

Cascading Style Sheet

What is CSS?

- CSS stands for Cascading Style Sheets
- CSS is the language we use to style an HTML document.
- CSS describes how HTML elements should be displayed on screen, paper, or in other media.
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

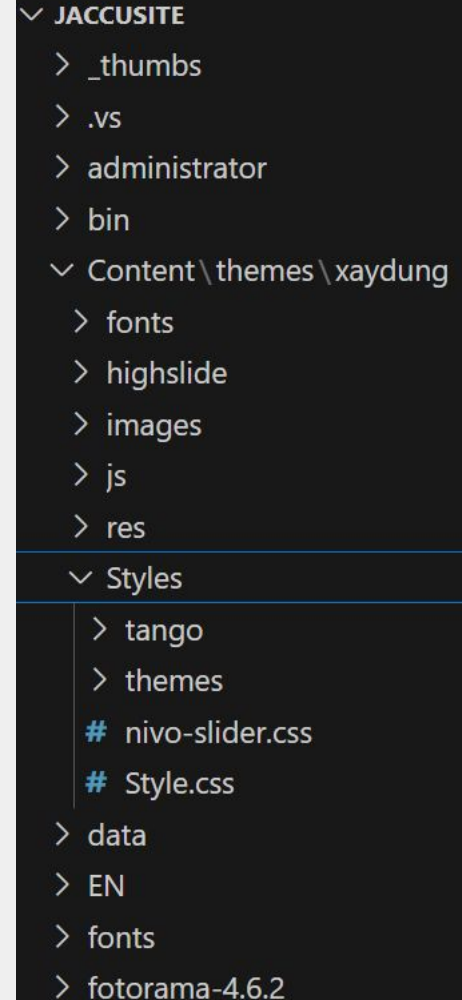
Where to place CSS file?

CSS file can put in any folder of project and the folder should contain only static files

1 E.g: /Content/css/style.css

2 E.g: /Themes/Content/css/style.css

3 E.g: /Resource/....



How to link a CSS file?

Question: Three Ways to Insert CSS?

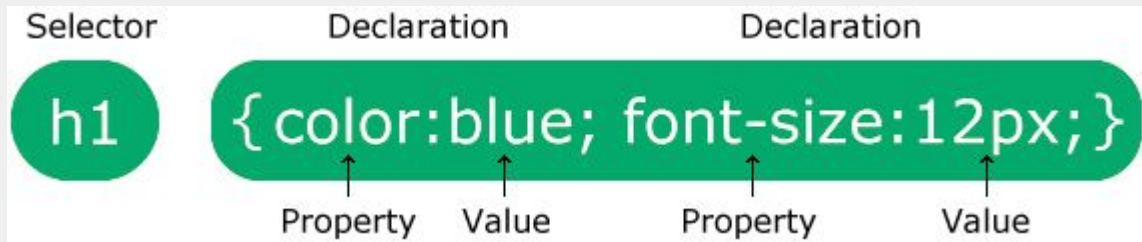
- 1 We can link multi CSS files on a html page
- 2 The priority of files are bottom to top.
(It mean the style in a file can overwrite the style in above files)

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

  <h1>This is a heading</h1>
  <p>This is a paragraph.</p>

</body>
</html>
```

CSS syntax



1 Selector

2 Declaration

- Property
- Value

Separated by a colon

3 Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

```
p {  
  color: red;  
  text-align: center;  
}
```

CSS selector

- 1 Simple selectors (select elements based on name, id, class)
- 2 Combinator selectors (select elements based on a specific relationship between them)
- 3 Pseudo-class selectors (select elements based on a certain state)
- 4 Pseudo-elements selectors (select and style a part of an element)
- 5 Attribute selectors (select elements based on an attribute or attribute value)

CSS selector

1 Simple selectors (select elements based on name, id, class)

- Element selector
- Id selector
- Class selector
- Universal selector
- Grouping selector

```
/* Element selector */
p {
  text-align: center;
  color: red;
}

/* id selector */
#para1 {
  text-align: center;
  color: red;
}

/* Class selector */
.center {
  text-align: center;
  color: red;
}

/* Combination element and class */
p.center {
  text-align: center;
  color: red;
}

