Where and how can I use the destructuring assignment in JavaScript? 🕵️🔥  
The destructuring syntax allows you to quickly get values out of arrays and objects into distinct variables. While it might seem scary at first, it's effortless to learn and use. 👋  
  
🕵️ Array destructuring  
Destructuring an array is very straightforward. All you have to do is declare a var for each value in the series. You can specify fewer vars than indexes in the array, skip some indexes or even use the rest pattern to unwrap any remaining values into a new collection.  
  
🕵️ Object destructuring  
It is a useful feature to extract properties from objects and bind them to vars. What's better, you can pull multiple properties in one statement, can access properties from nested objects, and can set a default value if the property doesn't exist.  
  
🕵️ Nested destructuring  
Nested objects and arrays can be restructured by following the same rules. The difference is that you can unpack nested keys or values directly to vars without storing the parent obj is a variable itself.  
  
🕵️ Advanced destructuring  
As arrays act much like objects, it's possible to use the destructuring syntax to get specific values from an arr by using the index as a key in an object destructuring assignment.

𝟱 𝗘𝘅𝘁𝗿𝗲𝗺𝗲𝗹𝘆 𝗣𝗼𝘄𝗲𝗿𝗳𝘂𝗹 𝗝𝗮𝘃𝗮𝗦𝗰𝗿𝗶𝗽𝘁 𝗧𝗿𝗶𝗰𝗸𝘀 𝘁𝗵𝗮𝘁 𝗬𝗼𝘂 𝗦𝗵𝗼𝘂𝗹𝗱 𝗞𝗻𝗼𝘄 🎯🔥  
This post shows you how to resolve everyday JavaScript tasks by writing less code and using some more straightforward tricks. Those some of the “techniques” that I usually use in my day-to-day work. 💃  
  
🚀 𝗴𝗲𝘁𝗢𝗯𝗷𝗕𝘆𝗖𝗼𝗻𝗱𝗶𝘁𝗶𝗼𝗻  
Returns the elements that exist in both arrays, after using the provided function to each array element of both. Create a Set by applying fn to all elements in "b". Use .filter() on "a" to only keep elements, which produce values contained in "b" when fn is applied to them. 😈  
  
🚀 𝗮𝘃𝗲𝗿𝗮𝗴𝗲𝗕𝘆𝗖𝗼𝗻𝗱𝗶a𝘁𝗶𝗼𝗻  
Calculates the average of an array, after mapping each element to a value using the provided function.  
  
🚀 𝘀𝘂𝗺𝗕𝘆𝗖𝗼𝗻𝗱𝗶𝘁𝗶𝗼𝗻  
This function calculates the sum of an array, after mapping each element to a value using the provided function.  
  
🚀 𝗺𝗮𝘅𝗕𝘆𝗖𝗼𝗻𝗱𝗶𝘁𝗶𝗼𝗻  
Returns the max value of an array, after mapping each element to a value using the provided function. Math.max() to get the maximum value. 🎅  
  
🚀 𝗺𝗶𝗻𝘅𝗕𝘆𝗖𝗼𝗻𝗱𝗶𝘁𝗶𝗼𝗻  
Returns the min value of an array, after mapping each element to a value using the provided function. Math.min() to get the minimum value

5 JavaScript Code Snippets to Solve Your Everyday Problems 💃🔥  
With JS changing all the time, it's important to create reuseable methods to saving our time.JS can also enable you to do some awesome stuff using just one single line.😎  
  
💃 𝗶𝘀𝗔𝗹𝗹  
This example returns true if the function returns true for all elements in arr and false otherwise. This snippet is one example I use daily and helps me solve a wide range of issues.  
  
💃 𝗴𝗲𝘁𝗜𝗻𝗱𝗲𝘅𝗲𝘀  
This code example can be used to get all indexes of a value in an array, which returns an empty array, in case this value is not included in it. By the way, one of my favorites, worth keeping it.🎓  
  
💃 𝗱𝗿𝗼𝗽𝗥𝗶𝗴𝗻𝘁𝗡 | 𝗗𝗿𝗼𝗽𝗟𝗲𝗳𝘁𝗡  
Those examples return a new array with N elements removed from the left/right.  
In plain English, I would recommend adding them to your arsenal, because you will definitely use them.💼  
  
💃 𝘀𝗺𝗮𝗹𝗹𝗲𝘀𝘁𝗡 | 𝗯𝗶𝗴𝗴𝗲𝘀𝘁𝗡  
It returns the n smallest/biggest elements from an array. If N is greater than or equal to the list’s length, then it will return the original array.  
  
