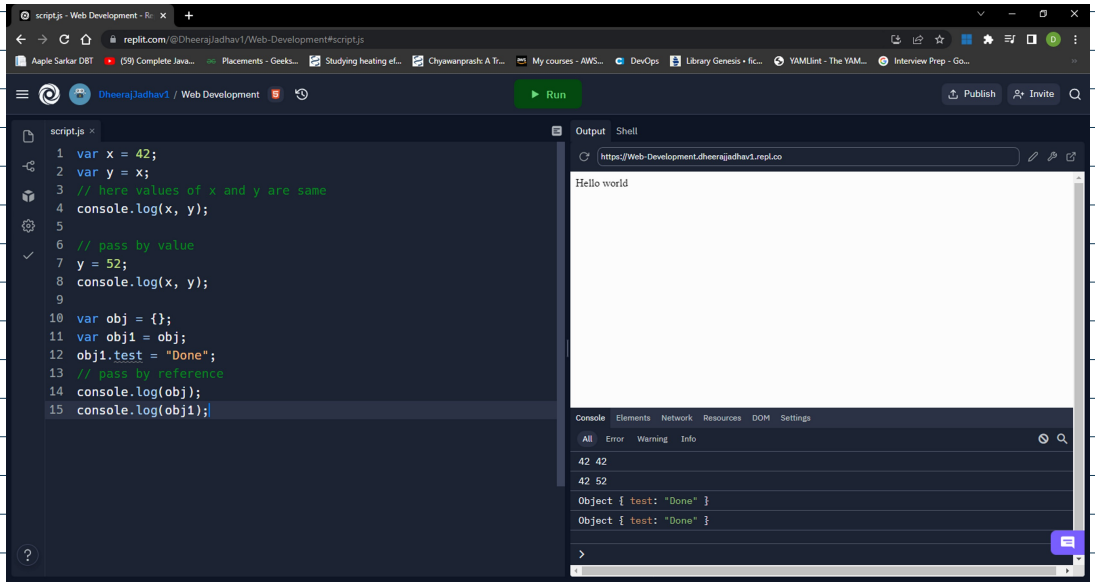
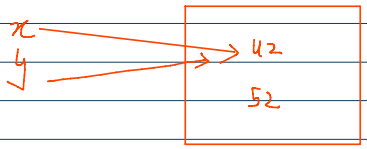
Advance JavaScript

Thursday, June 16, 2022 7:09 PM

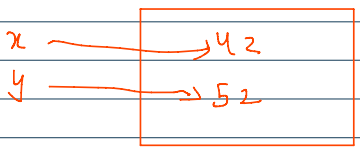


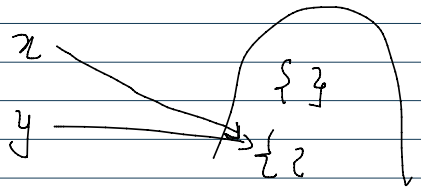
In the above code

when it comes to variable values are passed



If we change the value of one variable value of other variable doesn't change

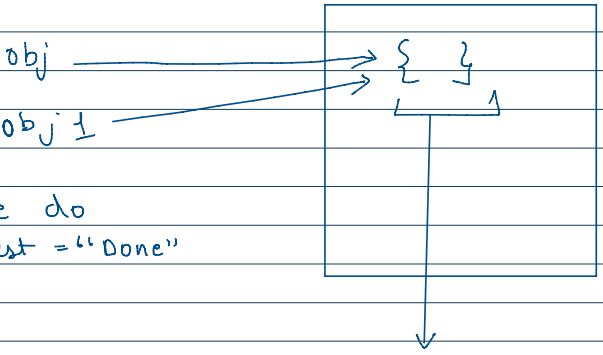




$x = y$

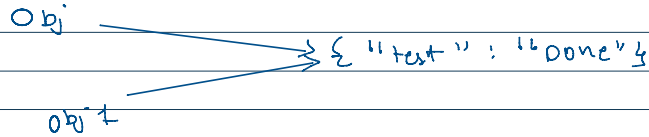
y.info = []

x.info[]



