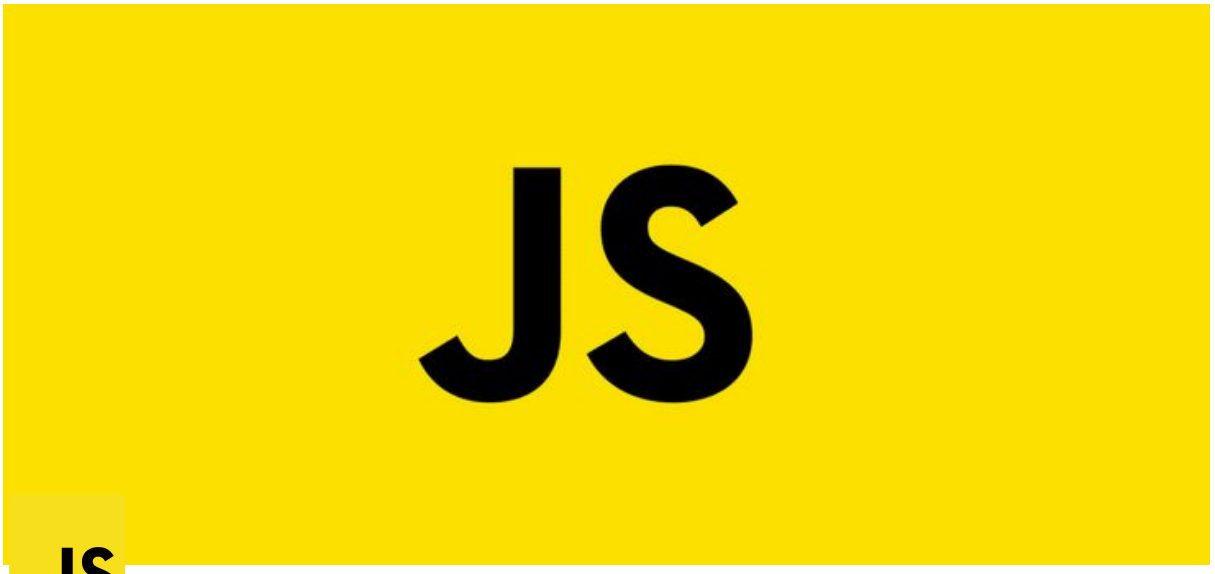


Javascript

Status	Done

Introduction

- Case sensitive
- Calculations using floats can be innacurate
- typeof = gives the type of the variables

```
console.log(typeof 3); //number
console.log(typeof var1); //string
```

Lesson 3 :Strings

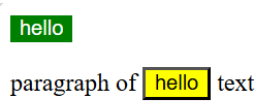
- Strings created with back-thiks are called ‘ template strings ‘ cause they have some special features . one od them is the use of interpolation.
- Interpolation : is more effective than concatenation , combination of characters , insert directly values into the string using ${} .
- multi-line strings → Only available when using template strings .

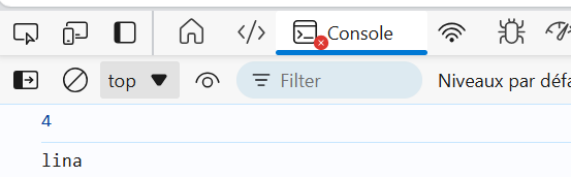
Lesson 4 : Html , css , console.log

- nesting : an element inside another element → <tag><element></element></tag>
- reminder of tooltips :

```
<button title="Good job!">
        hello
</button>
```

- Changing indentation and line-wrapping settings in vs code [Look into the video]



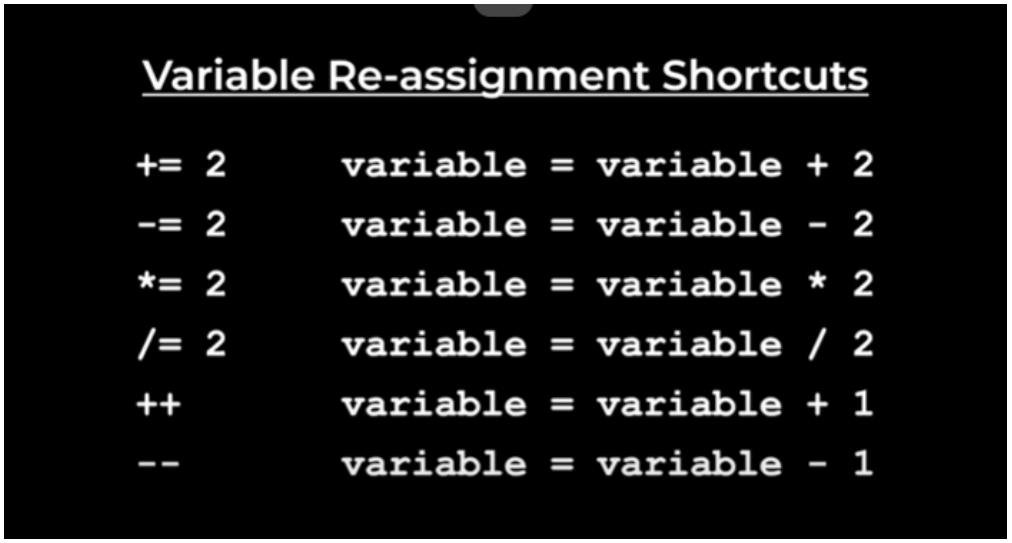


Lesson 5 : Variables

- ‘let var ’ → creates a new variable , and then we give these variables a name , ***we cannot have a variable named ‘let’*** , ***we cannot use numbers in the start of the names*** , ***we cannot use special charachter except for $ (Dollar) , _ (Underscore)***
- Console.log() → display whatever inside the bracket to the console of the website

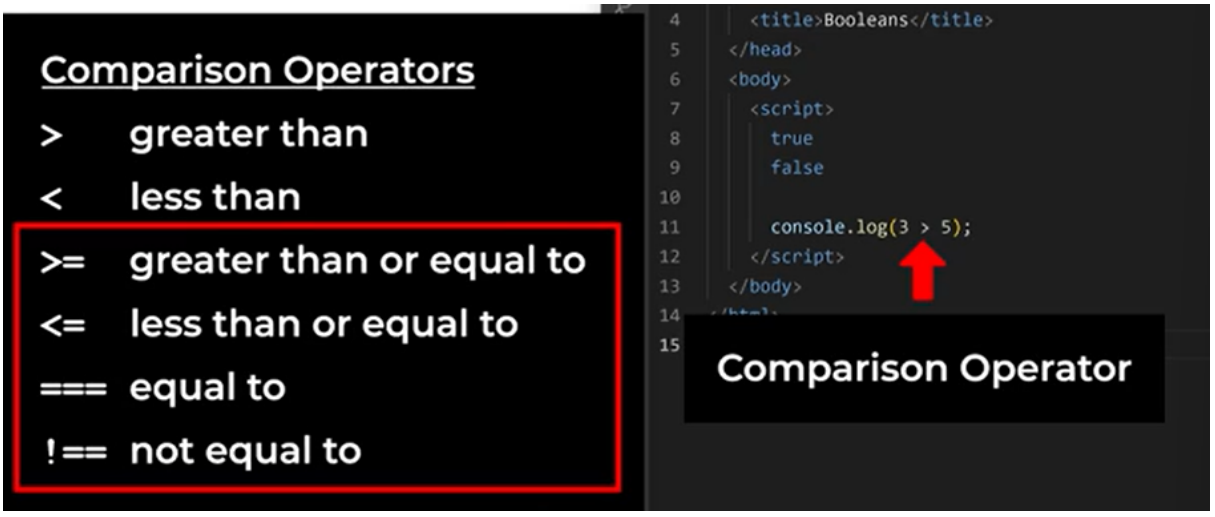
Shortcuts :

- Ctrl + A → to select all the code in an existing pagE



- Best Practices for naming variables : camelCase , PascalCase , kebab-case , snake_case —> in lower case .
- kebab-case is used in html and css , snake_case is not really used in js but used in other programming languages .
- There are 3 ways to create variables : const , let , var .

Lesson 6 : Booleans and if-statements



There are two ways to represent equality in javascript , it has two ways to check if two values are equal : === , ==

- The difference between == and === is that : double equal tries to convert the two values into the same type , so to check it is better to use the triple equals .
- To avoid the conversion version we use : !== to represent ***not equal to***

If-statements :

- var doesn't fully follow the rules of scope , scopes help us avoid naming conflicts .

Truthy-Falsy Values :

- Falsy Values : false 0 '' Nan undefined (any other value than those are called truthy values) , Nan = not a number , undefined = this variable is not defined
- Truthy Values : are values that behaves as true value

Some Operators :

- ?: = condition ? result1 : result2
- Guardian Operator — Short circuit evalution : evaluates only the first part and concludes some result (&& , ||)

Lesson 7 : Functions

- One of the best practices in JavaScript is to write clear and consistent names for functions, preferably in camelCase. Function names should typically use a verb or action to make them more expressive and easier to understand for anyone reading the code. The function name should describe what the function does .
- **Functions, like if-statements, create scopes.** Therefore, any variable created inside the curly brackets `{}` is confined to that scope and cannot be accessed outside the function.

we call those variables **local variables** , on the other hand there are g**lobal variables. Global variables** are variables that can be accessed from anywhere in the code.

- A function with a `return` statement without a value returns `undefined`.
- Inside a function, you can call another functions , which is often referred to as using a **callback**. A callback is a function passed as an argument to another function to be executed later, usually after some operation is completed.

Lesson 8 : Objects

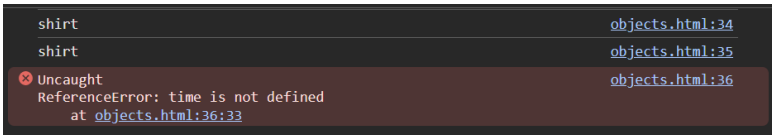
- An object groups multiple values togeher it contains values that can be of any type , strings , numbers or even objects we call the nameof these values ; properties , An object is just another type of value .
- We can use the property to change the value of the object's property , if we try to access a property that doesn't exist , it displays 'Undefined' , We can always **add a property to a object** and we can as well remove it .

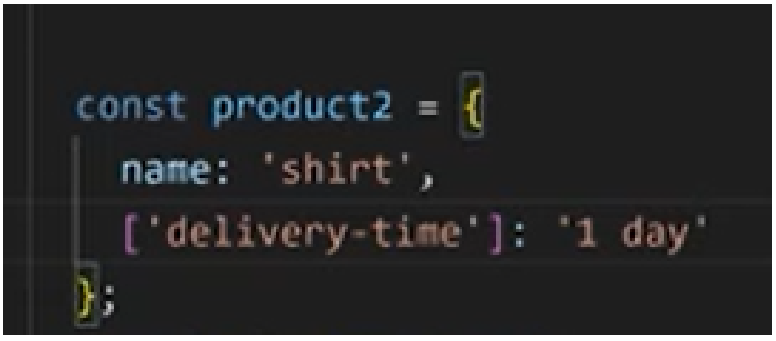
```
// Adding A property
product.newProperty = true //this value is added to the ob
console.log(product);

//Removing a property
delete product.newProprety;
consol.log(product.newProprety);
console.log(product);

//typeof used with objects
console.log(typeof product)
```

```
▶ {name: 'cotton socks', price: 1090, newProprety: true}
undefined
▶ {name: 'cotton socks', price: 1090}
```

We can access an object's properties in two ways: **dot notation** or **bracket notation**. When using bracket notation, if we try to access a non-existent property containing a '-' in its name, it displays 'undefined'. However, using dot notation in this case results in an error, as it interprets '-' as a mathematical operator. Bracket notation allows us to use properties that don't work with dot notation.

```
console.log(product2);
console.log(product2.name);
console.log(product2['name']);
console.log(product2.delivery-time);
```





- if the only thing inside the bracket is a string , we can remove the brackets , it is also important to note that inside the bracket , we can have strings, numbers , and other types of values .

Methods : functions inside objects

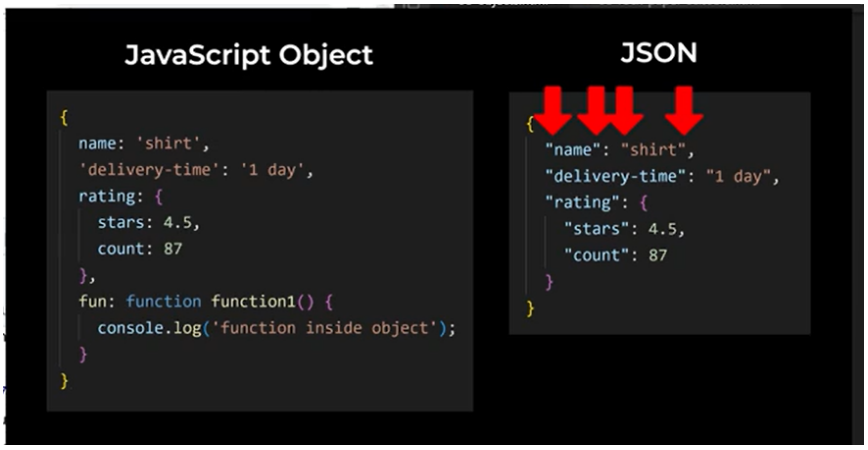
Methods are simply functions associated with a certain object .These functions can modify the object's properties or execute other instructions. Some examples of methods are `console.log()` , where `log` is a method of the `console` object and `Math.random()` , where `Math` is an object and `random` is a method saved within the `Math` object .

