what has previously been written on the whiteboard. It's still the same whiteboard, but with a new value written on it.

We can also compare variables to Lego bricks since every variable is (usually) different, and we can combine some variables to construct new ones. It's like introductory algebra. When we have two variables, say x and y, we can define a variable $z = x + y$.

Variables can have various types. In this lesson, we'll focus on numeric types: Numbers and decimals. We'll go through basic mathematical operations such as addition, subtraction, multiplication and division.

There are some other useful operators like type, logical and bitwise operators. We'll also mention some examples.

Learning outcomes

In this lesson, we are covering the following knowledge learning outcomes:

→ The candidate has knowledge of program syntax, program structure, control structures, data types, and variables as used in the JavaScript language.
→ The candidate has knowledge of industry relevant software used for writing JavaScript code.

In this lesson, we are covering the following skill learning outcomes:

- → The candidate can apply knowledge of browsers and development tools to set up a computer to develop and debug small JavaScript programs.
- → The candidate masters the use of relevant tools for writing, editing, and debugging JavaScript source code.

In this lesson, we are covering the following general competence learning outcome:

- → The candidate can carry out simple programming tasks with the JavaScript language.

Installation of Visual Studio Code

In this course, we'll learn how to program using JavaScript. Many other programming languages exist, but JavaScript is the perfect choice for creating web applications. It allows us to create numerous amazing visual effects while still creating the code necessary for our data's operations.

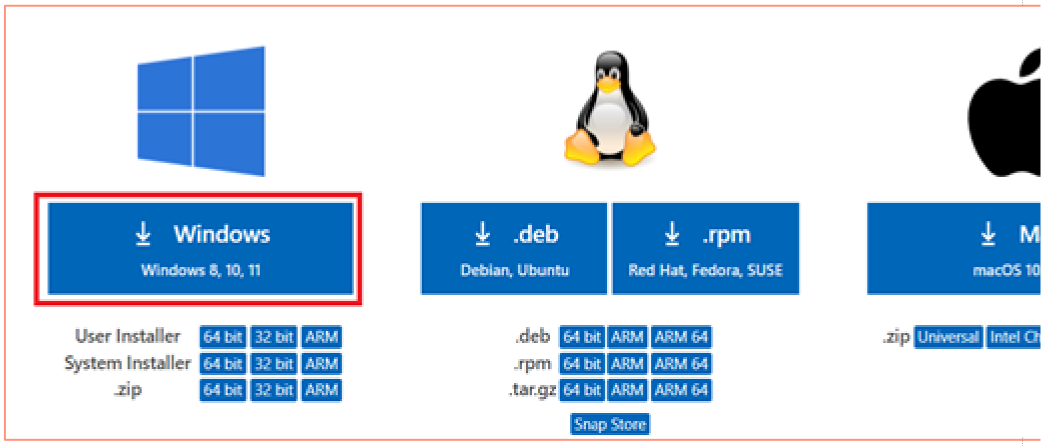
Source code for any programming language can be written in any text editor, such as Notepad or MS Word. The caveat is that some editors are specially tailored to developers who don't write books, letters or articles but scripts filled with source code necessary for developing a website or application. Such editors are called source code editors.

As a source code editor, we'll use Visual Studio Code, developed by Microsoft, very popular among developers and free. More specifically, according to a 2021 Stack Overflow Developer Survey, Visual Studio Code ranked as the most popular source code editor among professional developers and respondents. The name Visual Studio Code can be abbreviated as vsCode or VS Code.

DOWNLOAD

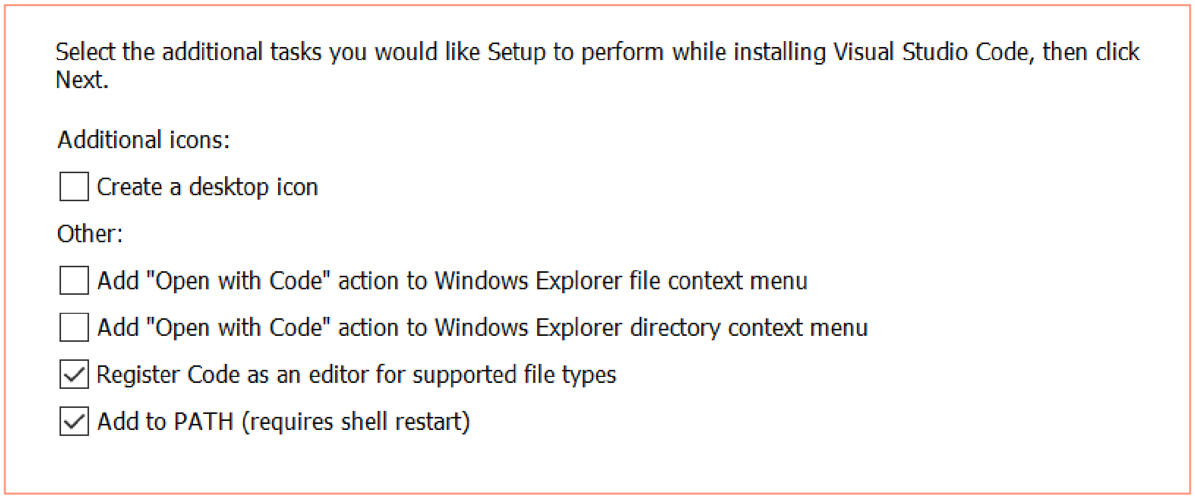
Visual Studio Code.

You can download a version for Windows, macOS and Linux. During this lesson, we'll install a Windows version. After clicking a 'Download' button from the above link (marked by a red rectangle on the image below), you'll download an installer with a name similar to or the same as this: VSCodeUserSetup-x64-1.67.2.exe.



In the next step, you should run the installer and select the language for installation. Choose English. Then you must accept a license agreement (standard MIT license, which means you can use this application free).

You then have five checkboxes:


Select the additional tasks you would like Setup to perform while installing Visual Studio Code, then click Next.

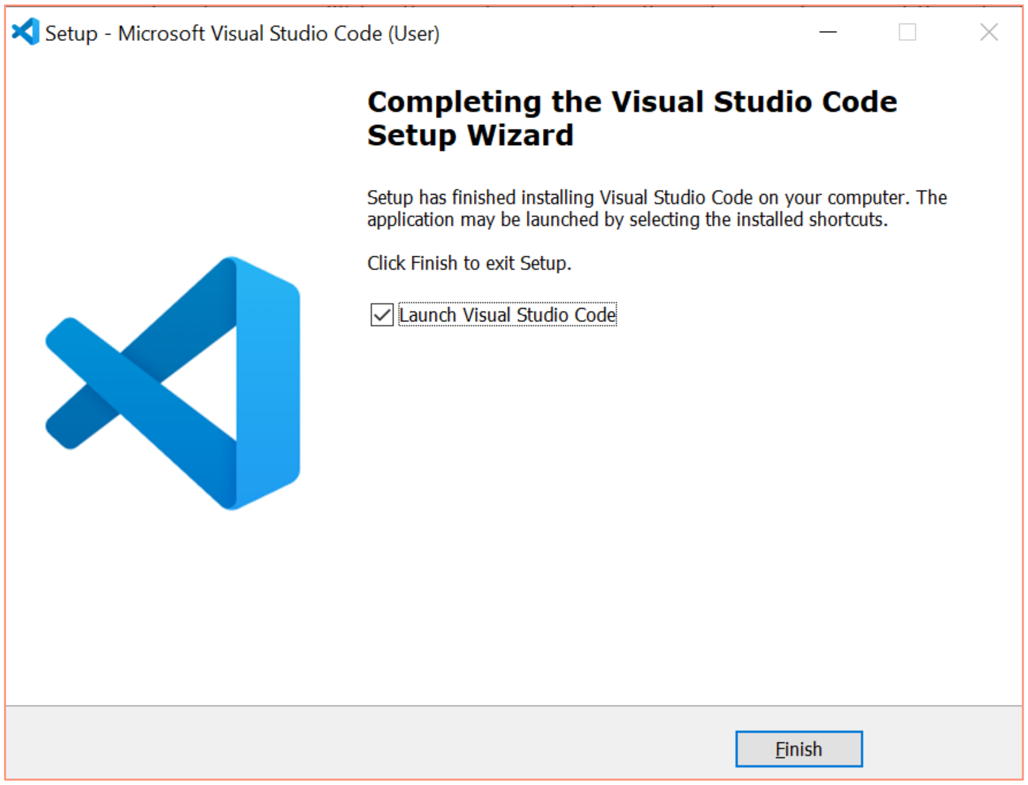
Additional icons:

- [ ] Create a desktop icon

Other:

- [ ] Add "Open with Code" action to Windows Explorer file context menu
- [ ] Add "Open with Code" action to Windows Explorer directory context menu
- [✓] Register Code as an editor for supported file types
- [✓] Add to PATH (requires shell restart)

Keeping the bottom two options checked is highly recommended, whereas the first three are up to you. The latter declares from where you'll be able to open the editor. Having the first one marked to create a desktop shortcut is convenient. Then you can click 'Install'. If everything goes correctly, you should see a message saying that the setup has finished installing Visual Studio Code:


Setup - Microsoft Visual Studio Code (User)

Completing the Visual Studio Code Setup Wizard

Setup has finished installing Visual Studio Code on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

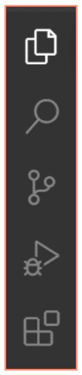
- [✓] Launch Visual Studio Code

You can finish installation and launch Visual Studio Code.

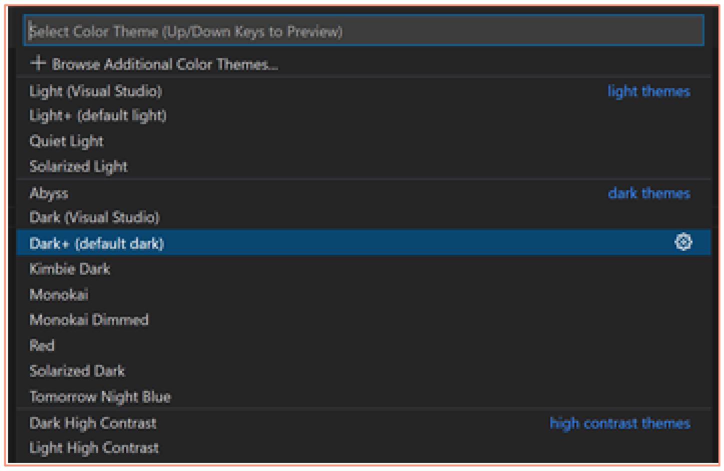
Introduction to Visual Studio Code

Visual Studio Code has numerous options to make programming as comfortable as possible. In this section, we'll go through the most important ones.

There are five buttons available on Visual Studio Code's left menu bar. Their icons are presented one by one, in a top-down arrangement, in the image below.

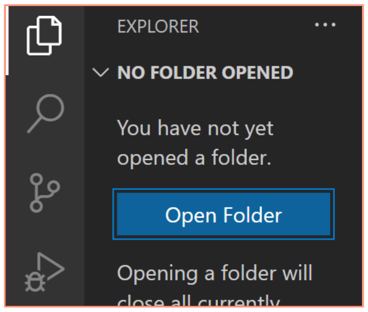
	The first one is most important for us since it allows us to see the preview of all the files in the currently opened folder.
	The second one is useful as well. It allows us to search for keywords and to replace them within all your files.
	The third one is for source control. It isn't that important, but we'll need it later.
	The fourth button is for debugging. Same as above, we'll learn when and how to use it later.
	The last one is for managing extensions. We'll go through some of the most important extensions in the next section of the current lesson.

VS Code gives many possibilities to make a custom configuration. First, we can choose a colour theme that suits us best. For example, we can try a standard one (dark+) and see if it works. If you're using Microsoft Windows, a list of all possible themes is available by typing CTRL+ K and then CTRL + T. We can look at the list of available themes and choose one (like on the image below). Shortcuts enable us to do things quickly. If you don't use Windows as your operating system, look at the shortcut list in the READ section.



Visual Studio Code allows us to configure much more options in the preferences menu (a shortcut to this is CTRL + PLUS), but default settings should do the job at the beginning.

Having chosen the basic options, we should first create a folder for our project. To do this, we click ‘Open Folder’ (as on the left image below) and create a new folder named ‘MyFirstApp’. On the right image, we see a menu to which we can easily add a new file to the folder we just created.


EXPLORER
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Opening a folder will close all currently

Let’s create our first file: Hello.html.

READ

1. Page: Getting started with Visual Studio Code by Visual Studio Code.
2. Page: Key Bindings for Visual Studio Code by Visual Studio Code.

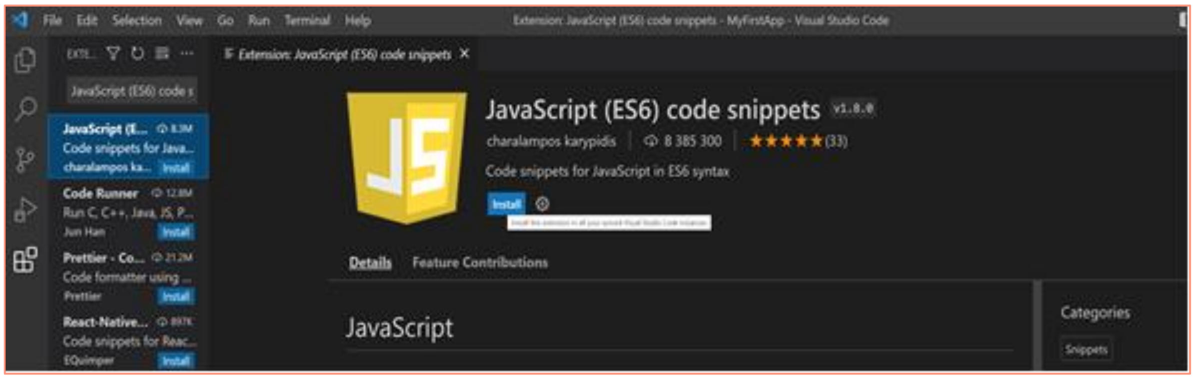
WATCH

1. Video: Learn Visual Studio Code in 7min (Official Beginner Tutorial) (7m 16s) by Visual Studio Code on YouTube.
2. Video: Visual Studio Code Tutorial for Beginners – Introduction (33m 48s) by Academind on YouTube.

Extensions

Visual Studio Code extensions are great add-ins that can be installed on top of a plain vanilla VS Code version to provide better performance and stability. Every tool has its own, so community-created extensions exist to mitigate them. All of these are usually for free. In the current section, we enlist some of the most popular extensions available for programming in JavaScript.

To install an extension, we use an extension management button. This is a bottom button on the left-hand side menu. We find a relevant extension by typing its name. Then we can install it by clicking an install button like the blue one on the print screen below. If we want to test something, we can disable or uninstall a previously installed extension whenever we want.


File
Edit
Selection
View
Go
Run
Terminal
Help
Extension: JavaScript (ES6) code snippets - MyFirstApp - Visual Studio Code
Extension: JavaScript (ES6) code snippets
JavaScript (ES6) code snippets v1.8.0
charalampos karypidis
8 385 300
★★★★★(33)
Code snippets for JavaScript in ES6 syntax
Install
JavaScript (E... 8.3M
Code snippets for Java...
charalampos ka... Install
Code Runner 12.8M
Run C, C++, Java, JS, P...
Jun Han Install
Prettier - Co... 21.2M
Code formatter using ...
Prettier Install
React-Native... 897K
Code snippets for Reac...
EQuimper Install
Details
Feature Contributions
JavaScript
Categories
Snippets

Some popular extensions include, but aren’t limited to:

- Snippet extensions – These extensions provide code snippets, i.e. short fragments of frequently used JavaScript code. With such extensions, when we type only the first fragment of a particular phrase and press the TAB button, the remaining fragment of the code is automatically generated. The most popular extension of this kind is an extension named ‘JavaScript (ES6) code snippets’ by Charalampos Karypidis. It has over three million installs.
- Syntax highlighting extensions – These extensions can be used to customise how code highlighting works in VS Code. Since default code highlighting works decently in VS Code, we don’t have to install any extension from this category for now.
- Linter extensions – This programming style has always been an issue for developers since some prefer different conventions. For example, some programmers prefer to use tabs as breaks, while others prefer spaces. It’s important, however, that we should use the same convention throughout the scope of a single project. Linter is an extension that helps with this. It checks whether the program has been written according to rules defined within a given programming style. We can read more about it in an article at the end of the current section.
- Formatting extensions – These extensions take care of code alignment (vertical formatting of code, so all the same operators in the following lines are precisely in the same positions). The most popular one is Prettier. It has over 5.7 million installs.

READ

1. Page: JavaScript extensions for VS Code by Visual Studio Code.
2. Article: 10 Must-have VS Code Extensions for JavaScript Developers by Michael Wanyoike.
3. Article: A Comparison of JavaScript Linting Tools by Jani Hartikainen.

WATCH

1. Video: 10 Helpful VS Code Extensions for JavaScript (19m 8s) by Coding Tech on YouTube.
2. Video: Top 10 Best VS Code Extensions (17m 36s) by codeSTACKr on YouTube.

HTML5 document

HTML is a standard language for creating web pages. HTML stands for HyperText Markup Language. It's responsible for putting small bricks on our side, from which we use more advanced technologies to create a fantastic website.

In this section, we'll go through the structure of the basic HTML5 file. HTML5 is the most recent version of HTML language.