💃 𝗿𝗮𝗻𝗱𝗼𝗺𝗡𝘂𝗺𝗯𝗲𝗿𝗜𝗻𝗥𝗮𝗻𝗴𝗲  
This code example can be used to generate a random integer in a specified range. It's really nice to do small tests, in a variety of variations.

4 JavaScript Shorthand Techniques that will save your time ⏰🔥  
Any programming language's shorthand techniques help you write more readable and optimized code and achieve your goal with less coding.  
  
⏳ 𝗢𝗯𝗷𝗲𝗰𝘁 𝗣𝗿𝗼𝗽𝗲𝗿𝘁𝘆 𝗔𝘀𝘀𝗶𝗴𝗻𝗺𝗲𝗻𝘁  
If the variable name and object key name are the same, we can mention the variable name in object literals instead of both key and value. JS will automatically set the key same as the variable name and assign it as a variable value.  
  
⏳ 𝗔𝘀𝘀𝗶𝗴𝗻 𝗮 𝗱𝗲𝗳𝗮𝘂𝗹𝘁 𝘃𝗮𝗹𝘂𝗲 𝘁𝗼 𝗮 𝘃𝗮𝗿𝗶𝗮𝗯𝗹𝗲  
We can use the "||" short circuit to assign a default value to a variable if the expected value is found falsy. This approach saves me much time, making my code much more readable.  
  
⏳ 𝗖𝗮𝗹𝗹𝗶𝗻𝗴 𝗮 𝗳𝘂𝗻𝗰𝘁𝗶𝗼𝗻 𝗼𝗻𝗹𝘆 𝗶𝗳 𝗮 𝘃𝗮𝗿𝗶𝗮𝗯𝗹𝗲 𝗶𝘀 𝘁𝗿𝘂𝗲  
If you want to call a function only if a variable is true, you can use the && short circuit as an alternative. This method is more useful to React devs when you want to render a component conditionally.  
  
⏳ 𝗔𝘀𝘀𝗶𝗴𝗻𝗶𝗻𝗴 𝘃𝗮𝗹𝘂𝗲𝘀 𝘁𝗼 𝗺𝘂𝗹𝘁𝗶𝗽𝗹𝗲 𝘃𝗮𝗿𝗶𝗮𝗯𝗹𝗲𝘀  
We can assign values to multiple variables in one line with array destructuring.

4 Useful JavaScript Snippets for Common Tasks You Can Use Today 👷🔥  
If anyone asks me what programming language they should learn as a beginner, I’ll tell them it’s JavaScript. This powerful language gets involved in almost every aspect of programming - frontend, backend, web, desktop, mobile.😍  
  
💃 Find the number of days between two dates in Javascript  
Let’s take an example of what we are going to do. We have a function that takes two dates as the input, and it returns the number of days.📆  
1) Convert the date object into a millisecond.  
2) Subtract the second date from the first date.  
3) Divide that by the number of milliseconds per calculated  
  
💃 How to test if a string is JSON or not?  
JSON.parse function will use string and converts to a JSON object and if it parses invalidate JSON data, it throws an exception (“Uncaught SyntaxError: Unexpected string in JSON”).  
  
💃 Validate if a number is valid  
This function is one of my favorites because it tries to handle all the end cases of validation. 💰  
  
💃 Set Multiple Default Parameters  
Sometimes you have to define a function with multiple parameters. Do you need to pass all the parameters values every time you invoke the function? Not really if you initialize the default values.

3 Most Powerful JavaScript Promise Methods 🎅🎁  
When working with promises in JavaScript, we have a lot of methods that can help us. In this post, we’re going to cover my 3 favorite methods. 🎉  
  
🔥 𝗣𝗿𝗼𝗺𝗶𝘀𝗲.𝗮𝗹𝗹()  
This method expects an array of promises (Iterable) and returns a single promise.  
It will only resolve if all the promises passed have been resolved. If any promise in an array of promises fails, then it will reject  
  
🔥 𝗣𝗿𝗼𝗺𝗶𝘀𝗲.𝗮𝗹𝗹𝗦𝗲𝘁𝘁𝗹𝗲𝗱()  
Promise.allSettled() accepts an array of promises. It returns a new promise that will resolve if all the array promises are settled, regardless of whether the promises are resolved or rejected. Upon resolving, it returns an array of objects. Each object has property “status” regardless of whether the promise was fulfilled or rejected, and “value” returned when the promise is resolved or rejected.  
  