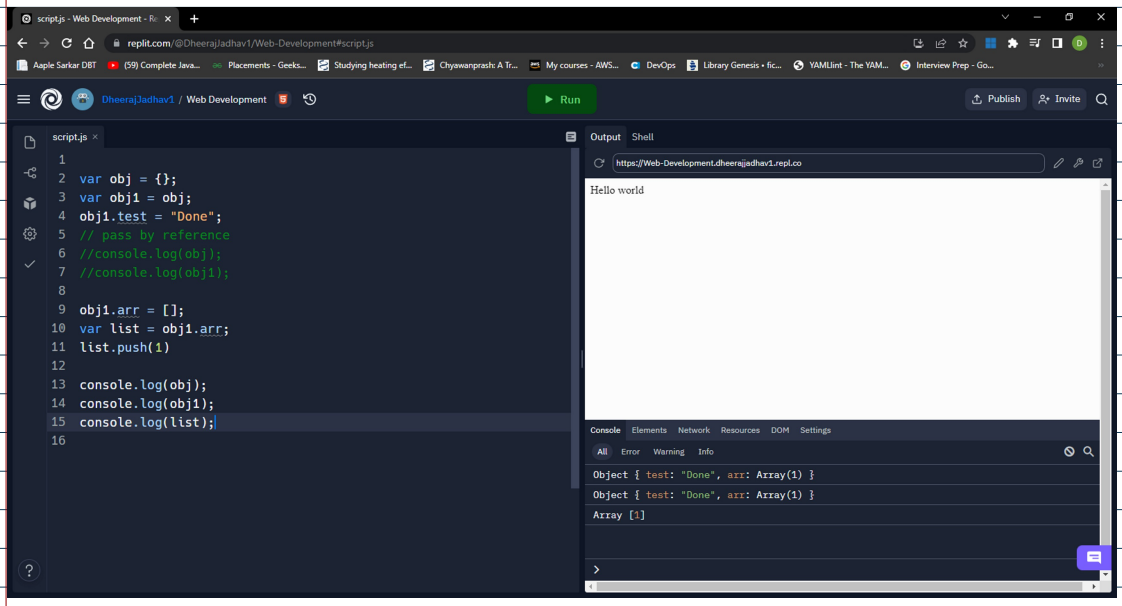
when we do
obj1.test = "Done"

This object will change



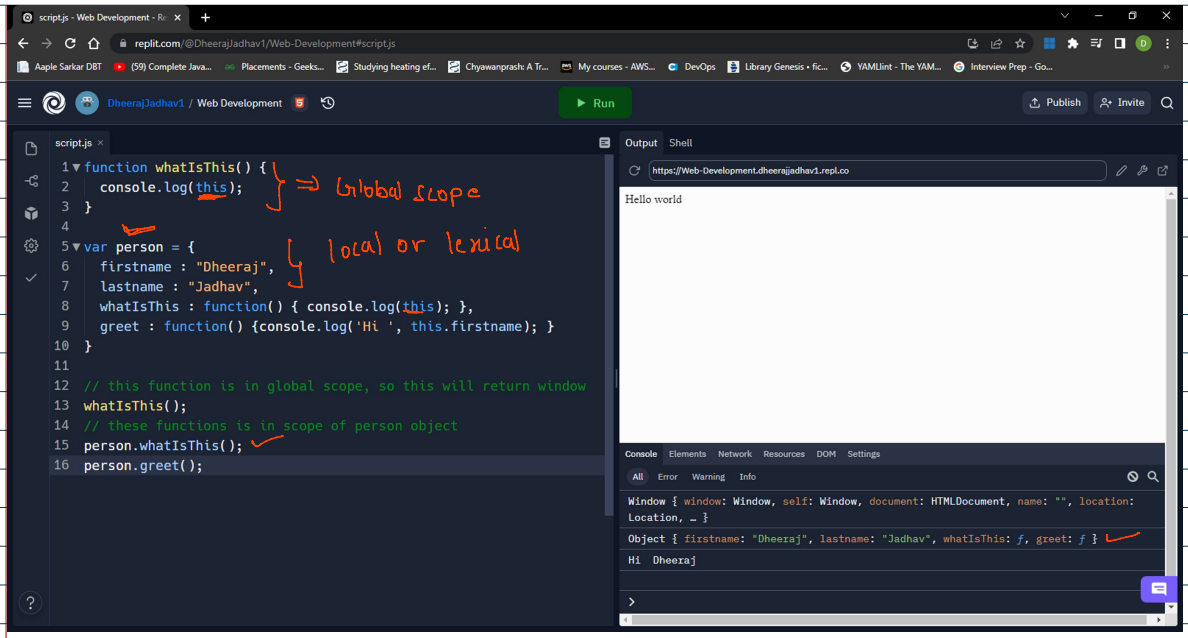
As obj & obj1 are pointing to same object, change in object through one reference variable, change will be shown on other reference variable also