Built-in Object :

Built-in objects are objects that are provided by the language; they are integrated into the language itself, there are two built-in objects :

[JSON OBJECT]

- JSON (Javascript object notation) : is a syntax that is similar to javascript object with fewer features , **JSON does not support single quotes** , **JSON does not support functions** .
- JSON syntax is more universal and can be understood by many programming languages, unlike JavasScript objects, it is commonly used to store data , or when we send data from a computer to another



- The JSON object let use convert a javascript object into a JSON object , for that we use `stringify` that actually returns string . we use `JSON.parse()` to convert a string into an object.

```
console.log(JSON.stringify(product2));
console.log(JSON.parse(JSON.stringify(product2)));
```

```
{"name":"shirt","delivery-time":"1 day","rating":
{"start":4.5,"count":87}}
▼ Object i
    delivery-time: "1 day"
    name: "shirt"
  ▶ rating: {start: 4.5, count: 87}
```

[Local Storage]

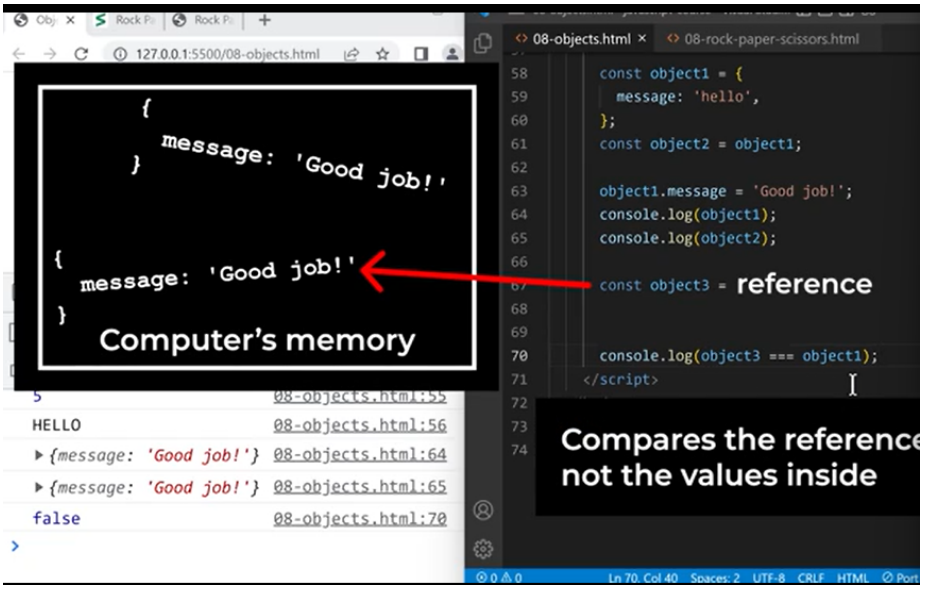
When a page is refreshed, all the variables of the objects are typically deleted, causing a loss of data. However, this can be resolved by using `localStorage`, which does not get cleared on page refresh. To save data in `localStorage`, the function `localStorage.setItem(name, value)` is used, where the value is stored under a specific key (name) in `localStorage`. To retrieve this data, we can use `localStorage.getItem(name)` to access the stored variable. It's important to note that `localStorage` only supports strings, so if you need to store objects or arrays, you must convert them to strings (e.g., using `JSON.stringify`) before saving.

[More details about objects]

- null = we intentionally want something to be empty , default value = if it is empty the default value is being used.
- Strings are objects and have properties and objects , one of the important tems related to strings is Auto-boxing ; it is a technique where JS automatically wrapps a string into an object , autoboxing does not work with null and undefined but works with numbers and booleans .
- Objects are actually references , we can copy an object by copying the reference of another object . when you compare objects , you actually compare references not objects .

```
const object2=object1;
```

- Even though we use const , we can always changet the values inside the object , constant prevent use to change the reference of the variable .



Shortcuts for objects : Destructing

- if we want to save an object property into another variable with the same name, we can use this syntax : it is called the destructuring shortcut. This is an easier way to take the object's variables out and save them into variables that have the same name .

```
//const message = object4.message;
const {message} = object4;
console.log(message);
console.log(object4.message);
```

```
const object4 {
    message : 'Good job!',
    price : 799
};

const {message , price} = object4;
console.log(message);
console.log(object4.message)
```

Shorthand shortcut:

The shorthand shortcut allows us to take whatever is in the variable name and save it into the object property with the same name. Additionally, this shortcut eliminates the need to specify that a method is considered a function, as well as removing the necessity to specify the name of the function itself.

```
const {message , price} = object4;
console.log(message);
console.log(object4.message);
const object5 = {
    message, //shorthand property syntax
    //method : fucntion fucntionNamr(){console.log('hello');}
    method(){
            console.log('hello');
    }
}

object5.method();
```

Lesson 9 : DOM

DOM is another built-in object , it contains properties and methods ., the special thisng about the DOM is that it is linked to the webpage .DOM is a short syntax for document object model , the document object represents the webpage that is way it is name document object model , the DOM combine HTML and javascript

The **DOM (Document Object Model)** is a built-in object in web development that provides a structured representation of a webpage. It acts as an interface that allows JavaScript to interact with and manipulate HTML content. The special thing about the DOM is that it's directly linked to the webpage, enabling dynamic changes to its structure, style, and content.

The DOM represents the entire document (or webpage) as a tree of objects, where each element, attribute, and piece of text is a node in this tree. This structure allows JavaScript to easily access and modify different parts of the page. For example, you can change the content of a paragraph, alter styles, or listen for events like button clicks.

In essence, the DOM bridges HTML and JavaScript, allowing web pages to be interactive and responsive to user input.

Properties and methods :

- `document.title` : returns the title of the document .

- `document.body` : body is an object , it represents the body tag in html , when an html tag is inside javascript it becomes an object .
- `document.querySelector()` : a method that helps get html elements
- `innerHTML` : Every html element have a proprety : innerHTML
- `Number()` : to manually convert a string into a number we can use the function : Number() , the number takes whatever inside the brakets and transforms it into a number .

Event Listeners :

Events in JavaScript are actions triggered by user interactions like clicks or key presses. The event object provides details about the event, allowing developers to respond with event listeners like `onclick` or `onkeydown` for interactive behavior.

- `onkeydown` : Runs JavaScript when a key is pressed down on the keyboard.
- `onclick` : Executes JavaScript when an element is clicked.These are examples of event listeners that respond to user actions.
- `onscroll` : Runs JavaScript when the user scrolls the page.
- `onmouseenter` : Runs JavaScript when the mouse pointer hovers over an element.
- `onmouseleave` : Runs JavaScript when the mouse pointer leaves an element (stops hovering).
- events is an object provided by javascript

Converting from string ↔ number :

- `Number()` to change a string or whatever inside the brackets into a number.
- `String()` to change the number into a string.

Implicit Type Coercion

: JavaScript automatically converts types based on the operation being performed.

- `console('25' - 5)` → The string `'25'` is automatically converted to a number, resulting in `20`.
- `console('25' + 5)` → The number `5` is converted to a string, resulting in `'255'` because the `+` operator concatenates strings.

Window Object :

The **window object** represents the browser and serves as the global object in JavaScript. The **document object** (representing the webpage) is part of the window object, along with other features like the **console** and **pop-ups**. The window is a built-in object provided by JavaScript and includes various properties and methods to interact with the browser environment.

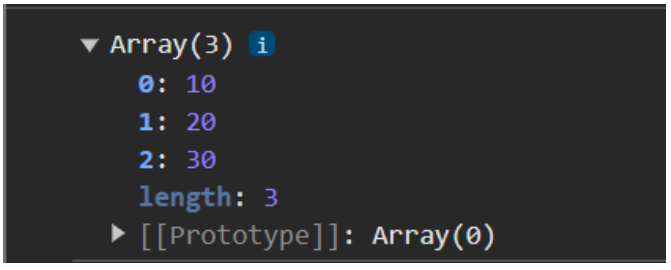
Lesson 10 : HTML , CSS and Javascript together

- Every element has a `classList` attribute, which provides control over the element's classes.
- `classList.add('nameOfClass')` : Adds a class to the element.
- **Border style**: The `border` property is used to define the width, style (e.g., solid, dotted, dashed), and color of an element's border. For example: `border: 2px solid red;`.

Lesson 11 : Arrays and loops

An Array is a value in Javascript that represents alist of values. It can contain various data types , including numbers ,strings , objects and other arrays. Additionally, arrays are a special type of object in JavaScript.

```
const myArray=[10, 20, 30];
console.log(myArray);
```



To check if a variable is an array, we use `Array.isArray(variable)`. Arrays in JavaScript have both properties and methods. One key property is `length`, which returns the number of elements in the array. Common array methods include `splice(from, to)`, which modifies the array by adding or removing elements, and `push(value)`, which adds a new element to the end of the array.

Standard and non-Standard loop

- **Standard Iteration**: This typically refers to loops that follow a predictable pattern of incrementing and decrementing an index. This type of iteration is common in structures like `for` loops.
- **Non-Standard Iteration**: This refers to loops that may not follow a predictable incrementing pattern. Instead, the loop's continuation is based solely on a condition. An example is the `while` loop, which is often used for non-standard iterations where we don’t need to increment the index value each time. Instead, the loop is primarily based on a condition.

Looping throught an array :

```
const todoList = [
    'make dinner',
    'wash dinner',
    'watch youtube'
];

for(let i=0;i<=todoList.length-1; i++){
    console.log(todoList[i]);
}
```

The Accumulator Pattern :

```
const num = [1,1,3];
let total=0;
for(i=0;i<num.length;i++){
    total += num[i];
}

console.log(total)
```

```
// A table can also be considered an accumulator
let numsDoubled=[];
for(i=0;i<num.length;i++){
    numsDoubled.push(num[i)*2;
    // numsDoubled[i] = num[i] * 2;
```

References and copies :

Arrays, like objects, are reference types. To avoid the behavior where changing one array also affects another because they share the same reference, we can use the `slice()` method, which creates a copy of the array without maintaining the same reference.

```
const array1 = [1,2,4];
const array2 = array1.slice();
array2.push(4);
console.log(array1); // [1,2,4]
console.log(array2); // [1,2,4,4]
```

Shortcut : Destructuring

We can easily extract multiple values from an array using this shortcut, which helps optimize the time it takes to retrieve those values.

```
  const array3=[1,2,3]; // To get the first and second value
  const [the , SecondValue] =[array3[2],2];
  console.log(the);
```

Lesson 12 : Advanced Functions

Functions are also values in JavaScript, meaning we can save a function inside a variable. To call this function, we simply use the variable name followed by parentheses. An anonymous function is a function that does not have a name and is saved in a variable.

```
// The use of advanced fucntions
// Advantages : Easier to read , Hoisitng
function greeting(){
    console,log('hello');
}
```

Hoisting is a feature in JavaScript that allows us to call functions regardless of the order in which they are defined, and this feature applies specifically to function declarations.

Additionally, we can add functions to objects, enabling us to associate behavior with those objects.

Asynchrounous and Synchronous Code :

- `Asynchronous Code` doesn’t wait for a line to finish before executing other lines. For example, `setTimeout()` uses asynchronous code, meaning it doesn't block the execution of other instructions while it waits to complete its task.
- `Synchronous Code` will wait for one line to finish before moving to the next. Each operation is executed one after the other, blocking the following instructions until the current one is done.

```
let isAutoPlaying = false;
let intervalIdl
function autoPlay(){
     clearInterval(intervalId);
     isAutoPlaying = false;
} else {
    isAutoPlaying = true;
    intervalId = setInterval(function(){
        const playerMove = pickComputerMove();
        yourMove(playerMove);
    }, 1000)
}
```

There’s a possibility to pass a function as an argument to another function. For example, the `setTimeout()` function is a built-in JavaScript function that takes a function as the first parameter and a number (representing time in milliseconds) as the second parameter, specifying how long it should wait before calling the given function.

setTimeout(function, param in millleseconds)

You can pass a function as an argument to another function. The

```
setTimeout()
```

function in JavaScript allows you to do this. It takes two parameters:

1. A function to execute after the delay.
2. The time (in milliseconds) to wait before executing the function.

```
setTimeout(function(){
  console.log('timeout');
} , 3000 ); //built-in functions
```

setInterval(function , param in milleseconds)

Another way to loop throught an array

We can loop throught an array using for or while , another method we can use is : forEach , we cannot use break in a forEach loop , we cannot use break inside forEaxhe , if we nwwd to use, it is better to use a regular for loop .

```
[
    'make dinner',
    'wash dishes',
    'watch youtube '
].forEach(function(value)){
    console.log(value);
}
```

Arrow Functions :

Arrow functions work similarly to regular functions but have a different syntax. An arrow function can be assigned to a constant, just like any other function.
Arrow functions offer some shortcuts: you can omit the parentheses if there is only one parameter, and you can also omit the
`return` keyword if the function contains only a single expression.

Additionally, features like method shorthand and destructuring can be used with arrow functions, making them even more concise and readable.

```
todoList.forEach((todoObject ,index)=>{
    const {name} = todoObject;
    const {dueDate} = todoObject;
    const html = `
        <div>${name}</div>
        <div>${dueDate}</div>
        <button class="deleteBtn" onclick="todoList.splice(${
        Delete
        </button>
    `;
    todoListHtml += html;
    })
```

```
cont object2={
    method: () => {

    },
    method(){

    }
};
```

Event-Listeners :

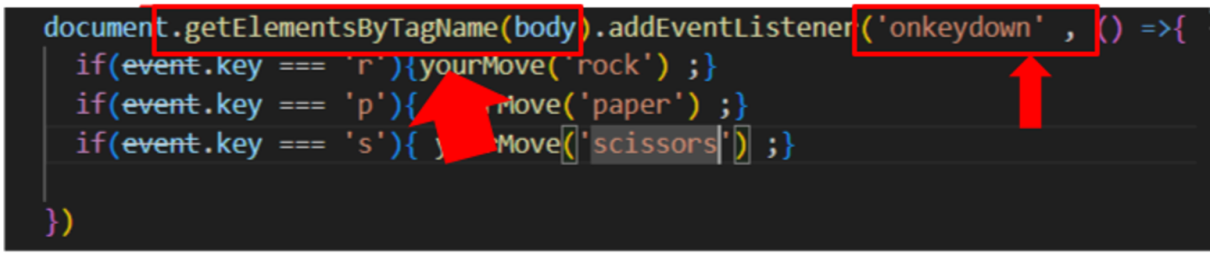
- `addEventListener()` takes two parameters: ***the first parameter*** specifies the event type (e.g., `'click'`, `'mouseover'`), and ***the second parameter*** is the function that should execute when the event occurs.
- We can add multiple event listeners for the same event type on a single element, and we can also remove multiple event listeners for the same event type from the same element.

```
document.querySelector('.js-rock-button'),addEventListener('c
```

Array Methods :

- The `filter()` method creates a new array containing only the elements that satisfy a given condition. For example, if you want to include only positive values from an array, `filter()` will generate a new array consisting solely of those positive values. Unlike `forEach()`, which does not return a value, `filter()` returns the new array.
- The `map()` method transforms each element of an array according to a specified function and returns a new array containing the transformed . It maps the original array to a new array with each element processed by the provided function.
- When a function (referred to as the inner function) is defined inside another function (the outer function), it gains access to the variables of the outer function. This access remains even after the outer function has finished executing. The inner function "remembers" the environment in which it was created, including the variables from the outer function. This behavior is known as a closure.

Side notes



- `document.body` returns the single `<body>` element of the document, while `getElementsByTagName()` will return an `HTMLCollection` containing all elements of the specified tag name. However, `getElementsByTagName()` is not used specifically for the `<body>` tag in practice, as `document.body` is a more straightforward way to access it.

Lesson 13 : Intro to git + Amazon prjt

Git provides a way to track changes in our code; it is both a tool and a technology. This is why it is effective to have a GitHub setup in Visual Studio, as it allows us to manage changes efficiently, including the ability to revert them if necessary. Git is particularly useful for larger projects. A repository serves as a folder where changes are tracked. To add a username or email to Git in Visual Studio, we can use the terminal with the following commands:

- `git config user.email "name@gmail.com"`
- `git config user.name "username"`
  To save changes made to files, we need to write a message in the input field above the

commit submit button in the Git panel within Visual Studio. Additionally, pressing `CTRL + F` allows us to search for specific elements in our code.