/* Universal selector */
* {
  text-align: center;
  color: blue;
}

/* Group selector */
h1, h2, p {
  text-align: center;
  color: red;
}
```

CSS selector

2 Combinator selectors (select elements based on a specific relationship between them)

1. Descendant combinator (space)
2. Child combinator (>)
3. Next sibling combinator (+)
4. Subsequent-sibling combinator (~)

https://www.w3schools.com/css/css_combinators.asp

```
/*Descendant Combinator*/  
div p {  
    background-color: yellow;  
}
```

```
/*Child Combinator (>)*/  
div p {  
    background-color: yellow;  
}
```

```
/*Next Sibling Combinator (+)*/  
div + p {  
    background-color: yellow;  
}
```

```
/*Subsequent-sibling Combinator (~)*/  
div ~ p {  
    background-color: yellow;  
}
```

CSS selector

3 Pseudo-class selectors (select elements based on a certain state)

What are Pseudo-classes?

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

- Style an element when a user moves the mouse over it
- Style visited and unvisited links differently
- Style an element when it gets focus
- Style valid/invalid/required/optional form elements

```
/* unvisited link */  
a:link {  
    color: #FF0000;  
}
```

```
/* visited link */  
a:visited {  
    color: #00FF00;  
}
```

```
/* mouse over link */  
a:hover {  
    color: #FF00FF;  
}
```

```
/* selected link */  
a:active {  
    color: #0000FF;  
}
```


CSS selector

4 Pseudo-elements selectors (select and style a part of an element)

A CSS pseudo-element is used to style specific parts of an element.

- `::first-line`: used to add a special style to the first line of a text
- `::first-letter`: used to add a special style to the first letter of a text
- `::before`: insert some content before the content of an element
- `::after`: insert some content after the content of an element
- `::marker`: selects the markers of list items.
- `::selection`: matches the portion of an element that is selected by a user

```
p::  color: #ff0000;  
  font-variant: small-caps;  
}
```

YOU CAN USE THE `::FIRST-LINE` PSEUDO-ELEMENT TO ADD A SPECIAL EFFECT TO THE FIRST LINE OF more, and more, and more, and more, and more, and more, and more, and more, and more.

```
p::  color: #ff0000;  
  font-size: xx-large;  
}
```

Y ou can use the `::first-letter` ps

CSS selector

5

Attribute selectors (select elements based on an attribute or attribute value)

The [attribute] selector is used to select elements with a specified attribute.

```
a[target] {  
  background-color: yellow;  
}
```



CSS [attribute] Selector

The links with a target attribute gets a yellow background:

[w3schools.com](#) [disney.com](#) [wikipedia.org](#)

The [attribute="value"] selector is used to select elements with a specified attribute and value.

```
a[target="_blank"] {  
  background-color: yellow;  
}
```



CSS [attribute="value"] Selector

The link with target="_blank" gets a yellow background:

[w3schools.com](#) [disney.com](#) [wikipedia.org](#)

Comment

Comments are used to explain the code, and may help when you edit the source code at a later date.

Comments are ignored by browsers.

```
/* This is a single-line comment */  
p {  
  color: red;  
}
```

```
p {  
  color: red; /* Set text color to red */  
}
```

```
p {  
  color: /*red*/blue;  
}
```

```
/* This is  
a multi-line  
comment */  
  
p {  
  color: red;  
}
```

Color

Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

- Background Color
- Text Color
- Border Color

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
```

```
<h1 style="color:Tomato;">Hello World</h1>
```

```
<h1 style="border:2px solid Tomato;">Hello World</h1>
```

Color value

Same as color name "Tomato":

rgb(255, 99, 71)

#ff6347

hsl(9, 100%, 64%)

Same as color name "Tomato", but 50% transparent:

rgba(255, 99, 71, 0.5)

hsla(9, 100%, 64%, 0.5)

Background

The CSS background properties are used to add background effects for elements.

- background-color
- background-image
- background-repeat
- background-attachment
- background-position
- background (shorthand property)

Border

The CSS border properties allow you to specify the style, width, and color of an element's border.

