



KodeGo

Introduction to Web Development

<https://nodejs.org/en/download/>

1. Go to the link provided above.
2. Download installer suitable for your OS
3. Verify installation through command prompt or Powershell

Enter **node -v**

Enter **npm -v**

Node Package Manager

NPM is a main package manager for Node.js, which is a repository of open source software available for everyone to use.

Why NPM?

1. Free, without registrations or logon
2. Makes coding easier
3. Easy to install packages

Example: **npm install *package name***

npm i *package name*

How to install NPM?

Local

installed in the directory where you run **npm install <package-name>**, and they are put in the **node_modules folder** under this directory

Global

placed in a single folder in your system, regardless of where you run **npm install -g <package-name>**

Javascript

- JavaScript is a scripting or programming language that allows you to implement complex features on web pages

Internal

JavaScript code must be inserted between **<script>** and **</script>** tags.

External

Practical when used in multiple web pages, uses the file extension **.js**

alert()

Uses and alert box to display data

window.alert():

```
<script>
```

```
    window.alert(5 + 6);
```

```
</script>
```


document.write()

For testing purposes, it is convenient to use **document.write()**:

```
<script>
```

```
document.write(5 + 6);
```

```
</script>
```

innerHTML

The **id attribute** defines the HTML element and the **innerHTML property** defines the HTML content

```
<body>
  <p id="demo"></p>
  <script>
    document.getElementById("demo").innerHTML = 5 +
6;
  </script>
</body>
```

JavaScript Variables

A variable stores the data value that can be changed later on.

```
<script>
```

```
var variable1 = "hello";
```

```
</script>
```

Let variable

The let keyword signals that the variable can be **re-assigned** a different value.

```
<script>  
  let variable1 = "hello";  
  variable1 = "hi";  
  //will change the initial value and  
  will not have any error  
</script>
```

Constant variable

a const variable **cannot be reassigned** because it is constant.

```
<script>  
  const variable1 = "hello";  
  variable1 = "hi";  
  //will create an error  
</script>
```

String

String is a series of characters.
Strings are written with quotes.

```
<script>  
  let variable1 = "string";  
</script>
```

Numbers

Numbers can be written with,
or without decimals.

```
<script>  
  let variable1 = 12;  
</script>
```

Booleans

Booleans can only have two values: **true** and **false**.

```
<script>  
  let variable1 = true;  
</script>
```

Array

Arrays are written with square brackets. It is used to store **multiple values** inside a **single variable**.

```
<script>  
  let variable1 = ["one", "two", "t"];  
</script>
```

JavaScript Events

An event is an action that occurs as per the user's instruction as input and gives the output in response.

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

Click Me

```
</button>
```

Different Types of Javascript Events

Event	Description
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

JavaScript Functions

A JavaScript function is a **block of code** designed to perform a **particular task**.

Function Syntax

Functions are created using the **function** keyword, followed by the **function name**, followed by the **parentheses** which stores **parameters** separated by commas.

Function keyword \Rightarrow `<script>` `function` **name** `myFunction` **Parameters w/ parentheses** `(p1, p2)` `{`
 ..`this` is where you will put
 the set of commands
 `}`
`</script>`

Arrow Functions

Arrow functions allow a short syntax for writing function expressions. In using an arrow function, you do not need to use the “function” keyword any more. You can also remove the use of curly brackets if you are going to use a single expression.

Arrow functions do not have their own `this`. They are not well suited for defining object methods.

Difference between Arrow Functions and a Normal Function

NORMAL FUNCTION:

```
var x = function(x, y) {  
  return x * y;  
}
```

ARROW FUNCTION:

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

Selecting element.

The **document object** is the **root node** of the HTML document.

