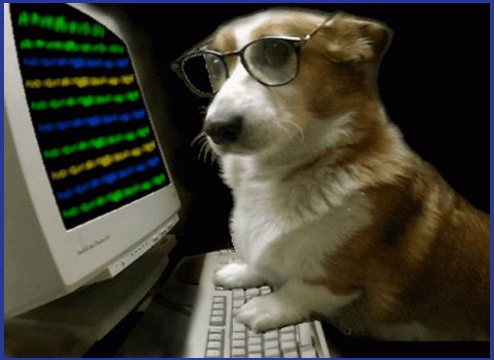
JS & HTML



What is the DOM?

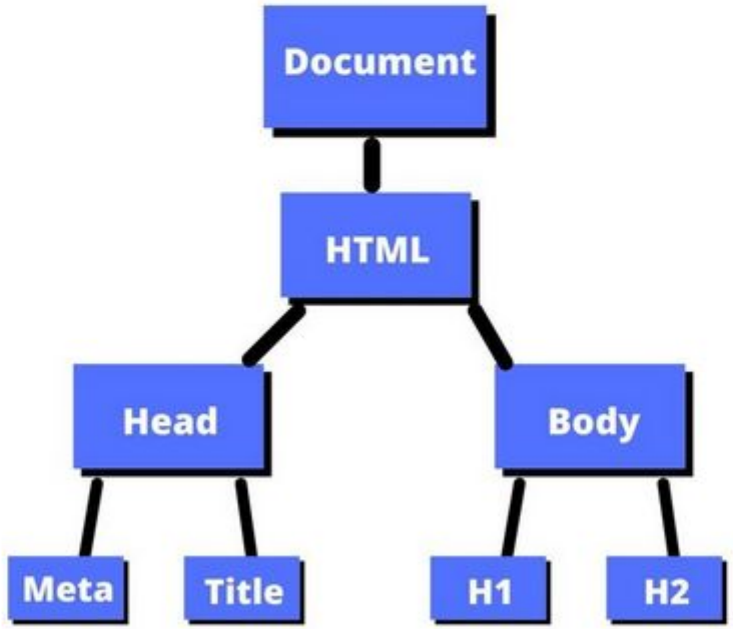
DOM (Document Object Model)

Programming interface that helps us to create, change or remove elements from a document. It allows to add events.

The DOM views an HTML document as a tree of nodes. A node represents an HTML element.

DOM tree structure

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>DOM tree structure</title>
  </head>
  <body>
    <h1>DOM tree structure</h1>
    <h2>Learn about the DOM</h2>
  </body>
</html>
```



Select Elements in the Document - getElementById()

getElementById(): returns an Element object representing the element whose id property matches the specified string.

```
<p id="p1">First paragraph.</p>

<p id="p2">Second paragraph.</p>
```

In JavaScript, we can grab an HTML tag by referencing the id name.

document.getElementById("id_here")

```
const paragraph1 = document.getElementById("p1");

console.log(paragraph1); => <p id="p1">First paragraph.</p>
```

Read the content of the paragraph:

```
const paragraph1 = document.getElementById("p1");

console.log(paragraph1.textContent); => "First paragraph."
```

Select Elements in the Document - querySelector()

querySelector(): returns the first Element within the document that matches the specified selector. If no matches are found, null is returned.

Find and print to the console the h1 element:

```
const h1Element = document.querySelector("h1");
```

With a class, use .list inside the querySelector() to target a class="list":

```
const list = document.querySelector(".list");
```

Select Elements in the Document - querySelectorAll()

querySelectorAll(): Returns a NodeList representing a list of the document's elements that match the specified group of selectors.

To obtain a NodeList of all of the <p> elements in the document:

```
const matches = document.querySelectorAll("p");
```

DOM AND JS

The browser implements all the properties and methods that the DOM will need, and with your JS code you will be able to use them and modify the web page.

Let's add some JS code to our movies webpage

In your movies.html:

```
<body>
   <h1>Movie Theater</h1>
   <input type="text" class="seat_selector" maxlength="3" size="3" />
  …..
  …….
</body>
```

In the console

```
// Save seat input in a variable
seat_input = document.querySelector('.seat_selector')
// Apply this function
seat_input.getBoundingClientRect()   => A DOMRect describes the size and position of a rectangle.
Seat_input.value => get the value of the element
seat_input.value = "345" => Modify the element
```

Events

With javascript we can create events. With an event, javascript knows it has to monitor an action, and when this action occurs, we can tell javascript to run a function.

Let's add a button in our html page.