- `dotted` - Defines a dotted border
- `dashed` - Defines a dashed border
- `solid` - Defines a solid border
- `double` - Defines a double border
- `groove` - Defines a 3D grooved border. The effect depends on the border-color value
- `ridge` - Defines a 3D ridged border. The effect depends on the border-color value
- `inset` - Defines a 3D inset border. The effect depends on the border-color value
- `outset` - Defines a 3D outset border. The effect depends on the border-color value
- `none` - Defines no border
- `hidden` - Defines a hidden border

Border

```
p.dotted {border-style: dotted;}  
p.dashed {border-style: dashed;}  
p.solid {border-style: solid;}  
p.double {border-style: double;}  
p.groove {border-style: groove;}  
p.ridge {border-style: ridge;}  
p.inset {border-style: inset;}  
p.outset {border-style: outset;}  
p.none {border-style: none;}  
p.hidden {border-style: hidden;}  
p.mix {border-style: dotted dashed solid double;}
```

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border. The effect depends on the border-color value.

A ridge border. The effect depends on the border-color value.

An inset border. The effect depends on the border-color value.

An outset border. The effect depends on the border-color value.

No border.

A hidden border.

A mixed border.

Margin

The CSS margin properties are used to create space around elements, outside of any defined borders.

```
p {  
  margin-top: 100px;  
  margin-bottom: 100px;  
  margin-right: 150px;  
  margin-left: 80px;  
}
```

```
p {  
  margin: 25px 50px 75px 100px;  
}
```

- top margin is 25px
- right margin is 50px
- bottom margin is 75px
- left margin is 100px

Padding

Padding is used to create space around an element's content, inside of any defined borders.

```
div {  
  padding-top: 50px;  
  padding-right: 30px;  
  padding-bottom: 50px;  
  padding-left: 80px;  
}
```

```
div {  
  padding: 25px 50px 75px 100px;  
}
```

Height, Width, Max-width

The CSS height and width properties are used to set the height and width of an element.

The height and width properties may have the following values:

- **auto** - This is default. The browser calculates the height and width
- **length** - Defines the height/width in px, cm, etc.
- **%** - Defines the height/width in percent of the containing block
- **initial** - Sets the height/width to its default value
- **inherit** - The height/width will be inherited from its parent value

The CSS max-width property is used to set the maximum width of an element.

```
div {  
  max-width: 500px;  
  height: 100px;  
  background-color: powderblue;  
}
```

Position

The position property specifies the type of positioning method used for an element.

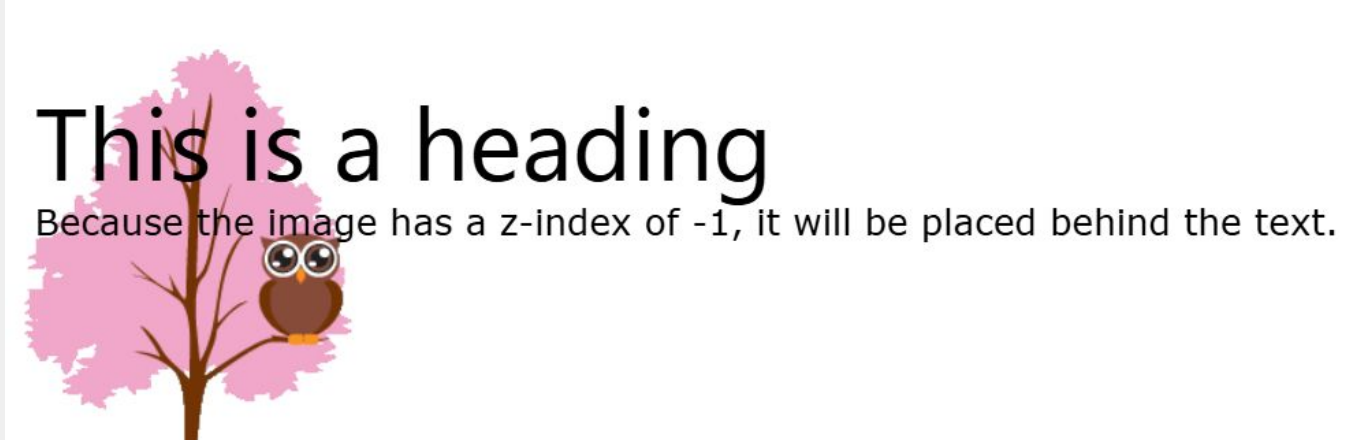
- **Static:** Static positioned elements are not affected by the top, bottom, left, and right properties.
- **Relative:** Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position.
- **Fixed:** An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled
- **Absolute:** is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed). Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.
- **Sticky:** An element with position: sticky; is positioned based on the user's scroll position

Eg: https://www.w3schools.com/css/css_positioning.asp

Z-index

When elements are positioned, they can overlap other elements.