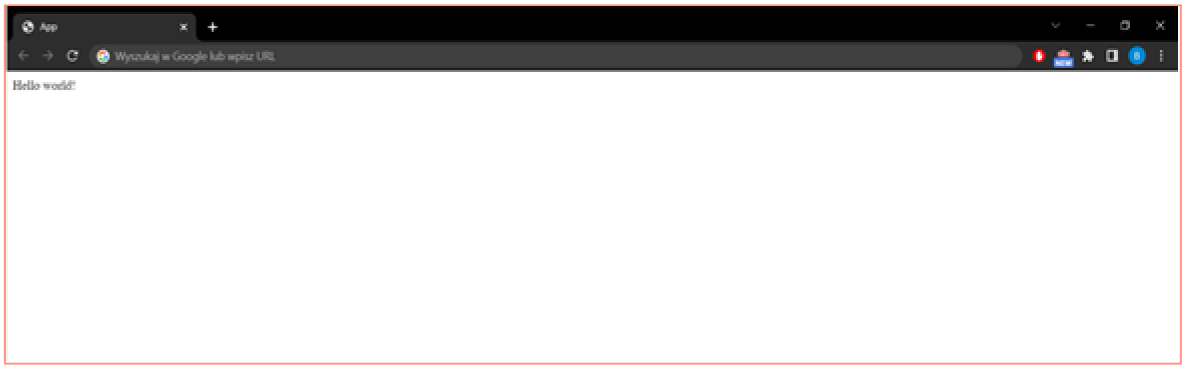
Let's open the hello.html file we previously created and write our first code example in it:

```
<!DOCTYPE html>
<html>
  <head>
        <title>App</title>
    </head>
    <body>
    Hello world!
    </body>
</html>
```

Always remember to save files after they've been edited. We can do it with a CTRL + S shortcut.

Then open the hello.html file with your browser. To do this, right-click on your file in VS Code and click Reveal in File Explorer. In File Explorer, right-click on it and choose the 'Open With' option.

Choose Google Chrome from the list. After opening the file, you should see a blank card with the title 'App' and dark text printed saying: 'Hello world!', similar to the image below:



Hello world!

We'll now go through this code line by line.

The first line is a so-called HTML signature. It informs the browser that the file is an HTML document. Older versions of HTML are more complicated:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

So, don't be surprised if you see such a longer declaration on some older web pages. It does exactly the same.

We'll name a word surrounded by angle brackets as 'tag'. 'Element' is everything between the opening and closing tag.

Next, we have an <html> tag. It represents a root of an HTML document and is a container for all other HTML elements. Note that every HTML element needs to be closed. At the very end of the program, we see an </html> tag, which corresponds to the <html> tag and closes it. If a tag has nothing inside, it can be opened and closed in one line.

So, instead of writing:

```
<tag></tag>
```

you can write:

```
<tag/>
```

An <html> tag has two children: <head> and <body>. The head element is responsible for providing the browser card's header, and the body element is responsible for providing the card's main content.

Note that <head> and <body> tags are closed by </head> and </body> tags respectively. Our header has a single child named <title> in which we set the card's title, 'App'.

In our example, the body contains just a 'Hello world!' string written in plain text. This is what a simple HTML document can look like.

ACTIVITY

Change the card's title to 'My website' and the main text from 'Hello world!' to 'It's my first app!'

Source code: Click here to reveal.

	WATCH Video: HTML basics (9m 44s) by Chris Mattia on LinkedIn Learning.

Adding JavaScript to HTML

HTML allows us to create fantastic websites, but without JavaScript (JS), they would always stay static. If we want to add dynamics to the site, we need to use JS. It allows controls to change in real-time and data change on the website without reloading. There are many more amazing features you can perform, thanks to JavaScript. You'll learn about these in this course.

In this section, we'll create our first JavaScript code. To add JavaScript code to the HTML file, we can use a <script> tag. It should be used within the head element, at the very bottom. Within this element, you can write JavaScript code. Let's use the most basic command we can think of – one that prints objects to the main content:

```
document.write("Hello world!")
```

Where a document responds to the main window of our HTML application, you'll learn more about this in chapters about DOM.

Our entire code should be similar to this:

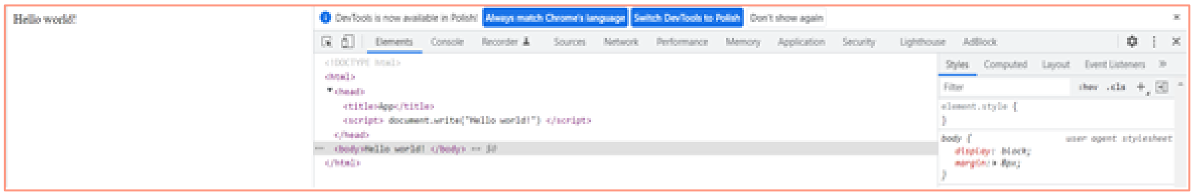
```
<!DOCTYPE html>
<html>
  <head>
        <title>App</title>
        <script>
            document.write("Hello world!")
        </script>
    </head>
</html>
```

Notice that our code contains indentations. They aren't mandatory but help a lot with the readability of our code. Moreover, in HTML code, indentations highlight its tree structure.

If we open the modified file in the browser, the final result should be exactly the same as in the previous case – a printed sentence 'Hello world!' It's typical to obtain the same result in many different ways in programming.

Now let's open the file again in the browser and press F12 on the keyboard to open the browser's development tools. This is a set of tools useful for web developers. You can inspect elements on site to see their code, change styles in the browser or debug applications. We can see a <body> part in this HTML code, even though it isn't in our code. It was generated by the document.write command.

You can see a preview of the development tools on a print screen below:



```
<!DOCTYPE html>
<html>
  <head>
    <title>App</title>
    <script> document.write("Hello world!") </script>
  </head>
  <body>Hello world! </body> == $0
</html>
```

```
element.style {
}
body {                      user agent stylesheet
  display: block;
  margin: 8px;
}
```

ACTIVITY

Change the main text from 'Hello world!' to 'It's my first app!'.

Source code: Click here to reveal.

JavaScript in the separate file

In this section, we'll achieve the same result as the previous one but will separate an HTML file from a JavaScript file. Let's create a new file named hello.js. and copy everything we have written within the scope of a script tag in the previous section.

Files with the extension '.js' will always contain source files written in JavaScript. The hello.js file should look like this:

```
document.write("Hello world!")
```

Now let's set the newly created hello.js file as a source file for a script element:

```
<script src="hello.js"></script>
```

Our hello.html file should look like this:

```
<!DOCTYPE html>
<html>
  <head>
        <title>App</title>
        <script src="hello.js"></script>
    </head>
</html>
```

When we open the file in the browser, we get the same result as on the last try. As expected!

Remember, we can use multiple scripts within one HTML file. To do this, we use script tags sequentially.

READ

Article: How To Add JavaScript to HTML by Lisa Tagliaferri.

WATCH

Video: Add JavaScript to HTML document (3m 1s) by Oluchukwu Okpala on LinkedIn Learning.

Storing information within variables

Variables are containers for storing data values. Every variable has its identifier and value. The identifier needs to fit the following rules:

- The name should start with a letter (a-z or A-Z), underscore (_) or dollar ($).
- The next characters can also be digits (0-9).
- Names using lower and upper case are different: identifier abc isn't the same as identifier ABC.
- Reserved words can't be used (for example, JavaScript keywords).

Below we have some basic examples of correct JavaScript variables.

```
var x = 55;
var X = "abc";
var _x = 1.2;
```

The keyword 'var' says that we're declaring a variable. Text that follows right after the var keyword is the variable's identifier (x, X or _x). The character '=' is the assignment operator, and 55, 'abc' and 1.2 are values our variables store.

A declaration of a variable is where a program says it needs a variable. We need to declare variables to use in further parts of our program.

There are two more keywords used for declaring variables: const and let. The keyword 'var' was used in JavaScript code from 1995 to 2015, so if we want our code to run in older browsers, we must use it.

Keywords 'let' and 'const' is the new standard. We should use the word 'const' when we declare variables and don't think it can change during the program. Otherwise, we should declare it with 'let'.

Below is a short code example with a declaration of variables using keywords 'const' and 'let':

```
const price1 = 5;
const price2 = 6;
let total = price1 + price2;
```

Variables price1 and price2 won't change unless we modify their values. On the other hand, the value of the variable total will change with every modification of the values of variables price1 and price2.

We can also declare multiple variables in the same line. The code below behaves in precisely the same way as our first example:

```
var x = 55, X = "abc", _x = 1.2;
```

We may notice every line ending with a semicolon. This means our statement is over, and we can start a new one. Putting the code in separate lines isn't necessary. It just makes it easier to read. The following code would also work in the same way as the previous examples:

```
var x = 55; var X = "abc"; var _x = 1.2;
```

In JavaScript, the majority of statements should end with a semicolon. Fortunately, JavaScript has Automatic Semicolon Insertion, so if you forget to add it, the JavaScript parser will do it for you. You can read more about it in the article in the READ section.

We can check our variables' values as we did during the previous lesson. Let's create a file named 'variables.js containing' this code and connect it to an HTML code, as we did previously.

variables.js file:

```
var x = 55;
var X = "abc";
var _x = 1.2;
document.writeln(x);
document.writeln(X);
document.writeln(_x);
```

variables.html file:

```
<!DOCTYPE html>
<html>
  <head>
        <title>App</title>
        <script src="variables.js"></script>
    </head>
    <body></body>
</html>
```

This is the last time we go through mapping a .js file to an HTML file. In future, we'll only see .js files, which we should map to our own .html file following the same lines.

	READ 1. Page: JavaScript Variables by W3 schools. 2. Article: Let's talk about semicolons in JavaScript by Flavio Copes.

	WATCH 1. Video: Declaring and assigning variables (5m 58s) by Joe Chellman on LinkedIn Learning. 2. Video: const vs. let vs. var (5m 16s) by Ray Villalobos on LinkedIn Learning.

	ACTIVITY Declare three variables: a,b and c. Variable a should be equal to 5, variable b to 8 and variable c should be the sum of the other variables. **Source code:** Click here to reveal.

Introduction to data types

In the last section, we learned what a variable is. Variables can hold different data types, for example, numbers, strings or objects. If the type isn't declared, the variable has the type 'undefined'.

Let's define two variables:

```
let x = 15;
let name = "John";
let u;
```

Variable x is a number, while the variable name is a string. It doesn't make any sense to add these variables. But if we try, the computer will convert one type to the other. We can check the results of the following code:

```
let x = 15;
let name = "John";
let result = name + x;
document.writeln(result);
```

In this example, 15 was converted to "15". The variable x remained as a type number variable, but the variable result has a type string.

In JavaScript, operations are performed from left to right.

```
let x = 15;
let y = 5
let name = "John";
let result1 = x + y + name;
let result2 = name + x + y;
document.writeln(result1)
document.writeln(result2)
```

After running the above code, we see that result1 performed addition x + y, but result2 didn't. This is because in result1, the first operation was x + y, both numbers. Then we added a name, which forced type conversion. In the case of result2, type conversion was already forced after the first operation.

In JavaScript, data types are dynamic. It means that one variable can change its type after creation. Let's look at the code example below:

```
let x;
x = 5;
x = "John";
```

In the beginning, variable x has the type ‘undefined’. Then it becomes a number, and changes to a string at the end.

JavaScript, unlike many other programming languages, has only one numeric type. It can be written with or without decimals. Numbers are stored in the computer’s memory using 64 bits. Because of that, integer precision will be accurate only up to 15 digits.

Let’s check the code example below:

```
let x = 999999999999999;
let y = 9999999999999999;
```

Variable x will be accurate, but y will be rounded up to 10000000000000000.

The maximum number of decimals is 17.

There’s also an exponential notation available:

```
let x = 56e5;
let y = 56e-5;
```

The code above does exactly the same as the code below (and is shorter):

```
let x = 5600000;
let y = 0.00056;
```

There are also types NaN (Not a Number) and Infinity.

NaN is used when the computer can’t perform an operation.

Infinity is a constant equal to the biggest number possible in JavaScript (since the memory is finite, there has to be the biggest number).

READ

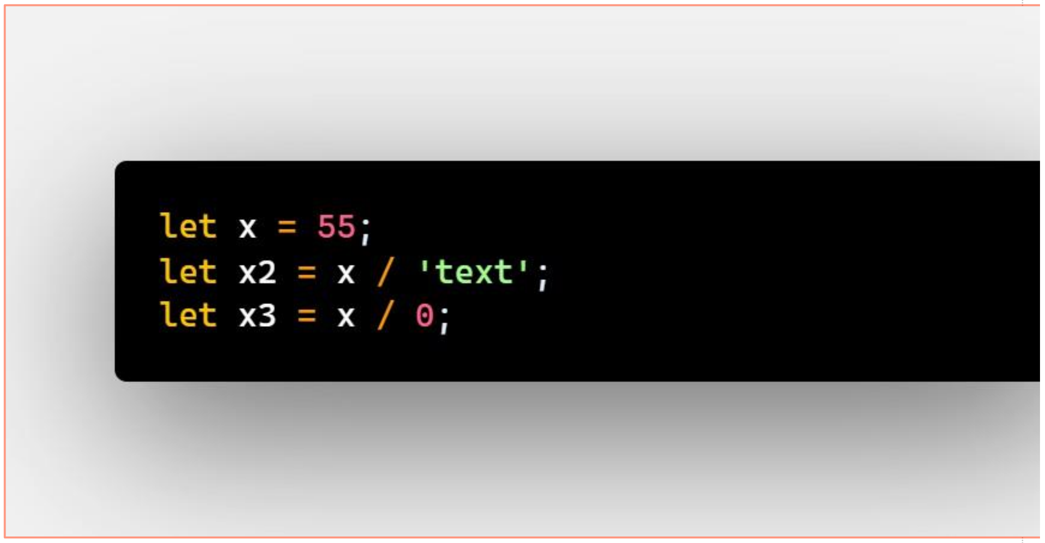
1. Page: JavaScript Data Types by W3 schools.
2. Page: JavaScript Numbers by W3 schools.

WATCH

1. Video: Data types in JavaScript (5m 3s) by Engin Arslan on LinkedIn Learning.
2. Video: JavaScript Tutorial For Beginners #22 – Numbers (6m 9s) by The Net Ninja on YouTube.

ACTIVITY

'/' operator stands for dividing the numbers. Analyse the values of the following variables:



```
let x = 55;
let x2 = x / 'text';
let x3 = x / 0;
```

Answer: Click here to reveal.

Assignment operators

Operators are used for performing specific operations on variables. We have already seen some operators in the previous code examples, for example, = operator and + operator.

In this section, we'll focus on assignment operators. The most standard one is the = operator:

```
x = 5;
```

The = operator assigns the value on the right-hand side of the operator to the variable on the left-hand side of the operator. The value can also be provided in the form of another variable. The code below copies the value from variable y to variable x.

```
x = y;
```

There are other assignment operators than the = operator:

- += operator
- -= operator
- *= operator
- /= operator
- %= operator
- <<= operator
- >>= operator
- >>>= operator
- &= operator
- ^= operator
- |= operator
- **= operator

Apart from assigning a value, these operators also modify it in a certain way. For example, let's analyse the += operator.

The code below presents how the += operator works:

```
let a = 0;
a += 5;
a = a + 5;
```

The second and the third line do exactly the same thing, i.e. they increase the value of variable a by 5. In the beginning, the value of 0 is assigned to variable a (line 1). Then it gets incremented by 5 (line 2). And then it gets incremented by 5 again, so it ultimately reaches a value of 10.

All other enlisted operators work analogously, but to understand how they do, we need to address arithmetic, logical and bitwise operators, which we'll do in the next sections of this lesson.

READ

1. Page: JavaScript Operators by W3 schools.
2. Page: JavaScript Assignment by W3 schools.

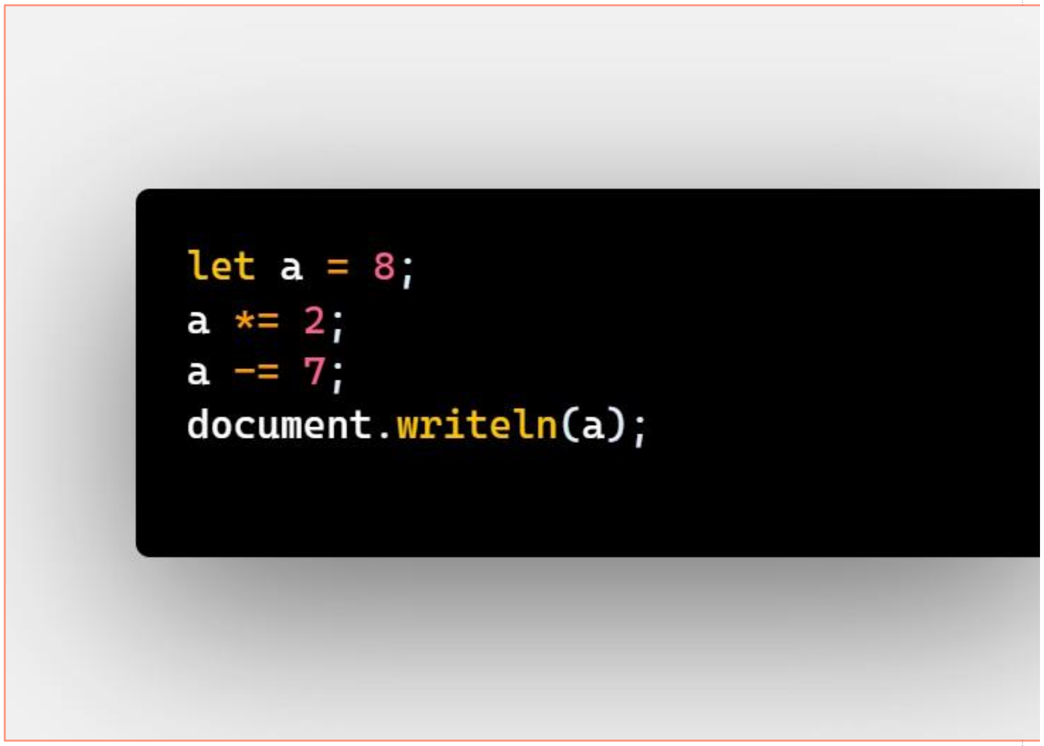
WATCH

Video: Simple comparisons (4m 10s) by Joe Chellman on LinkedIn Learning.

