

SSDP 2000 – Day 1

Course: JavaScript Fundamentals

Instructor: Gabbie Bade

Agenda

- Course Overview
- Introduction to JavaScript
 - JavaScript Syntax
 - Data Types
 - Variables
 - Window Methods
- Click Event Handlers
- Basic Debugging in JS
- Introduction to the Final Project
- Assignment #1

Course Overview

What you will learn

- Write basic JavaScript for execution in modern web browsers, including fundamental syntax and programming structure
- Understand and use core JavaScript concepts such as variables, functions, loops, arrays, conditionals, and objects
- Implement common interactive web behaviors including event handling, DOM manipulation, animations, modals, and form validation
- Apply asynchronous programming basics including working with APIs using fetch, async/await, and handling JSON data

What you won't learn

- Advanced JavaScript features and in-depth debugging
- Frameworks or libraries like React.js, Vue, or jQuery
- Backend tools like Node.js, databases, or server-side development
- Build tools, deployment, or production-level workflows

General Flow and Expectations

- Similar to SSDP 1100, all course materials and assignments will be available on the Learning Hub on the day of the lecture.
- Varying scripting ability levels:
 - If JS looks like alien language to you, do not be discouraged. Ask questions and use additional resources. Focus on YOUR learning.
 - If you know your stuff, great! Support your classmates when you can. Please be courteous, and please still pay attention during class.

Introduction to JavaScript

JavaScript in the Browser

- JS is the behaviour language of the web

HTML:

Content, Structure



CSS:

Design, Layout



JavaScript:

Behaviour, Advanced Interactions



What is JavaScript

- JavaScript runs directly inside web browsers, no extra plugins needed.
- It uses similar programming rules (like if, while, switch) as some other languages.
- The type of data in variables is set automatically as the code runs.
- It's widely used to create interactive features such as buttons, sliders, animations, and form validation.

What JavaScript Can Do

(within the Browser)

- Dynamic animations that run based on user interaction
- Inject new content into a document dynamically
- Update, change, delete and move existing content on the HTML **DOM**
- Validate forms
- Trigger code based on browser events such as scrolling, clicking, resizing, mouse movements and other events
- Browser based video games

Source: What is the DOM?

What JavaScript Can't Do

(within the Browser)

- The following points are only regarding JavaScript that runs **in a browser**. Other flavors of JavaScript such as Node.js may not have these restrictions
- JavaScript can not access a user's file system
 - Reading, writing, deleting and changing a user's files stored in the native operating system is not possible
- Change or effect code located on other external websites
 - You can not write JavaScript that will affect another web site

What JavaScript is NOT

- JavaScript is NOT Java
- JavaScript is NOT a light version of Java
- JavaScript has no relation to Java, they are completely separate languages



JavaScript Can Be Disabled by the User

- Be aware that a user can disable JavaScript in the browser by changing a simple browser setting
 - If they do, none of your JavaScript code will run
 - You can not force your users to run JavaScript
- The vast majority of people leave JavaScript enabled, so this is only a minor concern
- If your web app or web site requires JavaScript to run, you can display a message inside a "<noscript></noscript>" tag and ask your users to enable JavaScript

JavaScript & Ecma Script

- JavaScript follows the ECMA Script standard, which defines the language rules.
- Versions like ES5, ES6 (ES2015), ES2017, and others add new features and improvements.
- Modern browsers support most features from ES6 onward; older versions like ES5 are still common.

Sources: ECMA International (ecma-international.org), TC39 (tc39.es), MDN Web Docs

JavaScript & Ecma Script

- JavaScript follows the ECMA Script standard, which defines the language rules.
- Versions like ES5, **ES6** (ES2015), ES2017, and others add new features and improvements.
- Modern browsers support most features from ES6 onward; older versions like ES5 are still common.