```
<input
     type="text"
     class="seat_selector"
     maxlength="3"
     size="3"
  /><br /><br />
   <button type="button" class="button_check">Button</button>
```

In the JS file (movies.js):

```
"use strict"
document.querySelector(".button_check").addEventListener("click", function () {
 console.log("The button has been click");
});
```

If you click multiple times, the browser will display the message once, but there will be a number displayed on the right or left side (depending on the browser) of your consone. The number tells how many consecutive times this message has been displayed.

Let's create another button.

```
<button type="button" class="button_check">Button</button><br /><br />
<button type="button" class="button_css">My Second Button</button><br /><br />
```

In the javascript file, we are going to change the background color of the input element into red

```
document.querySelector(".button_css").addEventListener("click", function () {
 document.querySelector(".seat_selector").style.backgroundColor = "red";
});
```

So when we click on the button, the background color is going to be red.

Exercise

Let's do an exercise. Let's alternate the background color, one time red, one time clear.

Solution:

```
document.querySelector(".button_css").addEventListener("click", function () {
 let input_element_style = document.querySelector(".seat_selector").style;
 if (input_element_style.backgroundColor === "red") {
   input_element_style.backgroundColor = "";
 } else if (input_element_style.backgroundColor === "") {
   input_element_style.backgroundColor = "red";
 }
});
```

We can also create a variable to store how many times we change the background.

```
<button type="button" class="button_css">
    Change input background color</button><br /><br />
 <span class="bg_info">How many times have we changed the background? 0</span>
```

Then let's add some lines to the event:

```
let bg_change_times = 0; // Initialize a variable
document.querySelector(".button_css").addEventListener("click", function () {
 let input_element_style = document.querySelector(".seat_selector").style;
 if (input_element_style.backgroundColor === "red") {
   input_element_style.backgroundColor = "";
 } else if (input_element_style.backgroundColor === "") {
   input_element_style.backgroundColor = "red";
 }
 bg_change_times++;
 document.querySelector(
   ".bg_info"
 ).textContent = `How many times have we changed the background? ${bg_change_times}`;
});
```

Events using a class

Let's create a css:

```
<link rel="stylesheet" href="main.css" />
```

And inside the css:

```
.hidden {
 display: none;
}
```

In the html:

```
<button type="button" class="button_class">Change element class</button><br /><br />
<span class="hidden boo">I was hidden. Boooo!!!! Now you see me</span>
```

And in the javascript:

```
document.querySelector(".button_class").addEventListener("click", function () {
 document.querySelector(".boo").classList.remove("hidden");
});
```

We have removed the class "hidden" from the list of classes of the element.

Events using a class - Exercise

Use the previous code and modify it to alternate show and hide.

Hint: Your solution will probably use a list and you will need to check if a certain element is in the list, use **your_list.contains(item)**

Events using a class - Exercise - Solution

Use the previous code and modify it to alternate show and hide.

Hint: Your solution will probably use a list and you will need to check if a certain element is in the list, use **your_list.contains(item)**

Solution:

```
document.querySelector(".button_class").addEventListener("click", function () {
 let boo_class_list = document.querySelector(".boo").classList;
 if (boo_class_list.contains("hidden")) {
   boo_class_list.remove("hidden");
 } else {
   boo_class_list.add("hidden");
 }
});
```

Key Event

Keyboard event is a global event, because it cannot be assigned to an element.
There is an event that listens to any keyboard key that has been pressed down.

```
document.addEventListener('keydown')
```

So let's try:

```
document.addEventListener('keydown', function () {
        console.log('A key was pressed down')
})
```

Even when you write in the input field, it is going to catch the event.

Key Event

Let's see if we can avoid logging into the console when the text input field is active.

In the javascript:

```
const seat_selector = document.querySelector(".seat_selector")
document.addEventListener('keydown', function () {
	if (document.activeElement != seat_selector){
		console.log('A key was pressed down')
	}
})
```

Key Event

Let's filter which key can log into the console.

We can specify a variable to the function inside the event listener of 'keydown', which will match the event (https://developer.mozilla.org/en-US/docs/Web/API/Element/keydown_event)

```
document.addEventListener("keydown", function (event) {
 if (document.activeElement != seat_selector) {
   if (event.key == "r") {
     console.log("A key was press", `${event.key}`);
   }
 }
});
```

What is that code doing?

Create DOM Elements with JS

Add some code to your movies.html

```
<div>
    <span>Movie list</span>
    <ul class="movie_list">
      <li>Matrix</li>
      <li>Titanic</li>
    </ul>
</div>
```

Create DOM Elements with JS

In your js file, create an array with movies

let movie_list = [{'name': 'Forrest Gump'}, {'name': 'The Fast and the Furious'}, {'name':' Frozen'}]

Create a const variable that selects the class "movie_list".