Getting Element using ID

```
document.getElementById("sample");
```

Getting Element using Class

```
document.getElementsByClassName("sample");
```

Getting Element using name

```
document.getElementsByName("sample");
```

Getting Element using Tag name

```
document.getElementsByTagName("h1");
```

Javascript Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus
++	Increment
--	Decrement

Exercise 16: Create a Simple Conversion Calculator

1. With the topic of your choice create a simple conversion calculator
2. The current unit must be the commonly used unit of measurement
3. Add at least three (3) unit options

Example:

Topic: Length

Commonly used unit: Inches

Unit options for conversion: Millimeter, Centimeter and Meter

Comparison Operators

Operator	Description	X = 10
==	Equal. Checks if the values of two operands are equal or not, if yes the condition is true.	(x == 10) true (x == 20) false
!=	Not Equal. Checks if the values of two operands are equal or not, if values are not equal then the condition is true	(x != 5) true
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true	(x > 9) true
<	Checks if the value of left operand is less than the value of right operand, if yes condition becomes true	(x < 9) false
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true	(x >= 10) true
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes condition becomes true	(x <= 7) false

Comparison Operators

Operator	Description	X = 10
===	Exactly equals, including type	(x === 10) true (x === "20") false
!=	Not Equal in either value or type	(x !== 10) false (x !== "20") true

Logical Operators

Operator	Description	A = True B = False
&&	AND operator. Returns true if both statements are true	A && B is false; A && A is true
	OR operator. Returns true if one or both statements are true	A B is true; A A is true
!	NOT operator. Returns false if the statement is true.	!A is false; !(A && B) is true

As the most common type of conditional, the if statement only runs if the condition enclosed in parentheses () is truthy

```
if (10 > 5) {  
    var outcome = "if block";  
}
```

You can extend an if statement with an else statement, which adds another block to run when the if conditional doesn't pass.

```
if ("cat" === "dog") {  
    var outcome = "if block";  
} else {  
    var outcome = "else block";  
}
```

You can also extend an if statement with an else if statement, which adds another conditional with its own block.

```
if (false) {  
    var outcome = "if block";  
} else if (true) {  
    var outcome = "else if block";  
} else {  
    var outcome = "else block";  
}
```

JavaScript Switch Statement

The **switch statement** is used to perform different actions based on different conditions.

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

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

expression

An **expression** whose result is matched against each case clause.

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

case

If the expression matches the specified valueN, the statements inside the **matching case clause are executed**, and then the statements inside all **case** clauses which **follow** the **matching case** clause are executed – until either the end of the switch statement or a break.


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

default

this clause is executed if the value of **expression doesn't match any of the case clauses.**

same block of code to run over and over again in a row

1. For
2. While
3. Do...while

init counter

- Initialize the loop counter value

test counter

- Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends

increment counter

- Increases the loop counter value

```
For (init counter; test counter; increment counter)
{
    code to be executed;
}
```

```
while (condition is true) {  
    code to be executed;  
}
```

Do - While loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

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

Selecting element.

The **document object** is the **root node** of the **HTML document**.

Using `querySelector`

Used when selecting a single element.

`querySelectorAll`

Used when selecting multiple elements.

JavaScript Object

is an unordered collection of key-value pairs or named values.
Each key-value pair is called a property.

```
let car = {  
  brand: 'toyota',  
  model: 'vios',  
  year: '2022'  
}
```

The car object has three properties,
brand, model and year with the corresponding
values **toyota, vios** and **2022**.

Objects are mutable

```
let car = {  
    brand: 'toyota',  
    model: 'vios',  
    year: '2022',  
    color: 'black'  
};  
let x = car;  
x.color = white;
```

car.color and x.color value will be changed to white.

Object Literal Syntax Extensions

Prior to ES6, you could use the square brackets (`[]`) to enable the computed property names for the properties on objects.