Generate HTML code using JS :

- The `toFixed()` method is used to force having a number of decimals after the coma , The `toFixed()` method rounds the number to the specified number of decimal places and ensures the final result is clean, avoiding unnecessary decimal points.

```
<div class="product-price">
    $${(product.priceCents/100).toFixed(2)}
</div>
// 6.005.toFixed(2) => '6.00' ❌
// 7.005.toFixed(2) => '7.00' ❌
// 8.005.toFixed(2) => '8.01' ✅
```

A data attribute is an HTML attribute that allows us to attach any data to an element. Data attribute names always begin with
`data-`, and we use kebab-case for naming. This allows us to associate additional information with the element. The `Element.dataset` property provides access to these attributes related to the element

Lesson 14 : Modules

Problem :

```
<script src="data/cart.js"></script>
<script src="data/products.js"></script>
<script src="script/amazon.js"></script>
```

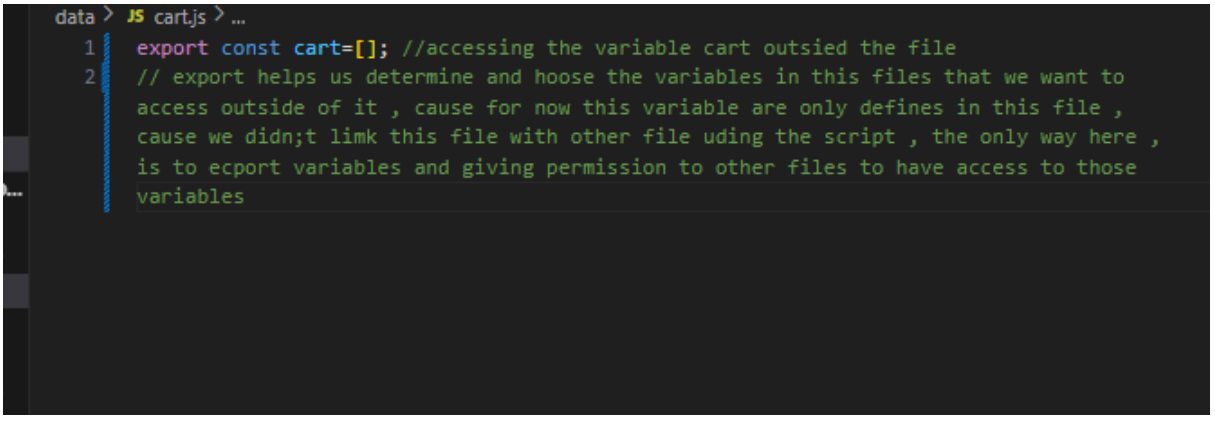
- This approach makes it hard to know which variables are already defined, increasing the likelihood of naming conflicts.
- Using too many script tags like this can cause naming conflicts because we may not be able to declare certain variables that are already defined in other JavaScript files.

Solution :

- To avoid this problem, we use modules. A module essentially contains variables within a file, preventing conflicts because the variables are only accessible within that file. To create a module, we need to create a file and load it with `<script type="module">`. Any variables we define inside the file will be contained within that file.

How to get a variable out of a module: To access a variable from a module, we must add the `type="module"` attribute to the `<script>` tag, and then we need to use export and import statements.

```
<script type="module" src="scripts/Amazon.js"></script>
```



Important notes :

- The import statements should always be at the beginning of the variables.
- Modules don't work without a live server. If we open the web page without using a live server, the modules won’t function, and therefore the variables cannot be accessed or exported. To ensure that modules work properly, we need to use Live Server.

Benefits of Modules :

- Modules help avoid naming conflicts, but conflicts can still occur if we import variables with the same name. We can resolve this by importing variables and renaming them to prevent naming conflicts.

```
import {cart as myCart} from '../data/cart.js';
```

- We don't have to worry about the order of the files when using modules. When we used script tags, we had to be aware of the order of the files because sometimes variables are defined and used in other files, making the order important. However, having too many script tags can be ineffective. Modules are the best way to organize our code, especially for complex websites.
- The entry point file is the file that links JavaScript files, module variables, and HTML all together.
- When importing from other files, we can import variables and functions in the same line from one page. We can also import everything from a file as an object, as shown in this image.
- Normalizing data is a technique where we assign an ID to an object to simplify the extraction of other data about that object. This process is also called deduplicating data.
- If a set of radio buttons has the same name attribute, we can only select one of them. We can use the ID in the name attribute of the input to separate the sections of the radio buttons.

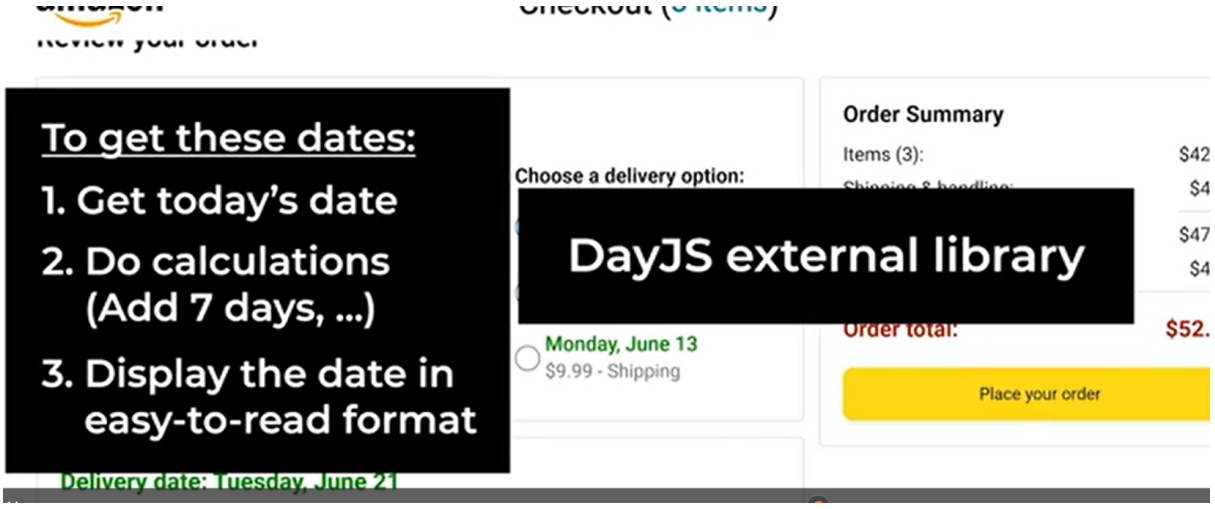
Each element in html has a function called remove() , which helps remove the element from html.

- Variables are reset when we refresh the page or when we go to another page .and this could be innefctive in many ways , in order to prevent this we use localStorage ,

localStorage takes two strings , the first strng is the name and the second is the value , localStorage only saves strings .

- Variables are reset when we refresh the page or navigate to another page, which can be ineffective in many ways. To prevent this, we use `localStorage`. `localStorage` takes two strings: the first string is the key (name), and the second is the value. Note that `localStorage` only stores strings.

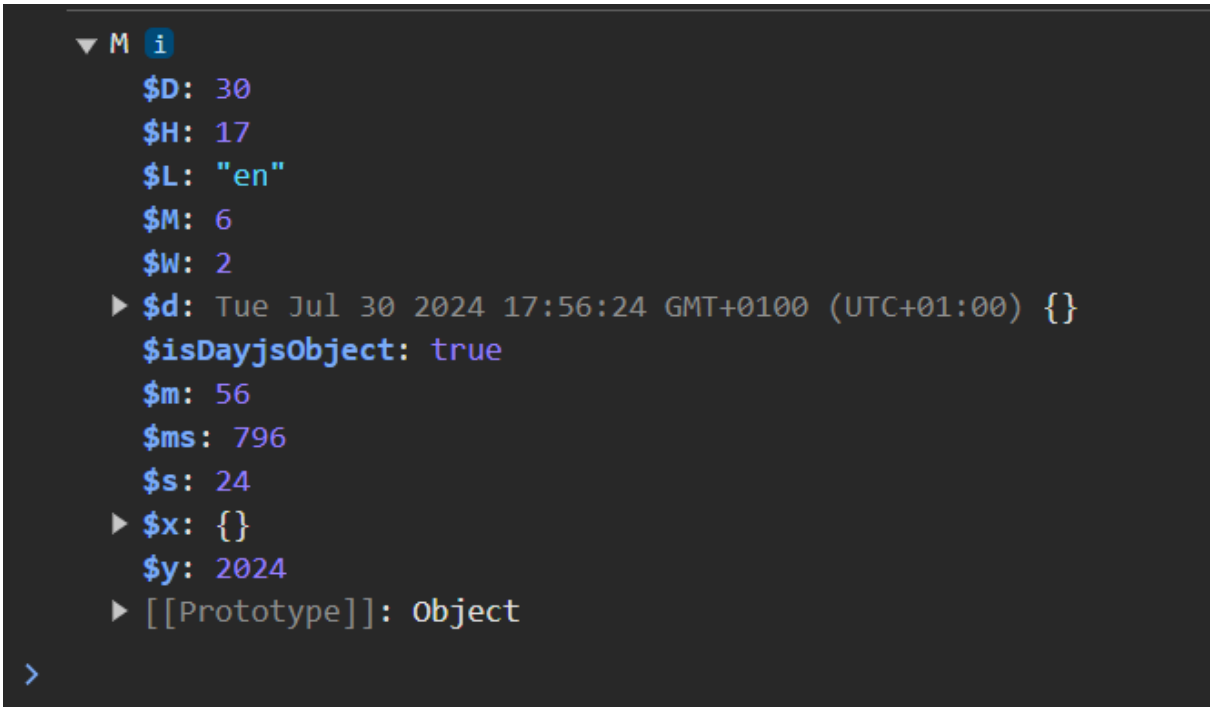
Lesson 15 :External Libraries



External libraries are pieces of code that exist outside of our project and can be easily loaded using the `<script>` tag. When integrating JavaScript code from the internet, we often compress it to enhance loading speed; this process is known as **minification**, which involves removing unnecessary characters such as whitespace, comments, and line breaks to reduce file size. For instance, the **Day.js** external library can be accessed from unpkg.com/dayjs@1.11.10/dayjs.min.js, and detailed usage instructions can be found in the Day.js documentation. The `dayjs()` function is particularly useful as it returns the current date and time, allowing developers to easily manipulate and format dates.

Using external libraries like Day.js simplifies common tasks, enabling developers to focus on building features rather than reinventing the wheel. Additionally, these libraries are often loaded from a Content Delivery Network (CDN), which provides caching benefits and faster load times. It's crucial to specify the version of the library being used to avoid unexpected changes in functionality that may arise with newer releases. Overall, leveraging external libraries not only enhances efficiency but also contributes to cleaner and more maintainable code.

```
console.log(dayjs());
```

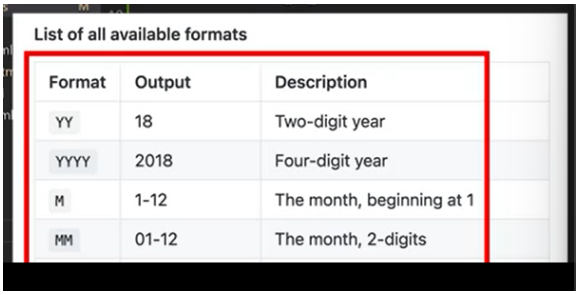


Calculation Using date with DayJS

The `add(a, b)` function takes two parameters: the first parameter, `a`, represents the number of units, while the second parameter, `b`, specifies the type of unit to add, such as days, minutes, hours, or even months, depending on our choice. The function returns a cloned Day.js object with the specified amount of time added.

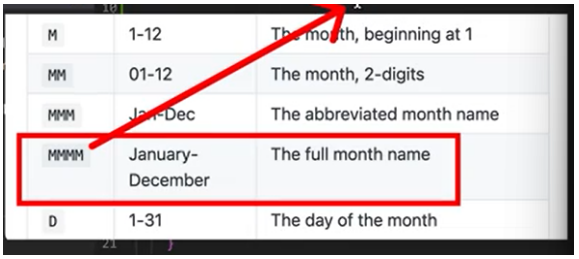
```
const a = dayjs();
const b = a.add(7,'day');
```

The `format` function retrieves the formatted date according to the string of tokens passed in. This format can take various forms. Below, you can find a table listing all available formats:

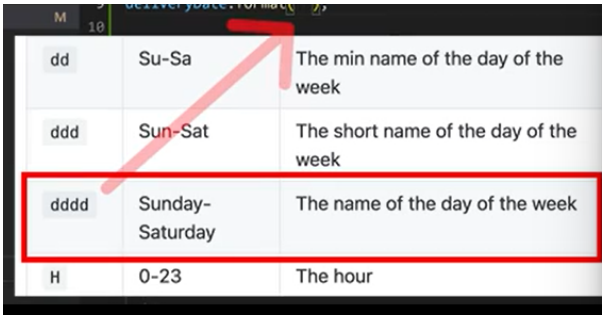


List of all available formats

Format	Output	Description
YY	18	Two-digit year
YYYY	2018	Four-digit year
M	1-12	The month, beginning at 1
MM	01-12	The month, 2-digits



M	1-12	The month, beginning at 1
MM	01-12	The month, 2-digits
MMM	Jan-Dec	The abbreviated month name
MMMM	January-December	The full month name
D	1-31	The day of the month



dd	Su-Sa	The min name of the day of the week
ddd	Sun-Sat	The short name of the day of the week
dddd	Sunday-Saturday	The name of the day of the week
H	0-23	The hour

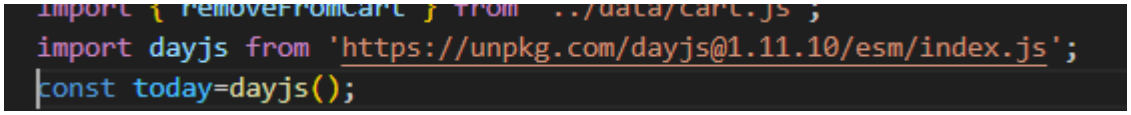
To use external libraries and JavaScript modules together, we need to use a special version of the library that is compatible with modules. This version is known as the ESM version, which stands for ECMAScript Modules. ECMAScript is another name for JavaScript, as it is the standardized specification upon which JavaScript is based.