```
const containerMovieList = document.querySelector('.movie_list')
```

Create a function addMovieList() that is going to insert a bullet point in the html. The funcion will receive an argument movie. This function will help us to use the function insertAdjacentHTML
https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentHTML
See beforebegin, afterbegin, beforeend…
https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentHTML#description

Create DOM Elements with JS

Code to insert new bullet points with movie names in array:

```
let movieList = [{'name': 'Forrest Gump'}, {'name': 'The Fast and the Furious'}, {'name': 'Frozen'}]
const containerMovieList = document.querySelector('.movie_list')

const addMovieList = function (movie) {
        const html = `<li>${movie.name}</li>`
        // write your code here to insert the values to the list
        // use insertAdjacentHTML('afterbegin', html)
}
for (const movie of movie_list) {
        addMovieList(movie)
}
```

Solution

```
let movieList = [{'name': 'Forrest Gump'}, {'name': 'The Fast and the Furious'}, {'name': 'Frozen'}]
const containerMovieList = document.querySelector('.movie_list')
const addMovieList = function (movie) {
	const html = `<li>${movie.name}</li>`
	containerMovieList.insertAdjacentHTML('afterbegin', html)
}
for (const movie of movieList) {
	addMovieList(movie)
}
```

With 'afterbegin', an element will be inserted before the first child.

Let's test it 'beforeend', it is going to insert it at the end, this is better. You can also test with the 2 other options, and you will see it is going to insert the html before and after the <ul></ul>

Do not forget to use the ` for the string to replace HTML

If we want to remove what we had in the <ul></ul> container, we do this,

containerMovieList.innerHTML = ''

before doing the loop.

Select, create and delete an Element

Select: You know this already document.querySelector

This will select the first element that match the specified selector
If you want to select all the element, you are going to use:

document.querySelectorAll
There are a lot of other function:
document.getElementById (no need to put the #)
document.getElementsByTagName
document.getElementsByClassName (no need to put the .)

See the MDN document for more information (https://developer.mozilla.org/en-US/docs/Web/JavaScript)

Select, create and delete an Element

Create: we can create the element as a javascript object

Let's delete the coder from your movies.html and add the following code:

```
<div class="movie_list_container">
    <span>Movie list</span>
</div>
```

Add this code to you js file:

```
const  containerMovieList = document.createElement('ul')
containerMovieList.classList.add('movie_list')
```

Exercise - Insert movie name

```
let movieList = [{'name': 'Forrest Gump'}, {'name': 'The Fast and the Furious'}, {'name': 'Frozen'}]
const  containerParentMovieList = document.querySelector('.movie_list_container')

const addMovieList = function (listElement, movie) {
	const li_element = document.createElement('li')
	// insert the movie name inside the li_element
	// then add li_element inside the listElement
}
const containerMovieList = document.createElement('ul')
containerMovieList.classList.add('movie_list')
for (const movie of movieList) {
	addMovieList(containerMovieList, movie)
}
containerParentMovieList.insertAdjacentElement('beforeend', containerMovieList)
```

Select, create and delete an Element

Delete: We can delete elements executing JS code.

Let's create a button to delete the ul list.

```
<button type="button" class="remove_ul_list">Delete ul list</button>
```

```
document.querySelector(".remove_ul_list").addEventListener("click", function () {
        containerMovieList.remove()
})
```

As the object containerMovieList is in the JS, we can use this object to use one of its base functions, which is remove, to delete it, and then it will not exist anymore in the DOM.

Styles, attributes and classes

Having a javascript object, you can edit the css inline with the property of the object.

For example:

```
containerParentMovieList.style.backgroundColor = 'lightgray'
```

If you want to see all the styling that is applying on an element, you can this:

```
getComputedStyle(containerParentMovieList)
```

Styles, attributes and classes

In the css, you can write variables that you will reuse all along (color, size font, ….).

Some we can edit the style attribute in our element, you can do the same thing for other attributes, for example the src attribute in an image

For the class, you can get the property classList or className. If you have multiple classes, className is going to give you the string with all the classes that the element belongs to.

You can also use the method getAttribute to get the value of an attribute if you have the attribute in a variable.

Styles, attributes and classes

For example, in the css, we can use a root variable this way

```
:root {
 --primary-color: #65d5ff;
}

.remove_ul_list {
		background-color: var(--primary-color);
}
```

We can modify it directly from javascript.