The z-index property specifies the stack order of an element (which element should be placed in front of, or behind, the others).



Overflow

The overflow property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.

The overflow property has the following values:

- **visible** - Default. The overflow is not clipped. The content renders outside the element's box
- **hidden** - The overflow is clipped, and the rest of the content will be invisible
- **scroll** - The overflow is clipped, and a scrollbar is added to see the rest of the content
- **auto** - Similar to scroll, but it adds scrollbars only when necessary

Display

The display property is used to specify how an element is shown on a web page.

Every HTML element has a default display value, depending on what type of element it is. The default display value for most elements is block or inline.

The display property is used to change the default display behavior of HTML elements.

Refer: https://www.w3schools.com/css/css_display_visibility.asp

Javascript

JavaScript is the world's most popular programming language.

JavaScript is the programming language of the Web.

JavaScript is easy to learn.

Commonly Asked Questions

1. How do I get JavaScript?
2. Where can I download JavaScript?
3. Is JavaScript Free?

JavaScript Can Change HTML Content

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hello JavaScript!";'>Click Me!</button>

</body>
</html>
```

JavaScript accepts both double and single quotes:

JavaScript Can Change HTML Attribute Values (e.g)

JavaScript Can Change HTML Styles (CSS)

```
document.getElementById("demo").style.fontSize = "35px";
```

JavaScript Can Hide HTML Elements

```
document.getElementById("demo").style.display = "none";
```

JavaScript Can Show HTML Elements

```
document.getElementById("demo").style.display = "block";
```

Where

JavaScript code is inserted between `<script>` and `</script>` tags

Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.

```
<script>  
document.getElementById("demo").innerHTML = "My First JavaScript";  
</script>
```

External JavaScript

Scripts can also be placed in external files

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

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

External JavaScript

```
<script src="https://www.w3schools.com/js/myScript.js"></script>
```

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

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

Variables

Variables are Containers for Storing Data

JavaScript Variables can be declared in 4 ways:

- Automatically : `x = 5;`
- Using var : `var x = 5;`
- Using let : `let x = 5;`
- Using const: `const x= 5;`

Variables

When to Use var, let, or const?

1. Always declare variables
2. Always use const if the value should not be changed
3. Always use const if the type should not be changed (Arrays and Objects)
4. Only use let if you can't use const
5. Only use var if you MUST support old browsers.

Variables

JavaScript variables can hold numbers like 100 and text values like "John Doe".

```
const pi = 3.14;  
let person = "John Doe";  
let answer = 'Yes I am!';
```

One Statement, Many Variables

```
let person = "John Doe", carName = "Volvo", price = 200;
```

Operators

The **Assignment Operator** = assigns values

The **Addition Operator** + adds values

The **Multiplication Operator** * multiplies values

The **Comparison Operator** > compares values

The **Division Operator** / compares values

The **Modulus Operator** % compares values

Increment ++, Decrement --

String comparison

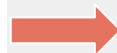
In javascript, strings are compared alphabetically

```
let text1 = "A";  
let text2 = "B";  
let result = text1 < text2;
```

```
let text1 = "20";  
let text2 = "5";  
let result = text1 < text2;
```

String Addition

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



John Doe

```
let text1 = "What a very ";  
text1 += "nice day";
```



What a very nice day

Adding Strings and Numbers

```
let x = 5 + 5;  
let y = "5" + 5;  
let z = "Hello" + 5;
```

The result of x, y, and z will be:

10

55

Hello5

Logical Operators

Operator	Description	
&&	logical and	<code>if(A && B){ console.log("result");}</code>
	logical or	<code>if(A B){ console.log("result");}</code>
!	logical not	<code>if(!A){ console.log("result");}</code>

Data Types

JavaScript has 8 Datatypes

String
Number
Bigint
Boolean
Undefined
Null
Symbol
Object

```
// Numbers:
```

```
let length = 16;
```

```
let weight = 7.5;
```

```
// Strings:
```

```
let color = "Yellow";
```

```
let lastName = "Johnson";
```

```
// Booleans
```

```
let x = true;
```

```
let y = false;
```

Object Datatype

The object data type can contain both built-in objects, and user defined objects:

Built-in object types can be:

objects, arrays, dates, maps, sets, intarrays, floatarrays, promises, and more.