The square brackets allow you to use the string literals and variables as the property names.

```
let name = 'machine name';  
let machine = {  
  [name]: 'server',  
  'machine hours': 10000  
};
```

```
alert(machine[name]); // server  
alert(machine['machine hours']); // 10000
```

Functions stored as an Object Property

```
const car = {  
  brand: "Toyota",  
  model: "Altis",  
  year: "2022",  
  unit: function() {  
    return this.brand + " " + this.model + " " +  
this.year;  
  }  
};  
  
alert(car.unit());
```

JavaScript Objects (for/in loop)

```
let car = {  
    brand: 'toyota',  
    model: 'vios',  
    year: '2022',  
    color: 'black'  
};  
  
let text = "",  
for (let x in car) {  
    text += car[x] + ",;  
}
```

We can use **for/in loop** to execute each property one at a time.

```
document.getElementById("display").innerHTML = text;
```

Object Literal Syntax Extensions

Prior to ES6, you could use the square brackets ([]) to enable the computed property names for the properties on objects.

The square brackets allow you to use the string literals and variables as the property names.

```
let name = 'machine name';  
let machine = {  
    [name]: 'server',  
    'machine hours': 10000  
};
```

```
alert(machine[name]); // server  
alert(machine['machine hours']); // 10000
```

Javascript Math Obejct

Allows you to perform **mathematical tasks** on numbers.

Math.round(x)

Math.round(x) returns the nearest integer.

Math.ceil(x)

returns the value of x rounded up to its nearest integer

Math.floor(x)

returns the value of x rounded down to its nearest integer.

Math.pow()

`Math.pow(x, y)` returns the value of `x` to the power of `y`.

Math.sqrt()

`Math.sqrt(x)` returns the square root of `x`.

Math.abs()

`Math.abs(x)` returns the absolute (positive) value of `x`.

Math.min(...)

Used to get the lowest value in a list of arguments.

Math.max(...)

Used to get the highest value in a list of arguments.

Math.random()

returns a random number between 0 (inclusive), and 1 (exclusive)

Javascript Date Object

is a **built-in object** in JavaScript that stores the date and time.

getFullYear()

Get the year as a four digit number (yyyy)

getMonth()

Get the month as a number (0-11)

getDate()

Get the day as a number (1-31)

getHours()

Get the hour (0-23)

getMinutes()

Get the minute (0-59)

getSeconds()

Get the second (0-59)

getDay()

Get the weekday as a number (0-6)

Working with Strings

Method	Description
length	Returns the length of a string
slice	Extracts a part of a string; <i>parameters (start, end)</i>
substring	Extracts a part of a string; <i>parameters (start, end)</i>
replace	Replaces a part of a string
Upper and Lower Case	The browser has finished loading the page

Array Methods

Method	Description
<code>every()</code>	goes through each element of the array and check if true
<code>fill()</code>	fills specified elements in an array with a value
<code>filter()</code>	creates a new array filled with elements that pass a test provided by a function
<code>find()</code>	returns the value of the first element that passes a test/condition
<code>forEach()</code>	calls a function for each element in an array

Object Methods

Method	Description
<code>Object.create()</code>	create a new object and link it to the prototype of an existing object
<code>Object.keys()</code>	creates an array containing the keys of an object
<code>Object.values()</code>	creates an array containing the values of an object
<code>Object.entries()</code>	creates a nested array of the key/value pairs of an object

HTML DOM

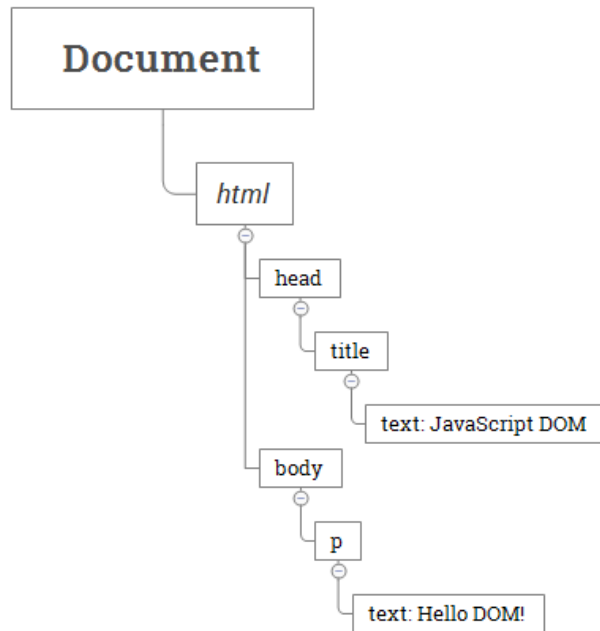
When an HTML document is loaded into a web browser, it becomes a document object.