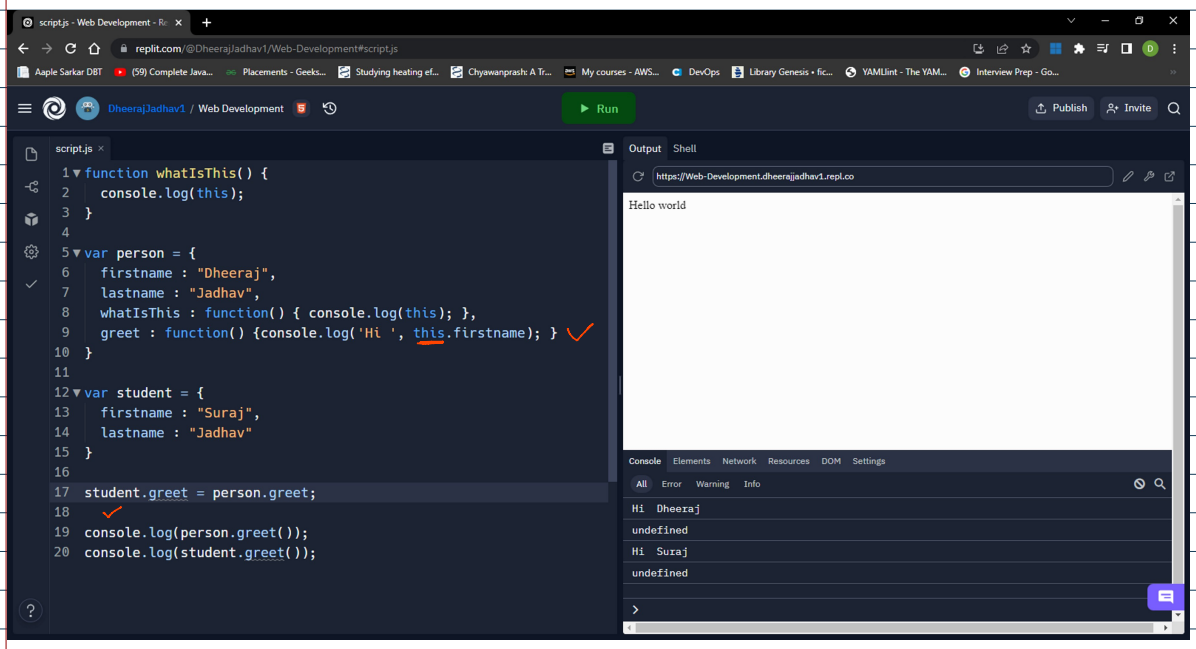
⇒ So, objects are passed by reference



'this' Key word

https://www.geeksforgeeks