```
// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```


Functions

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

A JavaScript function is executed when "something" invokes it (calls it).

```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

```
function myFunction(p1, p2) {  
    return p1 * p2;  
}
```

Function return

Why Functions?

With functions you can reuse code

You can write code that can be used many times.

You can use the same code with different arguments, to produce different results.

```
// Function is called, the return value will end up in x
let x = myFunction(4, 3);

function myFunction(a, b) {
  // Function returns the product of a and b
  return a * b;
}
```

Local variables

```
// code here can NOT use carName
```

```
function myFunction() {  
  let carName = "Volvo";  
  // code here CAN use carName  
}
```

```
// code here can NOT use carName
```

Objects

Objects are variables too. But objects can contain many values.

```
const car = {type:"Fiat", model:"500", color:"white"};
```

```
// Create an Object
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

```
// Create an Object
const person = new Object();

// Add Properties
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Object

You can access object properties in two ways:

```
person.lastName;
```

```
person["lastName"];
```

Object Methods

Methods are actions that can be performed on objects.

```
const person = {  
  firstName: "John",  
  lastName : "Doe",  
  id       : 5566,  
  fullName : function() {  
    return this.firstName + " " + this.lastName;  
  }  
};
```

```
person.fullName();
```

Object Methods

Adding method to object.

```
person.name = function () {  
    return this.firstName + " " + this.lastName;  
};
```

You can use javascript methods.

```
person.name = function () {  
    return (this.firstName + " " + this.lastName).toUpperCase();  
};
```

Display Object

Some solutions to display JavaScript objects are:

- Displaying the Object Properties by name
- Displaying the Object Properties in a Loop
- Displaying the Object using `Object.values()`
- Displaying the Object using `JSON.stringify()`

Display Object

Displaying the Object Properties by name.

```
// Create an Object
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

// Display Properties
document.getElementById("demo").innerHTML =
person.name + "," + person.age + "," + person.city;
```

Display Object

Displaying the Object Properties in a Loop

```
// Create an Object
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

// Build a Text
let text = "";
for (let x in person) {
  text += person[x] + " ";
};

// Display the Text
document.getElementById("demo").innerHTML = text;
```

Note:
You must use `person[x]` in the loop.
`person.x` will not work (Because `x` is the loop variable).



John 30 New York

```
const fruits = {Bananas:300, Oranges:200, Apples:500};

let text = "";
for (let [fruit, value] of Object.entries(fruits)) {
  text += fruit + ": " + value + "<br>";
}
```



Bananas: 300
Oranges: 200
Apples: 500

Display Object

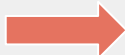
Displaying the Object using Object.values()

`Object.values()` creates an array from the property values:

```
// Create an Object
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

// Create an Array
const myArray = Object.values(person);

// Display the Array
document.getElementById("demo").innerHTML = myArray;
```



John,30,New York

Display Object

Displaying the Object using JSON.stringify()

```
// Create an Object
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

// Stringify Object
let myString = JSON.stringify(person);

// Display String
document.getElementById("demo").innerHTML = myString;
```

JSON.stringify() method can convert an object to string

Included in JavaScript and supported in all major browsers.



```
{"name":"John","age":50,"city":"New York"}
```

Object constructor

Sometimes we need to create many objects of the same type.

Object Type Person

```
function Person(first, last, age, eye) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eye;  
}
```



```
const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Sally", "Rally", 48, "green");  
const mySister = new Person("Anna", "Rally", 18, "green");  
  
const myself = new Person("Johnny", "Rally", 22, "green");
```

Note:

In the constructor function, `this` has no value.

The value of `this` will become the new object when a new object is created.

Object constructor

Create objects with default values.