Sources: ECMA International (ecma-international.org), TC39 (tc39.es), MDN Web Docs

Future Versions of JavaScript

- Ecma has decided to adopt a yearly release cycle for JavaScript
- Expect new features of JavaScript to be introduced on an annual basis
- Research what new features of JavaScript your target browsers support

JS Syntax


Adding JS to Your Web Page

- **Inline** – *Not recommended*
 - JavaScript is written inside an element as an event handler
- **Embedded** – *Useful in specific scenarios*
 - JavaScript code is embedded into the HTML page directly by writing JavaScript between "<script></script>" tags
- **External** - *Recommended method for most scripts*
 - JavaScript is written in separate ".js" files and attached to the HTML by adding a "src" attribute to a "script" element

Inline JavaScript Syntax

- JavaScript written inside an HTML tag (not recommended)

```
<button onclick="alert('Hello World')">Click Me</button>
```



This inline JavaScript code adds a "click" event handler that will show an alert pop-up box with the text of "Hello World" when the button is clicked

Embedded JavaScript Syntax

```
<button id="btn-say-hello">Click Me</button>

<script>

  document.getElementById('btn-say-
  hello').addEventListener('click', _ => alert('Hello World') );

</script>
</body>
</html>
```



This embedded JavaScript code adds a "click" event handler to the button with an "id" of "btn-say-hello". The "click" event handler will display an alert pop-up box with the text of "Hello World"

Embedded JavaScript Syntax

■ Advantages:

- Good solution for small single page web sites
- Good solution for small scripts that only need to run on a single page
- The "<script></script>" tags can go anywhere in the HTML file, but are most often placed in either the "head" section of an HTML file or just before the closing "</body>" element

■ Disadvantages:

- Different languages mixed into the same file
- Prevents browser caching of scripts
- Impossible to reuse scripts across multiple pages

External JavaScript Syntax

- JavaScript is written in external text files saved with the ".js" extension
- JavaScript files are attached to the HTML via a "src" attribute on a "<script></script>" element

```
<head>  
  <title>Document</title>  
  <link rel="stylesheet" href="styles/styles.css" >  
  <script src="scripts/script.js" defer></script>  
</head>
```



The external JavaScript file "script.js", which is located in a "scripts" folder is attached to an HTML document via a "src" attribute placed on a "<script></script>" element

Two Ways to Link External JavaScript

- **Option 1:**

Inside the head tag with defer attribute

- Script downloads asynchronously during HTML parsing
- Executes only after document is fully parsed
- Doesn't block page rendering or delay content display
- Maintains script execution order

- **Option 2:**

Before closing </body> tag

- Script loads after HTML content is parsed
- Ensures DOM is ready before JS runs
- May delay rendering if script is large
- Common in older web development

JavaScript Syntax

JavaScript is an object-based language that uses dot "." syntax to call methods (things an object can do) and properties (things that describe an object)

```
document.getElementById('p-01').innerHTML = 'Hello World';
```

This is the "document" object. This object is created for you by the browser and represents the HTML document

Method or property. Here we are calling the "getElementById" method on the document object. Methods have "()" after their name. Inside the parentheses is ID.

Objects can be chained. In this case the "getElementById" method selects an HTML element which is itself is a new object that has an "innerHTML" property

Here we are changing the value of the "innerHTML" property on the "p-01" HTML element to "Hello World".

JavaScript is Case Sensitive

```
document.getelementById('p-01');
```

BAD!!! This will not work. Properties, methods, objects, variables and keywords must match their case exactly

```
document.getElementById('p-01');
```

GOOD!!! This example calls the "getElementById" with the correct casing

What is Camel Case?

- Most built-in methods and properties in JavaScript are written in camel case
- To write a variable, method, property or function in camel case you write the first word in all lowercase and then the first letter in subsequent words are uppercased

```
const billTotalAfterTax = 23.57;
```


First word is all lowercase.

The first letter of subsequent words are uppercase.

The Semi-Colon

- Unlike many other languages, the ";" character at the end of a line of JavaScript code, called a "statement" is usually optional
- The choice to either add a semi-colon at the end of a JavaScript statement or not use a semi-colon is an endless debate
- No right or wrong answer here.

```
document.getElementById('p-01').innerHTML = 'Hello World';
```



The semi-colon at the end of a JavaScript statement is usually optional

JavaScript Code Comments

- Explaining code blocks to other developers or to your "future" self when looking at your code several months later
- Temporarily turning off blocks of code by commenting it out for troubleshooting or other development needs
- Script version numbers
- Providing attribution information for code sourced from other places
- Copyright and licence information

JavaScript Comment Syntax

Multi-line Comment

```
/*
```

- This is a multiline comment
- All text between the opening comment marker and the closing comment marker will be ignored

```
*/
```

Single-line Comment

```
// This is a single-line comment
```

JS Code Comments are Public

- JavaScript code is not compiled, it is delivered as is (in plain text) directly to the browser
- Even if you minify your code, it is still a one click operation to "prettify" your minified code and then view it directly in the browser
- Avoid the following in your comments:
 - Passwords and login information
 - User data
 - Private information that you do not want public
 - Probably best to avoid offensive content in your comments

Data Types & Variables

What is a Variable?