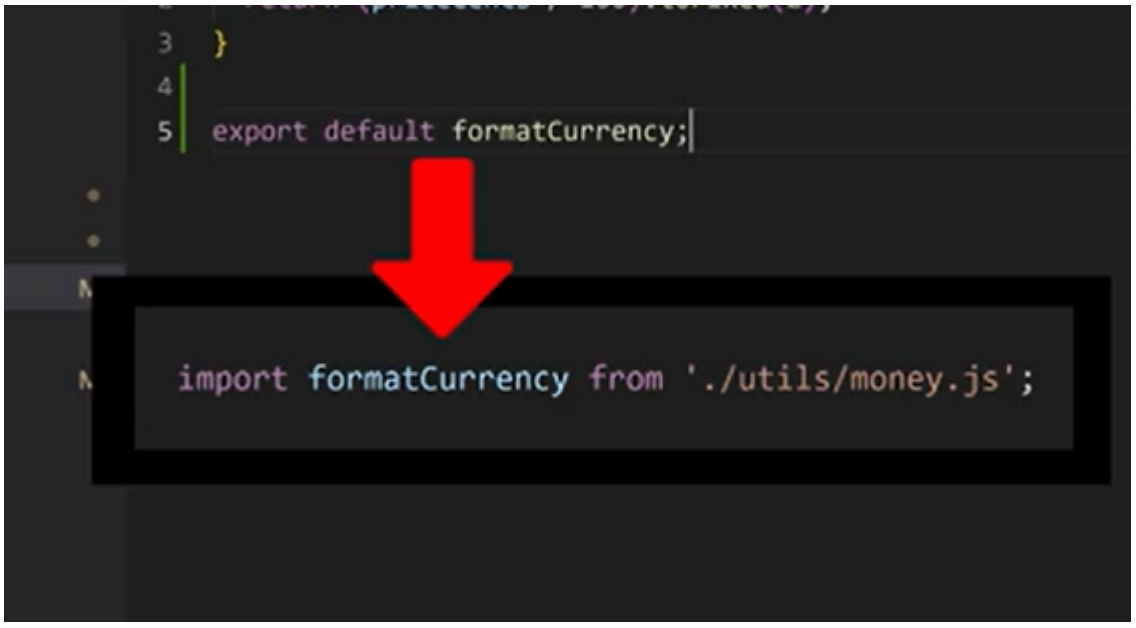
```
import {hello} from 'https://unpkg.com/supersimpledev@1.0.1/hello.
esm.js';
hello();
const today=dayjs();
```

```
hello
```

Default Export and naming Export :

A default export is another way of exporting. The type of export we used before is called a named export. Default exports are useful when we want to export only one value or function from a module. Each file can have only one default export. In contrast, **named exports** allow you to export multiple values or functions from a single file. When importing, **default exports** don’t require curly braces, while **named exports** do.





Default exports: When exporting a default value in JavaScript, you cannot rename it directly in the source file where it was defined. However, when importing the default export into another file, you can give it any name you'd like, as there can only be one default export per file.

```
//importing
import isSatSun from './15f.js';
let today = dayjs();

//Exporting
export default function isWeekend(date){
    const dayOfWeek = date.format('dddd');
    return dayOfWeek === "Saturday" || dayOfWeek === "Sunday"
}
```

Reload page , Updating js page :

Updating the page step by step: One way to update the content is by modifying individual elements on the page. However, another approach is to rerun all the relevant code, which

refreshes the page entirely. Instead of making changes directly through the DOM, it can be more efficient to recall the entire code, especially when dealing with complex updates.

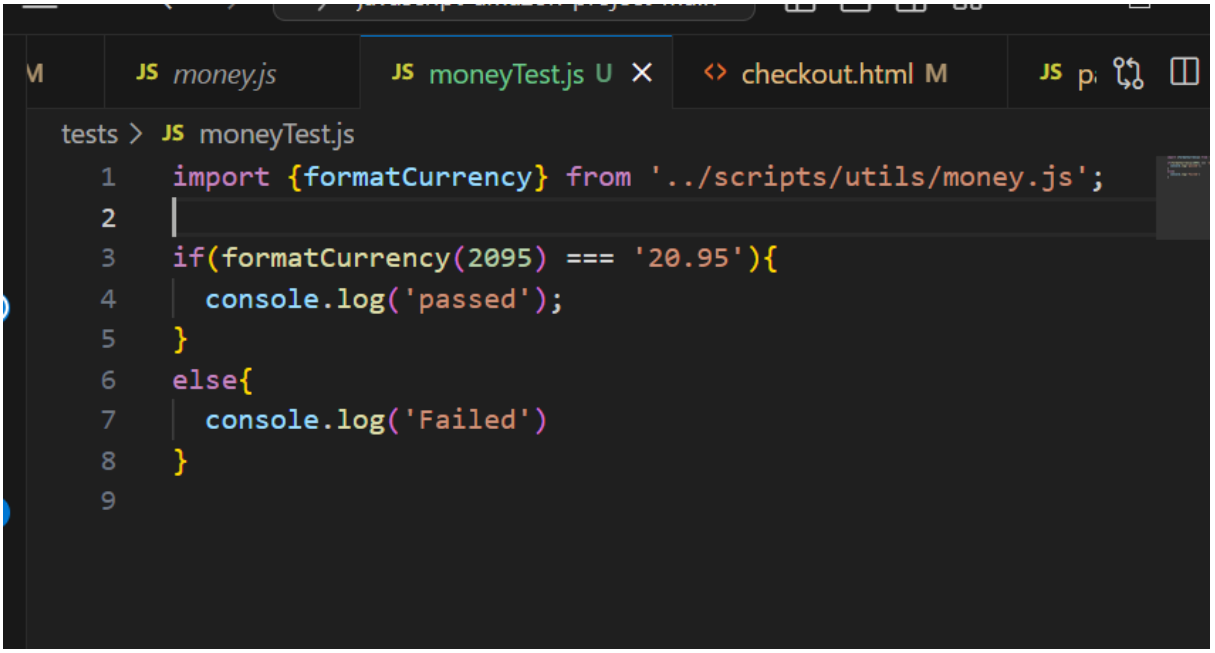
MVC (Model-View-Controller) Framework :

- **Model**: In MVC, the model is responsible for managing the data. This includes saving, retrieving, and manipulating the information needed by the application.
- **View**: The view is responsible for displaying the data. It takes information from the model and presents it to the user in a readable format.
- **Controller**: The controller acts as an intermediary between the model and the view. It runs code in response to user interactions, processing input and updating the view accordingly
- MVC = Model-View-Controller , in MVC we split our prjt into 3 parts , the firt part

Lesson 16 : Testing , Testing Frameworks

Manual and Automated Testing :

- **Manual Testing** : The easiest way to test our code is to open the website and try out the code. **Manual testing** is useful but has also disadvantages :
  - It is difficult to test every possible situation
  - Re-testing can be time-consuming and error-prone
- **Automated-testing :** instead of performing the tests manually we can have the computer execute them for us. Automated testing means using code to test other code allowing for more efficient and consitent testing .



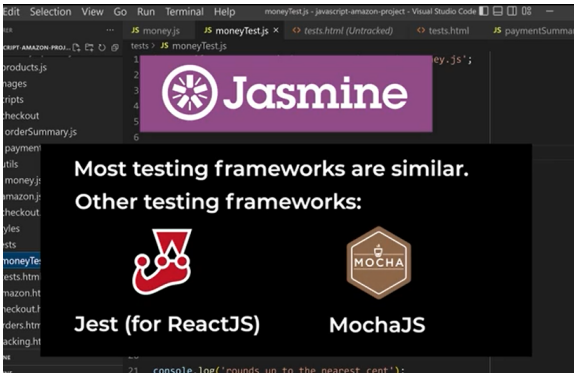

```
if(formatCurrency(0) === '0.00'){
  console.log('passed');
}
else{
  console.log('Failed')
}
```

Basic Test and Edge cases :

- There are two types of test cases: **Basic Test Cases** and **Edge Cases**. Basic Test Cases test whether the code is functioning as expected using normal or typical values. In contrast, Edge Cases test the code using unusual or extreme values that can be tricky or rare, often at the boundaries of normal input.
- A group of related tests is called a **test suite**. This allows related tests to be grouped and run together.

Testing Frameworks :

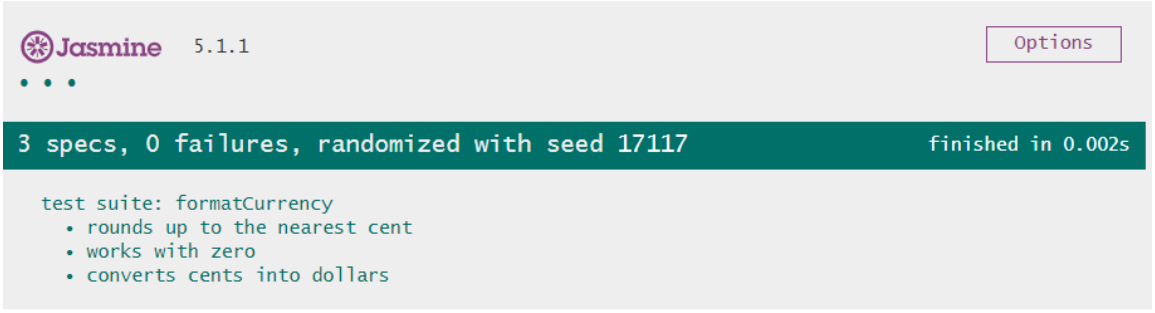
Testing frameworks are external libraries that help us test our code more easily and write our tests more efficiently.



- **Jasmine** is an external library that helps us easily test our code. Jasmine has a documentation page that you can check.
- The spec-running HTML file loads all the tests; in Jasmine, "spec" is just another name for a test.

- `describe()` :To create a group of test or test suite , `describe(description , specDefinitions)` , it creates a group of tests (often called a suite) , the calls to describe can be nested within other calls to compose your suite as a tree .
- `it (description, testfunction,  timeout )` : Define a single spec , A spec should contain one or more expectations , that test the state of the code . A spec whose expectations succeed will be passing and a spec with any failures will fail . The name `it` is a pronoun for the test target , it is not an abbreviation of anything . it.makes the spec more readable by connecting the function name it and the argument description as a complete sentence . the second parameter tand the third parameter are optional .
- The `expect()` function in jasmine is used instead of the if-statements , it creates an expectation for a spec , this fucntion returns object .

```
describe('test suite: formatCurrency', () =>{
    it('converts cents into dollars' , () =>{
        expect(formatCurrency(2095).toEqual('20.95'));
    });
});
```



Side Notes :

- **Test Coverage** refers to how much of the code is being tested. The best practice is to maximize test coverage.
- A flaky test is one that sometimes passes and sometimes fails.
- Mocks allow us to replace a method with a fake version. To create a mock, we can use the `spyOn()` function. `spyOn()` records every time a method is used. A mock lasts only for one test; once the test is finished, the method is no longer mocked

```
  expect(cart.length).toEqual(1);
  expect(localStorage.setItem).toHadBeenCalledTimes();
);
```

This method checks how many time localStorag.setItem is called : toHaveBeenCalledManyTimes(number)

Unit and integration Test :

- Unit tests is testing one piece of the code
- An integration test meaning testing many pieces , , like testing a whole page or a part of the page , test many pieces of code working together.
- When we test a page we need to test : how the page looks , and how the page behaves
- Sometimes when using test , we are testing in a different folder than the folder we are in , in our case we want to have an integration test for the orderSummary() , orderSummar function represents ap art of our page and therefore too many pieces of code working together , to make this kind of test we need to test how tha page looks + how it behaves . in this case we will need to have an element in our testing page .

```
expect(document.querySelectorAll('.js-cart-item-container'
        expect(document.querySelector(`.js-product-quantit
```

```
            .innerText).toContain('Quantity : 2')
```

Hooks :

Hooks are shortcuts used in Jasmine. A hook is a piece of code that runs before each test. To implement this, we use the `beforeEach(() => {})` function. However, we should be aware of the variables we define within this function. Any variable or constant defined in the `beforeEach` function will only be accessible within that function due to scope limitations.

- `beforeEach()`: This function runs the specified code before each individual test is executed. It is useful for setting up a consistent state before every test runs.
- `afterEach()`: This function runs the specified code after each individual test has completed. It is often used for cleanup tasks to ensure that each test does not interfere with others.
- `beforeAll()`: This function runs the specified code once before all the tests in a suite are executed. It is commonly used for setup tasks that only need to happen once, such as initializing resources.
- `afterAll()`: This function runs the specified code once after all tests in a suite have completed. It is typically used for cleanup tasks, such as closing database connections or clearing temporary files.

Lesson 17 : OOP

Object-Oriented programming :

Object oriented Programmig (OOP) is another style of programming , it is basically organising our code into objects . However Procedural programming is a step-by-step instructions , in **procedural programming** we organise our code into separate functions while in **OOP** everyting even the function are inside the object.

In **object-oriented programming**, we can create multiple objects, but excessive copy-pasting of code can lead to messy and unmanageable code. This redundancy can make maintenance difficult, as any changes would need to be replicated across all instances, increasing the risk of errors and inconsistencies.

In

object-oriented projects, it's recommended to use **PascalCase** for naming conventions. **PascalCase** involves capitalizing the first letter of each word in a name without spaces (e.g., `MyObject`, `UserAccount`). This convention is typically used for class names and constructors, helping to distinguish them from variables and functions, which usually use **camelCase** or **snake_case**. By following naming conventions like **PascalCase**, we improve code readability and maintainability, making it easier for developers to understand the structure and purpose of the code.

Classes :

A class serves as a blueprint for objects. It defines properties (attributes) and behaviors (methods) that the objects created from it will have. For example, if you have a class called `Car`, it might define properties like `color`, `model`, and methods like `drive()` or `stop()` , When you create an object from a class, that specific object is referred to as an instance of the class.

```
// To check if an object is an instance of a class
console.log(businessCart instanceof Cart);
```

Classes in Object-Oriented Programming (OOP) have special features such as **constructors**. A constructor is a special method that allows us to include setup code inside the class.

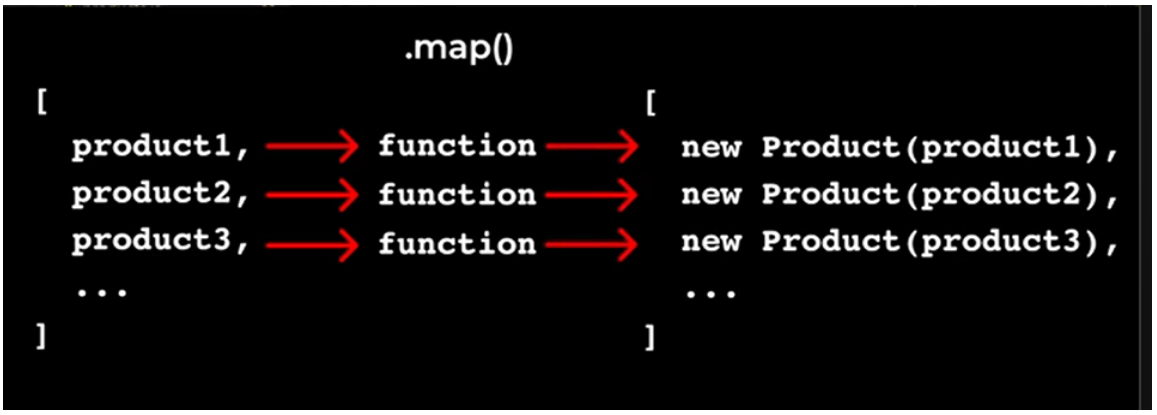
- The main difference between a constructor and other methods is that the **constructor is automatically called** when an object is created from the class. This means you don't have to manually call the constructor—it runs immediately after an object is instantiated.
- **Constructors can also have parameters**, allowing you to pass initial values when creating an object.
- The **constructor method** must always be named `constructor`, and it should not return anything (constructors implicitly return the new object instance they are creating

Private properities and methods :

In JavaScript, properties and methods can be made private by prefixing them with `#`, restricting access to within the class. This enforces encapsulation, ensuring that internal details are hidden and only accessible through public methods. If you attempt to access a private property or method from outside the class, it results in an error, protecting the internal state and functionality from accidental misuse.