🔥 𝗣𝗿𝗼𝗺𝗶𝘀𝗲.𝗿𝗮𝗰𝗲()  
This method also accepts an array of promises. Promise.race() returns a Promise that settles as soon as one of the promises in the iterable argument settles. A settled Promise is simply a promise that either has been resolved or rejected.  
  
⚠️Note: ES2021 will introduce a Promise.any(), So be ready!

4 Ways to Swap Variables in JavaScript 💂♨️  
In the past, swapping the value of two variables normally takes three lines and a temporary variable. While this approach still works, what if I told you there was an easier way to do this with JavaScript? 🤷  
  
🔥 Using a Temporary Variable  
⚪️ We created a temp variable to store the value of a temporarily.  
⚪️ We assigned the value of b to a.  
⚪️ The value of temp is assigned to b  
Note: The downside is the need for a specialized temporary variable, plus the swapping happens 3 times.  
  
🔥 Using Array Destructuring  
Destructuring assignment: If [a, b] = [1, 2, 3], the value of a=1, and b=2.  
⚪️ First, a temporary array [b, a] is created. Here the value of [b, a] will be [2, 4].  
⚪️ The destructuring of the array is done, i.e [a, b] = [2, 4].  
  
🔥 Using Mathematical Operations  
Let's see, Initially, a is 4 and b is 2.  
⚪️ a = a + b assigns the value 4 + 2 to a | now 6.  
⚪️ b = a - b assigns the value 6 - 2 to b | now 4.  
⚪️ a = a - b assign the value 6 - 4 to a | now 2, and finally a=2, b=4.  
  
🔥 Using Bitwise XOR operator  
Initially, a=4 and b=2.  
⚪️ a = a ^ b assigns the value 4 ^ 2 to a (now 6).  
⚪️ b = a ^ b assigns the value 6 ^ 2 to b (now 4).  
⚪️ a = a ^ b assign the value 6 ^ 4 to a (now 2), and finally, a=2 and b=4.

5 useful Spread Operator Tricks that you should know. 🔊🍞  
I recently came across some handy patterns using the spread operator so I thought I’d write up a quick and simple post🤙  
  
🚩 The spread operator is three dots (…), followed by an 'iterable' data type, iterables are anything that can be looped over such as strings, arrays, objects, or even a string.  
  
🚩The spread syntax works by copying the values of the original array, and then inserting them into another array or putting them in the order they appeared in the array as the list of arguments in a function in the same order.  
  
🚩With the 2018 version of [#javascript](https://www.linkedin.com/feed/hashtag/?keywords=javascript&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A6694904184371863552), we can also spread the properties of an object into another object, with keys and values spread into another object.

All You Need to Know to Get Started With JavaScript Promises. 🙋🎓  
Many people struggle with understanding how Promises work, so I will try to explain them as simply as I can. 😎  
  
Imagine you are a little kid, your mommy promises you that she'll buy you a new computer next week.🤵  
You don't know if you will get that computer until next week.  
Mom can either buy you a new computer, or she doesn't, because you didn't clean your room, and she isn't happy!.🛏️  
  
That is a promise! A promise has 3 states:  
🔶Pending: You don't know if you will get that computer.  
🔶Fulfilled: Mom is happy, she buys you a new computer.  
🔶Rejected: Mom is unhappy, she doesn't buy you a computer.  
  
  
🏁 Creating and Using A Promise 🏁  
Firstly, we use a constructor to create a Promise object, It takes two parameters, one for success (resolve) and one for fail (reject),  
Finally, there will be a condition. If the condition is met, the Promise will be resolved, otherwise, it will be rejected:  
  
✔️ .then( ) - for resolved:  
If the Promise success, then something will happen next depends on what we do with the resolved Promise  
  
❌ .catch( ) - for rejected:  
What if the Promise fails? it will jump to the catch( ), therefore we need to use the catch( ) method.

Understanding Deep and Shallow Copy in JavaScript 🏃‍♂️💨  
In programming, there are two types of copies, the first is called a “deep” and the second is called a “shallow”. Copying data in JavaScript looks simple. Well, it might actually be more complicated than you would think. 🕵️  
  
🎓 What is a Shallow copy?  
"Shallow copy" is also identified as pass by reference. ⚡️⭐️  
When you create a shallow copy, you create a new copy that is connected to the original. So when you change the original it will also affect and change the copied, and vice versa.  
  