ACTIVITY

Declare variable a equal to 8. Using only assignment operators, double its value and distract 7 from the doubled value. What is the final number?

Source code:



```
let a = 8;
a *= 2;
a -= 7;
document.writeln(a);
```

Answer: Click here to reveal.

Arithmetic operators

Arithmetic operators, unlike assignment operators, use more than one variable as an argument. Variables participating in the operation, i.e., the operator's arguments, are called operands. For example, in the following code:

```
let x = 1;
let a = 100 + 20 + x;
```

100, 20 and x are operands.

In JavaScript, there are the following arithmetic operators:

- + operator – addition
- - operator – subtraction
- * operator – multiplication
- ** operator – exponentiation
- / operator – division
- % operator – modulus
- ++ operator – increment
- -- operator – decrement

Below we provide some basic examples which show how to use operators. Results are written in the same line using //, which denotes a comment. Comments are texts that aren't executed in the program. To create a comment, you need to add a double backslash in the code. Everything we put on the right-hand side of the backslashes // will be treated as a comment.

```
let a = 1 + 4 //5
let b = 5 - 1 //4
let c = 2 * -4 //-8
let d = 2 / 4 // 0.5
let e = 5 % 3 // 2
e++ // 3
c-- //-9
```

We can verify these values using a document.writeln instruction.

	READ Page: JavaScript Arithmetic by W3 schools.

	WATCH Video: Arithmetic operators (5m 24s) by Joe Chellman on LinkedIn Learning.

Comparison operators

We already discussed two data types: numbers and strings. Now is the time for introducing the third new data called Boolean. A Boolean variable can take one of two possible values: true or false. True and false values are typically associated with results of logical expressions or answers to yes-no questions.

Comparison operators represent yes-no questions regarding two values, one of which is provided on the left-hand side of the operator and the other on the operator's right-hand side. Therefore, they return Boolean values.

Here is a list of helpful comparison operators:

- == operator – verifies if left-hand side is equal to the right-hand side.
- === operator – verifies if left side is equal to right and of the same type.
- != operator – verifies if left side isn't equal to right.
- !== operator – verifies if left side is equal to right, or not equal type.
- > operator – verifies if left side is greater than right.

- < operator – checks if left side is less than right.
- >= operator – checks if left side is greater or equal to right.
- <= operator – checks if left side is less or equal to right.

Let's assume:

```
const x = 1;
```

There are some not obvious examples of how these operators work. The value of returned Boolean is written in a comment on the same line.

```
x == 2 //false
x == 1 //true
x == "1" //true
x === 1 //true
x === "1" //false
x != 2 //true
x !== 1 //false
x !== "1" //true
x !== 2 //true
x > 2 //false
x < 2 //true
x >= 2 //false
x <= 2 //true
```

There are also logical operators that allow us to operate on Booleans.

- && operator – logical and (conjunction) – true only if the left-hand side and right-hand side is true simultaneously.
- || operator – logical or (alternative) – true if the left-hand side is true or if the right-hand side is true (also true if both are true).
- ! operator – logical not (negation) – changes true into false and false into true.

Let's examine the following examples:

```
(1 + 2 == 3) && !(2 + 3 > 4) //false
(1 + 2 == 3) || !(2 + 3 > 4) //true
!(1 + 2 == 3) && !(2 + 3 > 4)  //false
!(1 + 2 == 3) || !(2 + 3 > 4) //false
(1 + 2 == 3) && (2 + 3 > 4)  //true
```

In the case of the first example, we get a false value return. This is because the right-hand side is false. Initially, it is true since 2 + 3 > 4, but the negation operator switches it into false.

The second example returns true because the left-hand side is true, and in the case of an alternative operator, it suffices.

The next two examples return a value false because both sides are false, in which case both the conjunction and the alternative operators return a false value.

The fifth example returns a value true because both sides are true.

READ

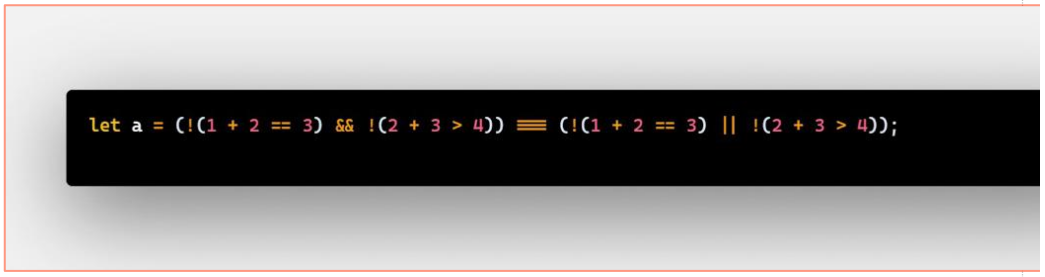
1. Page: JavaScript Comparison and Logical Operators by W3 schools.
2. Page: JavaScript Booleans by W3 schools.

WATCH

1. Video: JavaScript Tutorial For Beginners #11 – Booleans in JavaScript (6m 13s) by The Net Ninja on YouTube.
2. Video: JavaScript Tutorial For Beginners #14 – Comparison Operators (5m 25s) by The Net Ninja on YouTube.

ACTIVITY

What’s the value of variable a?



```
let a = (!(1 + 2 == 3) && !(2 + 3 > 4)) === (!(1 + 2 == 3) || !(2 + 3 > 4));
```

Do you think having such a long line of variable declarations is a good idea? If not, how would you fix it?

Long lines aren’t great because of poor readability. We can assign some of the statements to the variables to fix that.

Answer: Click here to reveal.

Other operators

There are more operators in JavaScript, but they are used less often, especially at the beginning. Special characters denoted the operators we discussed so far. Type operators are different since words denote them.

Type operators:

- typeof operator – returns the type of the variable.
- instanceof operator – verifies if an object is an instance of a given type and returns a Boolean value as an answer.

Bitwise Operators:

- & - bit AND operator
- | - bit OR operator
- ~ - bit NOT operator
- ^ - bit XOR operator
- << - left shift
- >> - right shift
- >>> - unsigned right shift

INFO

To discuss these operators, we need to introduce the workings of a binary system. It isn't that crucial for a JavaScript developer. Still, if you want to learn, you can watch the video 'Binary – The SIMPLEST explanation of Counting and Converting Binary numbers' by Practical Networking and read the page on JavaScript Bitwise Operations.

READ

1. JavaScript Operators by W3 schools.
2. JavaScript Bitwise Operations by W3 schools.

WATCH

1. Video: What is the difference between typeOf, isNan and instanceOf in JavaScript (3m 26s) by Study Box on YouTube.
2. Video: Bitwise Operators JavaScript (4m 36s) by techsith on YouTube.

ACTIVITY

Read the article 'Basic math in JavaScript — numbers and operators' by MDN Web Docs and complete the exercises as you go through.

What did I learn in this lesson?

This lesson provided the following insights:

- How to a set up text editor for web development.
- The structure of a basic HTML5 document.
- Creating a basic web application using JavaScript.
- How to create a variable, assign a value and change its value.
- The basic data types and what they represent.
- How to perform operations on variables using operators.
- The most important operators.

References

W3 Schools (n.d.) *HTML Tutorial*. Available at: https://www.w3schools.com/html/default.asp [Accessed 25 May 2022].

learn-html.org (n.d.) *Hello, World!* Available at: https://www.learn-html.org/en/Hello,_World! [Accessed 25 May 2022].

Visual Studio Code (2022) *Getting started with Visual Studio Code.* Available at: https://code.visualstudio.com/docs/introvideos/basics [Accessed 25 May 2022].

1.1. Lesson task - Your first application

The task

Part 1

We installed a text editor for web development in this lesson and created our first web application. The task for this lesson is very similar to what we have already done. You are asked to create a new website that shows three sentences.

The first sentence says, ‘I am sentence number 1 and I was printed without JS.’ It should be displayed without using JavaScript.

The second sentence says, ‘I am sentence number 2 and I was printed with JS from the HTML file.’ It should be displayed with JavaScript but within an HTML file.

The third sentence says, ‘I am sentence number 3 and I was printed with JS from a js file.’ It should be displayed with the use of a separate JavaScript file.

All three sentences need to be displayed on the same page. When you open your HTML file in the browser, what is the order of the sentences?

To complete this task, you are asked to perform desktop research to find out why sentences have been displayed in a given order and communicate your findings in a short report written in a Word document.

Your report should contain print screens of your code, print screens of your website, information about files that were created in Visual Studio Code and other relevant information.

Sentence order depends on the order of the sentences in the HTML source code. The sentences earlier in the source code will also be earlier on the website. It doesn’t depend on how we add them.

Part 2

This lesson discussed the basics of variables, data types and operators.

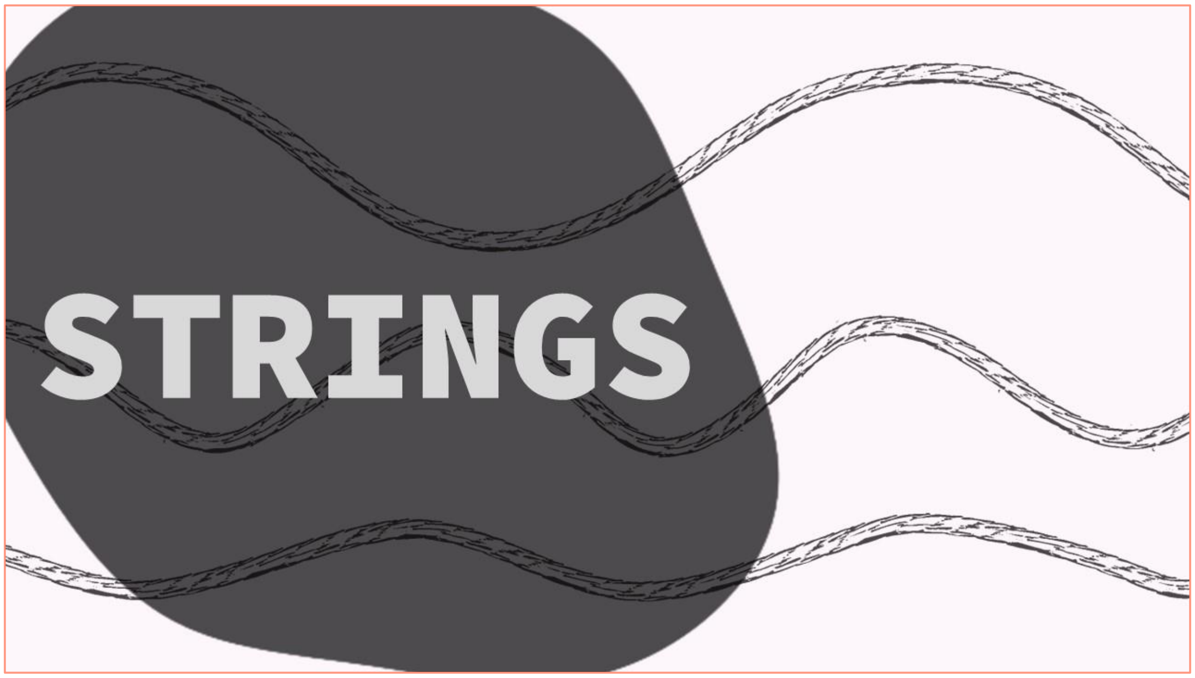
Practice what you have learned, by deriving the results of the following expressions (without JavaScript).

`let a = "a" + 1 + 2 + 1.2;`
`let b = 1 + 1.2 + "a" + 1;`
`let c = "" == 0;`
`let d = !(1 + 2 == 3) || !(2 > 4);`
`let e = 0.1 + 0.2;`
`let f = 9 % 2 ** 3;`
`let g = -1 / 0;`

	ACTIVITY After completing this task, you can try finishing the Math test on Mozilla documentation. Try to do as many task sub-items as possible!

1.2. Lesson - Strings

Introduction



‘String’ is a data type appropriate for storing and manipulating text. It’s one of the most important data types, especially in the case of web development. Websites have numerous text elements.

In JavaScript, various methods are available that improve the working experience with strings. These methods allow us to manipulate and access their parts efficiently.

In this lesson, we'll discuss these methods and use them to perform many activities on strings. Familiarity with strings is beneficial for a JavaScript developer.

When coding, we often want to perform actions depending on whether some conditions are satisfied. Conditionals are the most standard statements that enable us to do this.

In this lesson, we'll discuss three different types of conditionals. We'll also discuss the most convenient situations for using a given conditional.

When using conditional statements, we should be careful. We'll discuss what to avoid.

Learning outcomes

In this lesson, we are covering the following knowledge learning outcome:

- → The candidate has knowledge of program syntax, program structure, control structures, data types, and variables as used in the JavaScript language.

In this lesson, we are covering the following skill learning outcomes:

- → The candidate can apply knowledge of browsers and development tools to set up a computer to develop and debug small JavaScript programs.
- → The candidate masters the use of relevant tools for writing, editing, and debugging JavaScript source code.

In this lesson, we are covering the following general competence learning outcome:

- → The candidate can carry out simple programming tasks with the JavaScript language.

Introduction to strings

Strings in JavaScript consist of zero or more characters put between quotes:

```
let name1 = "John";
let name2 = 'John';
let name3 = '';
let name4 = ' "John" ';
```

Quotes can be either single or double. Both options are possible, but it makes sense to choose one and stick to it (unlike the example above). We'll use double quotes.

In the fourth line above, we can see that there can be double quotes inside a single quote. It also works the other way round.

We can get the length of a string by using a built-in property called length:

```
let s = "12345 12345";
let l = s.length; //12
```

In this case, string s, which consists of 10 digits and two spaces, has a length of 12.

	READ Page: JavaScript string by JavaScript Tutorial.

	WATCH Video: Strings (4m 10s) by Joe Chellman on LinkedIn Learning.

Escape sequences

Sometimes creating strings can be problematic. What if we needed to use quotes inside a string? Below is an example of such a situation:

```
let text = "My favourite book is "Harry Potter"";
```

JavaScript misunderstood our intention and created two strings instead of one. We could fix this by using the other type of quotes instead. But what if we have to use both? To address such issues, an escape character can be used.

The backslash (\) turns the next character inside a string into a special one. Here are the escape sequences:

- \' – single quote
- \" – double quote
- \\ - backslash
- \b – backspace
- \f – form feed
- \n – new line
- \r – carriage return
- \t – horizontal tabulator
- \v – vertical tabulator

Our code with the escape sentence applied looks like this:

```
let text = "My favourite book is \"Harry Potter\"";
```

There could be one more problem with the string edition. What if we have a very long string that takes more than one line to be printed? It's good programming practice to avoid lines longer than about 80 characters.

Fortunately, there's a way to break a string into multiple lines. For this purpose, we can use the backslash character again. We need to type the backlash in the place where we want the line to break and then press enter.

Below is an example where the value assigned to the variable text has been broken into two lines:

```
let text = "My favourite book \
is \"Harry Potter\"";
```

READ

Page: JavaScript Strings by W3 schools.

WATCH

1. Video: JavaScript Unicode Characters and Escape Sequences (10m 21s) by Steve Griffith – Prof3ssorSt3v3 on YouTube.
2. Video: Escaping Literal Quotes in Strings, freeCodeCamp Basic Javascript (2m 50s) by We Will Code on YouTube.

Operations on strings

Strings, like most variables, allow us to manipulate them using operations. The most standard thing we can do with two strings is to merge them into one. We can do this using a + operator, as well as using a concat() function, presented below:

```
let a = "abc";
let b = "de";
let c = "xyz";
let d = a + b;
```

```
document.writeln(d); //abcde
let e = a.concat(c);
document.writeln(e); //abcxyz
```

If we want to update an existing string by adding another string to it, we can also use the += assignment operator:

```
let a = "abc";
let b = "de";
a += b;
document.writeln(a); //abcde
```

Remember that this wouldn't work the same way with subtraction. Operator – doesn't work on strings, and erasing one string from the other is more complex than concatenating two strings.

Another thing we do with strings is comparison. We can use operators <, > and == for this purpose. Operators < and > compare strings according to a lexicographic, i.e. alphabetical order.

```
let a = "abc";
let b = "de";
let c = "abcd";
document.writeln(a > b); //false
document.writeln(b > c); //true
document.writeln(a > c); //false
document.writeln(a < b); //true
document.writeln(b < c); //false
document.writeln(a < c); //true
```

Variable 'a' turns out to be the smallest one of the three variables we declared, according to a lexicographic order, whereas variable 'b' is the biggest in those terms, even though it contains only two characters.