Converting an object into a class :

- `map()` loops through an array and runs a function for each value. It is an array method that creates a new array, and whatever is returned by the function will be placed inside the new array.

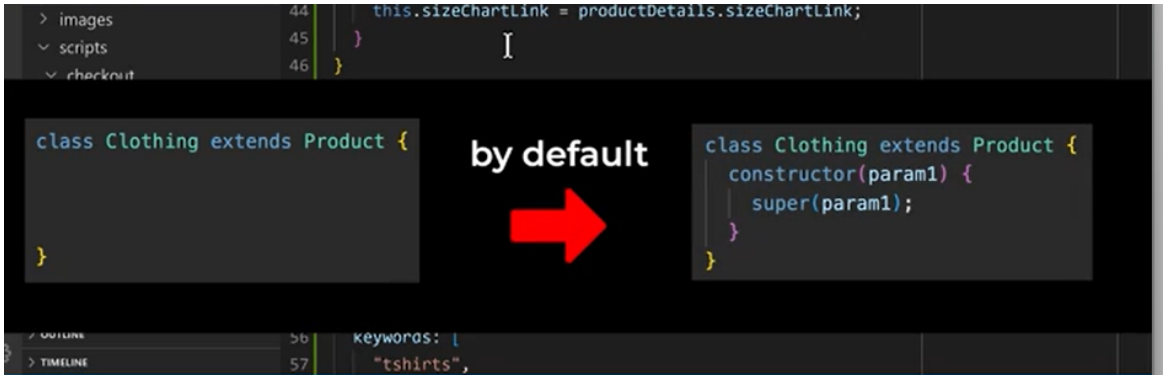


Inheritance :

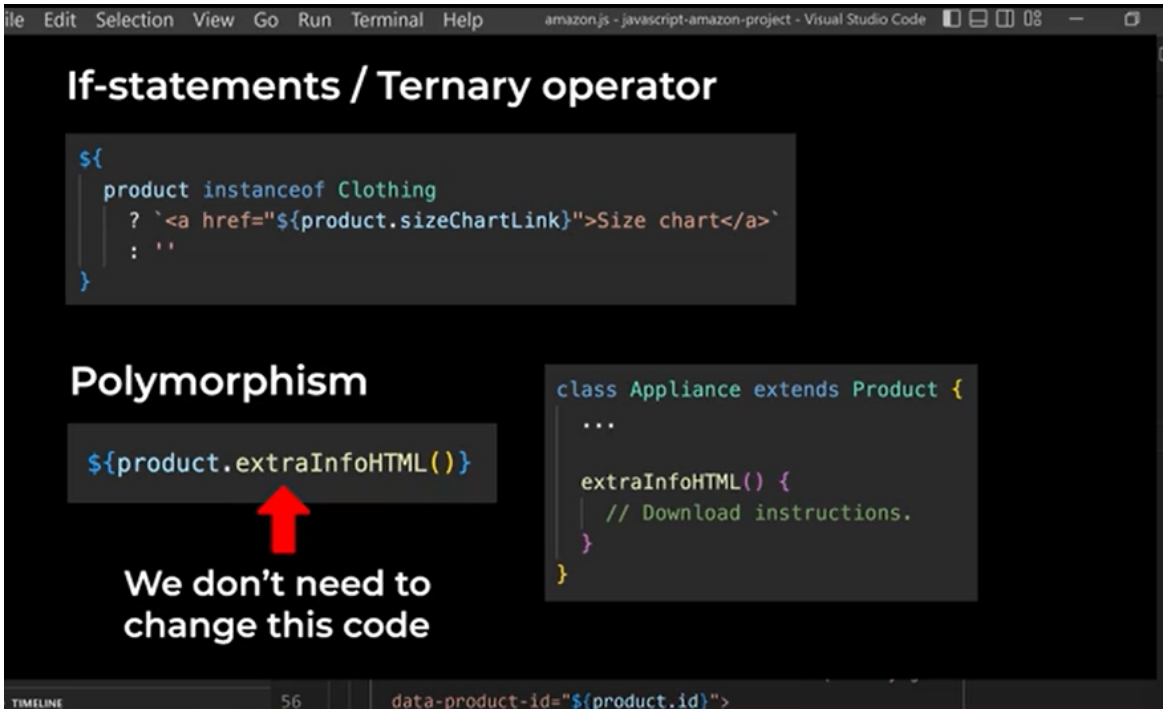
Inheritance means a class inherits certain properties (variables) and methods from another class, creating a parent-child relationship between these classes. In the example below, `Clothing` is the child class, which inherits from the parent class, `Product`. Inheritance allows us to reuse code between classes, reducing duplication. The `super()` method is used to call the parent class's constructor, helping to avoid redundancy by initializing the inherited properties.

```
class Clothing extends Product {
    sizeChartLink;
    constructor(productDetails){
        super(productDetails);
        this.sizeChartLink = productDetails
    }
}
```

- If we don’t create a constructor for the child class, the parent class's constructor is automatically run by default. This happens because the child class inherits the same properties and methods from the parent class without needing to explicitly define a constructor



- **Method Override**: Method overriding occurs when a method is defined in a child class that has the same name as a method in its parent class. The child class's method replaces the parent class's method when invoked on an object of the child class. However, if we still need to call the parent class's method within the overridden method, we can use the `super()` function. This allows access to the parent class's method, preventing complete replacement. The overridden method does not completely remove the parent method, but it takes precedence when called from the child.
- **Using `super()`**: The `super()` function can be used in two main ways. First, it can be used to call the constructor of the parent class from within the child class's constructor. This is essential when the parent class's constructor needs to be executed to properly initialize the inherited properties. Secondly, `super.methodName()` allows us to call a specific method from the parent class, rather than the constructor. This is useful when you want to reuse functionality from the parent class while extending or modifying it in the child class
- **Polymorphism**: Polymorphism refers to the ability to use the same method in different contexts, regardless of whether it belongs to the parent class or the child class. This allows for dynamic method binding, where the method call can be directed to the method in the parent class or the child class depending on the actual object instance. The idea behind polymorphism is that a parent class reference can refer to an object of a child class, and the method that gets executed depends on which class the object belongs to at runtime. This provides flexibility in how objects and methods are used in object-oriented programming



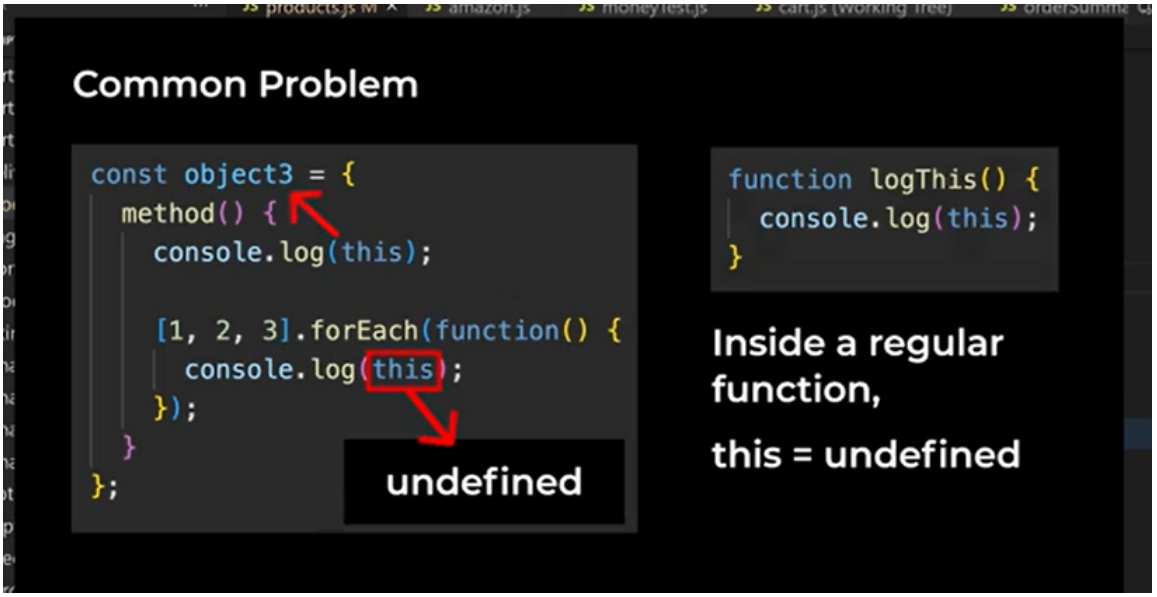
More details about classes :

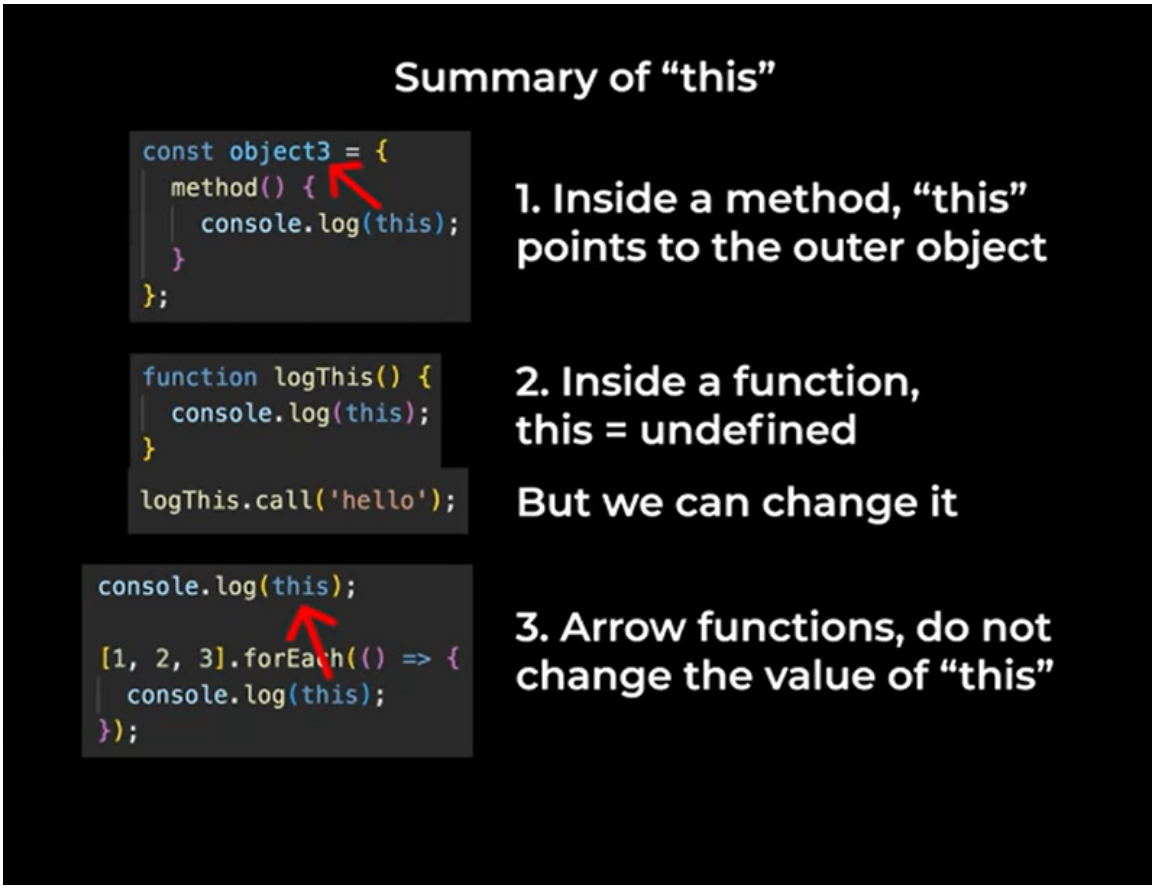
- **How to Test Classes**: To test a class, we create an instance (object) from the class and then verify whether its properties and methods function correctly.
- A built-in class is a class provided by the programming language. An example of a built-in class is:
  - `new Date()`: This generates an object that represents the current date and time. Every date object has a method called `.toLocaleTimeString()`, which returns the current time formatted as `H:m:s`. Additionally, libraries like Day.js utilize this built-in `Date` class under the hood to handle dates and times.

More details about 'this ' :

- The `this` keyword allows an object to access its own properties.
- In JavaScript, `this` can be used anywhere in your code.
- Using `console.log(this)` in the global context will typically output `Window` in a browser or `global` in Node.js, not `undefined`.
- Functions have a method called `.call()`, which executes the function and allows you to specify the value of `this` within that function. This method is similar to calling the function normally, but the main difference is that it can take an additional parameter that precedes other parameters, allowing you to pass context explicitly.

- Arrow functions do not change the value of `this`; instead, they lexically bind `this` from the surrounding context. This means that the value of `this` inside an arrow function is the same as it was outside the function.