- It is used to store information (data) to be referenced (retrieved) and manipulated in a computer program¹
- Developers use **descriptive variable names** as labels on data for easier retrieval of the data and to aid in understanding what kind of data is being stored in the variable¹



¹ <https://launchschool.com/books/ruby/read/variables>

Variable Scope

- Scope refers to where a variable is accessible in your code/ program.
- Unlike some other languages, JavaScript variables declared in a parent function (or the global space) are available inside any child functions

Scope Levels

- Global scope: Accessible everywhere
- Function scope: Accessible **only in the function** it was declared in
- Block Scope: Accessible only **within the nearest {} block** (ie. blocks, loops, conditionals)

const, let, or var?

Keyword	Scope	Can Reassign?	Recommended Use
const	Block, Global	No	Modern JS
let	Block, Global	Yes	Modern JS
var	Function, Global	Yes	Legacy only (IE10 and below)

Data Types & Variables in JS

- Developers **do not need** to declare a data type when creating variables
- JavaScript will automatically set the data type of a variable based on the value stored in a variable
- Variables in JavaScript can change their data type dynamically.

```
let foo = 23;  
foo = 'Bob';
```



This is valid JavaScript code, but is generally considered a bad practice

Some Common Data Types

- Data stored in variables are often associated with a type of data

- Strings

"Hello world" or 'Bob is 21'

- Numbers

123 or 3.14

- Booleans – *True or False*

String Data Type

- A string is a series of characters "strung" together.
- In JS, they are always wrapped in quotes ("", "", ``)
- There is no difference between using single quotes and double quotes
 - The string 'Hello World' and "Hello World" are functionally similar in JavaScript
- In the upcoming slides we will discuss template strings (``)

```
const user = 'Matthew';
```



The variable "user" is a string data type

Escaping Special Characters

- If we have a string that contains a special character that would otherwise confuse the JavaScript parser we can use the "\" to escape that special character

```
const someText = 'The Terminator said "I'll be back!";
```



```
const someText = 'The Terminator said "I\'ll be back!";
```



String Concatenation

- To concatenate a string means to combine 2 or more strings together to form a single string
- In JavaScript, one method to combine a string is to use the " + " character
- JavaScript does not know grammar

Example 1 – with no space

```
const firstname = 'John';  
const lastname = 'Smith';  
const fullname = firstName + lastName;  
// fullname -> JohnSmith
```

Example 2 – with space

```
const firstname = 'John';  
const lastname = 'Smith';  
const fullname = firstName + ' ' + lastName;  
// fullname -> John Smith
```


Number Data Type

- In JavaScript all numbers (integers and floating point decimal numbers) are assigned the number data type
- Most of the usual operations you may be familiar with from math or other programming languages are available by default in JavaScript

+ (add)

- (subtract)

* (multiplication)

/ (divide)

% (modulus)

What is the Modulus Operator (%)?

- The modulus operator returns the remainder of a division between two numbers
 - $17 \% 5 \rightarrow$ returns **2**
(17 divided by 5 equals 3 with a remainder of 2)
- If the dividend (the first number) is smaller than the divisor (the second number) then the modulus returns the dividend
 - $5 \% 17 \rightarrow$ returns **5**

The Order of Operations

- JavaScript follows the order of operations

- In Canada we learn the acronym BEDMAS

- JavaScript follows BEDMAS

- Make sure you write your equations correctly to avoid unpredictable results

```
const total = 5 + 5 * 3;  
// total -> 20
```

or...

```
const total = (5 + 5) * 3;  
// total -> 30
```

Beware of String Numbers in JS

- A string number is a string with a number as its value
 - '23' -> since this "number" is wrapped in quotes it is considered to have a string data type
- With JavaScript that runs in a browser we often get data from the HTML document from form fields or prompt boxes
 - Almost all data your script receives from the HTML document is a string, even if it is a number, it is always a string data type
 - This even applies to **form fields with an input type of "number"**.