The lexicographic order is a generalisation of the alphabetical order of the dictionaries to sequences of ordered symbols. It means that we compare strings like words in a dictionary. If we look at the letters of the English alphabet, we can see that: a < b = true, b < c = true. This is because alphabetically, 'a' comes before 'b' and, therefore, 'b' has a higher value lexicographically.

READ

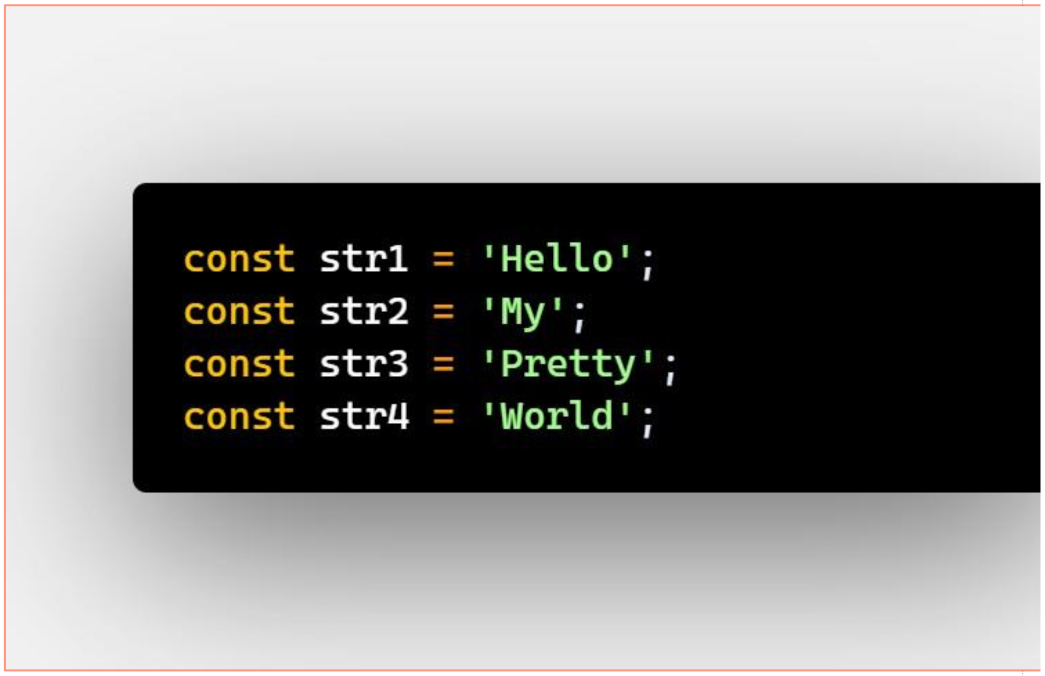
1. Page: JavaScript String Methods by W3 schools to learn more about other helpful string methods.
2. Page: String by MDN Web Docs.

WATCH

1. Video: Javascript String Concatenation | Javascript Tutorial For Beginners | ES6 Tutorial (11m 54s) by Dev Ed on YouTube.
2. Video: JavaScript String Concatenation (4m 16s) by Steve Griffith – Prof3ssorSt3v3 on YouTube.

ACTIVITY

We have given the following variables:



```
const str1 = 'Hello';
const str2 = 'My';
const str3 = 'Pretty';
const str4 = 'World';
```

1. What is their lexicographic order?
2. Concatenate all the strings into a "Hello My Pretty World" sentence.

Answer: Click here to reveal.

Extraction of string elements

Sometimes we want to extract a specific substring from a string object. The most basic example is when we need to know the first character of a string, for example, to sort words alphabetically. There's a method charAt(), which returns the character at a specified index. In JavaScript, the position of the first character is 0. JavaScript follows zero-based indexing, which, in the beginning, might be a bit confusing.

On other occasions, we are interested in extracting more characters from a string object and have three useful methods: slice, substring and substr.

Method slice() extracts a substring from a string and returns it so that we can assign the returned substring to another variable. It takes two input parameters, the start index position and the end index position (slice extracts up to, but not including, the end index character). Remember, indexing is zero-based.

```
let text = "My favourite book is \"Harry Potter\"";
let part = text.slice(8, 14);
document.writeln(part); //rite b
```

Negative indexes are also allowed. A negative index means we're counting indexes starting not from the beginning but at the end of the string:

```
let text = "My favourite book is \"Harry Potter\"";
let part = text.slice(-27, -21);
document.writeln(part); //rite b
```

If we omit the second parameter, the slice method will extract the remainder of the string, starting from the value indicated by the first parameter:

```
let text = "My favourite book is \"Harry Potter\"";
let part = text.slice(4);
document.writeln(part); //avourite book is "Harry Potter"
let part2 = text.slice(-4);
document.writeln(part2); //ter"
```

The method substring() works analogically to the slice() method, but negative indexes are treated as 0.

The method substr() is also similar, but the second parameter doesn't indicate the end position of the extracted substring, but the length of it:

```
let text = "My favourite book is \"Harry Potter\"";
let part = text.substr(4, 5);
document.writeln(part); //avour
```

WATCH

1. Video: substring and substr String methods (5m 29s) by Steve Griffith - Prof3ssorSt3v3 on YouTube.
2. Video: JavaScript Substring Example | Slice, Substr, and Substring Methods in JS (5m 38s) by Cem Eygi Media.

Replacement of string elements

When processing very long strings, we might need to replace every occurrence of a particular word. The replace() method is ideally suited for this. Below we show a basic example:

```
let text = "My favourite book is \"Harry Potter\". I love
this book so much.";
let result = text.replace("book", "movie");
document.writeln(result); //My favourite movie is "Harry
Potter". I love this book so much.
let result2 = text.replace(/book/g, "movie");
document.writeln(result2); //My favourite movie is "Harry
Potter". I love this movie so much.
```

This method only replaces the first occurrence by default, but we can change that using regular expressions. You can read more about regular expressions in the article below.

READ

1. Article: An introduction to regular expressions by Thomas Nield.
2. Page: JavaScript Regular Expressions by W3 schools.

WATCH

Video: Using the String.replace() method – JavaScript Tutorial (9m 19s) by dcode.

Conversion to lower and upper case

JavaScript strings have simple methods in which we can convert all alphabetic characters of the string into upper or lower-case. These methods are toUpperCase() and toLowerCase() respectively.

```
let text = "My favourite book is \"Harry Potter\". I love
this book so  much.";
let result = text.toLowerCase();
document.writeln(result); //my favourite book is "harry
potter". i love this book so much.
let result2 = text.toUpperCase();
document.writeln(result2); //MY FAVOURITE BOOK IS "HARRY
POTTER". I LOVE THIS BOOK SO MUCH.
```

READ

Page: JavaScript Strings by Pluralsight.

	WATCH Video: How to convert strings to Uppercase and lowercase with vanilla JavaScript (3m 3s) by TechStacker.

String search methods

Sometimes we want to find the position of a certain phrase within a string. JavaScript offers numerous methods to help us with that. The first one we'll discuss is indexOf(). indexOf() finds the first occurrence of specified text within the string.

Let's look at an example:

```
let text = "My favourite book is \"Harry Potter\". I love
this book so much.";
let result = text.indexOf("book");
document.writeln(result); //13
```

The indexOf("book") method finds the first occurrence of the string given to the method, which was 'book', then returns the index of the first character 'b', which happens to be the index value of 13.

There's also a similar method, lastIndexOf(), which works analogically. However, it doesn't find the first occurrence of a given string, but the last:

```
let text = "My favourite book is \"Harry Potter\". I love
this book so much.";
let result = text.lastIndexOf("book");
document.writeln(result); //49
```

Another useful method is search(). This is similar to indexOf() but allows us to use regular expressions. Regular expressions are very powerful when it comes to searching strings.

Sometimes we don't need an exact location of a given phrase within a string but only want to know if that string includes the phrase in question. In such a case, we have the following methods: includes(), startsWith() and endsWith().

These methods verify whether a string contains a particular phrase, starts with a particular phrase or ends with a particular phrase. All these methods return Boolean values.

Let’s look at some examples:

```
let text = "My favourite book is \"Harry Potter\". I love
this book so much.";
let result1 = text.startsWith("My");
let result2 = text.includes("My");
let result3 = text.includes("My", 4);
let result4 = text.endsWith("book");
document.writeln(result1); //true
document.writeln(result2); //true
document.writeln(result3); //false
document.writeln(result4); //false
```

The second parameter of the includes() method is optional, but when provided, it represents an index from which we start searching. There were no occurrences of the phrase “My” after the fourth index, so the method returned a value false.

READ

1. Page: JavaScript String Search by W3 schools.
2. Page: Useful string methods by MDN Web Docs.

WATCH

1. Video: indexOf and lastIndexOf String Methods (11m 47s) by Steve Griffith – Prof3ssorSt3v3 on YouTube.
2. Video: startsWith, endsWith, and Finding Strings (6m 40s) by Steve Griffith – Prof3ssorSt3v3.

Booleans

We discussed Boolean variables, comparison, and logical operators in previous lessons. Booleans are essential for the current lesson, so let's revise the most important things. Booleans are variables that can be either true or false. Both comparison and logical operators return a Boolean value.

As types in JavaScript are dynamic, variables of various types can be converted to Booleans. In such cases, when a variable x has a 'value', the Boolean(x) result is true, and when a variable x doesn't, Boolean(x) is false.

For example, the following expressions are cast to true:

- 33
- 1.12
- 1 + 2 + 3
- "true"
- 'Hello world'
- -12

On the other hand, the following expressions are cast to false:

- 0
- -0

- “”
- null
- NaN
- Undefined

	READ Page: JavaScript Booleans by W3 schools.

If statement

The ‘if statement’ is the most basic conditional statement (or conditional). We use this statement with a specified condition, which must be Boolean. If the condition is true, the code inside the if block will be executed. Otherwise, it won’t be executed.

```
if (condition) {
    //  block of code to be executed
}
```

Curly brackets ‘{‘ and ‘}’ define the scope of the if statement block. These brackets don’t have to be spaced exactly as in the example. Our code would still run, but this spacing is the most popular in JavaScript. The spacing of curly brackets in programming is known as indentation style. It’s a convention governing the indentation of code blocks to convey program structure. The two most popular styles are:

Allman style:

```
if (condition) {
    //  block of code to be executed
}
```

And K&R style (Kernighan & Richie Style):

```
if (condition)
{
    //  block of code to be executed
}
```

The code to be executed inside the if statement starts right after the first bracket '{' and ends right before the second one '}'. The condition is a variable, specifically a Boolean variable. Keyword 'if' defines the start of the if statement. Remember, it has to be written in lowercase. Otherwise, it will throw an error.

Below we have a simple code example, which prints a variable only when its value is bigger than 5:

```
let a = 10;
if (a > 5) {
    document.writeln(a);
}
```

ACTIVITY

Declare a name as a variable. If it has a length of less or equal to 10 characters, print that on the website.

Source code: Click here to reveal.

Else statement

We can use the 'else statement' to prepare an alternative for situations in which the if statement conditions are false. The basic else statement looks like this:

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

Apart from the ‘if’ condition, we also use the keyword ‘else’ and start the next code block. Code inside this block will be executed if the condition is false.

But what if we want to enter the block of code that follows the keyword ‘else’ only if some other condition is satisfied? In such a case, we can use the else-if statement. It’s very similar to the if statement. The difference is that after the keyword ‘else’, we add a keyword ‘if’ again, along with the other condition, the satisfaction of which results in the execution of the code inside the block.

Note that sequential application of the else-if statement allows us to verify as many conditions (cases) as we want. This is because we can add more else if statements one after another. We will go through an example to illustrate.

```
if (condition1) {
    //  block of code to be executed if condition1 == true
} else if(condition2) {
    //  block of code to be executed if condition1 == false
and condition2 == true
} else if(condition3) {
    //  block of code to be executed if condition1 == false
and condition2 == false and condition3 == true
} else {
    //  block of code to be executed if all conditions are
false
}
```

At first, we verify if condition1 is satisfied. If it is, we enter the first block of code. If it isn’t, we verify condition2; if satisfied, we enter its block of code. Otherwise, we verify condition3 and, if satisfied, enter its block of code. Finally, if none of the conditions 1, 2 and 3 are satisfied, we enter the block of code of the else statement at the end of the structure.

READ

1. Page: JavaScript if, else, and else if by W3 schools.
2. Page: JavaScript Conditionals by Pluralsight.

WATCH

1. Video: Conditionals: If (6m 21s) by Joe Chellman on LinkedIn Learning.
2. Video: Conditional if...else statement (5m 52s) by Morten Rand-Hendriksen on LinkedIn Learning.

Nested if statements

We already know how statements if, else, and if-else work, but we didn't specify what can be used within their code blocks.

Within the block of code of an if, else, or if-else statement, we can use variables defined outside of the block, but it doesn't work the other way around. This is because variables created within inner blocks of code are local variables. Let's see what this means.

```
let a = 1;
if ( a > 0 ) {
  let b = 3;
    document.writeln(a);
}
document.writeln(b);
```

The code above will print the value of variable a, although this variable was defined outside the if statement's block. Variable b, on the other hand, exists only within the if statement's code block – its scope is limited to this block, so we'll fail when we try to print it outside of this block.

In the code block of an if statement, we can execute any code we want. In particular, we can use another conditional. This way, we can verify whether the outer condition is satisfied. If it is, we can perform some actions in the second stage of such a hierarchical structure depending on whether the inner condition is satisfied.

Here's an example of a nested if statement structure:

```
let a = 1;
let b = 3;
if ( a > 0) {
    if( b > 0) {
        document.writeln(b);
    }
    else {
        document.writeln(a);
    }
}
```

It does the job, but the code becomes illegible when we introduce too many levels of nested statements. Too many levels of nested statements can also cause the execution of our code to take longer. This is because the computer will have to check unnecessary conditions before executing our code. It makes sense to reduce the structure by using appropriately specified conditions.

For example, the previous code can also be written as:

```
let a = 1;
let b = 3;
if ( a > 0 && b > 0) {
    document.writeln(b);
}
else if( a > 0 && b <= 0) {
    document.writeln(a);
}
```

This code doesn't use a nested if statement structure but verifies a condition a > 0 twice.

Nested if statements can make code simpler, more efficient and reduce redundancies, but we need to be careful because keeping the code clean can be one of the main priorities. Numerous levels of nested conditionals can make the code less clear.

	WATCH Video: JavaScript 2021 Tutorial 16 – Nested if statements (7m 52s) by Daniel Wood on YouTube.

Switch statement

Another conditional that can be found helpful in many cases is a switch statement. We can use it when a given expression can take several possible outcomes. A switch statement looks like this:

```
switch(expression) {
  case x:
        // code block
        break;
    case y:
        // code block
        break;
    case z:
        // code block
        break;
    default:
        // code block
}
```

It starts with the keyword ‘switch’. Then we have an expression, which can be of any type (note that for the if statement, it was a condition, not an expression). Inside the brackets, we enlist possible values of the expression.

Every instance starts with the keyword ‘case’ followed by the relevant value of the expression and the colon. After the colon, we provide a code block which will be executed when the expression takes the relevant value.

The block ends with a keyword break. In the end, we have a keyword ‘default’, which represents a case where we couldn’t determine the value of the expression. The default will be executed if none of the above expressions is matched.

Below we can see an example:

```
let x = 2;
switch(x) {
    case 1:
        document.write("1");
        break;
    case 2:
        document.write("2");
        break;
    default:
        document.write("3");
}
```

This switch statement verifies if the value of variable x equals 1 or 2. If it does, a digit 1 or 2 is printed. If x is neither 1 nor 2, a digit 3 is printed by default. Since we have assigned a value of 2 to a variable x right before the switch statement, our program will print a digit 2.

Notice that every switch statement can be replaced with if statements. This is how:

```
let x = 2
if (x == 1) {
        // code block
        document.write("1");
} else if (x == 2) {
        // code block
        document.write("2");
} else {
    document.write("3");
}
```

However, the code looks more complicated this way, so it’s good practice to use switch statements.

READ

Page: JavaScript Switch Statement by W3 schools.

WATCH

Video: Conditionals: Switch (3m 53s) by Joe Chellman on LinkedIn Learning.

The ternary operator

A ternary operator provides us with a concise way of using conditionals. We can build the entire conditional in only one line. The syntax of the ternary operator looks like this:

```
x = condition ? expression1 : expression2
```

It verifies whether the condition is satisfied (i.e. whether it's true). If the condition is satisfied, a value specified by the expression1 gets assigned to variable x. When the condition isn't satisfied or false, a value specified by the expression2 gets assigned to variable x. So, a ternary operator works like an if-else statement; only the syntax is simpler.

Here we have an example of the ternary operator in action:

```
let a = 1;
let b = 2;
let c = a > b ? a : b;
```

A maximum of a and b will be assigned to the variable c.

It is possible to nest ternary operators, but usually, it isn't good practice. Code with multiple '?' and ':' characters can be hard to understand.

READ

Page: JavaScript Ternary Operator by Programiz.

WATCH

Video: The ternary operator (2m 48s) by Ray Villalobos on LinkedIn Learning.

ACTIVITY

Read the article 'Making decisions in your code — conditionals' by MDN Web Docs and complete the exercises as you go through.

What did I learn in this lesson?

This lesson provided the following insights:

- Escape sequences and how they work.
- Which string methods are most useful for a beginner JavaScript developer.
- How to manipulate strings.

- The general idea behind conditionals.
- When it's better to use the if statement versus the switch statement.
- Nested statements and when to avoid them.