Lesson 18 : Backend

- A backend is a server that manages the data for our projects or websites, often using technologies like databases to store and retrieve information. To send information from the frontend to the backend, we use a protocol called HTTP (HyperText Transfer Protocol). An HTTP request can carry various types of information, such as data sent to be processed or retrieved from the server.
- To facilitate this communication, we use a built-in object called `XMLHttpRequest`. This object allows us to create and send asynchronous requests to the server without having to reload the entire web page, enabling a smoother user experience. Additionally, with modern JavaScript, the Fetch API is often used as a more powerful and flexible alternative to `XMLHttpRequest`.

  ```
  new XMLHttpRequest();
  //Creates a new HTTP meassage to send to the backend. mess
  ```

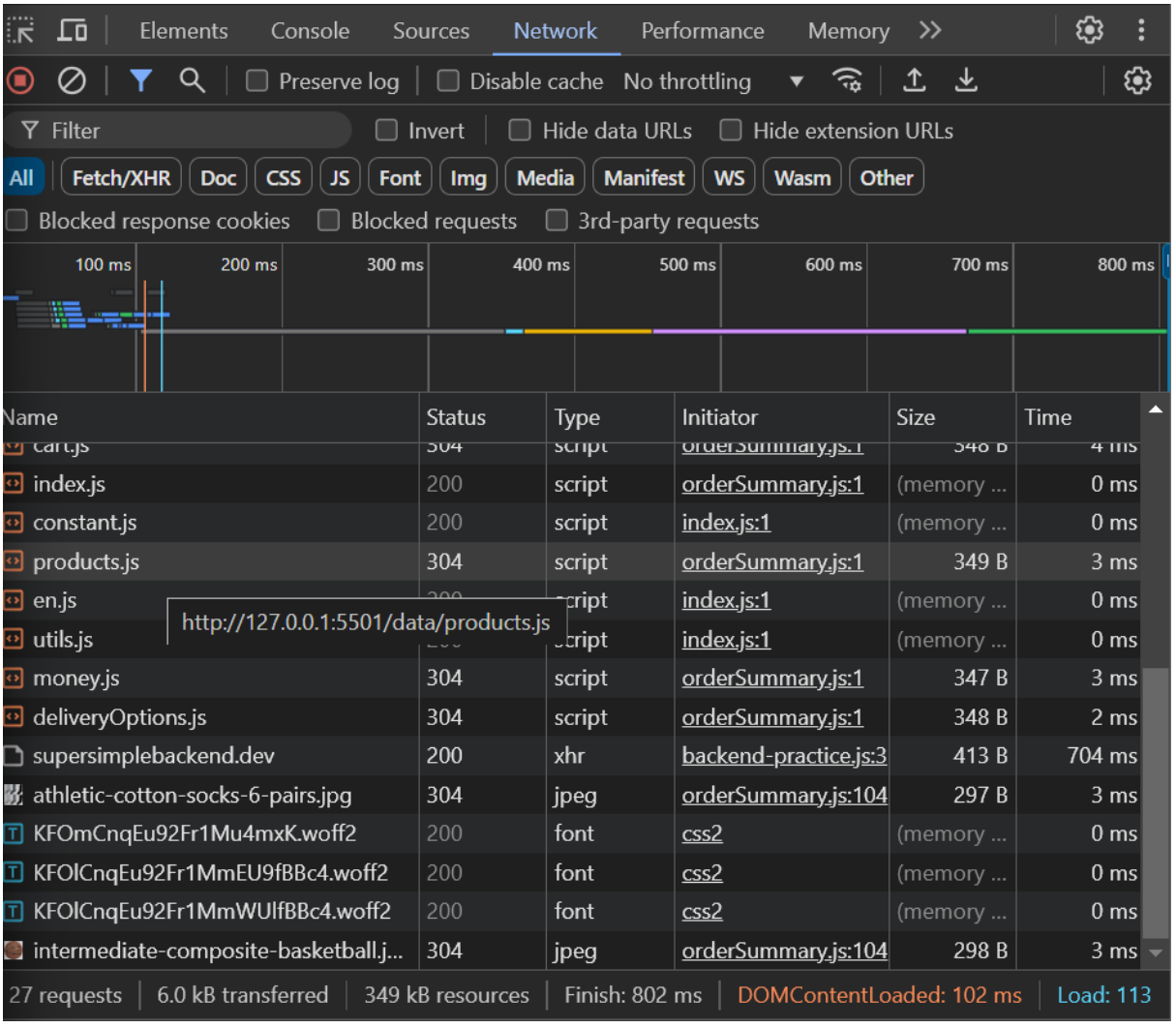
  To set up a request, we use the `open` method, which takes two parameters. The first parameter specifies the type of request or HTTP method (e.g., `GET`, `POST`, `PUT`,

`DELETE`). The second parameter indicates the URL to which the HTTP request will be sent. This URL can point to any server connected to the internet.

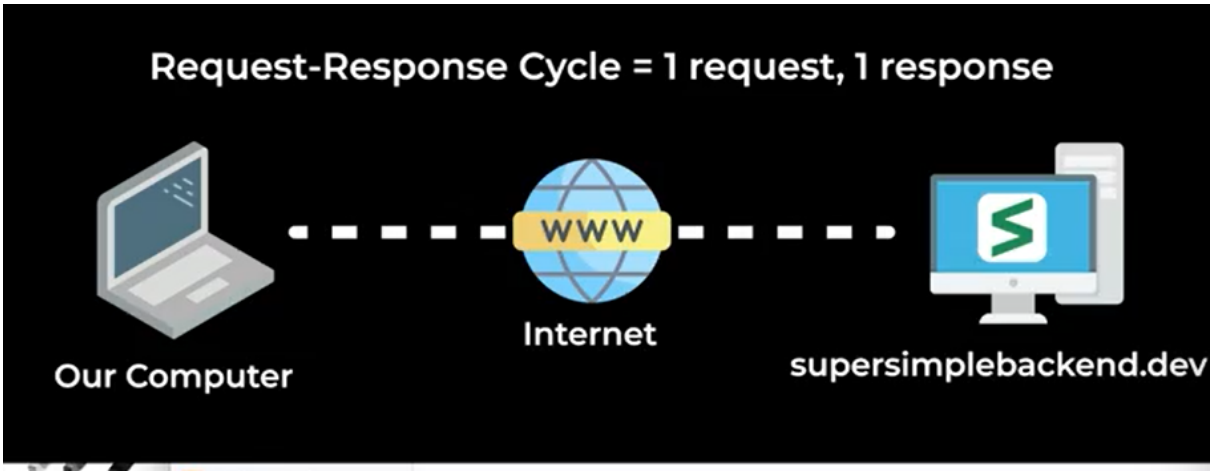
Urls [Uniform ressorce Locator]

A URL (Uniform Resource Locator) is like an address but for the internet. It helps locate resources (such as websites, files, or services) on another computer or server connected to the internet. A URL specifies the location of the resource and can include information about how to access it, such as the protocol (e.g., `http`, `https`) and the specific path to the resource.

Example : https://amazon.com → https : we are using http to communicate with this computer . [amazon.com] is a domain name itpoints to a backend amazon computer .



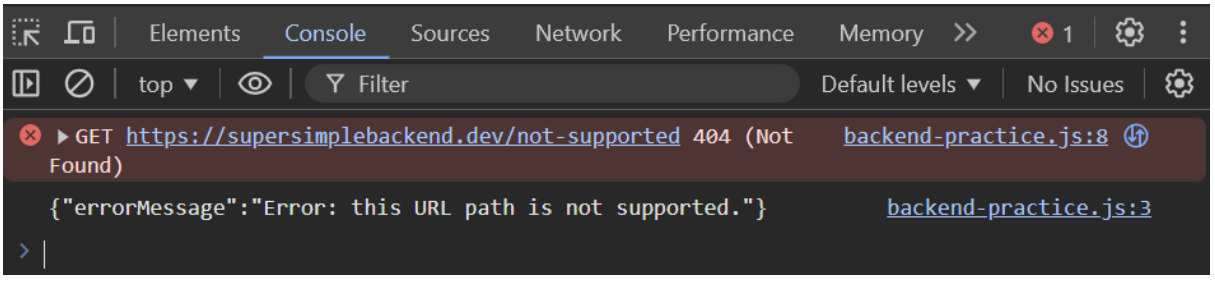
In the network tab we can see all of the http messages that we get and we send



It sometimes takes time to receive a response from the server, so to handle this delay, we can use `xhr.addEventListener()`. This method takes two parameters. The first parameter is the event type (e.g., `'load'`), which refers to when the request has successfully completed. The second parameter is a callback function that will execute when the event occurs. Before receiving the response, `xhr.response` will be `undefined`.

```
xhr.addEventListener('load',function(){
    console.log(xhr.response);
});
```

URL paths refer to the portion of the URL that comes after the domain name. These paths direct the request to specific resources on the server. For example, in the URL `https://www.example.com/api/data`, the path is `/api/data`. It is important to note that not all url Paths are supported , A backend only supports a specific set of URL paths, depending on what has been programmed on the server. If a request is made to a URL path that the backend does not recognize, the server will return an error, often a 404 (Not Found) or 400 (Bad Request).



Here’s the improved version of your content, incorporating the additional notes seamlessly into the main paragraphs:

The number **404** is a **status code**, which indicates the result of a request to the server. Status codes starting with 4 (like 404) indicate a **client-side issue**—a problem with the request (e.g., a page not found). Status codes starting with 5 (like 500) indicate a **server-side issue**, such as the backend crashing. On the other hand, status codes that start with 2 (like 200) indicate that the request was **successful**.

To know which
URLs (or paths) are supported by a backend, we usually cannot have access to all paths due to security reasons. However, some backend systems provide documentation or specific pages that list the **supported paths**. These paths form part of the **backend API** (Application Programming Interface), which defines how we can interact with the backend. An **API** essentially serves as an interface that tells us which functions or endpoints are available and how to use them.

When making requests to a backend, it can respond with different types of data, such as **text**, **JSON**, **HTML**, or **images**. The format of the response depends on what the backend is designed to return based on the URL path.

For instance, when you type a **URL** into your browser, the browser sends a **GET request** to the server. If the request is successful, the server responds with the requested resource (such as a webpage or data), which is then displayed on the browser. If you make the same GET request from a tool like **VSCode** using something like `XMLHttpRequest`, the response is received programmatically, often displayed via `console.log(xhr.response)` or examined in the **network tab** in the browser's developer tools. This tab is especially useful for monitoring the status codes, request times, and the returned data for each request.

Callbacks :

- A **callback** is a function passed as an argument to another function and is intended to be executed later, typically after some operation is completed.
- We've already used callbacks with the `setTimeout()` function. When we pass two parameters to `setTimeout()`, the first parameter is a function (the callback), and the second parameter is the delay (in milliseconds) after which we want the callback to be executed.
- Although slightly out of context, it's important to note that the `setTimeout()` function returns an ID. This ID can be useful if we want to cancel the timeout using the `clearTimeout()` function.

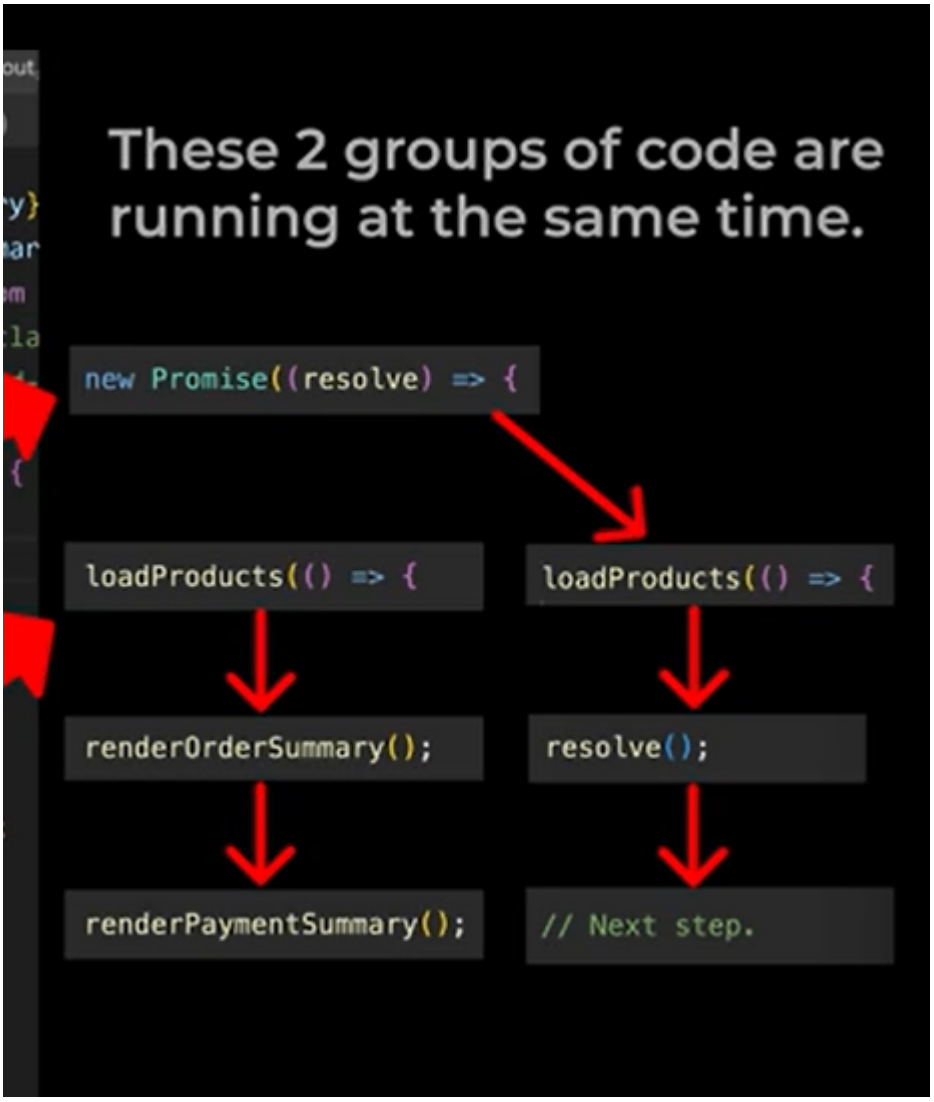
Testing with backend :

Jasmine provides a function called `done()` that is used in asynchronous tests. It allows Jasmine to wait for the asynchronous code to complete before proceeding with the test assertions. This is particularly useful when testing code t hat involves timers, API calls, or any other non-blocking operations. Without calling `done()`, Jasmine would assume the test has finished and might fail prematurely.

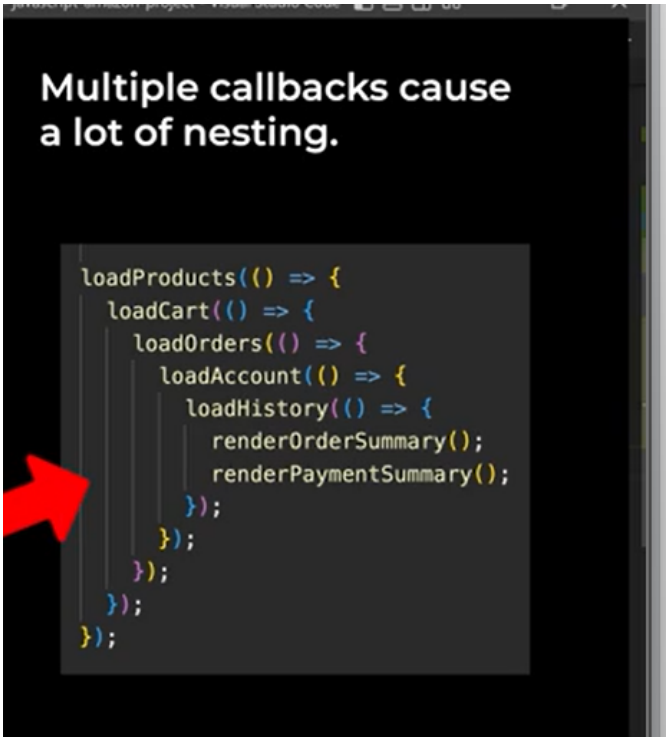
Promises :

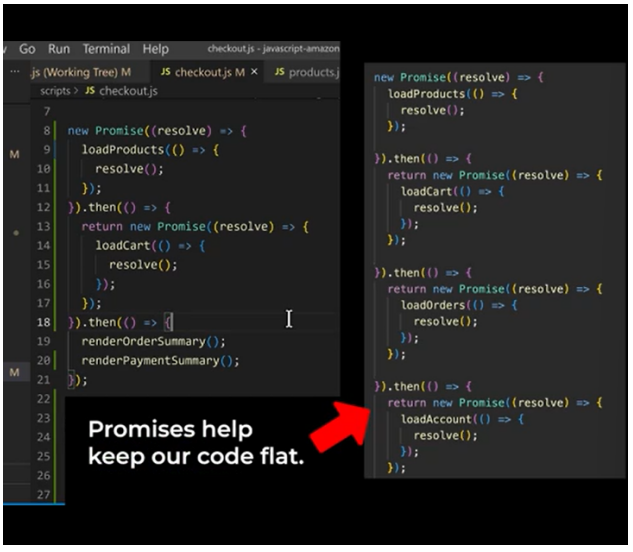
Promises are similar to callbacks, but they offer a more powerful and flexible way to handle asynchronous code. Like the `done()` function in Jasmine, promises allow us to wait for some code to finish before moving on to the next step, but they also provide better error handling and chaining mechanisms.