DOM stands for **Document Object Model**.

The document object is the **root node of the HTML document**.

Hierarchy of Nodes

```
<html>  
  <head>  
    <title>JavaScript DOM</title>  
  </head>  
  <body>  
    <p>Hello DOM!</p>  
  </body>  
</html>
```



Setting Inline Styles

The **.style** manipulates the inline style of the HTML elements.

element.style

```
div.style.backgroundColor
```

```
div.style.color
```

```
div.style.margin
```



Traversing ELEMENTS

parentNode

The parentNode property returns the parent node of an element or node.

How to select the child of a parentNode?

firstChild

the **firstChild** property returns the first child node of a node.

lastChild

the **lastChild** property returns the last child node of a node.

firstElementChild.tagName

The **firstElementChild** property returns the first child element of the specified element.

childElementCount


The **childElementCount** property returns the number of child elements of an element.

nextElementSibling

The **nextElementSibling** property returns the next element in the same tree level.

previousElementSibling

The **previousElementSibling** property returns the previous element in the same tree level.



Manipulating HTML Elements

Create Element

The **document.createElement()** creates a new element.

```
<script>
    let div = document.createElement('div');
    div.id = 'content';
    div.innerHTML = '<p>createElement example</p>';
    document.body.appendChild(div);
</script>
```

Append Child

The **appendChild()** allows you to add a node to the end of the list of child nodes of a specified parent node.

parentNode.insertBefore(newNode, existingNode);

```
<script>  
  let div = document.createElement('div');  
  div.id = 'content';  
  div.innerHTML = '<p>createElement example</p>';  
  document.body.appendChild(div);  
</script>
```

Insert Before

The **insertBefore()** to insert a node before another node as a child node of a specified parent node.

```
parentNode.insertBefore(newNode, existingNode);
```

```
<script>  
    let menu = document.getElementById('menu');  
    let li = document.createElement('li');  
    li.textContent = 'Home';  
    menu.insertBefore(li, menu.firstChild);  
</script>
```

RemoveChild

The **removeChild()** method to remove a child node from a parent node.

```
<script>  
    let menu = document.getElementById('menu');  
    menu.removeChild(menu.lastElementChild);  
</script>
```

textContent

The **textContent()** property gets the text content of a node and its descendants.

```
<div id="note">
  <span style="display:none">Hidden Text!</span>
</div>
<script>
  let note = document.getElementById('note');
  alert(note.textContent);
</script>
```

Event Listener

The **addEventListener()** method attaches an event handler to a document sample.

```
document.addEventListener("click", myFunction);  
  
function myFunction() {  
    document.getElementById("demo").innerHTML = "Hello World";  
}
```

Replace Child

The **replaceChild()** method replaces an HTML element by a new one.

parentNode.replaceChild(newChild, oldChild);

```
<script>
  let menu = document.getElementById('menu');
  let li = document.createElement('li');
  li.textContent = 'Home';

  menu.replaceChild(li, menu.firstElementChild);
</script>
```

Clone Node

The **cloneNode()** method clones an element.

let clonedNode = originalNode.cloneNode(deep);

```
<script>  
  let menu = document.querySelector('#menu');  
  let clonedMenu = menu.cloneNode(true);  
  clonedMenu.id = 'menu-mobile';  
  document.body.appendChild(clonedMenu);  
</script>
```


setAttribute()

The `setAttribute()` method sets a new value to an attribute.

removeAttribute()

The `removeAttribute()` method removes the current attribute.

getAttribute()

The `getAttribute()` method returns the value of an element's attribute.