```
<button type="button" class="change_color">Change button color</button>

document.querySelector(".change_color").addEventListener("click", function () {
		document.documentElement.style.setProperty('--primary-color', 'red')
})
```

eventListener shortcut

Here is a list of events: https://developer.mozilla.org/en-US/docs/Web/Events

You know that we can trigger an event using the method addEventListener

```
containerMovieList.addEventListener('mouseenter', function (event) {
  console.log("mouse passed over the div of the movie list")
})
```

You can use a shortcut:

```
containerMovieList.onmouseenter = function (event) {
 console.log("mouse clicked on the movie list dev")
}
```

Remove eventListener

You can also remove the event listener.

Let's create a function first for the event listener.

```
const eventFunc = function (event) {
        console.log('mouse passed over the div of the movie list')
        containerMovieList.removeEventListener('mouseenter', eventFunc)
}

containerMovieList.addEventListener('mouseenter', eventFunc)
```

Where **'mouseenter'** represents a string which specifies the type of event for which to remove an event listener **eventFunc** is event listener function of the event handler to remove from the event target.

Test it, you will see the log only once. When the event is triggered, it is going to remove it also. It is a one time event.

Remove eventListener

You can remove the event lister with a timer.

```
const eventFunc = function (event) {
        console.log('mouse passed over the div of the movie list')
}

containerMovieList.addEventListener('mouseenter', eventFunc)
setTimeout(() => containerMovieList.removeEventListener('mouseenter', eventFunc), 10000)
```

After 10 second, the event will not be triggered

OOP Javascript

This is the way to create a new object with a **constructor function**

```
const Student = function (firstName, lastName, schoolClass) {
        this.firstName = firstName
        this.lastName = lastName
        this.schoolClass = schoolClass
}
```

To create the new object, just do this:

const ariana = new Student('Ariana', 'Smith', '7th Grade')

On the console, you can write ariana, and you see the object, the properties (firstName, ….).

OOP Javascript

We can set a **function** with it:

```
const Student = function (firstName, lastName, schoolClass) {
        this.firstName = firstName
        this.lastName = lastName
        this.schoolClass = schoolClass

        this.saveGrade = function(grade, curve) {
                grade = grade + curve
                if (grade > 100) {
                  this.grade = 100
                } else if (grade < 0) {
                  this.grade = 0
                }
        }
}
```

And run the function saveGrade:

```
ariana.saveGrade(95, 10)
```

Prototype Object

You may have noticed that we have a prototype object attached to the object ariana.

Prototype is like a chain (until it is null).

This prototype object is going to include the methods by inheritance, and the method in this prototype properties are not going to be created, but they are going to refer to. This is the javascript way.

Prototype Object - Example

```
const Student = function (firstName, lastName,
schoolClass) {
 this.firstName = firstName
 this.lastName = lastName
 this.schoolClass = schoolClass
}

Student.prototype.saveGrade = function(grade, curve) {
   grade = grade + curve
   if (grade > 100) {
     this.grade = 100
   } else if (grade < 0) {
     this.grade = 0
   }
 }
```

And you can still do ariana.saveGrade(95, 10)

We can even modify the function saveGrade and even after the object ariana has been created, you will see the new saveGrade function applied to the object ariana.

After the Student construction, and the Student.prototype.saveGrade function definition, and after creating the object ariana, let modify the saveGrade function:

```
Student.prototype.saveGrade = function(grade, curve) {
   this.grade = grade + curve
}

ariana.saveGrade(95, 10)
```

The grade for ariana will be 105 this time.

For more information: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain

Prototype Object

For example, you can create an object array,
list = [1, 2, 3, 4] and in the console, you can lookup at all the functions in the prototype property, or also write list.__proto__ to see the same thing.
You will see all the methods relative to the array, it is there in the prototype property.
list = [1, 2, 3, 4] is the same thing as we would have writing list = new Array(1, 2, 3, 4).
[ ] use the Array constructor (shortcut), and in the documentation, you will use the function of all the constructor as in constructor_name.prototype.function_name, so now you know why.

Practice

1. Let's create a constructor function to implement a Car. A car has a brand and a speed property. The speed property is the current speed of the car (no unit for the moment)
2. Implement an 'accelerate' method that will increase the car's speed by the number of the parameter, and log the new speed to the console
3. Implement a 'brake' method that will decrease the car's speed by the number of the parameter and log the new speed to the console
4. Create 2 car objects and experiment
   a. Data car 1: audi with speed at 130
   b. Data car 2: renault with speed at 100

ES6 Classes

```
class Student {
	constructor(firstName, lastName, schoolClass) {
		this.firstName = firstName
		this.lastName = lastName
		this.schoolClass = schoolClass
	}

	saveGrade(grade, curve) {
	   grade = grade + curve
	   if (grade > 100) {
	     this.grade = 100
	   } else if (grade < 0) {
	     this.grade = 0
	   }
	 }
}
```

const ariana = new Student('Ariana', 'Smith', '7th Grade')

This replaces what we did before with the constructor function. You notice that the **saveGrade function has been placed automatically in the prototype property** because "This is the way" of javascript. By the way, do you have this expression "This is the way"? Star Wars? The Mandalorian? https://www.youtube.com/watch?v=uelA7KRLINA

ES6 Classes - Set a function (prototype)

Of course, you can set a function on the prototype of the class as before, it will work as well, ariana is still the same object as previously:

```
Student.prototype.saysHi = function () {
        console.log(this.firstName, this.lastName, "says hi!!!!")
}
```

Setters and Getters

Let build an object, grade_object:

```
const grade_object = {
        'student': 'Kate',
        'grades': [95, 90, 98],
        get latest() {
                return this.grades.slice(-1)
        },
        set latest(grade) {
                this.grades.push(grade)
        }
}
```

Get latest grade: Execute grade_object.latest (you use it as **property**, not a function)

Set latest grade: Execute grade_object.latest = 93

We can implement **setters and getters in a class** as well:

```
class Student {
        constructor(firstName, lastName, schoolClass) {
                this.firstName = firstName
                this.lastName = lastName
                this.schoolClass = schoolClass
                this.grades = []
        }
        saveGrade(grade, curve) {
          grade = grade + curve
          if (grade > 100) {
            this.grade = 100
          } else if (grade < 0) {
            this.grade = 0
          }
         }
        get latest() {
                return this.grades.slice(-1)
        }
        set latest(grade) {
                this.grades.push(grade)
        }
}
```

The get and set latest are also placed in the prototype property.

Static methods

To create static methods, you just need to attach a function to the object:

```
const Car = function (make, speed) {
	this.make = make
	this.speed = speed
}

Car.static_method = function() {
	console.log('Hey, I am a static method')
}
```

This will work when you call the function as "class function", like this:
Car.static_method()

Static methods

If you create a Car instance object and then tried to run the static_method in it, it will not work:

```
const bmw = new Car('bmw', 10)
bmw.static_method() => error
```

Static methods in classes

```
class Student {
        constructor(firstName, lastName, schoolClass) {
                this.firstName = firstName
                this.lastName = lastName
                this.schoolClass = schoolClass
                this.grades = []
        }
        saveGrade(grade, curve) {
          grade = grade + curve
          if (grade > 100) {
            this.grade = 100
          } else if (grade < 0) {
            this.grade = 0
          }
         }
        get latest() {
                return this.grades.slice(-1)
        }
        set latest(grade) {
                this.grades.push(grade)
        }
        static static_method() {
                console.log('I am a static method in Student class')
        }
}
```

Inheritance

JS support inheritance and the syntax is similar to Java:

```
class Class1 extends Class2 {
…..
}
```

Exercise

```
class Animal {
  constructor(name, favFood) {
    this.name = name;
    this.food = favFood;
  }
  identifier() {
    return `I am ${this.name}`;
  }
}

class Dog extends Animal {
  constructor(name, favFood, sound) {
        // find out how to call the constructor
       // of the Animal class
        this.sound = sound;
  }
}
```

Write a function makeNoise() in Dog class, that returns the sound variable 2 times.

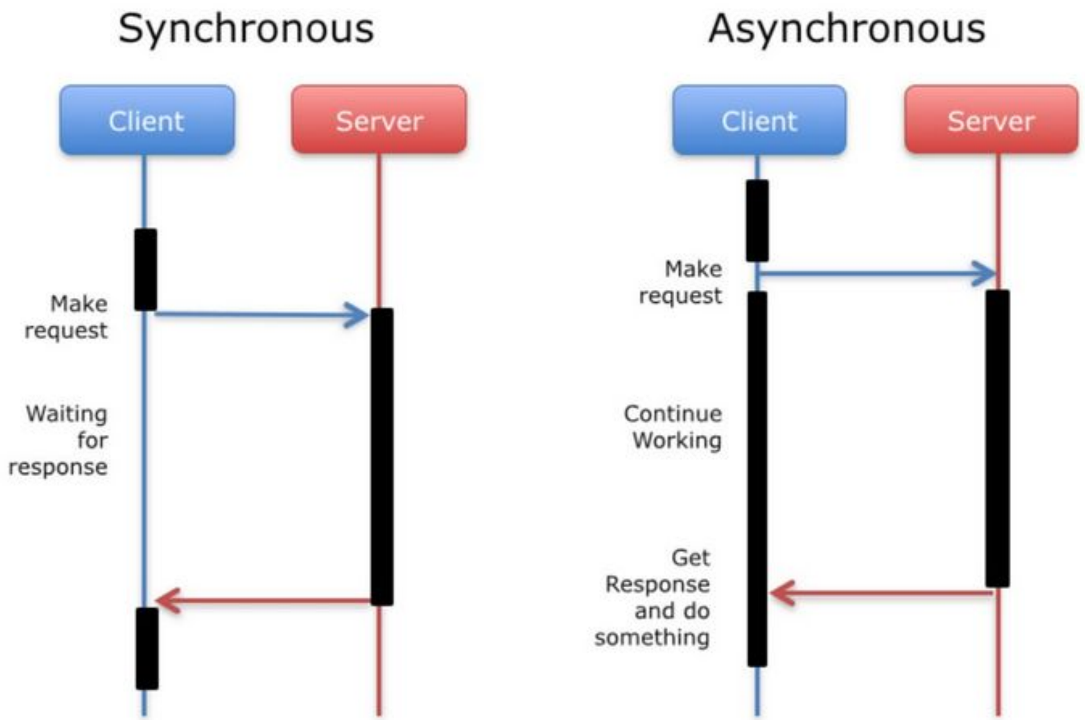
Create a class Poddle (subclass of Dog), that will overwrite the makeNoise function, calling the parent makeNoise, and then you add to the string, ", moving the tail" and return the string.

AJAX

For the European soccer fans, it is not not Ajax Amsterdam, it is **Asynchronous JavaScript and Xml.**

Do you know what asynchronous means?

Synchronous - Asynchronous



Synchronous - Asynchronous

Asynchronous: It is is a technique that enables your program to start a potentially long-running task and still be able to be responsive to other events while that task runs, rather than having to wait until that task has finished. Once that task has finished, your program is presented with the result.

Synchronous: As the name suggests synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed.

Synchronous Example

First let's setup the color blue on a –primary-color:

```
document.documentElement.style.setProperty('--primary-color', 'blue')

console.log('I am starting synchronous, and I am going to wait until you click on the alert')
alert('Hey, the code is blocking right now')
console.log("ok I am back working here")
document.documentElement.style.setProperty('--primary-color', 'purple')
```

The alert box was blocking the code, you did not see the console log 'ok I am back….'.

Let's set the color blue again:

```
document.documentElement.style.setProperty('--primary-color', 'blue')
```

Asynchronous - Example

We are going to insert an asynchronous function setTimeout. The setTimeout will wait in the background, and the following line is going to proceed, and when the background timer is done (and when javascript has finished what it had next), then it will proceed with the instruction inside the setTimeout (refer to the picture of the previous page)

```
console.log('I am going to go asynchronous, and I am not going to wait like a benet this time')
setTimeout(function () {
        document.documentElement.style.setProperty('--primary-color', 'purple')
        console.log("it is purple now, what? You logged something in the console while I was waiting")
}, 5000)
console.log("ok I finished here, quicker that the settimeout, now I am going to wait, because I finished first,
lalalala")
```