- When we create a **promise**, it runs the inner function immediately. This inner function takes two parameters: `resolve` and `reject`.
- `resolve` is similar to the `done()` function in Jasmine. It allows us to control when to proceed from one step to another. It's called when the asynchronous operation is successful.
- **Reject** is used to handle errors or failures in the asynchronous operation.



Callback Hell: Using multiple nested callbacks can lead to "callback hell," which is difficult to read and maintain due to deep nesting. Promises provide a cleaner alternative, making the code easier to read and understand by flattening the structure. With promises, you can chain operations using `.then()` methods, allowing for a more linear flow of code.

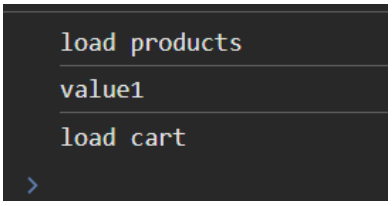




- Instead of using `.then()`, which allows us to execute the next step in a promise chain, we **cannot use the** `resolve` **function directly** outside of the context in which the promise was created. The `resolve` function is meant to be called within the executor function of the promise to indicate that the asynchronous operation has completed successfully. To achieve subsequent actions, we typically utilize the `.then()` method to handle the resolved value and execute further steps.
- We can **share parameters between promises** using the `resolve` function. By passing a value to the `resolve` function when fulfilling a promise, that value can be accessed in the

next `.then()` call. This allows us to pass data through the promise chain easily.

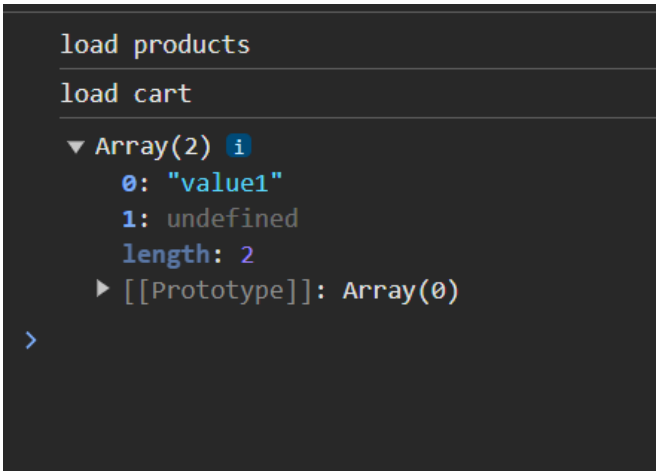
```
new Promise((resolve) => {
    loadProducts(() => {
        resolve('value1');
    });
}).then((value) => {
    console.log(value);
    return new Promise((resolve) => {
        loadCart(() => {
            resolve();
        });
    });
}).then(() => {
    renderOrderSummary();
    renderPaymentSummary();
});
```



- `Promise.all()` , lets us run multiple promises a the same time and waits for all of them to finish . we give `Promise.all()` an array of promises . we can always share values and they aer going to be in an array of values

```
Promise.all([
    new Promise((resolve) => {
        loadProducts(() => {
            resolve('value1');
        });
    }),
    new Promise((resolve) => {
        loadCart(() => {
            resolve();
        });
```

```
    })
]).then((values) => {
    console.log(values);
    renderOrderSummary();
    renderPaymentSummary();
});
```



Fetch :

- `fetch()` is a modern and more powerful way to make HTTP requests. By default, `fetch()` makes a GET request and only requires the URL to which we want to send the request.
- When we call `fetch()`, it creates a promise that resolves to the Response object representing the completion of the request. We can chain `.then()` to handle the next steps in the promise chain.
- `fetch()` sends a request to the backend URL and waits for a response. To access the response, we add a parameter inside the `.then()` function to use the response object.
- To retrieve data from the response (which corresponds to the HTTP request sent to the backend), we use `response.json()`. This method is asynchronous and returns a promise, which resolves to the result of parsing the body text as JSON. It's important to handle this promise to access the data.

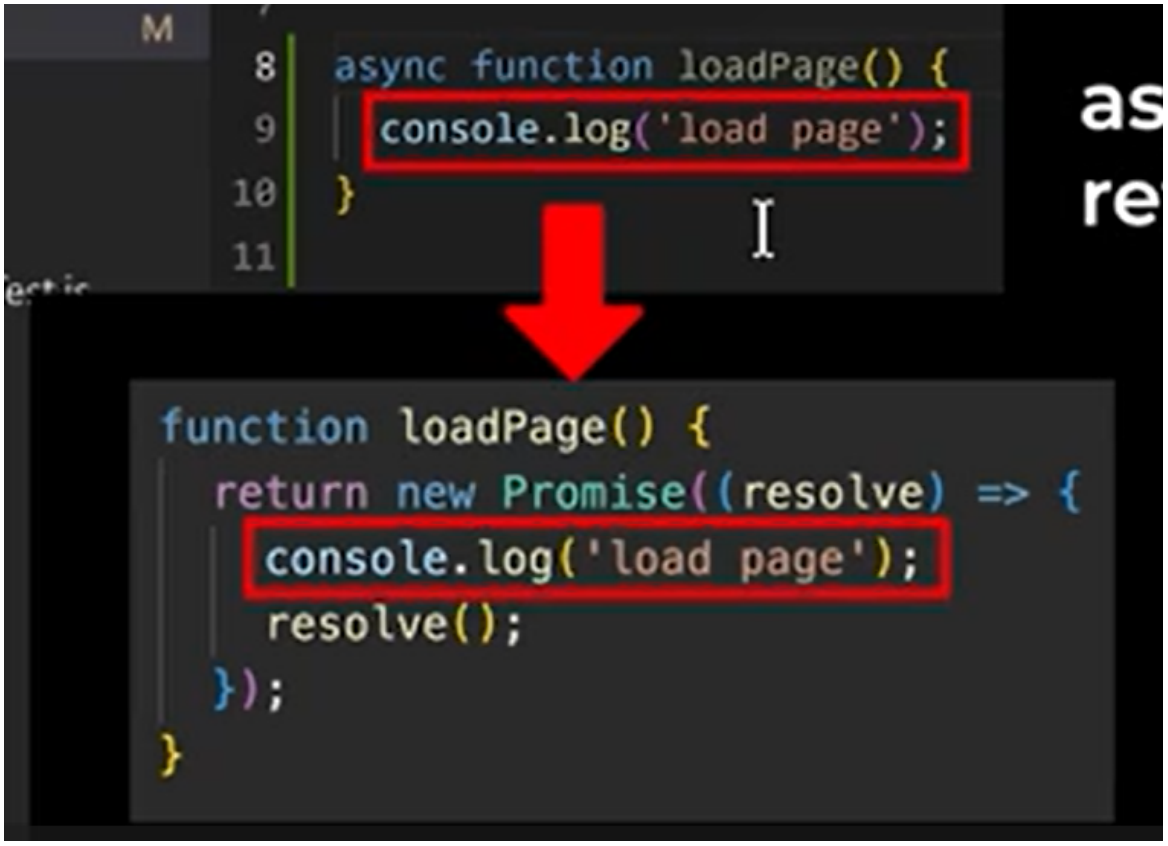
```
export let products = [];

export function loadProductsFetch() {
```

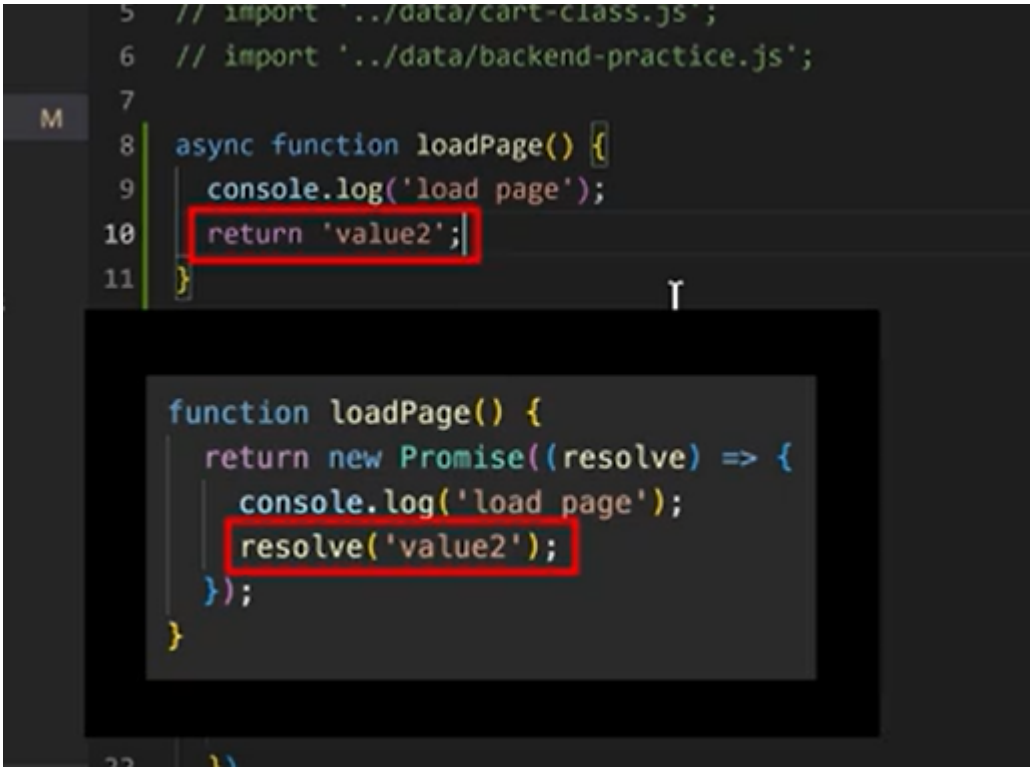
```
  const promise = fetch('https://supersimplebackend.dev/pr
    .then((response) => {
      return response.json(); // gives the data attached t
      // response.json() is Asynchronous, it returns a pro
    })
    .then((productsData) => {
      products = productsData.map((productDetails) => {
        if (productDetails.type === 'clothing') {
          return new Clothing(productDetails);
        }
        return new Product(productDetails);
      });

      console.log('load products');
    });

  return promise;
}
```

Async Await :

A more efficient approach to handling asynchronous code is by using the `async` and `await` keywords, which make working with promises more straightforward. When we add `async` in front of a function, it ensures that the function returns a promise. If we return a value inside the function, it automatically becomes resolved, allowing us to catch that value in the subsequent `.then()` method, just like with regular promises.

.





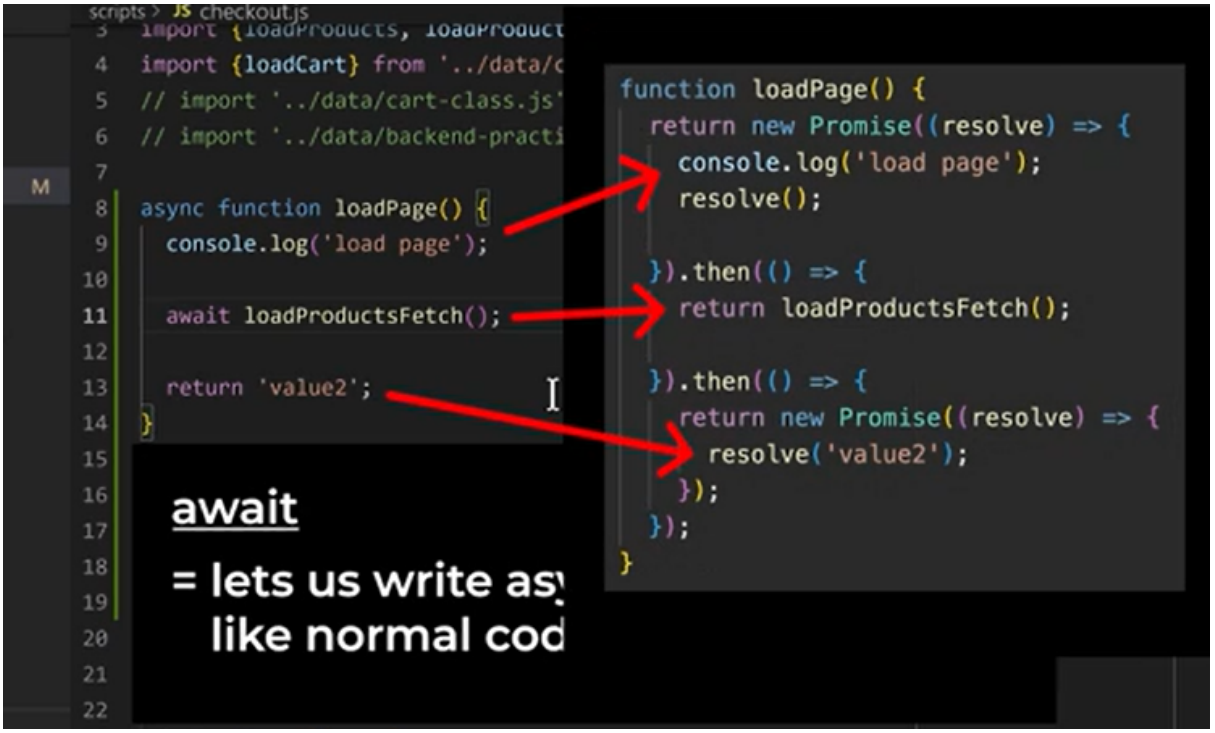
The

`async` keyword allows us to use `await`, which pauses the execution of the function until the

promise is resolved. This provides another way to wait for a promise to finish before moving to the next line of code. While we can already achieve this with the `.then()` method, `await` offers a cleaner, more readable syntax, making asynchronous code look more like synchronous code.