.org/this-in-javascript/

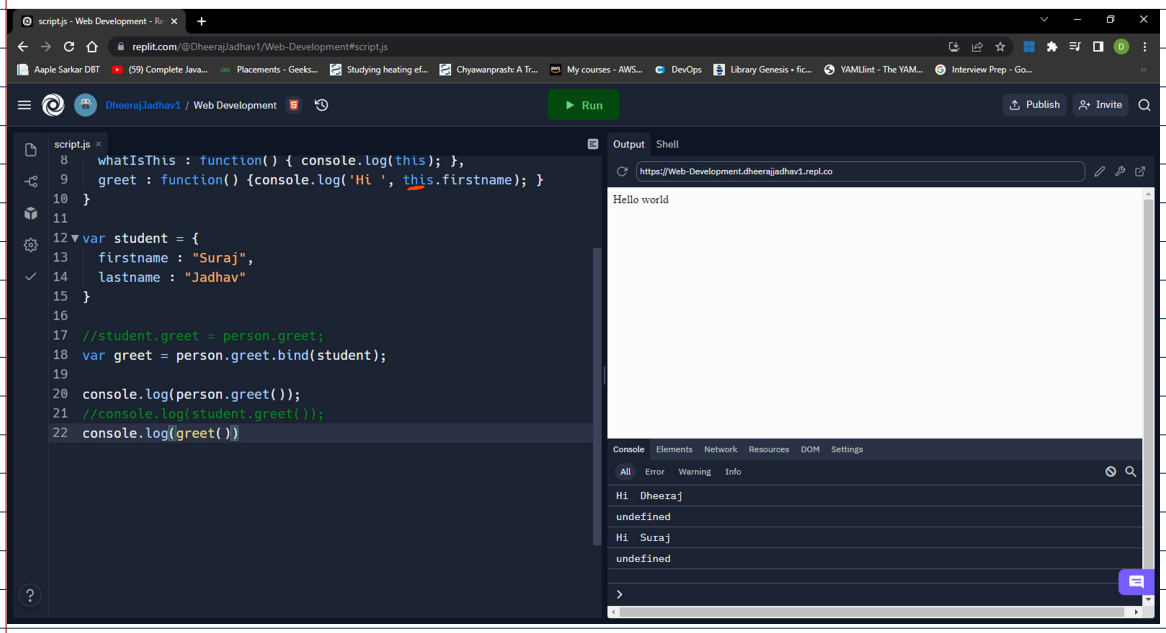


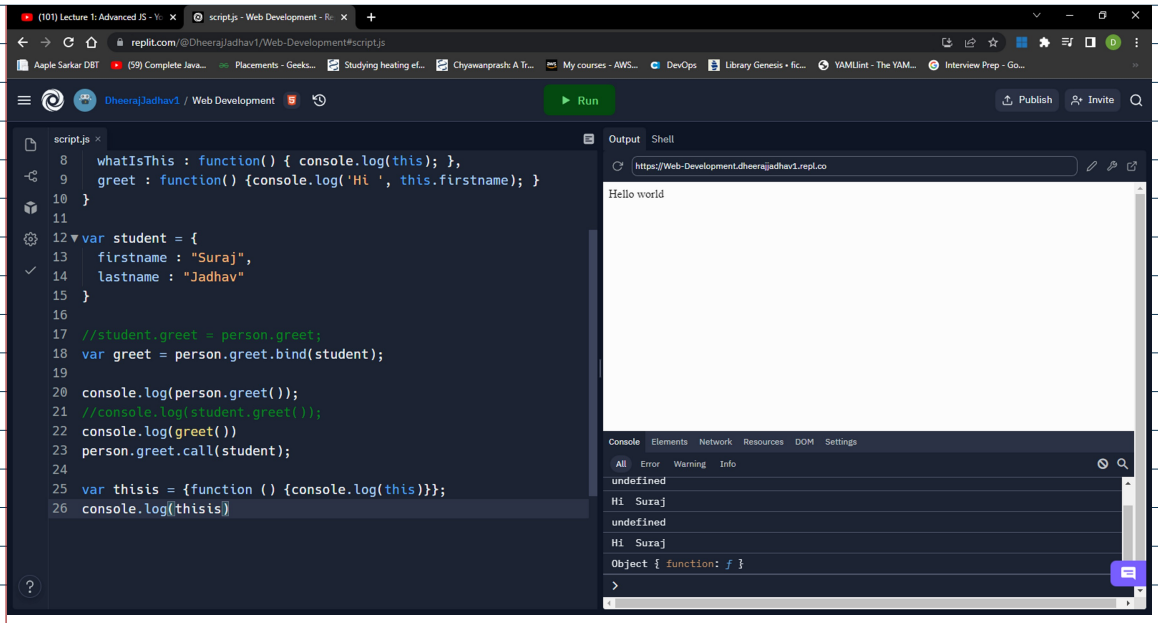