Setting Inline Styles

The **.style** manipulates the inline style of the HTML elements.

element.style

```
div.style.backgroundColor
```

```
div.style.color
```

```
div.style.margin
```

Add CSS classes to an element

The **add()** adds one or more CSS classes to the class list of an element

element.classList.add

```
<div id="content" class="main red">JavaScript classList</div>
```

```
<script>
```

```
let div = document.querySelector('#content');  
div.classList.add('info', 'visible', 'block');
```

```
</script>
```

Remove CSS classes from an element

The **remove()** removes a CSS class from the class list of an element

element.classList.remove

```
<div id="content" class="main red">JavaScript classList</div>
```

```
<script>
```

```
let div = document.querySelector('#content');
```

```
div.classList.remove('block', 'red');
```

```
</script>
```

Replace CSS classes from an element

The **replace()** replaces an existing CSS class with a new one

element.classList.replace

```
<div id="content" class="main red">JavaScript classList</div>
```

```
<script>  
let div = document.querySelector('#content');  
    div.classList.replace('info', 'warning');  
</script>
```

preventDefault()

The `preventDefault()` method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

Exercise 23: Create a Game using Javascript

1. Create a Rock, Paper, Scissors game using JavaScript.
2. Add three (3) buttons where the player can choose their pick.
3. A prompt will appear stating what the bot picked and if you win/lose.
4. **You may add images, styles and other features (scoreboard, match history etc.) to make the game more exciting.**

An abstract network diagram in the top right corner of the slide. It features three circular nodes of varying sizes connected by thin lines. The largest node is at the bottom right, with two smaller nodes above it to the left. Dotted lines extend from the top-left and bottom-right nodes, suggesting a larger network.

JavaScript ES6

let

when you declare a variable using the `var` keyword, the scope of the variable is either global or local. If you declare a variable outside of a function, the scope of the variable is global. When you declare a variable inside a function, the scope of the variable is local.

const

Like the `let` keyword, the `const` keyword declares block-scope variables. However, the block-scoped variables declared by the `const` keyword can't be reassigned.

rest parameters

allows you to represent an indefinite number of arguments as an array.

```
function sum(...args) {  
    let total = 0;  
    for (let a of args) {  
        total += a;  
    }  
    return total;  
}
```

```
alert(sum(1, 2, 3));
```

Spread parameters

spread operator allows you to spread out elements of an iterable objects.

```
const odd = [1,3,5];  
const combined = [2,4,6, ...odd];  
alert(combined);
```

The For/Of Loop

for/of lets you loop over data structures that are iterable such as **Arrays**, **Strings**, **Maps**, **NodeLists**, and **more**.

The For/Of Loop Parts

```
const persons = ["John", "Mike", "Sasha"];  
let text = "";
```

```
for (let x of persons) {  
    text += x + " ";  
}
```

variable - For every iteration the value of the next property is assigned to the variable. Variable can be declared with `const`, `let`, or `var`.

iterable - An object that has iterable properties.

Template Literals

Before ES6, you use **single quotes (')** or **double quotes (")** to wrap a string literal. And the strings have very limited functionality.

To enable you to solve more complex problems, ES6 template literals provide the syntax that allows you to work with strings more safely and cleanly.

In ES6, you create a template literal by wrapping your text in **backticks (`)** as follows:

```
let simple = `This is a template literal`;
```

Template Literals

- A multiline string: a string that can **span multiple lines**.
- String formatting: the ability to substitute part of the string for the values of variables or expressions. This feature is also called **string interpolation**.
- HTML escaping: the ability to transform a string so that it is safe to include in HTML.

```
let simple = `This is a template literal`;
```


Quotes Inside Strings

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

```
let title= `The book “Les Miserables” centers on the  
character named “Jean Valjean”`;
```

Multiline Strings

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

```
let title = `The book “Les Miserables”  
centers on the character  
named “Jean Valjean”`;
```

Variable and Expression Substitution

Allows you to embed variables and expressions in a string. The JavaScript engine will automatically replace these variables and expressions with their values. This feature is known as **string interpolation**.