```
function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
  this.nationality = "English";  
}
```



```
const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Sally", "Rally", 48, "green");  
const mySister = new Person("Anna", "Rally", 18, "green");  
  
const mySelf = new Person("Johnny", "Rally", 22, "green");
```



`myMother.nationality = ?`

Event

HTML events are "things" that happen to HTML elements.

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

```
<element event='some JavaScript'>
```

```
<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>
```

```
<button onclick="displayDate()">The time is?</button>
```

Event

Common events:

Event	Description
onchange	An HTML element has been changed
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

Event

Event handlers can be used to handle and verify user input, user actions, and browser actions:

- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Many different methods can be used to let JavaScript work with events:

- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

Array

An array is a special variable, which can hold more than one value:

```
const cars = ["Saab", "Volvo", "BMW"];
```

Why use Arrays?

Creating Arrays?

```
const array_name = [item1, item2, ...];
```



```
const cars = ["Saab", "Volvo", "BMW"];
```

Another ways:

```
const cars = [  
  "Saab",  
  "Volvo",  
  "BMW"  
];
```

```
const cars = [];  
cars[0] = "Saab";  
cars[1] = "Volvo";  
cars[2] = "BMW";
```

```
const cars = new Array("Saab", "Volvo", "BMW");
```

Better for performance

Access Array elements

Array indexes start with 0

```
const cars = ["Saab", "Volvo", "BMW"];  
let car = cars[0];
```



?

Changing an Array Element

```
cars[0] = "Opel";
```

Converting an Array to a String

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML = fruits.toString();
```



Banana,Orange,Apple,Mango

```
const cars = ["Saab", "Volvo", "BMW"];  
document.getElementById("demo").innerHTML = cars;
```



Saab,Volvo,BMW

Array are Objects

What value of person[2]?

```
const person = ["John", "Doe", 46];
```



Array are Objects

```
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
const mySister = new Person("Anna", "Rally", 18, "green");

const arr = [myFather, myMother, mySister];

// Display age
document.getElementById("demo").innerHTML =
  "My father is " + arr[0].age + ". My mother is " + arr[1].age + ".";
```



My father is 50. My mother is 48.

Array Properties, Methods

```
cars.length // Returns the number of elements  
cars.sort() // Sorts the array
```

Adding array element


```
const fruits = ["Banana", "Orange", "Apple"];  
fruits.push("Lemon"); // Adds a new element (Lemon) to fruits
```

WARNING !

Adding elements with high indexes can create undefined "holes" in an array:

```
const fruits = ["Banana", "Orange", "Apple"];  
fruits[6] = "Lemon"; // Creates undefined "holes" in fruits
```

```
const fruits = ["Banana", "Orange", "Apple"];  
fruits[fruits.length] = "Lemon"; // Adds "Lemon" to fruits
```



Banana
Orange
Apple
undefined
undefined
undefined
Lemon

Basic array methods

Array_length

Array_toString()

Array_at()

Array_join()

Array_pop()

Array_push()

Array_shift()

Array_unshift()

Array_delete()

Array_concat()

Array_copyWithin()

Array_flat()

Array_splice()

Array_toSpliced()

Array_slice()

Array search

indexOf()

```
array.indexOf(item, start)
```

<i>item</i>	Required. The item to search for.
<i>start</i>	Optional. Where to start the search.

- `Array.indexOf()` returns -1 if the item is not found.

If the item is present more than once, it returns the position of the first occurrence.

- `Array.lastIndexOf()` is the same as `Array.indexOf()`, but returns the position of the last occurrence of the specified element.

- `Array.includes()` to arrays. This allows us to check if an element is present in an array.

- `Array.find()` method returns the value of the first array element that passes a test function

Array Sort methods

Alphabetic Sort

Array.sort()

Array.reverse()

Array.toSorted()

Array.toReversed()

Sorting Objects

Numeric Sort

Numeric Sort

Random Sort

Math.min()

Math.max()

Home made Min()

Home made Max()

Sorting an Array

The **sort()** method sorts an array alphabetically:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();
```

The **reverse()** method reverses the elements in an array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.reverse();
```

Array Iteration

Array forEach

Array map().

Array flatMap().

Array filter().

Array reduce().