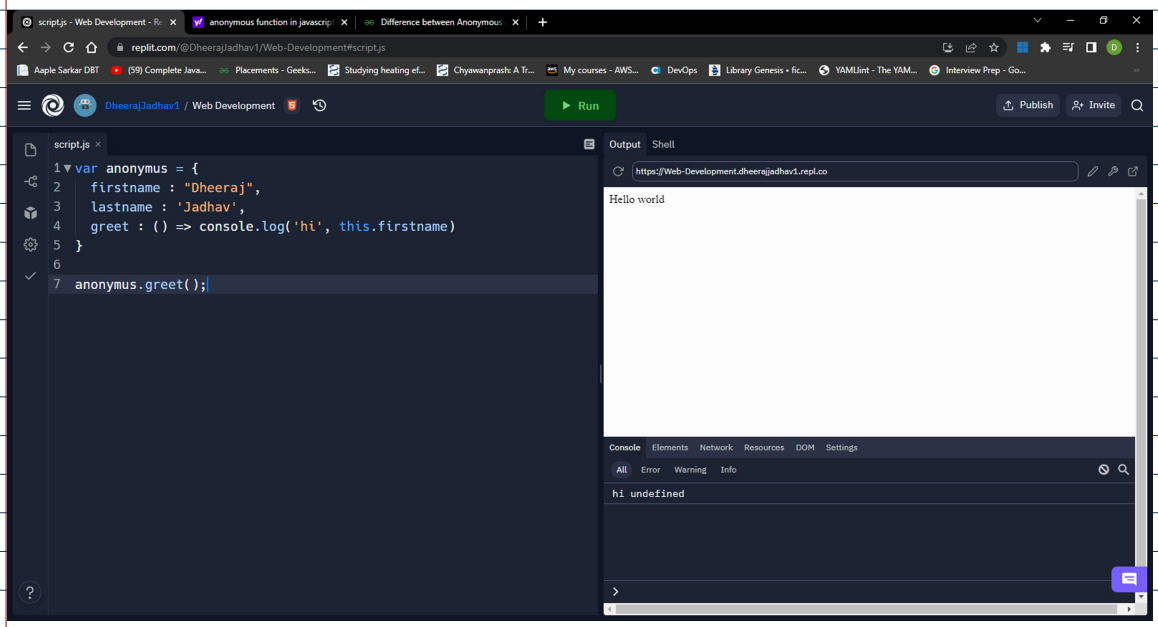
Student.greet
=
person.greet

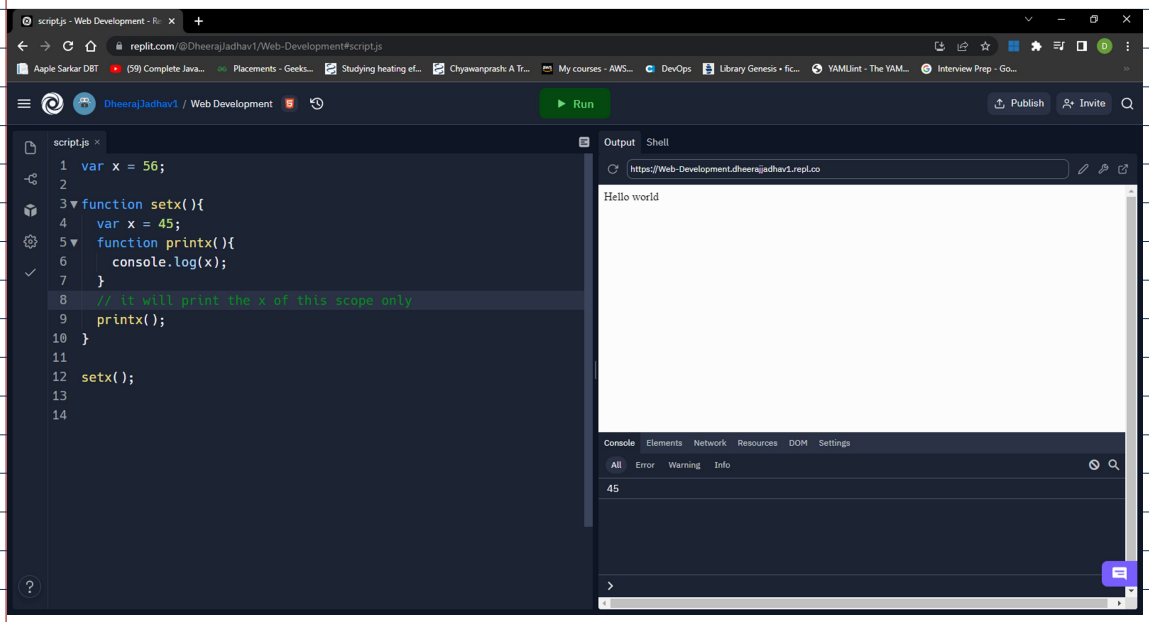
Student.greet
= ()