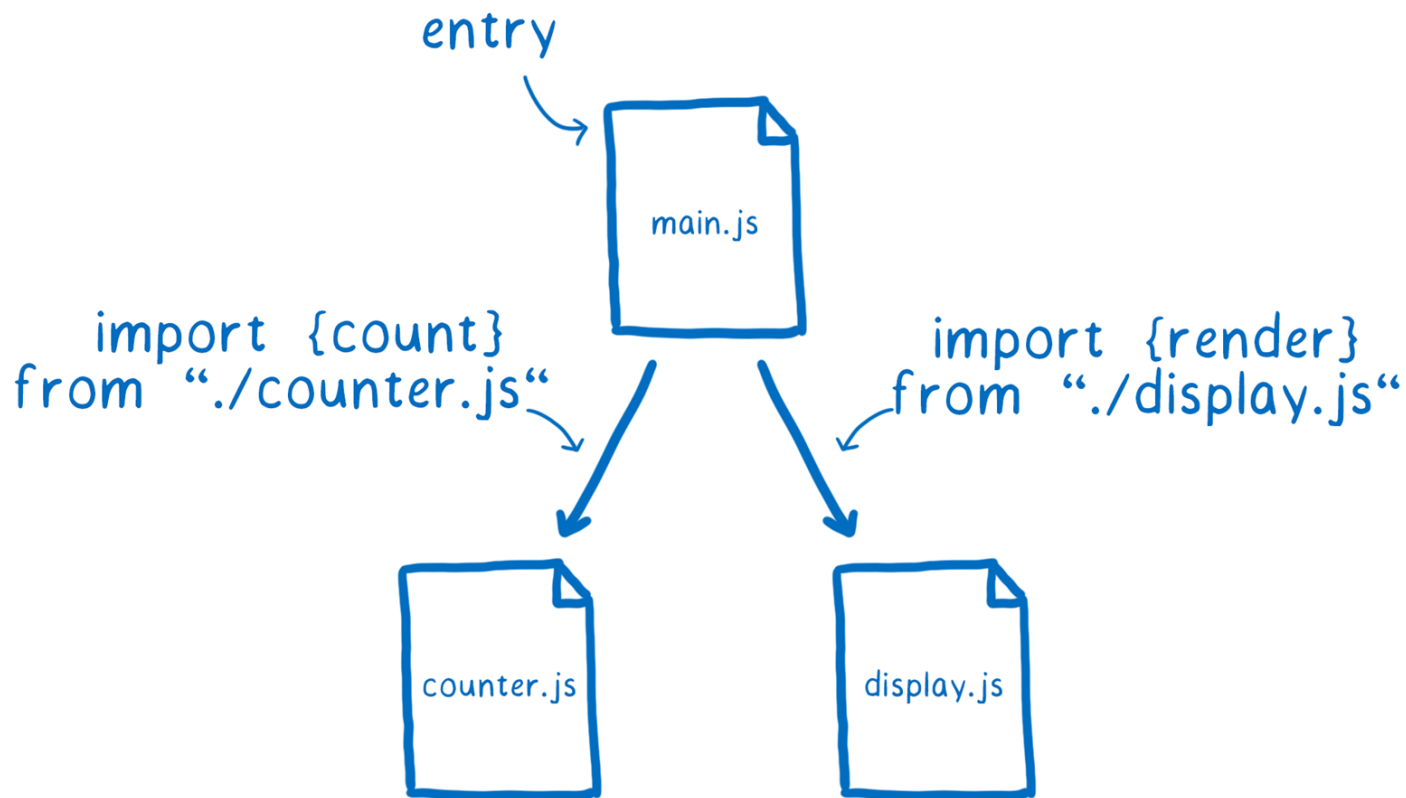
```
${variable_name}
```



Modules

A module is a function or group of similar functions. They are grouped together within a file and contain the code to execute a specific task when called into a larger application.