References

W3 schools (n.d.) *HTML Tutorial*. Available at: https://www.w3schools.com/html/default.asp [Accessed 25 May 2022].

Thomas Nield (2017) *An introduction to regular expressions*. Available at: https://www.oreilly.com/content/an-introduction-to-regular-expressions/ [Accessed 25 May 2022].

Programiz (n.d.). JavaScript Ternary Operator. Available at: https://www.programiz.com/javascript/ternary-operator [Accessed 25 May 2022].

1.2. Lesson task - Strings

The task

This lesson taught us about strings and the operations we can perform on them. We also discussed extracting substrings and searching for phrases within a string.

In this task, we have a string:

```
let text = "My favourite book is \"Harry Potter\". I love
this book so much.";
```

You are asked to create the following strings using the variable text and the operators we discussed:

resultText1: favourite book is “Harry Potter”
resultText2: My book is love.
resultText3: My book is “HARRY POTTER”.
resultText4: potter harry

Try to minimise the number of variables you use, but try not to perform all the operations in one line.

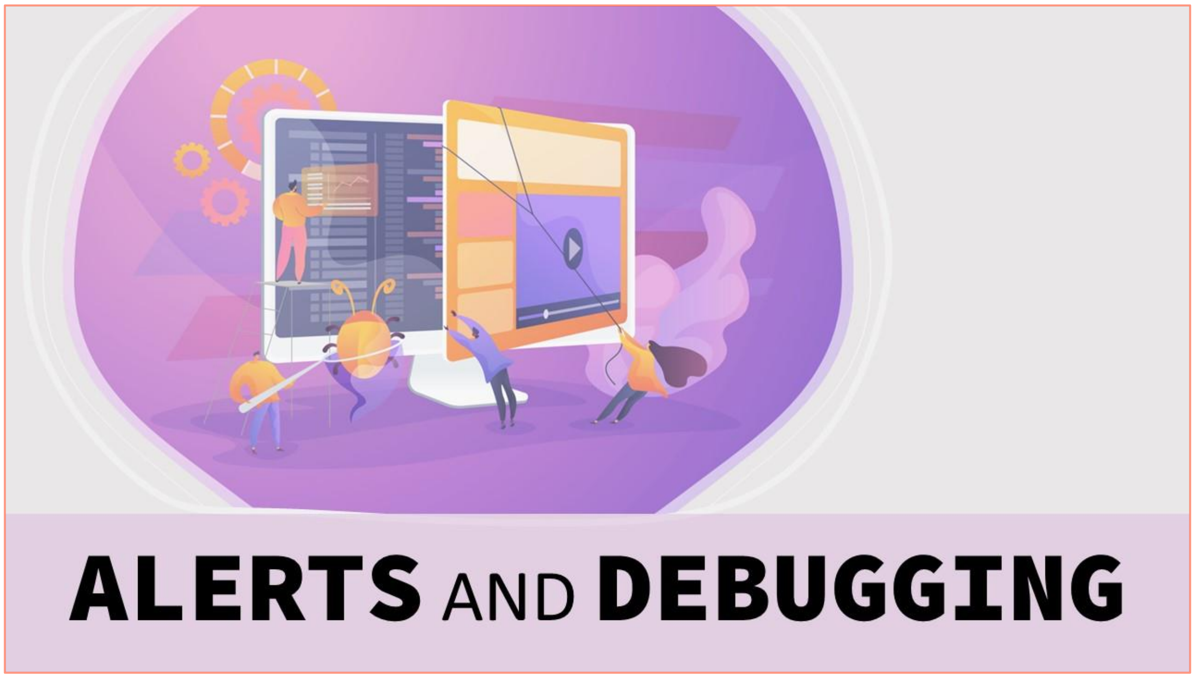
ACTIVITY

After completing this lesson task, you can try finishing the Strings test on Mozilla documentation. Try to do as many task sub-items as you can!

You can also do some more complicated exercises on JavaScript String - Exercises, Practice, Solution by w3resource. After trying them on your own, you can check their solutions.

1.3. Lesson - Alerts and debugging

Introduction



We already have some essential knowledge about computer programming in JavaScript, but we don't interact with the user much. For now, our application testing consists mainly of printing values using the document.writeln method.

We also haven't learned an efficient way to deal with potential errors within our code. Sometimes our application can stop working, and it's not always instantly clear why.

In this lesson, we'll solve these problems using new tools. Prompts and alerts will solve the first issue, while debugging and logging will take care of the second.

Alerts are pop-up windows relaying important information to the user. Prompts are dialogues which allow the web page to request information from the web page user. Debugging pauses the execution of the code in a particular line to see the current values of our variables. Logging prints helpful information in the browser's console. While we can see it, it isn't easily accessible to the average user of our application.

Variables are perfect for storing single pieces of information or data, but web development usually needs to store a large number of data. Using variables to store this data would require hundreds of variables and remembering every name. Fortunately, arrays provide us with an elegant way to solve this problem.

An array is a collection of variables, each identified by an index number. It allows us to perform certain operations on an entire set of variables. For example, after creating an array of variables, we can easily find the minimum value or sort all values.

Learning outcomes

In this lesson, we are covering the following knowledge learning outcomes:

- → The candidate has knowledge of development and debugging within a web browser.
- → The candidate has knowledge of program syntax, program structure, control structures, data types, and variables as used in the JavaScript language.

In this lesson, we are covering the following skill learning outcomes:

- → The candidate can apply knowledge of web browsers and development tools to set up a computer for use in developing and debugging small JavaScript programs.
- → The candidate masters the use of relevant tools for writing, editing, and debugging JavaScript source code.

In this lesson, we are covering the following general competence learning outcome:

- → The candidate can carry out simple programming tasks with the JavaScript language.

Browser developer tools - Inspector

Most popular browsers have built-in tools that deliver practical help to web developers. This lesson will go through the essential functions of developer tools included in Google Chrome.

DOWNLOAD

The newest version of Google Chrome.

To start, create a new empty folder and open it in VS Code. In that folder, create the following two files with the following code:

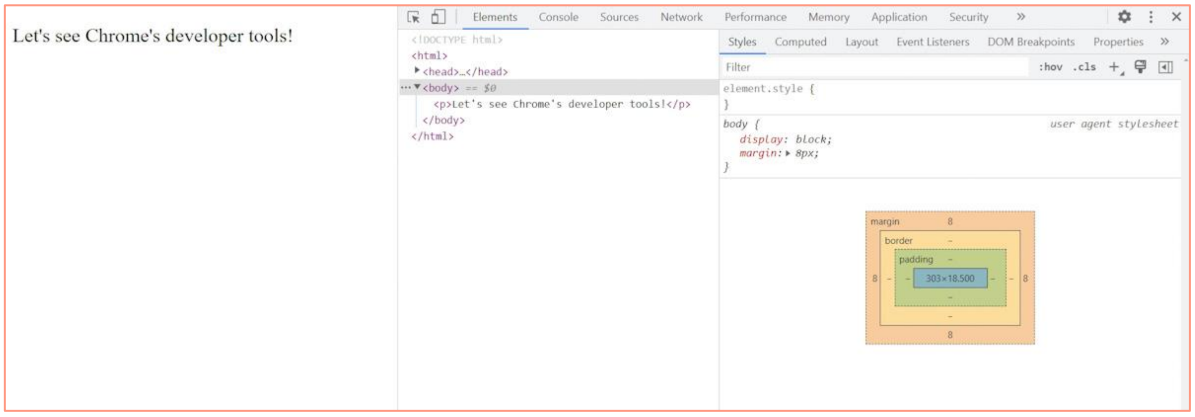
tools.html:

```
<!DOCTYPE html>
<html>
  <head>
        <title>App</title>
        <script src="tools.js"></script>
    </head>
    <body>
        <p>Let's see Chrome's developer tools!</p>
    </body>
</html>
```

tools.js:

```
let x = Math.random();
if (x > 0.5) {
    console.log("Hello, it's nice to see you here.");
}
else {
    console.log(x);
}
```

Let’s open the tool's HTML file with Google Chrome and click the F12 button. We should see something like this:


Let's see Chrome's developer tools!

```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
··· ▼<body> == $0
    <p>Let's see Chrome's developer tools!</p>
  </body>
</html>
```

```
element.style {
}
body {                                   user agent stylesheet
   display: block;
   margin: ▶ 8px;
}
```

The right part of the screen represents the browser’s developer tools. In the upper menu, we can see many tabs to choose from. In this section, we’ll focus on the current one – Elements.

On the main left part, we can see the HTML code responsible for our current website. We can edit this code in real-time. In other words, the effects of our code changes will be visible on the website, but after refreshing the site, they will be lost. This is because the changes we make to the code in the web browser don’t change our local source code. This is the perfect solution to test code; we can write the code and see the results immediately.

If satisfied with the results, we can copy this code to the VS Code and save the solution. If not, we can refresh the site (by clicking F5 or the refresh button) and start from our local source code’s state.

Let’s add a new paragraph on the website from developer tools. To do this, click the right button on the line with the current paragraph (text: Let’s see Chrome’s developer tools!) and choose the option ‘Edit as HTML’.

Go to the end of the line, click Enter, put the line (below) here and click in any other space in the HTML editor area:

```
<p>I am adding this line from developer tools!</p>
```

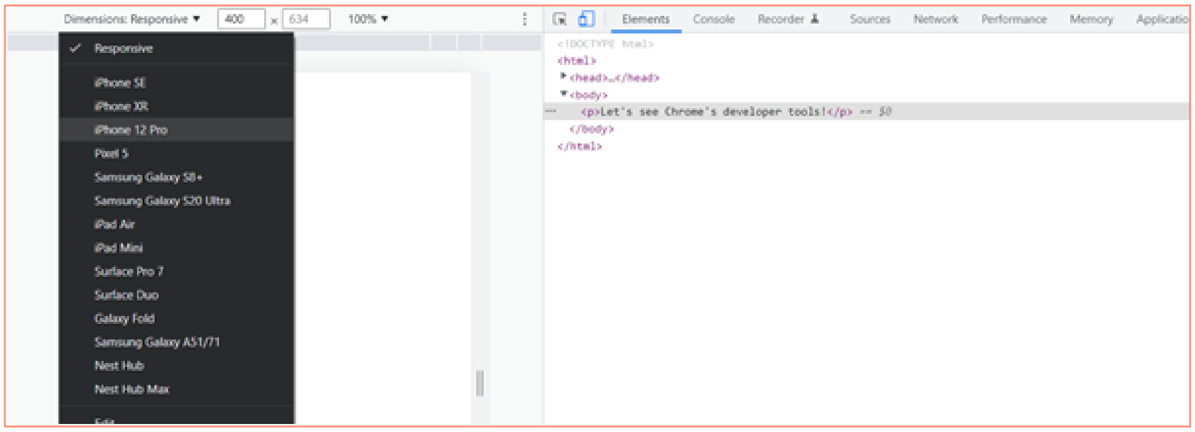
We can see that this line was added below our current paragraph on the web page to the left.

If we check our local source code file in VSCode, we can see that although the new text shows in the web browser, it doesn't show in the source code.

The right part of the Developer Tools view is about styles. We'll skip this for now, but remember that the code edition of styles works the same way as the HTML code edition; results are lost after refresh.

Now notice two icons on the left from the Elements tab. The left is called Inspector, while the right is responsible for the mobile view. We can select the right one to simulate displaying the website on a mobile phone. We can select different phone models from the dropdown menu.

As we can see below, there are plenty of models to choose from:

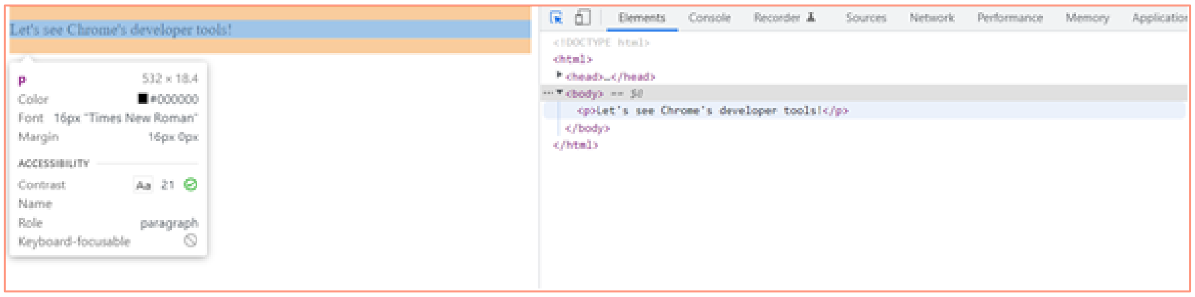


Dimensions: Responsive ▾ 400 × 634 100% ▾

```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body>
    <p>Let's see Chrome's developer tools!</p> == $0
  </body>
</html>
```

To come back to the personal computer view, deselect the right icon.

Now let's focus on the Inspector itself. Select the Inspector icon and move your mouse cursor on the paragraph text. We get the following result:



p 532 × 18.4
Color #000000
Font 16px "Times New Roman"
Margin 16px 0px
ACCESSIBILITY
Contrast Aa 21
Name
Role paragraph
Keyboard-focusable

```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body> == $0
    <p>Let's see Chrome's developer tools!</p>
  </body>
</html>
```

We can see that our paragraph is highlighted. Basic style info popped up close to the paragraph, and the HTML code responsible for the paragraph is highlighted in light blue. After clicking on the paragraph, we are moved to this line in the HTML code. It allows us to quickly move to the code we're interested in, which is helpful, especially in more complex applications. After clicking on the paragraph, Inspector is deselected. If we want to inspect another element, we must select it again.

WATCH

Video: Streamlining development with Chrome DevTools (1m 9s) by Sasha Vodnik on LinkedIn Learning.

ACTIVITY

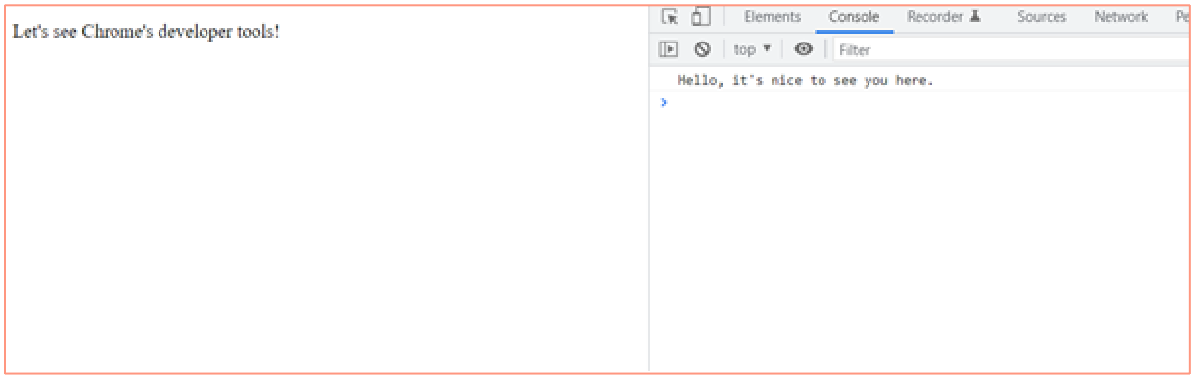
Open your favourite website and change any text element to “Hello from the Inspector!”

Browser developer tools - Console

In this section, we’ll talk about the Console tab. It allows us to show logs and apply JavaScript code during runtime without saving it permanently. Logs are printed information from the JavaScript code visible only in the console. We’ll use them as one of the debugging tools. For example, if our variable changes its value multiple times and its final value is other than expected, we can print the variable’s value after every change. We can also use logs to communicate with other developers or advanced users. For example, we can write logs about errors.

The primary tool to create logs is the method: console.log(). To invoke this, you need to put the parameter you want to see inside the parenthesis. It can either be a string or any other variable. We used this method twice in our JavaScript code. Our code has a 50% chance of showing us greetings and 50% of showing us the value of our variable.

The console tab looks like this:


Let's see Chrome's developer tools!

```
Hello, it's nice to see you here.
>
```

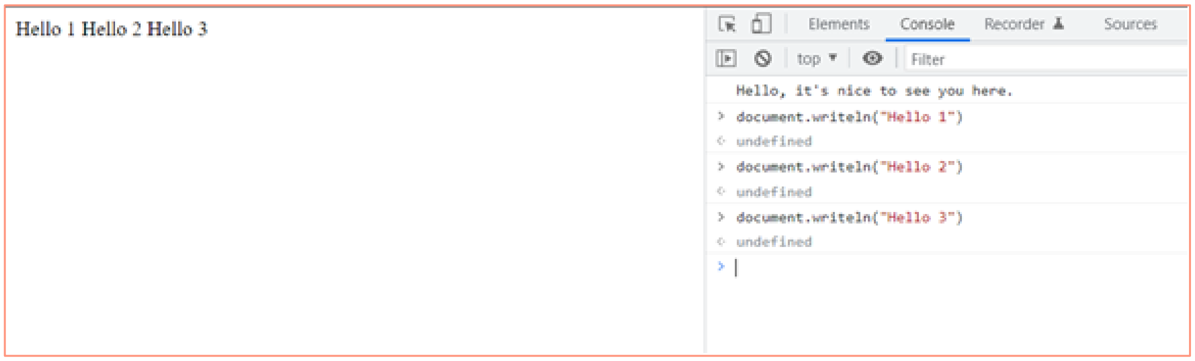
You can refresh the site to see that you'll get the greetings once and the number another time.