🎓 What is a Deep copy?  
"Deep copy" is also identified as pass by value. ⚡️ ⭐️  
When you create a deep copy, you make an identical copy of the original and its properties. These properties are not connected, so If you change the original, that change, will not have any effect on that copy.  
  
There are many ways to copy elements, but which one is a deep copy and which is a shallow copy⁉️ 🤷🚦  
  
🔸 Primitive types: all elements with primitive types like numbers, strings, and boolean are deep copied which means that the copied element and the original are not connected  
  
🔸 Non Primitive Types: for type likes objects, Arrays, etc. Javascript does a shallow copy by default, which means they are connected to each other.

Quick Introduction to Sets in JavaScript 🧯🔥  
What sets allow you to do is to create collections of values.  
These values can be anything, from numbers and strings to arrays and objects. 📚  
  
Sets contain only unique values. When you try to add multiple same values into a set it will accept only the first. 🧨🧲  
  
📍 Creating sets in JS-New Set()  
When you want to create sets in JavaScript you do it always with the set constructor set(), preceded by the new keyword.  
  
📍 Adding values to sets - .add()  
You can add values to a set using the add() method. This method accepts either one value or an iterable. Iterable means an array of values.  
  
📍 Removing values from Sets - .delete()  
by using the delete() method, You pass the value as an argument when you call it. If deletion is successful, delete() will return true. If not, it will return false.  
  
📍 Removing all values from sets - .clear()  
Sometimes, you may want to remove all values from the set. This can be done using the clear() method.  
  
📍 Checking for existing values in sets - .has()  
The has() method is probably the simplest way to test if the set contains a specific value.  
  
📍 Finding out how big a set is-size  
When you want to know how many items are in an array, you can use its length property.

A Simple Guide to Javascript Generators 🔥🌶️  
Generators are functions that you can use to control the iterator. They can be suspended and later resumed at any time.💃  
  
Here’s a simple analogy to generators ✍️  
Imagine you are reading an action book, and suddenly your doorbell rings. This is the pizza guy. You get up, however before you do, you set a bookmark on the last page you read. Once you return, you start the book from the page where you set the bookmark. You do not start it from scratch. In a sense, you acted as a function of a generator.🎯  
  
How does it Work? 🕵️💡  
  
🔮 Calling a generator does not execute the function completely as soon as it's called. It will return an iterator object that can be used to use the function.  
  
🔮 So this function can be executed part by part, where these parts are decided by the yield expression.  
  
🔮 To execute these parts, the .next() is used on the iterator. When .next() is called, the function resumes execution until the next yield is found, or the function completes or a return statement is executed.  
  
🔮 Every time you execute the .next(), the generator returns an object that  
The value here is the value sent by the yield and the done indicates if the generator has run completely.

What is Local Storage and how does it work with JavaScript? 🕵️📬  
Local Storage is part of the web storage API, it allows you to store data inside a user’s browser which your application can access.🔥  
  
Local Storage allows JavaScript sites and apps to store and access data right in the browser with no expiration date. ⚡️💾  
This means the data stored in the browser will persist even after the browser window has been closed. 💡💻  
  
It is important that you know ⚠️  
Local Storage can only save strings, so storing objects requires that they be turned into strings using JSON.stringify(). That also means the object must be run through JSON.parse() on the way out of Local Storage.  
  
There are 4 basic JavaScript methods you can use with Local Storage 🔮  
⚪️ setItem(): Add key and value to localStorage.  
⚪️ getItem(): Retrieve a value by the key from Local Storage.  
⚪️ removeItem(): Remove an item by key from Local Storage.  
⚪️ clear(): Clear all Local Storage.  
  
The following are some limitations, and also ways to NOT use Local Storage ✔️❌  
◾️ Do not store sensitive user information in Local Storage.  
◾️ Local Storage is limited to 5MB across all major browsers.

5 Powerful JavaScript Array Tricks You Should Know 💃🔥  
An array is one of the most common concepts of Javascript, which gives us a lot of possibilities to work with data. Check this tricks which can be very helpful! Let’s get started.  
  
🚩 Removing Falsy Values  
In Javascript, falsy values are false, 0, ” ", null, NaN, undefined. Now we can find out how to remove this kind of values from our array. To achieve this, we are going to use the .filter()  
  
🚩 Get random value form the array  
Sometimes we need to select a value from the array randomly. To create it we can get a random index number according to the array length.  
  