Array reduceRight().

Array every().

Array some().

Array from().

Array keys().

Array entries().

Array with().

Array Spread (...).

Array Iteration

```
const numbers = [45, 4, 9, 16, 25];  
let txt = "";  
numbers.forEach(myFunction);  
  
function myFunction(value, index, array) {  
  txt += value + "<br>";  
}
```



45
4
9
16
25

```
const numbers1 = [45, 4, 9, 16, 25];  
const numbers2 = numbers1.map(myFunction);  
  
function myFunction(value, index, array) {  
  return value * 2;  
}
```



90,8,18,32,50

Dates

```
const d = new Date();
```

Sat Jan 18 2025 06:24:15 GMT+0700 (Indochina Time)

```
const d = new Date("2022-03-25");
```

Fri Mar 25 2022 07:00:00 GMT+0700 (Indochina Time)

```
new Date()  
new Date(date string)  
  
new Date(year, month)  
new Date(year, month, day)  
new Date(year, month, day, hours)  
new Date(year, month, day, hours, minutes)  
new Date(year, month, day, hours, minutes, seconds)  
new Date(year, month, day, hours, minutes, seconds, ms)  
  
new Date(milliseconds)
```

Date Get methods

Method	Description
getFullYear()	Get year as a four digit number (yyyy)
getMonth()	Get month as a number (0-11)
getDate()	Get day as a number (1-31)
getDay()	Get weekday as a number (0-6)
getHours()	Get hour (0-23)
getMinutes()	Get minute (0-59)
getSeconds()	Get second (0-59)
getMilliseconds()	Get millisecond (0-999)
getTime()	Get time (milliseconds since January 1, 1970)

Date Set methods

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

If else

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

Switch

Use the switch statement to select one of many code blocks to be executed

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


For Loop

```
for (let i = 0; i < cars.length; i++) {  
  text += cars[i] + "<br>";  
}
```

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **for/of** - loops through the values of an iterable object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true

Maps

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

You can create a JavaScript Map by:

Passing an Array to **new Map()**

Create a Map and use **Map.set()**

Map.get()

Map.size

Map.delete()/ Map.clear()

Map.has()

Map.forEach()/ Map.entries()/ Map.key()/ Map.values()

```
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
```

```
// Create a Map
const fruits = new Map();

// Set Map Values
fruits.set("apples", 500);
fruits.set("bananas", 300);
fruits.set("oranges", 200);
```

```
fruits.get("apples");    // Returns 500
```

Class

Use the keyword class to create a class. Always add a method named constructor()

Class Methods

```
class Car {  
  constructor(name, year) {  
    this.name = name;  
    this.year = year;  
  }  
  age() {  
    const date = new Date();  
    return date.getFullYear() - this.year;  
  }  
}
```

```
const myCar1 = new Car("Ford", 2014);  
const myCar2 = new Car("Audi", 2019);
```

```
const myCar = new Car("Ford", 2014);  
document.getElementById("demo").innerHTML =  
"My car is " + myCar.age() + " years old.";
```

Class Inheritance

To create a class inheritance, use the extends keyword.

A class created with a class inheritance inherits all the methods from another class:

The `super()` method refers to the parent class.

By calling the **`super()`** method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
```

Getters and Setters

Classes also allows you to use getters and setters.

To add getters and setters in the class, use the get and set keywords.

The name of the getter/setter method cannot be the same as the name of the property, in this case **carname**.

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  get cnam() {
    return this.carname;
  }
  set cnam(x) {
    this.carname = x;
  }
}

const myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.cnam;
```

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

const myCar = new Car("Ford");
myCar.carname = "Volvo";
document.getElementById("demo").innerHTML = myCar.carname;
```

Class static

Static class methods are defined on the class itself.

You cannot call a static method on an object, only on an object class.

```
class Car {  
  constructor(name) {  
    this.name = name;  
  }  
  static hello() {  
    return "Hello!!";  
  }  
}  
  
const myCar = new Car("Ford");  
  
// You can call 'hello()' on the Car Class:  
document.getElementById("demo").innerHTML = Car.hello();  
  
// But NOT on a Car Object:  
// document.getElementById("demo").innerHTML = myCar.hello();  
// this will raise an error.
```