Beware of String Numbers in JS

- JavaScript will not always automatically convert string numbers to numbers
- Be especially cautious when adding two string numbers together in JavaScript as you will get unpredictable results

```
const formInput1 = '23';  
const formInput2 = '7';  
const total = formInput1 + formInput2;  
// total -> 237
```

String to Number Conversion

- Dealing with string numbers is a common problem in JavaScript that runs in the browser
- JavaScript provides several ways to convert a string number to a number data type
- On the next slides we will look at three options

parseFloat()

- **parseFloat()** will convert a floating point (decimal number) and an integer string number to a number

```
const strNum1 = '23.571';  
const strNum2 = '7.258';  
const total = parseFloat(strNum1) + parseFloat(strNum2);  
// total -> 30.829  
// If these were added WITHOUT parseFloat() -> 23.5717.258
```

parseInt()

- **parseInt()** will convert an integer number (whole number) into a number
 - If you pass in a floating point number into parseInt() then JavaScript will simply remove the decimals and return an integer
 - Use parseInt() if you are sure you will dealing with a whole number

```
const strNum1 = '23';  
const strNum2 = '7';  
const total = parseInt(strNum1) + parseInt(strNum2);  
// total -> 30
```


Type Conversion with Operators

- JS will convert your string numbers to numbers if you try to multiply them or do a few other mathematical operations on the string number

MULTIPLYING BY 1

```
const strNum1 = '23';  
const strNum2 = '7';  
const total = strNum1*1 + strNum2*1;  
// total -> 30
```

PREPENDING “+”

```
const strNum1 = '23';  
const strNum2 = '7';  
const total = +strNum1 + +strNum2;  
// total -> 30
```

Boolean Data Type

- Computers at their core are just a series of electronic switches that either let electricity flow (on or true) or prevent the flow of electricity (off or false)
- Most computer languages have a Boolean data type that can store one of two values
 - true (on)
 - false (off)
- Boolean values are **never written with quotes**
- We often use Booleans with conditional logic in our scripts

Boolean Data Type Example

- Booleans combined with a conditional statement provide an easy way to direct your script in different directions.

```
let userCanVote = true;

if(userCanVote == true){
    console.log('User can vote');
} else {
    console.log('User can\'t vote');
}
```

Template Literals

- Template literals or template strings allow you to insert variables directly inside a string using the "\${yourVariable}" syntax
- They are delimited by backticks ``.

```
const firstname = 'John';  
const lastname = 'Smith';  
const fullname = `The user's name is ${firstname} ${lastname}.`;   
// fullname -> ...is John Smith.
```

Template Literals

- They also allow you to run code directly inside the string

```
const subTotal = 23.57;  
const taxRate = 0.05;  
const output = `Your total including tax is  
${(subTotal + (subTotal * taxRate)).toFixed(2)}`;   
// output -> Your total including tax is 24.75
```

Template Literals

- White space is honored inside template literals
- This is handy if you want to insert dynamic HTML into the DOM and wish to maintain indentation

```
const dynamicHTML =  
`<ul>  
  <li>Foo</li>  
  <li>Bar</li>  
</ul>`;   
/* dynamicHTML ->  
<ul>  
  <li>Foo</li>  
  <li>Bar</li>  
</ul>  
*/
```

Window Methods

Built-In JS Window Methods

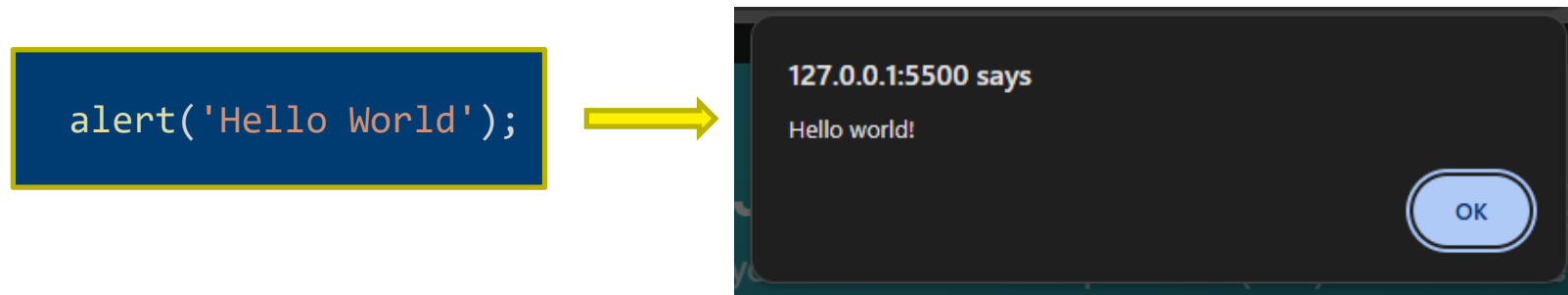
- The built-in window methods "alert", "prompt" and "confirm" are useful for quick prototyping and learning the JavaScript language.
- The pop-up windows produced by these methods can not be custom styled or have their functionality changed by the developer, so these are not used often in final production.
 - If you need a custom pop-up window with custom functionality and style (often called a "modal" window) you will have to create it yourself with HTML/CSS and JavaScript