Call
this = Student



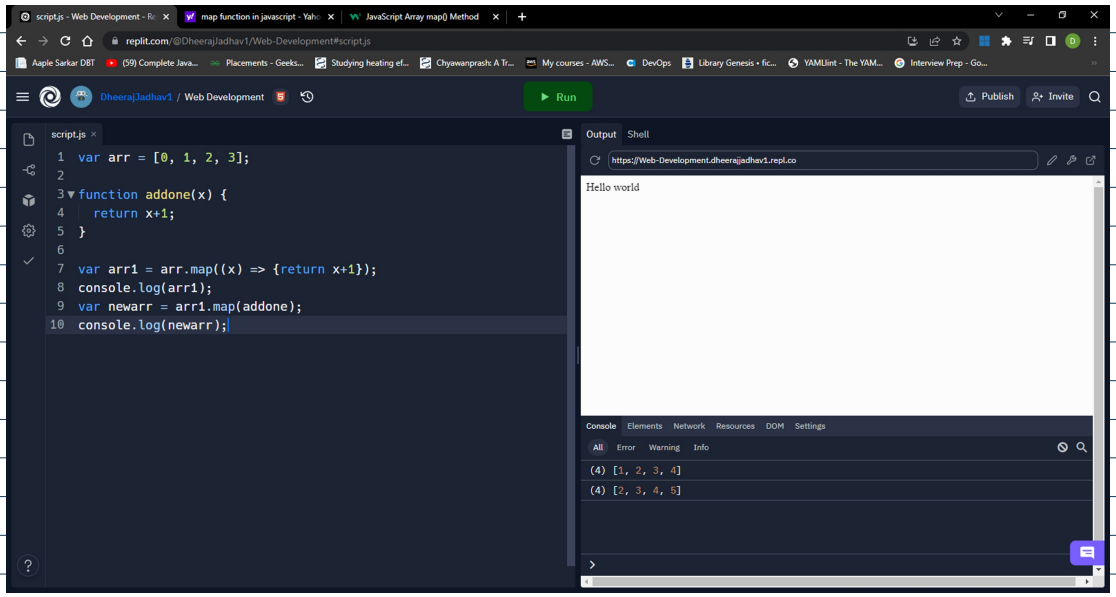






➢ **Map in JavaScript**

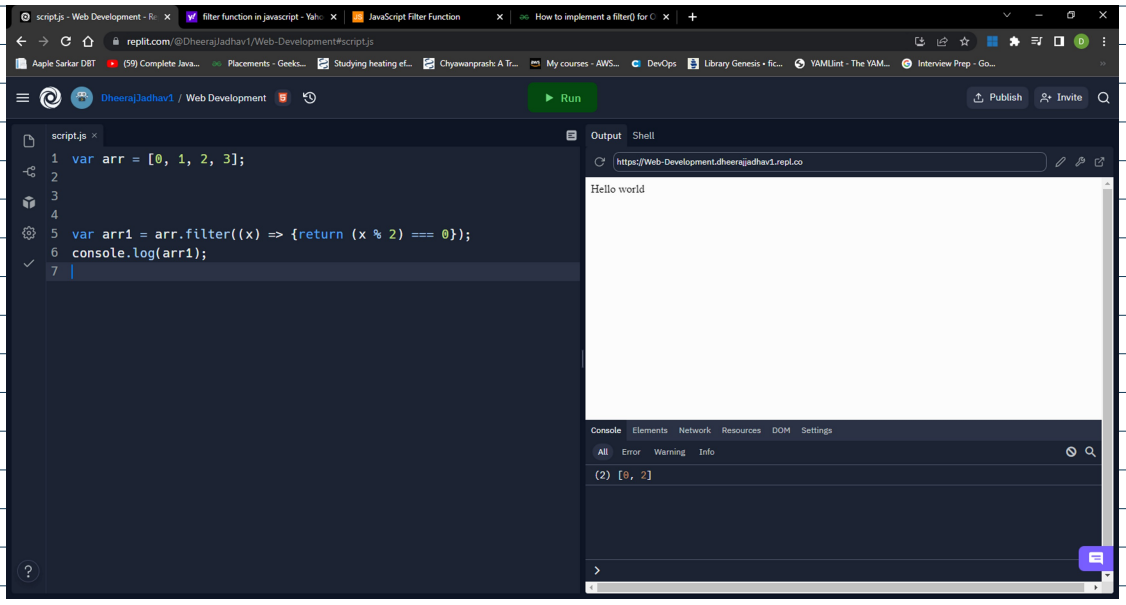
1. Runs a function on each value of array.