- **Independent/Self-contained**
- **Specific**
- **Reusable**



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>ES6 Modules</title>
</head>
<body>
<script type="module" src="./main.js"></script>
</body>
</html>
```

message.js

```
export let message = 'ES6 Modules';
```

main.js

```
import { message } from './message.js'

const h1 = document.createElement('h1');
h1.textContent = message

document.body.appendChild(h1)
```


Exporting

To export a variable, a function, or a class, you place the export keyword in front of it as follows:

```
export let message = 'Hi';
```

```
export function getMessage() {  
  return message;  
}
```

```
export function setMessage(msg) {  
  message = msg;  
}
```

```
export class Logger {  
}
```

Importing

Once you define a module with exports, you can access the exported variables, functions, and classes in another module by using the import keyword. The following illustrates the syntax:

```
import { message } from './other_module.js';
```

- Specify **what to import** inside the curly braces, which are called **bindings**.
- Specify the **module from which you import** the given bindings.

Importing Multiple Bindings //calculate.js

```
export let a = 10,  
          b = 20,  
          result = 0;
```

```
export function sum() {  
  result = a + b;  
  return result;  
}
```

```
export function multiply() {  
  result = a * b;  
  return result;  
}
```

Importing and Exporting Multiple Bindings //main.js

```
import {a, b, result, sum, multiply } from './cal.js';
```

```
sum();
```

```
alert(result); // 30
```

```
multiply();
```

```
alert(result); // 200
```

Aliasing //main.js

```
//math.js  
export function add( a, b ) {  
    return a + b;  
}
```

```
//main.js  
import {add as total} from './math.js';  
  
alert(total(8, 5));
```

Aliasing //main.js

```
//math.js  
function add( a, b ) {  
    return a + b;  
}
```

```
export { add as sum };
```

```
//main.js  
import { sum } from './math.js';
```

Default Exports

A module can have one and only one default export. The default export is easier to import. The default for a module can be a variable, a function, or a class.

```
// sort.js
export default function(arr) {
  // sorting here
}
```

```
//main.js
import sort from './sort.js';
```

Re-exporting a Binding

It's possible to export bindings that you have imported. This is called re-exporting. For example:

```
// sort.js  
export function math() {  
  // sorting here  
}
```

```
-----  
//random.js  
export { math } from './sort.js';  
-----
```

```
//main.js  
import { math } from './random.js';
```


An abstract network diagram in the top right corner of the slide. It features three circular nodes of varying sizes connected by thin lines. The largest node is at the bottom right, with two smaller nodes above it to the left. Dotted lines extend from the top and bottom of the diagram, suggesting a larger network.

JavaScript Asynchronous

Callbacks

a function that you pass into another function as an argument for executing later.

- **Synchronous callbacks** are executed during the execution of the high-order function that uses the callback.
- **Asynchronous callbacks** are executed after the execution of the high-order function that uses the callback.

Asynchronicity means that if JavaScript has to wait for an operation to complete, it will execute the rest of the code while waiting.

setTimeout()

Lets you specify a callback function to be executed **on time out**.

```
setTimeout(myFunction, 5000);
```

```
function myFunction() {  
    alert("Hello Batch 12!");  
}
```

setInterval()

Lets you specify a callback function to be executed **for each interval**.

```
function toggleColor() {  
    let e = document.getElementById('flashtext');  
    e.style.color = e.style.color == 'red' ? 'blue' : 'red';  
}
```

```
function stop() {  
    clearInterval(intervalID);  
}
```

```
function start() {  
    intervalID = setInterval(toggleColor, 100);  
}
```

Promises

A promise is an object that encapsulates the result of an asynchronous operation. A promise object has a state that can be one of the following:

- **Pending**
- **Fulfilled with a value**
- **Rejected for a reason**

Creating a Promise

To create a promise object, you use the **Promise()** constructor:

```
const promise = new Promise((resolve, reject) => {  
  // contain an operation  
  // ...  
  