Where is the Window Object

- The "alert()", "prompt()" and "confirm()" methods belong to the "window" object
 - The "window" object represents the browser window and is built-in to all versions of JavaScript that run in the browser
- The "window" object is assumed to be the default object, so writing "`window.alert()`" (which is still correct) can be shortened to just "`alert()`"

The alert() Method

- Creates a generic pop-up window that displays the text passed into the method (the text in quotes between the "()")



The prompt() Method

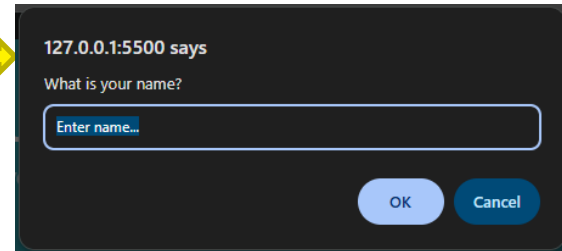
- Creates a generic pop-up window that allows a user to enter text into the pop-up window
- The prompt method "returns" the text that the user entered into the pop-up window

```
const user = prompt('What is your name?', 'Enter name...');
```

The variable we will use to store the text that the user enters into the prompt window

This required parameter represents the text you wish to have displayed in the prompt box

This is an optional parameter and is used to display helper text in the input field



What does "return" mean?

- In many computer languages, methods and functions often process data and then output that processed data
- When a function outputs data it is often done by a return statement
- This is true with built-in methods as well
- We often use a "variable" to capture and store the returned data from a function or method

Capture & Store Returned Data

- To capture and store returned data from a function or method we can use a variable which has its value set to the function or method call

```
const user = prompt('What is your name?', 'Enter name...');
```

The variable "user" is used to store the data "returned" by the prompt() method

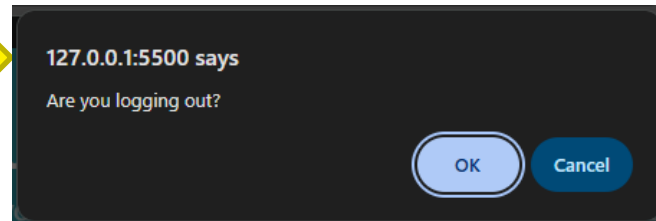
The prompt() method returns text that the user enters into the prompt window

The confirm() Method

- The confirm() method displays a pop-up box that asks the user to "confirm" an action
- The confirm() method returns a Boolean data type

```
const logout = confirm('Are logging out?');
```

The text you want to display within the confirm box



Modifying the DOM

- Think of the DOM as a representation of the HTML document
 - Each HTML element is a node
 - Each HTML element is an object with several built-in properties and methods

Using innerHTML

- This property allows you to set the HTML of an element or get the existing HTML of an element
- The "innerHTML" property is just one of several ways to set or get the text HTML of an element

Using innerHTML to Set the HTML

```
<p id="foo">Hello World!</p>
```



```
document.getElementById('foo').innerHTML = 'Hello Class!';
```



Hello Class!

What are Event Handlers?

- Event handlers are functions in JavaScript that run in response to events on web page elements.
- Events represent user interactions like clicks, key presses, mouse movements, and more.
- Event handlers make web pages interactive by reacting to these events automatically.

Click Event Handler

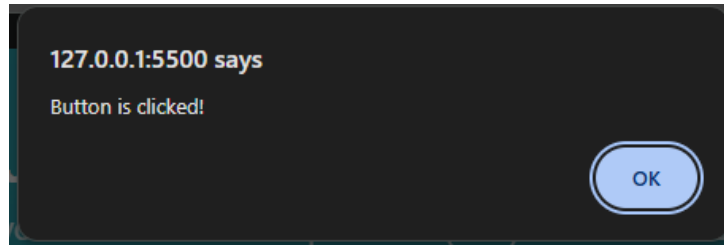
- This runs when a user clicks on an element like button or a hamburger menu icon.
- Click events typically help you trigger actions like displaying a message or changing content.