The small blue arrow under the greeting points to the space where we can write new JavaScript code. Let's add another line to the body of our application by pasting this command here:

```
document.writeln("Hello 1")
```

We can repeat this with the text "Hello 2" and "Hello 3 " as well:

We got the following result:


Hello 1 Hello 2 Hello 3

```
Hello, it's nice to see you here.
> document.writeln("Hello 1")
< undefined
> document.writeln("Hello 2")
< undefined
> document.writeln("Hello 3")
< undefined
> |
```

After refreshing the website, we're back to our standard page.

	WATCH Video: How to test JavaScript in the browser using the browser console – Ep 02 (4m 13s) by Jamis Charles on YouTube.

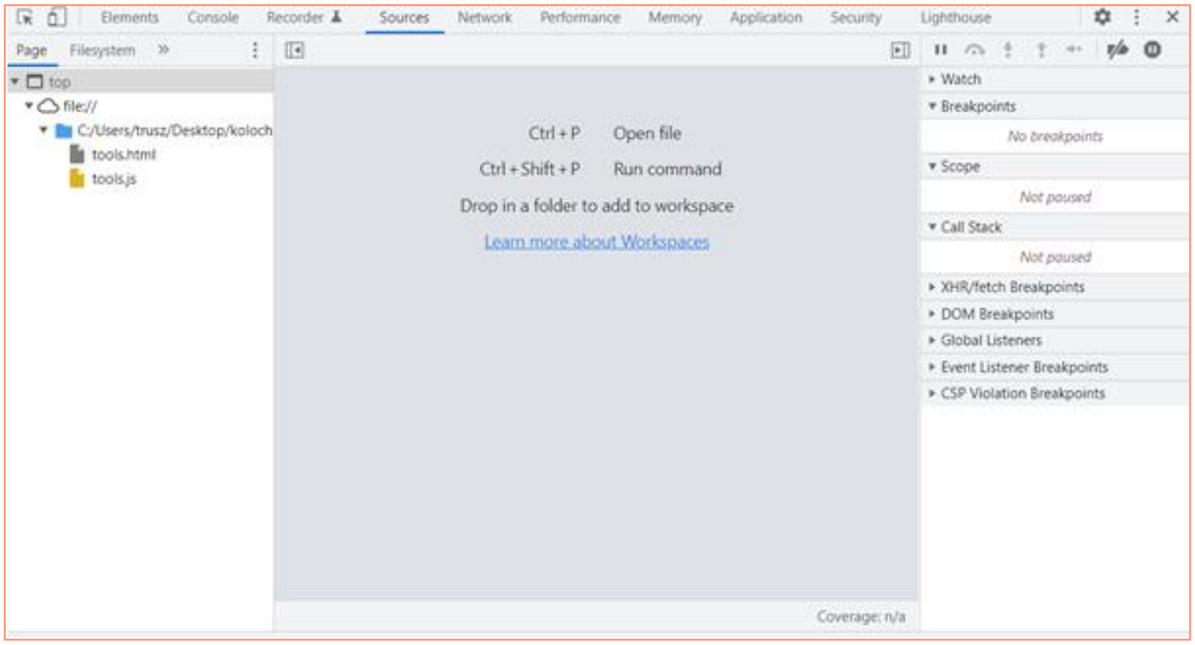
ACTIVITY

Open the browser’s console. Declare two variables and initialise them with numbers 1 and 4. Log their sum using the console.log() method.

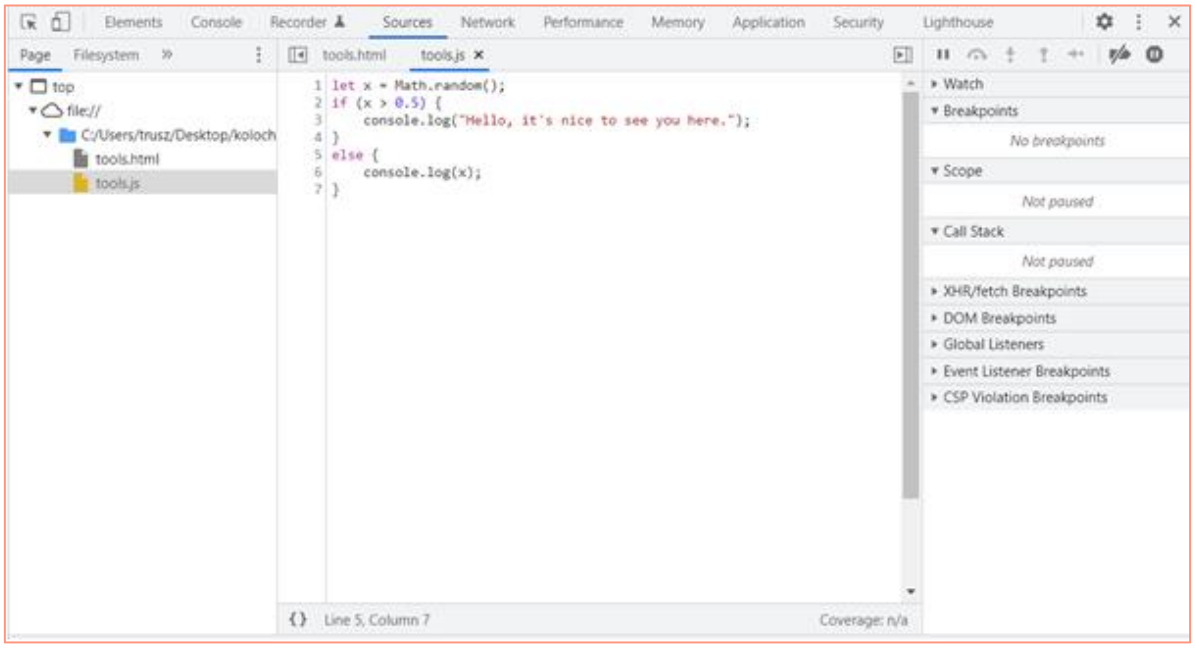
Browser developer tools - Debugger

Debugging is probably the most time-consuming part of a programmer’s job. Discovering why the application failed is one of the most important programming skills. To do this efficiently, we need additional tools. Logs are a useful and common way of debugging JavaScript, but Debugger enables other possibilities. Thanks to Debugger, we can pause the execution of our application in the selected place and look at the current values of our data. It helps us inspect the next phases of the application run and when the errors occur.

We should select the Source tab in the Developer Tools to run the Debugger. It should look like this:



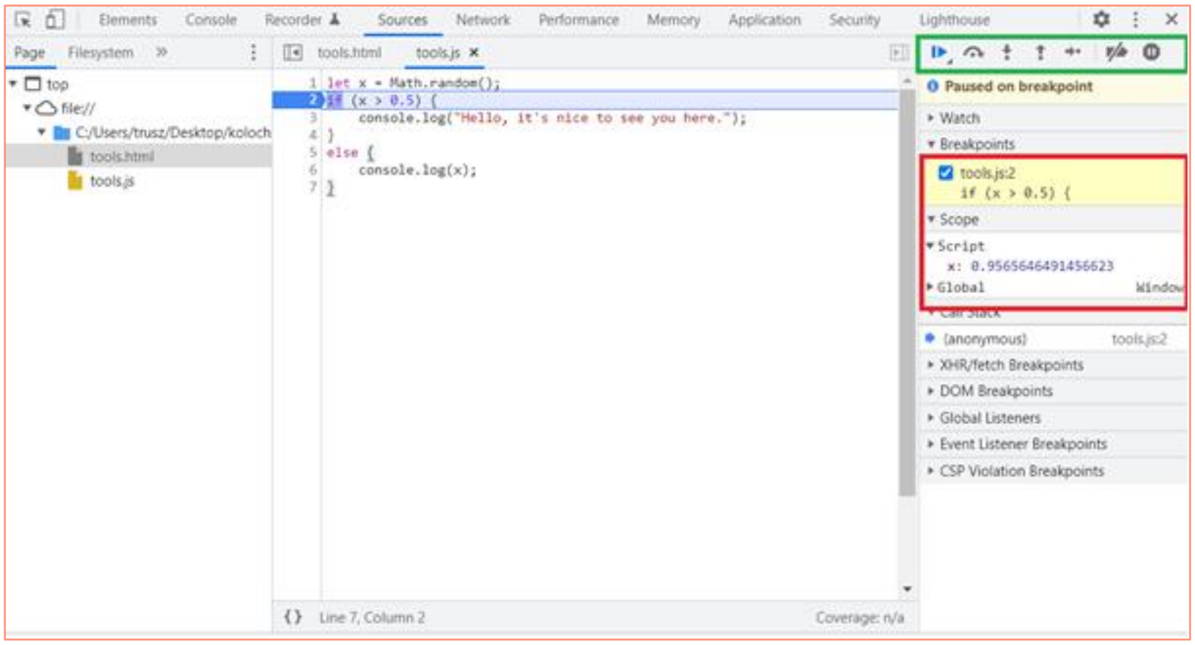
On the left-hand side, we can see the list of local files to select. Let's select the tools.js file. After clicking on the file's name on the middle part of the screen, we can see the code within that file. This code is what you'll see in VSCode in the same file:



```
1 let x = Math.random();
2 if (x > 0.5) {
3     console.log("Hello, it's nice to see you here.");
4 }
5 else {
6     console.log(x);
7 }
```

Notice the line numbers to the left of our code. We can create breakpoints by clicking on them. Breakpoints will cause our app to pause when the line number of the breakpoint is reached during code execution. Breakpoints save even after refreshing the website. This is because these breakpoints are saved in the web browser, not your local files.

Let's create a breakpoint on the second line (by clicking on the second line number) and refresh the application:



```
let x = Math.random();
if (x > 0.5) {
    console.log("Hello, it's nice to see you here.");
}
else {
    console.log(x);
}
```

```
Paused on breakpoint
Breakpoints
  tools.js:2
    if (x > 0.5) {
Scope
  Script
    x: 0.9565646491456623
  Global                Window
Call Stack
  (anonymous)           tools.js:2
```

The most interesting part is on the right side. In the scope menu (marked with the red rectangle), we can see that our x is close to 0.95. So, after resuming, our application will show the greetings in the console. The part marked with the green rectangle is very interesting as well. Starting from the left, clicking these buttons will cause the following effects:

- Resuming the application (after it goes into another breakpoint if there are more).
- Only the next line will execute.
- Only the next line will execute. We'll move to its implementation when going into the method.
- The entire code in this function or file will perform.
- Deactivates all breakpoints.
- Determines if the application should stop on exceptions. Exceptions are the special types of errors caused by the application code itself.

We can now click the first icon to resume our application. We can also click the second line number again to disable this breakpoint.

WATCH

1. Video: JavaScript breakpoints (4m 9s) by Christina Truong on LinkedIn Learning.
2. Video: Streamlining development with Chrome DevTools (1m 9s) by Sasha Vodnik on LinkedIn Learning.
3. Video: Best Debugging Tips For Beginners (11m 56s) by Web Dev Simplified on YouTube.
4. Video: Accessing and using the developer tools (3m 49s) by Sasha Vodnik on LinkedIn Learning.

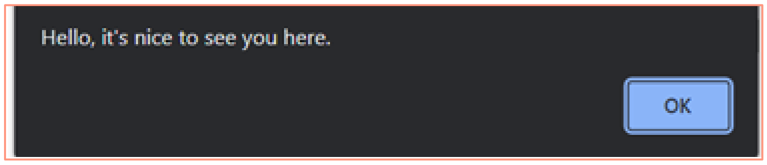
Alerts

Alerts are pop-up windows that provide critical information to the user. We use the alert() method to create an alert and put the information we want to provide inside the parenthesis. For example, this information could be incorrect input data from the user or some exception that occurred.

Let's investigate this method by replacing the console.log() method with alert() in our JavaScript code:

```
let x = Math.random();
if (x > 0.5) {
    alert("Hello, it's nice to see you here.");
}
else {
    alert(x);
}
```

After running the application, we see a pop-up similar to this:



Hello, it's nice to see you here.

After clicking the OK button, the pop-up closes, and we're back on our website.

There is also the analogous method: confirm(), which has the Cancel button.

You should use the alert method when you want to give the user plain information, while it's better to use the confirm method if you want to provide the user with a choice.

READ

1. Page: Window.alert() by MDN Web Docs.
2. Page: Window.confirm() by MDN Web Docs.

Prompts

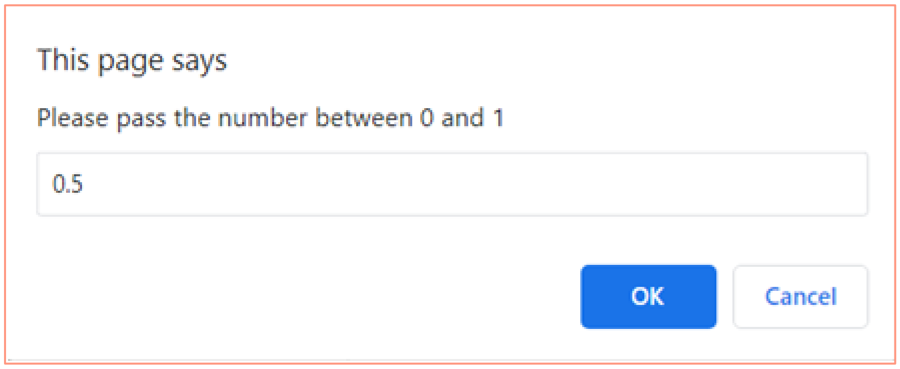
Prompts are the easiest way to communicate with users. They also help us a lot with testing the application. Passing parameters inside dialogue by temporarily editing code in the browser is way faster than changing values inside our code in VS Code, as we don't need to switch from the browser.

To create the prompt, we'll use the prompt() method. It has two parameters. The first one is the text of the question that will be visible to the user. The second one is optional and declares the default value. A comma separates parameters in the method's parenthesis.

Let's use the prompt method in our application. We'll replace generating random x by getting it directly from the user. Below we can see modified JavaScript code:

```
let x = prompt("Please pass the number between 0 and 1",
"0.5");
if (x > 0.5) {
    alert("Hello, it's nice to see you here.");
}
else {
    alert(x);
}
```

Variable x has a default value equal to 0.5. We can now run the application and see results similar to this:



After passing the value and clicking the OK button, we can see the appropriate alert based on the conditions in the if-else statement.

	READ Page: Window prompt() by W3 schools.

	WATCH Video: How to do JavaScript Popups (alert, confirm, prompt examples) (9m 45s) by Tony Teaches Tech on YouTube.

Comments

Comments are a valuable tool for improving code quality and providing an explanation to anyone who reads your code. They can also be used as a temporary block of code, but we're not interested in that now. There are two types of comments, one line and a long one. You can probably recall one-line comments from the previous lessons.

```
var a = 1; //this is one line comment
```

However, there is also the possibility of commenting on multiple lines. You can do this by starting with the sentence /* and finishing with */. You can see an example of a long comment below:

```
var a = 1; /*thiscomment is a bitlonger*/ var b = 2;
```

The code above works but is unclear. To follow good coding practices, long comments shouldn't be on the same lines as working code.

Be careful with the comments. You should use them mainly if your code is too complicated to understand. High-quality code usually has minimal, concise comments, as it is self-explanatory.

To comment and uncomment code easily in Visual Studio Code, you can use the shortcut: CTRL and/. This shortcut comments the line if it was uncommented and uncomments if it was commented.

	WATCH Video: The Art of Code Comments – Sarah Drasner | JSConf Hawaii 2020 (21m 12s) by JSConf on YouTube.

Introduction to arrays

Arrays are the perfect type to store a list of items. The easiest way of creating an array is using an array literal:

```
const my_array = [item1, item2, item3];
```

In our example, my_array stands for an array name. After choosing the name, we put the list of array items within square brackets. They can be of different types but should be connected in some way.

Notice that we used the keyword ‘const’ instead of ‘let’. A declared array can’t be reassigned, so it’s good practice to use the keyword ‘const’. However, it doesn’t define a constant array. Elements of the array can be and usually are reassigned. You can read more about this on the W3 schools tutorial linked in the READ section.

We can access elements of the array by invoking the array name and putting square brackets inside an index of an element we’re looking for. The index of items in an array begins at ‘0’, not ‘1’.

Therefore, the first item in an array is referenced by array[0]. We can also change the array’s values or add new items.

Let’s look at an example of creating the array element by element:

```
const my_array = [];
my_array[0] = "The First item";
my_array[1] = "The Second item";
my_array[2] = "The Third item";
my_array[3] = "The Fourth item";
console.log(my_array);
console.log(my_array[0]);
```

Remember that array indexing starts with 0.

We can use the length property to access the count of elements inside the array.

Let’s add the following line to our application and see the results:

```
console.log(my_array.length); //4
```

Knowing the number of elements inside the array allows us to access the last element of the array easily. Let’s change the first and last elements of our array now:

```
const my_array = []; my_array[0] = "The First item";
my_array[1] = "The Second item"; my_array[2] = "The Third
item"; my_array[3] = "The Fourth item"; my_array[0] = "The
new First item"; my_array[my_array.length - 1] = "The new
Last item"; console.log(my_array);
```

In logs, we can see that values of our first and last elements changed as expected.

	READ 1. Page: Arrays by Pluralsight. 2. Page: JavaScript Array Const by W3 schools.

	WATCH Video section: 7. Arrays (22m 6s) by Morten Rand-Hendriksen on LinkedIn Learning.

	ACTIVITY Create an array containing your name, age and the name of your favourite colour. Print this in the console. **Answer**: Click here to reveal.