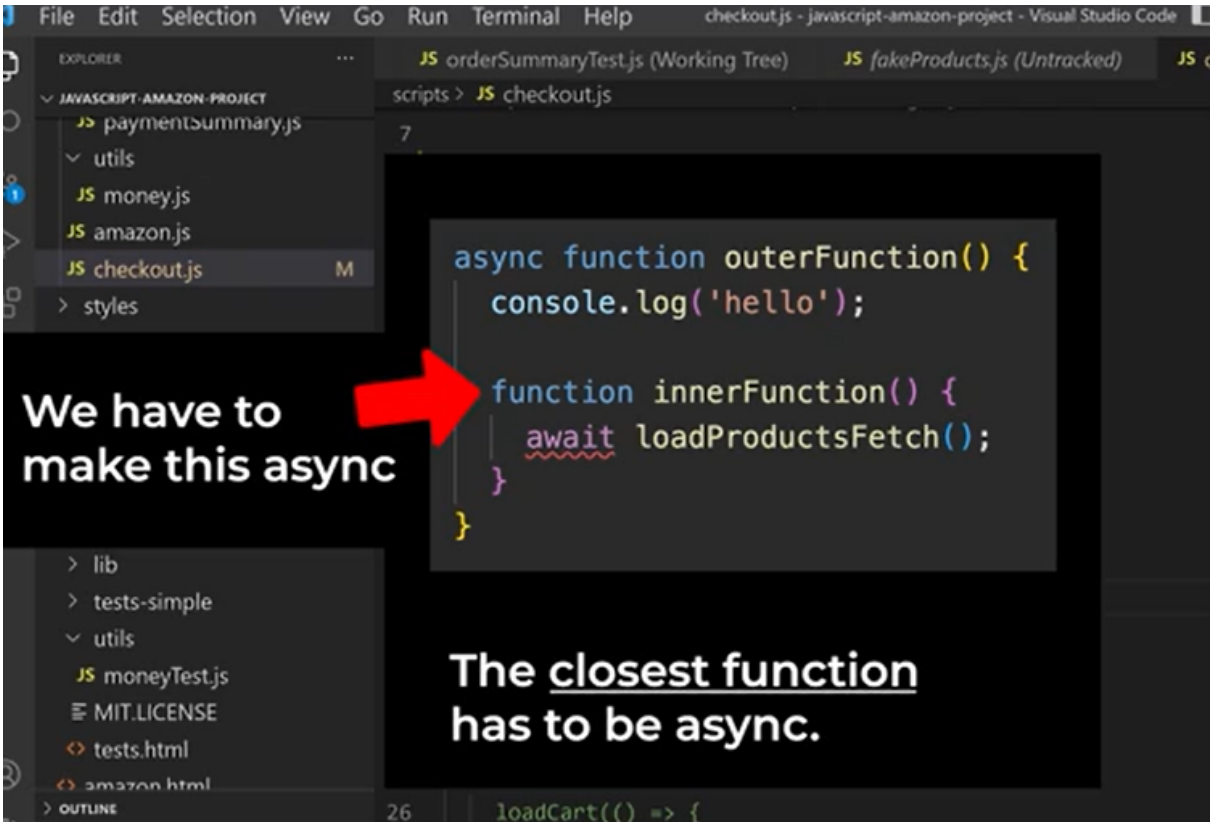
It's important to note that
`await` can only be used inside an `async` function, and it simplifies the structure of our code by eliminating the need for chaining `.then()` calls. This results in more readable and maintainable asynchronous operations.

```
loadProductsFetch().then(()=>{});
//or
await loadProductsFetch();
```



When using a `resolve` function inside an `await` function, there's no need to catch the value being passed using `.then()`. This is because the `await` function automatically returns the resolved value. As a result, we can directly assign this value to a variable if needed.

This means that `await` simplifies the process by eliminating the necessity of chaining `.then()` calls. Instead, we can store the resolved value in a variable and use it as part of the normal function flow, making the code more readable and easier to manage.



Error Handling :

Handle errors in callbacks :

For callbacks we usually set a separate callback just for errors .

```
async function exampleFunction() {
  // Simulating a promise that resolves a value
  const promise = new Promise((resolve) => {
    setTimeout(() => resolve('Resolved value'), 1000);
  });

  // Using await to get the resolved value without .then()
  const result = await promise;

  // Now we can use the result as a variable
  console.log(result); // Outputs: Resolved value
}

exampleFunction();
```

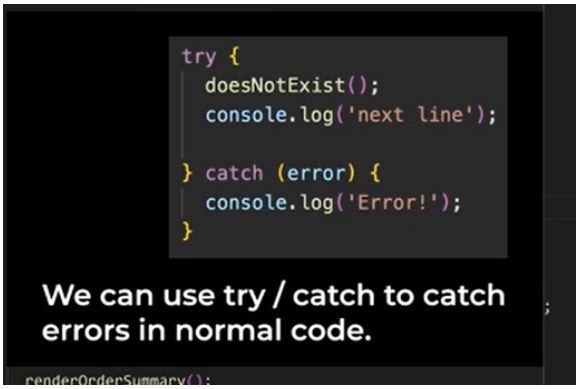
Handle errors in promises:

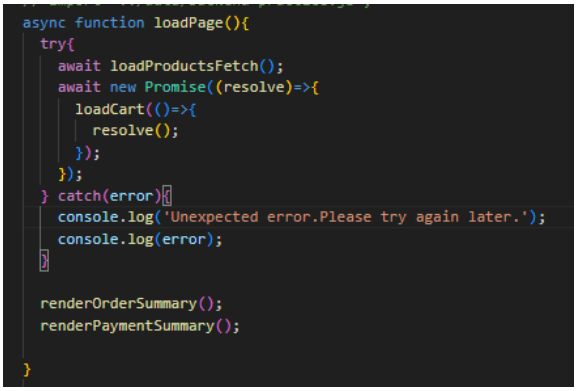
- We can handle errror in promises by two ways , using `then()` or using the method .`catch()`
- Just like with callbacks, you can always have a parameter inside the `.catch()` function called `error`, in case you want to display more information about the error. A simple example of how you can handle errors using the `.catch()` function is shown in the code below

```
promise.catch((error)=>{
    console.log(error);
})
```

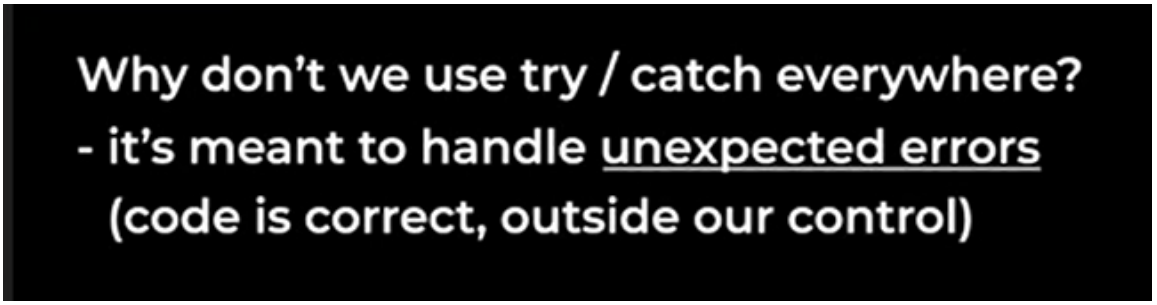
Handle errors in async await :

- A new syntax is used here , called `try/catch` , `try/catch` can be use outside async wait , it can be used with normal synchrounous code .





- When using `try/catch`, if an error occurs in the `try` block, the code execution will skip to the `catch` block, bypassing any subsequent code in the `try` block.



Manually creating errors :

If we use promises and want to manually create an error, we can do so in two ways:

- We can use the `throw` statement inside the promise. This is used to create an error synchronously within the promise.
- If we want to create an error in the future, we need to use a different approach. We can use the `reject()` function, which is the second parameter in the promise executor function. This function allows us to create an error asynchronously.

Intro to backEnd :

To send data in a request, we use different types of HTTP methods. There are four common types of requests: `GET`, `POST`, `PUT`, and `DELETE`.

- `GET`: This method retrieves data from the backend but does not send data.
- `POST`: This method sends data to the backend, typically to create a new resource.
- `PUT`: Similar to `POST`, but it's usually used to update an existing resource.

- `DELETE`: This method is used to remove a resource from the server.

```
document.querySelector('.js-place-order').addEventListener('c
  const response = await fetch('https://supersimplebackend.de
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      cart: cartItems // Assuming cartItems is an array or ob
    })
  });

  const order = await response.json();
  console.log(order); // Logs the response (order details)
});
```

When sending data using `fetch()`, you need to provide a second parameter, which is an object. This object includes several important options:

- `method`: Specifies the type of request (e.g., `'POST'` for sending data).
- `headers`: An object that provides additional information about the request, such as the content type (e.g., `'Content-Type': 'application/json'`).
- `body`: Contains the actual content you want to send to the server. If you're sending an object, you should first convert it to a JSON string using `JSON.stringify()`.

```
> localStorage.getItem('orders')
'[{"id":"f3246a79-a796-4c43-992d-f15e1d239c50","orderTime":"2024-09-02T19:09:27.551Z","t
otalCostCents":9860,"products":[{"productId":"e43638ce-6aa0-4b85-b27f-e1d07eb678c6","qua
ntity":2,"estimatedDeliveryTime":"2024-09-05T19:09:27.550Z","variation":null},{"productI
d":"15b6fc6f-327a-4ec4-896f-486349e85a3d","quantity":3,"estimatedDeliveryTime":"2024-09-
09T19:09:27.551Z","variation":null}]}]'
> localStorage.removeItem('orders')
undefined
```

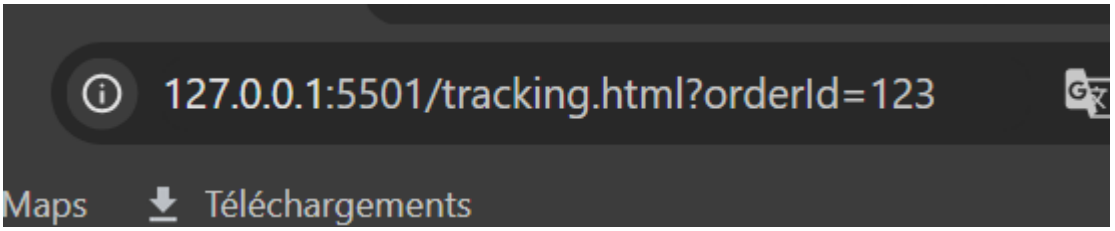
The
`.unshift()` method is a built-in function in JavaScript that is used with arrays to add one or more elements to the beginning of the array. This method modifies the original array and

returns the new length of the array after the elements have been added.

The `window.location.href` property is used to get or set the entire URL of the current page displayed in the browser's address bar. When accessed, it retrieves the complete URL as a string, including the protocol (e.g., `http` or `https`), the domain, the path, and any query parameters

URL Parameters :

URL parameters allow us to include data directly within the URL, enabling the transmission of information between web pages or applications. This is especially useful for sharing state, filters, or specific identifiers in a query string format. URL parameters are appended to the end of a URL after a question mark (`?`), with each parameter represented as a key-value pair separated by an ampersand (`&`).

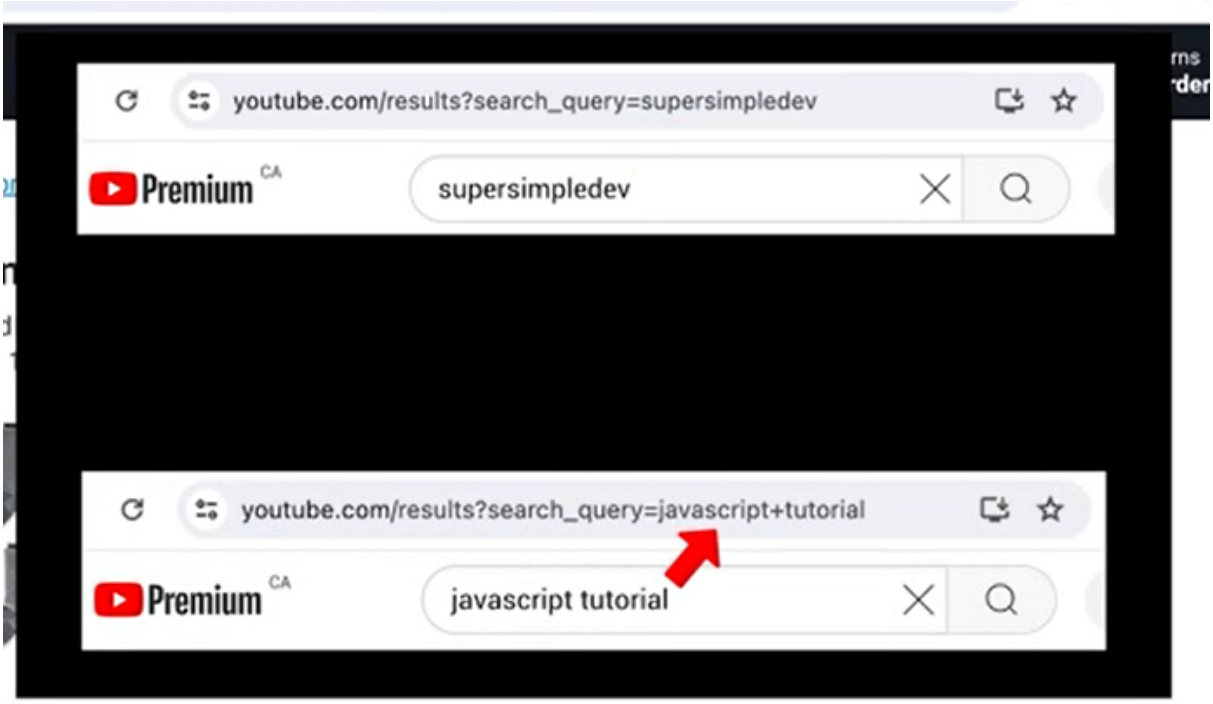


The `URL()` class is a built-in JavaScript class that allows us to work with URLs easily. To use it, we need to create an instance of the `URL` class and pass the URL from which we want to extract values.

URL parameters, often referred to as search parameters, are the key-value pairs that follow the question mark (`?`) in a URL. They provide additional information to the server.

To retrieve URL parameters, we use the `searchParams` property of the `URL` object. This property contains a `URLSearchParams` object that provides various methods for working with search parameters. One commonly used method is `.get(parameterName)`, which allows us to retrieve the value of a specific parameter by its name.

```
const url = new URL(window.location.href);
console.log(url.searchParams.get('orderId'));
```


youtube.com/results?search_query=supersimpledev
Premium CA
supersimpledev
youtube.com/results?search_query=javascript+tutorial
Premium CA
javascript tutorial


What's the next step after this course?
Learn how to create our own backend.
1. Command Line
2. NodeJS = create our own backend
node JS
Also after this course,
learn how to put a website
on the Internet
(links in description)
Put a Website Online
WWW
29:37
How to put an HTML website online (on the Internet)