Click Event Handler Example

```
<button id="clickMe">Click me!</button>
```



```
document.getElementById('clickMe').addEventListener('click', () => {  
  alert('Button is clicked!');  
});
```



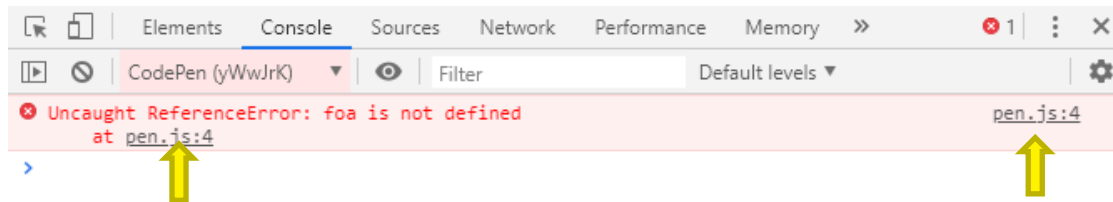
Moving Forward

- We will practice this more and explore more how we can interact with the DOM.

Basic Debugging in JS

The Browser JavaScript Console

- The console will report script errors that the browser found while trying to execute your JavaScript code
- The reported errors will report the error type, the error and the file and code line number where the error occurred



The error type and
description

The location of the
error

Running Code in the Console

- You can run a single line or multiple lines of JavaScript code from within the console by entering code directly into the console
 - This code is not saved and will be deleted once you close the developer tools session
 - To run a single line of code press the "Enter" key after you write your code
 - To run multiple lines of code (Chrome Browser) press "Shift+Enter" (Windows) or "Option-Enter" to add an additional line

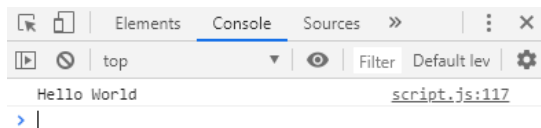
Logging Text or Data to the Console

- You write messages to yourself (or other developers) that display in the browser's console window
- Use the `console.log('Your message...')` to output messages to the console
- You can also output a variables value to the console by simply writing the variable's label as an argument to the `console.log()` method
- You can also combine a message with a variable value in a `console.log()` message

console.log() Examples

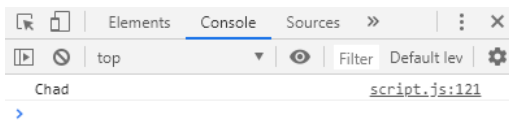
Output a message

```
// Output a text message to  
// the console  
console.log('Hello World');
```



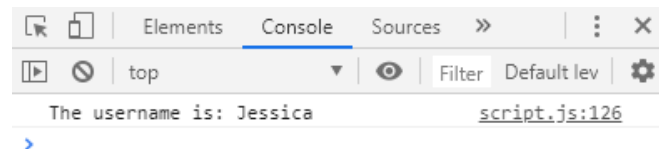
Output a variable

```
// Output a variable...  
const username = 'Chad';  
console.log(username);
```



Output a message with a variable (this is one way of doing this, there are other ways to do this as well)

```
// Output a message  
// with a variable  
const username = 'Jessica';  
console.log('The username is: ' + username);
```



Assignment #1

Assignment #1

- Please refer to Assignment #1 in the Learning Hub.
- To submit the assignment, you can do **one** of these:
 - **Have me check your assignment in class before 4pm.**
 - Zip today's **folder** and upload it to the Learning Hub before next class.
- If you have questions or need guidance, just ask!

JS Learning Resources

- [MDN \(Mozilla Developer Network\) - JavaScript](#)
- [W3Schools](#)
- [LinkedIn Learning](#)
 - Check with your city's library, Vancouver and Burnaby both have access to LinkedIn Learning for free!

JS Learning Resources - YouTube

- Lots of free "How to" videos on YouTube about JavaScript
- Quality can vary, here are some good ones:
 - [Traversy Media](#)
 - [The Net Ninja](#)
 - [freeCodeCamp.org](#)
 - [Programming with Mosh](#)

Resources

- [Ecma International](#)
- [W3 Schools](#)

QUESTIONS & ANSWERS