🚩 Shuffle Elements in an Array  
Shuffling is easy with the sort method. As long as it returns a random number that can be positive or negative, we can sort it with a random order.  
  
🚩 Find the intersection of two arrays  
we can use a Set() to make sure that values in the array we are checking are not duplicated and we will use .filter and .includes methods. As a result, we will get the array with values that were presented in both arrays.  
  
🚩 Swap values with array restructuring  
The destructuring assignment syntax is a JS expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

How JavaScript Async/Await Works 🎯🕵️  
Async/Await is the next step in the evolution of handling asynchronous operations in JavaScript. 🏆  
It gives you two new keywords to use in your code: “async” and “await”.  
Async is for declaring that a function will handle asynchronous operations and await is used to declare that we want to “await” the result of an asynchronous operation inside a function that has the async keyword.  
  
Async functions return a Promise 🙋  
If the function throws an error, the Promise will be rejected. ✔️  
If the function returns a value, the Promise will be resolved. ❌  
  
🚀 How it works 🚀  
◾️There are Async Functions. These are declared by prepending the word async before their declaration.  
◾ Your code can be paused waiting for an Async Function with await  
◾ Await returns whatever the async function returns when it is done.  
◾ Await can only be used inside an async function.  
◾ If an Async Function throws an exception, the exception will bubble up to the parent ◾ Functions just like in normal Javascript and can be caught with try/catch.  
  
Key Points 📝  
🔸Async/await is a new way to write asynchronous code.  
🔸Async/await is actually just syntax sugar built on top of promises.

4 Powerful Ways to Level up Your JavaScript Array Manipulations. 🚜💨  
When it comes to iterating over an array, finding elements, sorting it, or whatever you want, there’s probably an array method out there that you can use. 🔥  
  
🚩 Filter  
The .filter() method is used to create a new array from a given array consisting of only those elements from the given array which satisfy a condition set by the argument method.  
Again, to clarify, the filter function doesn’t change the values in the array in any way, it’s just a filter, a gatekeeper.  
  
🚩 Some  
The .some() tests whether at least one element in an array passes the provided condition.  
It returns true if at least one element passes the test, or false otherwise.  
  
🚩 Every  
The .every() function tests if all the elements contained in an array pass the provided condition.  
This method returns the Boolean value true if all the elements in the array pass the test, or false otherwise  
  
🚩 Find  
The arr.find() method is used to get the value of the first element in the array that satisfies the provided condition. It checks all the elements of the array and whichever the first element satisfies the condition It returns, otherwise it returns undefined.

setTimeout, setInterval and How to Schedule Tasks in JavaScript 🤦⏰  
  
Usually, you want to execute your code as you write it. ✍️  
That said, there will be situations when you will want to delay the execution.  
Or, you may want to repeat the execution at specific intervals. JavaScript provides two methods, one for each of these goals. The setTimeout() to delay execution and setInterval() to repeat it. 🔥  
  
⭐setTimeout() - Executing Code After a Delay 🧭  
It is used to execute a function or specified piece of code just once after a certain period of time. Its basic syntax - setTimeout(function, milliseconds).  
It accepts two parameters: a function, which is the function to execute, and an optional delay parameter, which is the number of milliseconds to wait before executing the function.  
  
⭐setInterval() - Executing Code at Regular Intervals ⏳  
Similarly, you can use the setInterval() to execute a function or specified piece of code repeatedly at fixed time intervals.  
  
⭐Stopping Code Execution or Cancelling a Timer ⛔️  
Both setTimeout and setInterval method return something called “timer identifier," which identifies the timer created by these methods.  
You can stop your code execution using these two functions: clearTimeout() and clearInterval().

A Simple Guide for Understanding Closures in JavaScript 🔥🚁  
Closures in JavaScript are one of those concepts that many struggles to get their heads around.  
  
📣 Closures, also known as lexical or function closures, are combinations of functions bundled together with references to their surrounding state. In other words, Closure give you access to an outer function’s scope from an inner function outside of its original usage.  
  
⭐A simple example for understanding closures.⭐  
The adder function returns the add function, but it doesn't invoke it (exp: add()). This indicates that when we invoke the adder with a firstNum argument (exp: const addSeven = adder(7)), it returns a function that now knows what the value of firstNum is, and we can invoke it (like so: addFive(7)) to get the final result.  
  