Adding and removing elements from an array

As in the previous section, we can add new elements to the arrays by passing the value at the chosen index. If we choose the index higher than the current length of our array, the array will automatically be filled with undefined elements.

Let’s check the results of the following code:

```
const my_array = [];
my_array[0] = "The First item";
my_array[1] = "The Second item";
my_array[2] = "The Third item";
my_array[3] = "The Fourth item";
my_array[10] = "The Last item";
console.log(my_array);
console.log(my_array[4]);
```

We can see the array contains 11 elements, and six of them are undefined. Undefined is the default value of the new element, and arrays always have to stay compact.

But this isn’t the only way to add new elements to our array. The array contains methods push() and unshift(). Method push adds a new element to the end of the array, while method unshift adds the new element at the very beginning and then unshifts all other elements by one. This moves every value in the array up to the next index. We pass the parameter to be added inside the parenthesis.

Let’s look at a code example using these methods:

```
const my_array = [1,2,3,4];
my_array.push(10); /// [1, 2, 3, 4, 10]
my_array.unshift(11); // [11, 1, 2, 3, 4, 10]
my_array.push(12); // [11, 1, 2, 3, 4, 10, 12]
console.log(my_array); // [11, 1, 2, 3, 4, 10, 12]
```

Firstly, we add 10 at the end, 11 at the beginning and 12 at the end.

We now know how to add new elements to the array. Deleting elements from the array is similar. We can use method pop() to eliminate the last element or method shift() to delete the first element and shift all other elements by one position to the left. Unlike methods of adding new elements, these methods don't take any parameters.

Let's look at the code example:

```
const my_array = [1,2,3,4];
my_array.pop(); /// [1, 2, 3]
my_array.shift(); // [2, 3]
my_array.pop(); // [2]
console.log(my_array); // [2]
```

The first pop deletes 4 from the array. Shift deletes 1 and second pop deletes 3.

	WATCH Video: Arrays in code (4m 36s) by Morten Rand-Hendriksen on LinkedIn Learning.

	ACTIVITY With prompts, ask the user about their name, surname and age. Parse the age to int with the parseInt() method (converts string to the number if possible). Save that data into an array and show it in the console. **Source code:** Click here to reveal.

Other array methods

We can see that arrays are similar to strings in some ways. We could merge all values from the array into one string or split the string into smaller parts and store them in the array. There are methods to do this in JavaScript.

The toString() method is the most standard which returns the string from the array. In JavaScript, we can get a string from almost everything using this method. The string created with this method should look this way:

```
const my_array = ["Hey", "Hi", "Hello"];
console.log(my_array.toString()); //Hey,Hi,Hello
```

We can see the result is far from perfect. Elements are always separated with a comma, and there are no spaces between them. Fortunately, there's a better way to do the same thing. Method join() takes the separator we want to use as a parameter.

Let's split every greeting with the dash and spaces:

```
const my_array = ["Hey", "Hi", "Hello"];
console.log(my_array.join(" - ")); //Hey - Hi - Hello
```

Now let's focus on creating an array from the string. An analogous method to join does this: split(). We pass the separator as the parameter and get an array containing all the string parts. The split() method splits the string, putting the split into the next index of a new array every time a separator is found.

Let's look at an example of creating an array of all words used in a string:

```
const text = "Hello, it's nice to see you here.";
const words = text.split(" ");
console.log(words); //['Hello,', "it's", 'nice', 'to', 'see',
'you', 'here.']
```

Similarly to the strings, we can also merge multiple arrays. We'll use the method concat() to do this. We invoke this method on the array which contains the elements we want our final array to start with.

Let's look at an easy example of how to merge three different arrays into one:

```
const arr1 = [1, 2, 3, 4];
const arr2 = [11, 12, 13];
const arr3 = [5, 6, 7];
const final = arr1.concat(arr2, arr3);
console.log(final); //[1, 2, 3, 4, 11, 12, 13, 5, 6, 7]
```

Notice that we can pass as many arrays as we want in the concat() method. The final array elements order depends on the order of the parameters we pass.

We can achieve the same thing using the spread operator "…". It enables us to access all array elements.

Let's look at the code doing the same thing as in the previous one, but with the use of a spread operator:

```
const arr1 = [1, 2, 3, 4];
const arr2 = [11, 12, 13];
const arr3 = [5, 6, 7];
const final = [...arr1, ...arr2, ...arr3];
console.log(final); //[1, 2, 3, 4, 11, 12, 13, 5, 6, 7]
```

WATCH

1. Video: JavaScript String.split and Array.join (8m 41s) by Steve Griffith – Prof3ssorSt3v3 on YouTube.
2. Video: How to Merge Arrays – JavaScript Tutorial (7m 34s) by dcode on YouTube.

ACTIVITY

With prompts, ask the user about their name, surname and age. Create a one-element array for each of these properties. Concatenate all arrays and print the result in the console.

Source code: Click here to reveal.

Splicing and slicing

Arrays share many similarities with the string. Remember that strings have the powerful method slice(), which is used to extract a certain part from them. So do the arrays! The slice method works similarly to a string version. Arrays also have a very powerful splice method. It enables us to modify the input array; we can add or remove elements on the chosen indices.

Let's see a code example of using the array's slice method:

```
const numbers = [1,2,3,4,5,6];
const newNumbers = numbers.slice(1,3);
console.log(numbers); // [1, 2, 3, 4, 5, 6]
console.log(newNumbers); // [2, 3]
```

Above, we can see how to extract part of the array. Notice that our input array hasn't been modified. Remember, we can use negative indices to start counting from the end of the array. The slice method also allows us to pass only one parameter. In this case, we'll get all the values to the end:

```
const numbers = [1,2,3,4,5,6];
const newNumbers = numbers.slice(-3);
console.log(numbers); // [1, 2, 3, 4, 5, 6]
console.log(newNumbers); // [4, 5, 6]
```

Let's go through an example of using the splice method:

```
const numbers = [1,2,3,4,5,6];
numbers.splice(2,1,7,8);
console.log(numbers); // [1, 2, 7, 8, 4, 5, 6]
```

The first parameter stands for the index we're starting from. In this case, we start from the index: 2 – the beginning value stored at that index is 3.

The second parameter tells us how many values we want to delete. We want to delete one value, so we delete the first value starting from index 2, which is 3.

All other parameters stand for values we want to add to our array. We add them at the position specified as a first parameter.

	READ Page: JavaScript Array Methods by W3 schools.

	WATCH Video: Slice and Splice in Javascript (7m 55s) by Hitesh Choudhary on YouTube.

ACTIVITY

With prompts, ask the user about three numbers. Create an array containing all of them. Delete the middle element and add the mean of the first and the last element at the end of the array.

Source code: Click here to reveal.

Iterator methods

Array iterator methods allow us to perform general operations on every element in an array. To understand them, we'll need some basic knowledge about functions. We have already seen many methods. JavaScript allows us to create our own.

We create functions with the use of function keywords. After the function keyword, we give the function a descriptive name. Then we declare the number of parameters and define what our function will do within the code block. We can return a value from the function using the return keyword.

Let's look at a basic example of a function which takes one value and returns it multiplied by 2:

```
function myFunction(value) {
  return value * 2;
}
```

We'll go through the four most popular iteration methods: forEach, map, filter, and reduce. All these methods take functions as the parameters and apply that functions to all values of the array. These functions passed as the parameters are often named callback functions.

The forEach() method calls a callback function once for every array element. First, let’s create a simple callback function. Our function should take three parameters: the item value, its index and the entire array.

```
function callback(value, index, array) {
    return console.log("value: ", value, "index: ", index);
}
```

This callback function logs the value and the index of the element. Let’s look at the entire code example:

```
const numbers = [1,2,3,4,5,6];
numbers.forEach(callback);
function callback(value, index, array) {
    return console.log("value: ", value, "index: ", index);
}
```

In the console, we see that every element of the numbers array is printed, as expected. Notice that we can delete the third parameter (array) from the callback, since we aren’t using it here and won’t affect our application.

The map method modifies every element of our array. Let’s apply myFunction declared at the very beginning of the section on every element of the numbers array:

```
const numbers = [1,2,3,4,5,6];
const newNumbers = numbers.map(myFunction);
console.log(numbers); //[1, 2, 3, 4, 5, 6]
console.log(newNumbers); //[2, 4, 6, 8, 10, 12]
function myFunction(value) {
    return value * 2;
}
```

Notice that the map method doesn’t affect our input array but returns the results as the new separate array.

The filter method checks every element if it fits our condition. The callback function passed to the filter function should return a Boolean. The element from the input array will be in the result array only if the callback function returns true for its value.

Let’s use the filter function to only get even numbers from our array:

```
const numbers = [1,2,3,4,5,6];
const even = numbers.filter(isEven);
```

```
console.log(numbers); //[1, 2, 3, 4, 5, 6]
console.log(even); //[2, 4, 6]
function isEven(value) {
    return value % 2 == 0;
}
```

The filter function also doesn't modify our main array.

The last method is Reduce. Unlike the other methods, its callback function takes four parameters, and the entire method returns only one value instead of the entire array. As the first parameter in the callback function, we pass the current value of reduce function from all the previous elements. The reduce method runs from left to right in an array and produces a single value.

We can use the reduce function to calculate the sum of all parameters from the array:

```
const numbers = [1,2,3,4,5,6];
const sum = numbers.reduce(add);
console.log(numbers); //[1, 2, 3, 4, 5, 6]
console.log(sum); //21
function add(total, value) {
    return total + value;
}
```

Index and array parameters aren't necessary in our case. Reduce function can be hard to understand, so let's go iteration by iteration. The parameter total at the beginning is always equal to 0.

At first, we call the add function for 1. It returns 0 + 1 = 1. The returned value becomes the total parameter when we call add function for 2. Then it returns: 1 + 2 = 3. Then it becomes the total parameter when we call add function for 3. We have: 3 + 3 = 6. Analogously we further have: 6 + 4 = 10, 10 + 5 = 15 and 15 + 6 = 21, which is our final result.

In the iterator methods, we often use the arrow functions. Arrow functions are the other, shorter way to create functions. At first, we pass the parameters in the parenthesis, then write an arrow, and at the end, put our code block. We don't need to put the brackets if the block only has one line.

Let's look at the example above, but with the use of the arrow function:

```
const numbers = [1,2,3,4,5,6];
const sum = numbers.reduce((total, value) => total + value);
console.log(numbers); //[1, 2, 3, 4, 5, 6]
console.log(sum); //21
```

READ

Page: JavaScript Array Iteration by W3 schools.

WATCH

Video: Array methods (5m 53s) by Morten Rand-Hendriksen on LinkedIn Learning.

ACTIVITY

With the use of prompts, ask the user about three colours. Create an array containing all of them. Print an array containing all the colours' names but written in capital letters in the console. We can use the toUpperCase() method to write a string in capital letters.

Multidimensional arrays

We already know how to store variables within arrays and when it's a good solution. But what if we store arrays within arrays? JavaScript allows for this. An array storing another array is called a multidimensional array. Every time an array is nested inside another, it stands for one dimension of the array.

Let's look at some examples of two and three-dimensional arrays:

```
const numbers2d = [[0,1,2], [1,2,3], [1,2,4]];
const numbers3d = [[[0],[1],[2,3]], [[0, 1],[2,3]],
[[1],[1],[2,3]], [[1, 5],[2,3]]];
```

We can add as many dimensions as we want, but we'll rarely have to use arrays with dimensions greater than two.

To access the elements of a multidimensional array, we need to add as many square brackets as the array's dimension. Let's access the elements of the arrays we see above:

```
const numbers2d = [[0,1,2], [1,2,3], [1,2,4]];
const numbers3d = [[[0],[1],[2,3]], [[0, 1],[2,3]],
[[1],[1],[2,3]], [[1, 5],[2,3]]];
console.log(numbers2d[0][1]); //1
console.log(numbers3d[0][0][0]); //0
```

READ

1. Page: Arrays by MDN Web Docs.
2. Page: JavaScript Multidimensional Array by Programiz.

What did I learn in this lesson?

This lesson provided the following insights:

- Web browser developer tools and why they are essential for a JavaScript programmer.
- How to communicate with a user using alerts and prompts.
- How to comment on your code.
- Arrays and why we should use them.
- How to modify elements of the array one-by-one.
- How to perform a general operation on all elements of the array.

References

W3 schools (n.d.) *HTML Tutorial*. Available at: https://www.w3schools.com/html/default.asp [Accessed 25 May 2022].

1.3. Lesson task - Alerts and debugging

The task

Part 1

In this lesson, we learnt about debugging and communicating with a user. We saw why the browser's developer tool is handy and how to use it. We also modified our code to interact with the user. However, one of the tasks we did in this class wasn't completed, specifically the one about prompts. What if the user passes the wrong value with the prompt?

In this task, you must improve our code from this class:

```
let x = prompt("Please pass the number between 0 and 1",
"0.5");
```

```
if (x > 0.5) {
  alert("Hello, it's nice to see you here.");
}
else {
    alert(x);
}
```

The number x should only be visible if x is equal to or greater than 0 while being equal to or less than 0.5. If x isn't between 0 and 1, you need to ask the user to pass the number again. If the new data is correct, the application behaves like now.

However, if the new data is also incorrect, you should create an alert with the text "Your data is not correct!" and don't show any other alert. You can use the parseFloat() method to parse the string to float number.

ACTIVITY

You don't need to write any code in the second part of this task. You need to play a game – console.log() adventure.

View the code of this game and analyse it.

Part 2

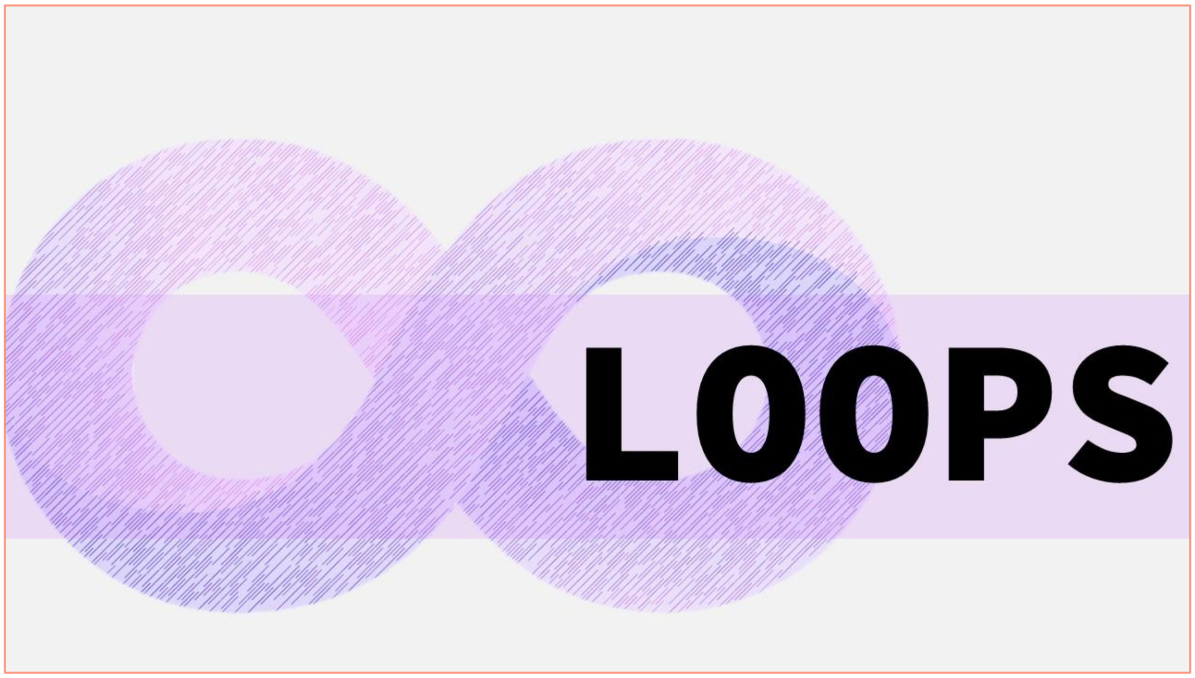
In this lesson, we learnt about arrays. You should know how to modify the array's elements and perform general operations.

For this task, create an array containing 10 names of vegetables. At least one vegetable name should have an even length. Using the iterator method, show the user an alert containing this name if the length of the vegetable name is even.

	ACTIVITY After completing this task, you can try your hand at finishing the Arrays test on Mozilla documentation. Try to do as many task sub-items as you can. You can also check more complicated exercises on JavaScript array - Exercises, Practice, Solution by w3 resource. After doing it, or if you aren’t able to do it, you can check their solutions.

1.4. Lesson - Loops

Introduction



Computer programming is usually associated with executing similar code multiple times. To sum up 10 numbers, we can quickly create 10 variables and add all of them. But what if we need to sum up 100 variables? Or 1000? Summing up that many are incredibly time-consuming and make our code unclean.

Fortunately, JavaScript provides us with loops. These loops enable us to perform certain operations, with different values each time, as often as we want.

There are many types of loops. Most problems can be solved with loops, but sometimes, using a specific loop type makes our code more straightforward. In this lesson, we'll learn when to use each.

For now, we already know some basics about programming, but our programs are usually still very small. However, even for now, some pieces of our code are very similar to each other. Fortunately, JavaScript provides a tool to reuse code in multiple places once written.

This tool is known as a function. Thanks to functions, we can use the same code many times with different arguments for different results.

Functions also enable us to interact with users. We can bind some functions to HTML elements to execute code after the user clicks this element. There are many other usages of functions, so learning them will help us become better programmers.