  // return the state  
  if (success) {  
    resolve(value);  
  } else {  
    reject(error);  
  }  
});
```

then() Method

To get the value of a promise when it's fulfilled, you call the `then()` method of the promise object. The following shows the syntax of the `then()` method:

```
promise.then(onFulfilled, onRejected);
```

`then()` method **accepts two callback functions**: `onResolve` and `onRejected`

- calls the **onFulfilled()** with a value, if the promise is fulfilled
- Calls the **onRejected()** with an error if the promise is rejected

catch() Method

If you want to get the error only when the state of the promise is rejected, you can use the `catch()` method of the Promise object:

```
promise.catch(onRejected);
```

Internally, the `catch()` method **invokes the `then(onFulfilled, onRejected)` method**.

async

The keyword `async`, which is placed before a function makes the function return a promise.

```
async function myFunction() {  
    return "Hello";  
}
```

```
myFunction().then(  
    function(value) {myDisplayer(value);},  
    function(error) {myDisplayer(error);}
```

await

You use the await keyword to wait for a Promise to settle either in resolved or rejected state. And you can use the await keyword only inside an async function:

```
async function display() {  
    let result = await sayHi();  
    console.log(result);  
}
```

Introduction to Web API



What is Web API?

Application Programming Interface

API is some kind of interface which has a set of functions that allow programmers to access specific features or data of an application, operating system or other services

Web History API

Allows manipulation of the browser session history, that is the pages visited in the tab or frame that the current page is loaded in

History back() Method

Same as the back arrow in your browser, **back() method** loads the previous URL in the windows.history list.

```
<button onclick="myFunction()">Go Back</button>
```

```
<script>  
function myFunction() {  
    window.history.back();  
}  
</script>
```

History go() Method

go() method loads a specific URL from the history list.

```
<button onclick="myFunction()">Go Back 2  
Pages</button>
```

```
<script>  
function myFunction() {  
    window.history.go(-2);  
}  
</script>
```

History length property

Returns the **number of URLs** in the history list

```
let length = history.length;
```


Web Storage API

A simple syntax for storing and retrieving data in the browser

- **localStorage** - data is persisted until user clears the browser cache or web app clears the data
- **sessionStorage** - data is persisted only until the window or tab is closed

setItem() Method

stores a data item in a storage; two parameters: **name** and **value**

```
localStorage.setItem("name", "John Doe");
```

```
sessionStorage.setItem("name", "John Doe");
```

getItem() Method

retrieves a data item from the storage; only one parameters: **name**

```
localStorage.getItem("name");
```

```
sessionStorage.getItem("name");
```

Fetch API

Allows web browsers to make HTTP request to web servers.

fetch()

-allows you to fetch resources such txt,header, and etc. It will also provide you with “promise” result.

then()

-will be used to get the promise object.

Getting the value of a .txt file via fetch API

```
function getText(){  
    fetch('sample.txt')  
    .then(function(res){  
        return res.text();  
    })  
    .then(function(data){  
        alert(data);  
    });  
}
```

Will be used to get the resource from the .txt file. It will be returned as a promise.

Getting the value of a .txt file via fetch API

```
function getText(){  
    fetch('sample.txt')  
    .then(function(res){  
        return res.text();  
    })  
    .then(function(data){  
        alert(data);  
    });  
}
```

This then function will select the text value inside the promise.

Getting the value of a .txt file via fetch API

```
function getText(){  
    fetch('sample.txt')  
    .then(function(res){  
        return res.text();  
    })  
    .then(function(data){  
        alert(data);  
    });  
}
```

This then function will output
the value inside res.text()

Geolocation API

Geolocation API is used to locate a user's position.

getCurrentPosition()

-Is a method that returns a coordinates object to the function specified in the parameter

showPosition()

-Is a function that outputs the Latitude and Longitude