⭐A real-life example of closures in production code.⭐  
We make a generic dataFetcher function that takes in a URL, and returns a function that takes an additional path parameter. We using this to be able to change the path dynamically to get results for different cases.  
Therefore, a specific data fetcher named "getUsersDataFetcher" that uses the dataFetcher function and passes in the URL for the JSONPlaceholder Users API.

8 Useful String Methods You Must Know as a JavaScript Developer. 😎⚡️  
JavaScript has many powerful built-in functions that make working with strings easy for developers, this list covers the most important string functions for you. 💯🚀  
  
🔹 Replace 🔹  
The .replace() method replaces a specified value with another value in a string.  
  
🔹 Concat 🔹  
The .concat()method is used to join two or more strings.  
  
🔹 Trim 🔹  
The .trim() method is used to remove white-space from either side of a string.  
  
🔹 Charat 🔹  
The .charAt() method returns the character at a specified index in a string.  
  
🔹 Repeat 🔹  
The .repeat() function returns a string consisting of the elements of the object.  
  
🔹 Changing 🔹  
Case .toUpperCase() and to .LowerCase() are the methods used to convert a string to uppercase or lowercase.  
  
🔹 Slice 🔹  
The .slice() method extracts a part of the string and returns the extracted part in a new string.  
  
🔹 Split 🔹  
The split() method converts a string into an array.  
You have to pass a character such as a comma (,) or space to tell where to split the string, If nothing is passed, the string is split between each character.

The Secret features of JSON.stringify(). 🤫🔮  
Simply the stringify() method is used to create a JSON string.  
While developing an application using JavaScript many times it is needed to serialize the data to strings for storing the data into a database or for sending the data to an API or web server.  
  
Actually JSON.stringify() Takes 3 Arguments:  
  
◾The first parameter of the method convert a JavaScript object or value to a JSON string.  
  
◾The second parameter is called the replacer, It takes 2 types of arguments: Array and Function.  
If the argument is an Array - key/s passed in the array is used to pick the key-value pair that you want to output,  
If the argument is a Function - you can loop over each item and with each pass, manipulate with the logic defined in your function.  
  
◾The third parameter controls the spacing in the final string.  
If the argument is a number, each level in the stringification will be indented with this number of space characters.  
If the third argument is a string, it will be used instead of the space character as displayed above.  
  
The .stringify() function is a core JavaScript fundamental.  
so a solid understanding of .stringify() lets you do more with your favorite npm modules.

How Type Coercion Works in JavaScript 🙋‍♀️🔥🚀  
In JavaScript, Type coercion is the process of converting a value from one type to another (string to a number, object to boolean). Any type, be it primitive or an object, is a valid subject for type coercion. 👍  
  
Type coercion can be explicit and implicit 🤔  
When a developer converts between types, like Number(value), it’s called explicit type coercion. However, implicit type coercion usually happens when you apply operators to values of different types, like 1==null, 8>'12'.  
  
The first rule to know is there are only three types of conversion  
  
🚩 String Conversion  
To explicitly convert values to a string, apply the String() function. Implicit coercion is triggered by the binary + operator when any operand is a string  
  
🚩 Boolean Conversion  
To explicitly convert a value to a boolean apply the Boolean() function. Implicit conversion happens in logical context, or is triggered by logical operators ( || && !)  
Logical operators, do boolean conversions internally, but actually return the value of original operands.  
  
🚩 Number Conversion  
For an explicit conversion just apply the Number() function.  
Implicit conversion is tricky because it’s triggered in more cases:  
comparison operators (>, <), arithmetic operators.

Object Destructuring may sound scary at first, but it's really very simple, believe me! 👨‍🚒  
Once you know about destructuring it's probably something that you're going to be using every single day.  
  
But what exactly is destructuring?  
Destructuring is a JavaScript expression that allows you to extract data from arrays, objects, maps, sets, into their own variable.  
  
Destructuring might seem tricky when you're starting out, but once you found your feet, it'll help you quickly pick up what you need by creating certain patterns.

What if I told you can do other things with console besides console.log()? 💻 🔥  
The console object has more methods than just log and can help you bring your JavaScript debugging skills to a new level.⚡️🤘  
  
🏁 console.assert():  
  
The thing with .log() is that the only way to conditionally log something is to put it in an if-block statement.  
This is where .assert() comes to help us.  
.assert() only logs if the first item passed in is false, otherwise, it doesn’t log anything.  
  