Learning outcomes

In this lesson, we are covering the following knowledge learning outcome:

- → The candidate has knowledge of program syntax, program structure, control structures, data types, and variables as used in the JavaScript language.

In this lesson, we are covering the following skill learning outcomes:

- → The candidate can apply knowledge of browsers and development tools to set up a computer to develop and debug small JavaScript programs.
- → The candidate masters the use of relevant tools for writing, editing, and debugging JavaScript source code.

In this lesson, we are covering the following general competence learning outcome:

- → The candidate can carry out simple programming tasks with the JavaScript language.

The for-loop

It's best to use the for-loop when we need to loop over a single piece of code multiple times. The for-loop consists of three elements: the initialiser, the condition and the final expression. It has the following syntax:

```
for(initializer; condition; finalExpression) {
    //execute this code
}
```

We will understand a single code execution within the block by iteration.

At the very beginning, the initialiser expression is executed. Then we perform the first iteration of our code within the code block. After this iteration, we perform the final expression and check if our condition is true. If it is, we perform the next iteration. If not, we go out of the loop.

Let's look at an example of the program logging every number from 0 to 100:

```
for(let i = 0; i <= 100; i++) {
    console.log(i);
}
```

Usually, we use the initialiser to set our iterator to the starting value. In this case, the first value we want to print is 0, so we set this to 0.

As we don't want to log numbers greater than 100, we want our condition part to check if our iterator is still within the accepted range after every iteration. If our iterator goes to 101, we want our loop to interrupt.

The final expression is usually about the iterator change. In our scenario, we want to write all integers between 0 and 100, so we increase the iterator by one every time. Using 'i++' is a shorter way of coding "i=i+1".

All three parts aren't mandatory. We can initialise our iterator before starting the loop. Without a condition, our loop will also work but never stop. We'll have to stop it manually at some point using the break keyword.

We can also always move our final expression to the last line of our code block without changing our program performance. However, using a structure for using all these three parts is commonly used, as in the example:

	READ Page: JavaScript For Loop by W3 schools.

	WATCH Video: For loops (3m 16s) by J. David Eisenberg on LinkedIn Learning.

	ACTIVITY Create an array containing 10 values. Print all of them using the for-loop. **Source code:** Click here to reveal.

The loop scope

What if we have many loops with many iterators? Do we always have to name our iterator differently? How do variables with the same name declared inside and outside the loop affect each other? We'll try to answer these questions.

The scope is a concept that refers to where values can be accessed. There are several scopes, for example:

- Global scope – values are accessible from anywhere.
- File scope – values can be accessed only within the file.
- Function scope – values are accessible within the function.
- Code block scope – values are only visible within a code block.

It turns out that sometimes the variable's scope depends on which keyword we use to initialise it. Let's look at the following examples:

```
var i = 5;
for (var i = 0; i < 10; i++) {
  // some code
}
// Here i is 10
```

```
let i = 5;
for (let i = 0; i < 10; i++) {
  // some code
}
// Here i is 5
```

In the first program, create the initialiser instead of the new iterator. Set its value to 0. The other program created the new iterator visible only within the loop scope.

It turns out that variables declared with the 'var' keyword are in the same scope as the for-loop, while variables declared with the 'let' keyword are local to the loop scope.

As long as we use the let keyword to initialise iterators, they will stay local and won't disturb each other.

	READ Page: for by MDN Web Docs.

	WATCH Video: Anatomy of a JavaScript For Loop (Lexical Scope vs Global Scope) (9m 8s) by GosuCoder on YouTube.

The for-in loop

The for-in loop iterates through all keys in our object. An array could be an example of the object, and its indices are equivalent to the object's keys. We'll demonstrate how for-in loops work on the arrays, but this loop can also work on other collections we don't know yet.

For-in loops have the following syntax:

```
for (key in object) {
    // code block to be executed
}
```

Let's use this loop to print all the elements from the array:

```
const numbers = [1,2,3,4,5,6];
for (index in numbers) {
    console.log(numbers[index]);
}
```

Be careful not to use this method when index order is important. The values might not be accessed in the order we expect. In this scenario, it's better to use for-loops, for-of loops or the Array.forEach() method.

Let’s look at another example of for-in loops. This time, we’ll use a string instead of an array. We’ll log every single character of our string:

```
let text = "Some pretty long string.";
for (index in text) {
    console.log(text.charAt(index));
}
```

This example illustrates that the for-in loop works for many types of objects, not only for arrays.

READ

Page: JavaScript For In by W3 schools.

WATCH

Video: JavaScript for-in loop (2m 58s) by Bro Code on YouTube.

The for-of loop

The for-of method is similar to the for-in method but iterates through values instead of keys. It has the following syntax:

```
for (value of object) {
    // code block to be executed
}
```

Let’s see how our code changes to achieve the same result as programs from the for-in loop section:

```
const numbers = [1,2,3,4,5,6];
for (value of numbers) {
    console.log(value);
}

let text = "Some pretty long string.";
for (value of text) {
    console.log(value);
}
```

Our code is shorter and cleaner. But if we needed to perform calculations based on indices, we wouldn't have that information easily accessible from here.

	READ Page: JavaScript For Of by W3 schools.

	WATCH Video: For-of loop with an Array in JavaScript (5m 2s) by Telusko on YouTube.

The while-loop

The while-loop is different from all the previous loops. The previous loops focus on performing an established number of iterations on our data. The while-loop focuses on the specified condition. It has the following syntax:

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

The while-loop can be dangerous to beginners, as the programmer should always make sure that our condition will be false at some point. Otherwise, we'll be stuck in the infinite loop, suspending our application.

Let's see a working example of the while-loop:

```
let isLessThanTen = true;
let i = 0;
while (isLessThanTen) {
    i++;
    if (i < 10) {
        isLessThanTen = true;
    }
    else {
        isLessThanTen = false;
    }
    console.log(isLessThanTen);
}
```

The program checks if our variable is less than 10. If the variable reaches 10, the program stops. Note that if we should cut the *else* part from the *if* (setting isLessThanTen to false), we would reach the infinite loop.

There is another variant of the while-loop. Do-while loops work the same but always execute the first iteration, even if the condition is false. Do-while loops have the following syntax:

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

The standard variant of the while-loop is mainly used, but if you need your first iteration always to execute, the use of the do-while variant is a perfect solution.

READ

Page: JavaScript While Loop by W3 schools.

WATCH

1. Video: While Loops (5m 36s) by Joe Chellman on LinkedIn Learning.
2. Video: While structure (2m 37s) by J. David Eisenberg on LinkedIn Learning.

ACTIVITY

Create an array containing some numbers. Print all numbers from the array beginning until becoming a number which is a multiplicity of 7. Print information that the first multiplicity of 7 has been found and get out of the loop. To check whether the number is a multiplicity of 7, we can use the remainder operator (%).

Source code: Click here to reveal.

Disrupting the loop - break and continue

Sometimes you might want to interrupt the loop. For example, we started the loop only to check if a particular value was in our array. After some iterations, we found it, and the job was done, but there are still plenty of iterations to execute.

Fortunately, JavaScript gives us the ability to interrupt the loop. We can use the break keyword to jump out of a loop.

On the other side, the keyword ‘continue’ enables skipping only one iteration of the loop, which also can be very useful. For example, suppose we have to check multiple conditions in one iteration, and one of these has already failed. In that case, we can go to the next iteration to save time instead of executing pointless calculations.

Let’s look at some working examples of how to use the break and continue keywords:

```
for(let i = 0; i < 10; i++) {
    if(i == 4) {
        continue;
    }
    console.log(i);
}
```

The code above logs every number from 0 to 9 instead of 4. If the iterator equals 4, the iteration is skipped.

```
let i = 0;
while(true) {
    i++;
    if(i >= 10) {
        break;
    }
    console.log(i);
}
```

This code fragment illustrates another way to deal with infinite loops. We can use the break keyword to interrupt them.

READ

Page: JavaScript Break and Continue by W3 schools.

WATCH

Video: JavaScript for Beginners #39 Break and Continue in Loops (12m 8s) by GreeneMath.com on YouTube.

ACTIVITY

Create an infinite loop and ask the user (via prompt) to pass an even number. If the number was even, end the loop and show that number in the alert. Otherwise, show another prompt and ask for an even number again.

Source code: Click here to reveal.

Introduction to functions

Let's remind ourselves of what the function syntax looks like:

```
function functionName(parameters) {
    // code to be executed
}
```

We use the keyword function to define the function, followed by the name of our function. We define as many parameters as we want to pass, and at the very end, we implement the code to be executed within the function code block.

Let's look at a basic example of a function that multiplies two variables and returns their multiplication:

```
function multiply(a, b) {
    return a * b;
}
```

We say the function is called or invoked when we use the function name with the () operator. We can invoke this function with the code below:

```
multiply(1,2);
```

The return keyword stops the execution of the function. If we assigned a variable to the function invocation, the value after the return keyword would be assigned to our variable:

```
let x = multiply(1,2); //x = 2
```

We can also use functions as variable values, like in the code example below:

```
let x = multiply(1,2); //x = 2
let text1 = "One times two equals" + x;
let text2 = "One times two equals" + multiply(1,2);
```

If we call the function this way, we must remember to put its declaration at the very beginning of the file. In JavaScript, we need to be careful not to invoke a function until it's declared. The code below results in error, as we call the function before its declaration:

```
multiply(1,2);
function multiply(a, b) {
    return a * b;
}
```

READ

Page: JavaScript Functions by W3 schools.

WATCH

Video: What are functions? (5m 32s) by Ray Villalobos on LinkedIn Learning.

ACTIVITY

Declare and create a function that adds three numbers. Call this function and set the returned sum of numbers to a variable.

Source code: Click here to reveal.

Function parameters

As we already know, our function can have as many parameters as we want. Since 2015, we can also set up default values of the parameters. The function below will log the default value (2) if we don't pass the second parameter (i.e. if we don't provide the value of the second parameter):

```
function myFunction(x, y = 2) {
    console.log(y);
}
myFunction(1); //2
myFunction(1, 3); //2
```

There's also a way to deal with a scenario where we have provided more arguments than expected. We can access them via the Arguments Object. Let's look at a working example of summing any number of provided arguments:

```
function sum() {
  let sum = 0;
  for (let i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
}
x = sum(1, 23, 45, 67, 89); //225
```

We accessed all arguments from the Arguments Object and adequately calculated the sum.

Arguments can be passed either by value or reference. When an argument is passed by value, we only get to know its value, and we're working on a copy of it. Because of that, our changes won't affect the original object and won't be visible outside the function.

When we pass the argument by reference, we work on the actual object, not the copy. Any changes made to it will be visible even outside the function.

In JavaScript, arguments are passed by value, so we don't have to worry about modifying them. Only objects behave like 'passed by reference', but we haven't had objects in this course yet.

READ

1. Page: JavaScript Function Parameters by W3 schools.
2. Page: JavaScript Function Invocation by W3 schools.
3. Page: JavaScript ES6 by W3 schools.

	WATCH Video: Using the arguments parameter (2m 20s) by Ray Villalobos on LinkedIn Learning.

	ACTIVITY Write a function that multiplies all arguments (we assume all are numbers) and returns the result. We should be able to pass as many arguments as we want. **Source code:** Click here to reveal.

Function expressions

There are also other ways to define a function. We can use function expressions to do this. Using these, we can store the function with a single variable. Let's look at an example of the function expression:

```
const x = function (a, b) {return a * b};
let z = x(3, 4);
```

Note that our function doesn't have a name. We don't need a name, as we always invoke it using the variable name. Functions without names are also called anonymous functions.

READ

1. Page: JavaScript Function Definitions by W3 schools.
2. Article: JavaScript | Function expression by Abhishek Kharmale on GeeksforGeeks.

WATCH

Video: Returning values (2m 43s) by Ray Villalobos on LinkedIn Learning.

ACTIVITY

Write a function expression that multiplies all arguments (we assume all are numbers) and returns the result. We should be able to pass as many arguments as we want.

Source code: Click here to reveal.

Arrow functions

With ECMAScript 2015, we can use arrow functions instead of function expressions. It allows us to write a short function without using the function keyword. We can even omit the return keyword if our function is only a single statement.

Let's look at a simple arrow function which does the same job as our multiply function:

```
const x = (x, y) => x * y;
```

If our function is longer than one statement, it would also consist of curly brackets and the return keyword:

```
const x = (x, y) => { return x * y };
```

Arrow functions don't have binding to the Arguments Object like the standard functions. Usually, we can solve this problem with the use of the rest parameter. The rest parameter stores an array of excess parameters. It allows us to represent an indefinite number of arguments as an array.

We can see the arrow function with the rest parameter on the code example below:

```
var add = (...args) => {
    var total = 0;
    for (var i = 0; i < args.length; i++) {
        total += args[i];
    }
    return total;
};
```

Our arrow function sums up all parameters. The rest parameter always starts with … (ellipses). We can access all passed parameters from the args array, no matter how many.

READ

1. Page: JavaScript Function Definitions by W3 schools.
2. Page: Arrow function expressions by MDN Web Docs.

WATCH

1. Video: JS Arrow Functions: Everything You Need to Know (20m 22s) by Colt Steele on YouTube.
2. Video: The arrow function (4m 49s) by Morten Rand-Hendriksen on LinkedIn Learning.

Binding functions to the HTML

After learning the functions, we can use them to improve our view. Let’s copy this HTML and JS code:

functions.html:

```
<!DOCTYPE html>
<html>
    <head>
        <title>App</title>
        <script src="functions.js"></script>
    </head>
    <body>
        <button style="width: 100px; height: 100px;"
onclick="increment()"></button>
    </body>
</html>
```

functions.js:

```
let x = 0;
function increment() {
    x = x + 1;
    console.log(x);
}
```

We added one button to our HTML view. We also created a function increment() which increases the value of x by 1 with every invocation. Let's focus on the onclick part of the HTML file.

Onclick is an event. In JavaScript, events enable us to execute our code every time something happens (in our case, a button is clicked). In this way, we bind our JavaScript code to the button. Every button click starts an event that invocates our function. We'll learn more about events later.

Let's move to the website and click our button. In the console, we can see that our x value changes with every click by one, which means that clicking the button invokes our increment() function as expected.

WATCH

1. Video: How to add onclick to button in JavaScript | JavaScript Tutorial (8m 29s) by Chart JS on YouTube.
2. Video: DOM events explained (1m 41s) by Morten Rand-Hendriksen on LinkedIn Learning.

READ

Page: Functions by MDN Web Docs.

What did I learn in this lesson?

This lesson provided the following insights:

- How to access excess function parameters.
- Arrow functions and how to use them.

- Binding the function to simple HTML controls.
- Loops and why we should use them.
- When to use every type of the loop.
- How to disrupt the loopPage 15/15.

References

W3 schools (n.d.) *HTML Tutorial*. Available at: https://www.w3schools.com/html/default.asp [Accessed 25 May 2022].

MDN Web Docs (2022) *JavaScript*. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript [Accessed 25 May 2022].

1.4. Lesson task - Loops

The task

Part 1

In this lesson, we learnt about loops: for-loops, for-in loops, for-of loops, while-loops and how to interrupt them. To convert strings to numbers, use the parseFloat() method.

This task is divided into two sections:

1. Load the data from the user (via prompt). You should get a number greater than 1 but less than 20. If the data from the user is incorrect, you should show them another prompt repeatedly until you get the correct number.
2. Print all multiplicities of these positive integer numbers in the console that are less than 1000 and not multiplicities of 23. Which loop is best for our task?

ACTIVITY

After completing this task, you can try finishing the Loops test on Mozilla documentation. Try to do as many task sub-items as you can.

You can also do more complicated exercises on JavaScript conditional statements and loops - Exercises, Practice, Solution by w3resource. After doing it, or if you aren't able to do it, you can check their solutions.

Part 2

In this lesson, we learnt about the functions, function expressions and arrow functions. We also finally used functions to improve our web application.

In this task, we'll continue the development of our web application:

In the beginning, get the input value from the user. The value should be between 0 and 100. Create two buttons and apply two functions to them on click: one will increase the stored value by 1, while the second function will decrease it.

After every button click, show the current variable value in the console. After reaching either 0 or 100, instead of increasing/decreasing the value, communicate "Game over" in the console.

```
<!DOCTYPE html>
<html>
    <head>
        <title>App</title>
    </head>
    <body>
        <button onclick="increment()">Increment</button>
        <button onclick="decrement()">Decrement</button>
        <script src="homework.js"></script>
    </body>
</html>
```

ACTIVITY

After completing this task, you can try your hand at finishing some function exercises on the w3resource platform. You can find many exciting activities with their solutions. Completing some of them will help you understand the functions.

1.5. Lesson - Self-study

Introduction

This is a self-study lesson which consolidates knowledge from the first module. It contains a video summarising the basics of JavaScript programming, including variables, data types, operators and conditionals. It also includes exercises.

Materials

WATCH

1. Video course: Learning the JavaScript Language (2h 53m) by Joe Chellman on LinkedIn Learning.
2. Video sections: 1–9 Coding for Visual Learners: Learning JavaScript from Scratch by Engin Arslan on LinkedIn Learning.

1.5. Lesson task - Self-study

The task

Here are the online exercises on the W3 school's page.

Chapters to complete:

- Variables
- Operators
- Data Types
- Strings
- String Methods
- Comparisons
- Conditions
- Switch

	NOTE Try not to use the ‘Show answer’ option before trying to solve the exercise.

Noroff
Education