🏁 console.time(), console.timeEnd():  
  
When you start having junky and slow performance, one of the most important things to do is to determine where your performance bottlenecks are.  
A useful tool for that is controlled with 2 methods:  
Start a timer with .time()  
Stop the timer and logging its final state using .timeEnd()  
  
🏁 console.count() and console.countReset():  
  
Sometimes all you need is to keep a counter variable, to do that with just .log() would necessitate keeping track of the count in a global variable which is not an ideal solution.  
Luckily, we have .count() to do this for us.  
Each time you call .count() it will give the running total.  
The count will reset on either a page refresh or by calling .countReset().

Today it is time to talk about the Object.entries and Object.fromEntries methods which allow us to convert an object into an array of key-value arrays and vice versa. 💡  
  
What's this for?  
Its main utility is to be able to operate on objects taking advantage of the methods of arrays such as filter, map...  
  
So, by combining the two methods we have the possibility to perform object-to-object transformations.  
  
For example:  
📍 Object.fromEntries(Object.entries(object).filter( ... ))  
📍 Object.fromEntries(Object.entries(object).map( ... ))

Optional Chaining is one of my new favorite [#javascript](https://www.linkedin.com/feed/hashtag/?keywords=javascript&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A6679311837453987840) ES2020 features 🚀 💪  
  
The '?.' operator allows you to safely access the properties of nested objects, without having to check for each of them.  
  
When you want to reach the property of an object, usually you can use '&&' operator to avoid getting errors when the object is null or undefined.  
  
With these new JavaScript feature it makes the code more elegant and avoids some errors.  
You can just use '?.' instead of adding '&&' operator for each level of the tree.  
  
The optional chaining '?.' stops the evaluation and returns undefined if the part before '?.' is undefined or null.

A Quick Introduction To The ES6 Map Data Structure in JavaScript. 🗺️🕵️  
Map is a collection of elements where each element is stored as a key, value pair. 🔑  
   
A map can hold both objects and primitive values as either key or value.  
Let’s look at the differences. We have a couple of built-in methods:  
  
🤙 set(key, value) → Set the value for the key in the Map, and return the Map.  
If the key is already present in the map, then the value is overwritten.  
  
🤙 get(key) → Returns the value associated with the key or returns undefined if there is no key on the Map that matches the passed key.  
  
🤙 delete(key) → Delete the entry matching the key and returns true.  
If the key is not found in the map, then it returns false.  
  
🤙 has(key) →  Returns true if the map has the key provided, else return false.  
  
🤙 size() → This is how we get the number of entries currently in the Map.  
  
🤙 clear() → Delete all key-value pairs from the Map.  
  
🤙 keys() → Returns a new iterator that contains the keys in insertion order.  
  
🤙 values() → Returns a new iterator that contains the values in insertion order.  
  
🤙 entries() → Returns an iterator that has an array of key-value pairs in insertion order.

Dynamic import( . . . ) is another freshness [#javascript](https://www.linkedin.com/feed/hashtag/?keywords=javascript&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A6681837245504212992) new feature in the ES2020 🤙🎯  
  
It gives you the option to import JS files dynamically as modules in your application natively.  
  
This feature will help you ship on-demand-request code, better known as code splitting, without the overhead of webpack or other module bundlers.  
Also, you can call it in any place in the code!  
  
The dynamic import allows developers to be able to dynamically load parts of a JavaScript application at run time.  
This comes with multiple advantages, for example:  
  
📌 Lazy loading of routes of your application (improving performance).  
  
📌 Load user's language instead of loading them all.  
  
📌 Can handle failure if the module isn't found.

What is the difference between the `==` and the `===` operators in [#javascript](https://www.linkedin.com/feed/hashtag/?keywords=javascript&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A6683327591773937665) ?🧐🤷‍♂️  
  
in JavaScript `==` and `===` both are used to compare two values.  
The question arises here is that if `==` operator was already there to compare values then what is the need to introduce `===` operator in JS.  
The reason is quite interesting!💡  
  
In all programming languages `==` operator is used to comparing two values and it is the same for the JavaScript but sometimes `==` operator behaves very weird and can give the wrong result.  
  
Soo to sum up...  
The `==` operator is used to check the value which means it compares the value only.  
The `===` operator is used to check the datatype along with the result (it is known as Strict Comparison).