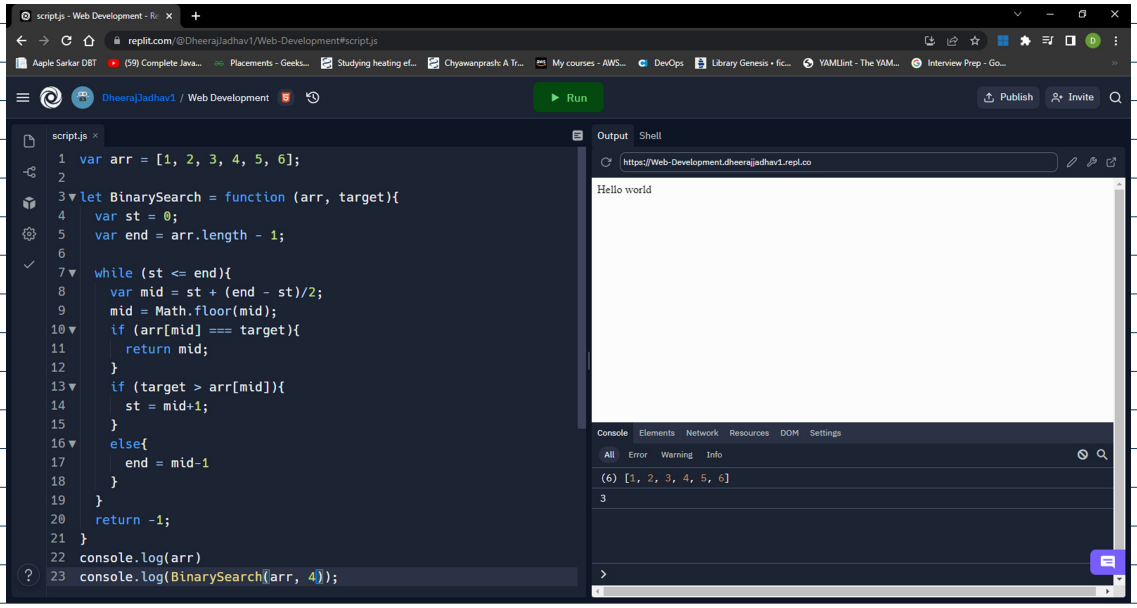


➢ Filter function in JavaScript

The filter() method basically outputs all the element object that pass a specific test or satisfies a specific function. The return type of the filter() method is an array that consists of all the element(s)/object(s) satisfying the specified function.

From <https://www.geeksforgeeks.org/how-to-implement-a-filter-for-objects-in-javascript/>





➢ **Synchronous and Asynchronous in JavaScript**

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

Let us understand this with the help of an example.

Example:

```
<script>
    document.write("Hi"); // First
    document.write("<br>");

    document.write("Mayukh") ;// Second
    document.write("<br>");

    document.write("How are you"); // Third
</script>
```

Output:

In the above code snippet, the first line of the code **Hi** will be logged first then the second line **Mayukh** will be logged and then after its completion, the third line would be logged **How are you**.

So as we can see the codes work in a sequence. Every line of code waits for its previous one to get executed first and then it gets executed.

Asynchronous JavaScript: Asynchronous code allows the program to be executed immediately where the synchronous code will block further execution of the remaining code until it finishes the current one. This may not look like a big problem but when you see it in a bigger picture you realize that it may lead to delaying the User Interface.

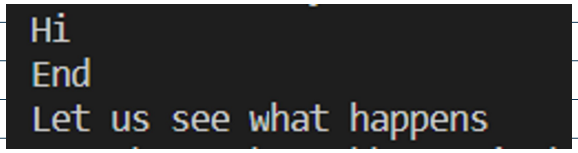
Let us see the example how Asynchronous JavaScript runs.

```
<script>
    document.write("Hi");
    document.write("<br>");

    setTimeout(() => {
        document.write("Let us see what happens");
    }, 2000);
```

```
    document.write("<br>");
    document.write("End");
    document.write("<br>");
</script>
```

Output:



So, what the code does is first it logs in **Hi** then rather than executing the **setTimeout** function it logs in **End** and then it runs the **setTimeout** function.

At first, as usual, the **Hi** statement got logged in. As we use browsers to run JavaScript, there are the web APIs that handle these things for users. So, what JavaScript does is, it passes the **setTimeout** function in such web API and then we keep on running our code as usual. So it does not block the rest of the code from executing and after all the code its execution, it gets pushed to the call stack and then finally gets executed. This is what happens in asynchronous JavaScript.

From <https://www.geeksforgeeks.org/synchronous-and-asynchronous-in-javascript/>

★➢ ***Introduction to Asynchronous JavaScript***

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing

★➢ ***Why we put JavaScript tags at the bottom of HTML ?***

It is a best practice to put JavaScript tags just before the closing tag rather than in the section of your HTML.

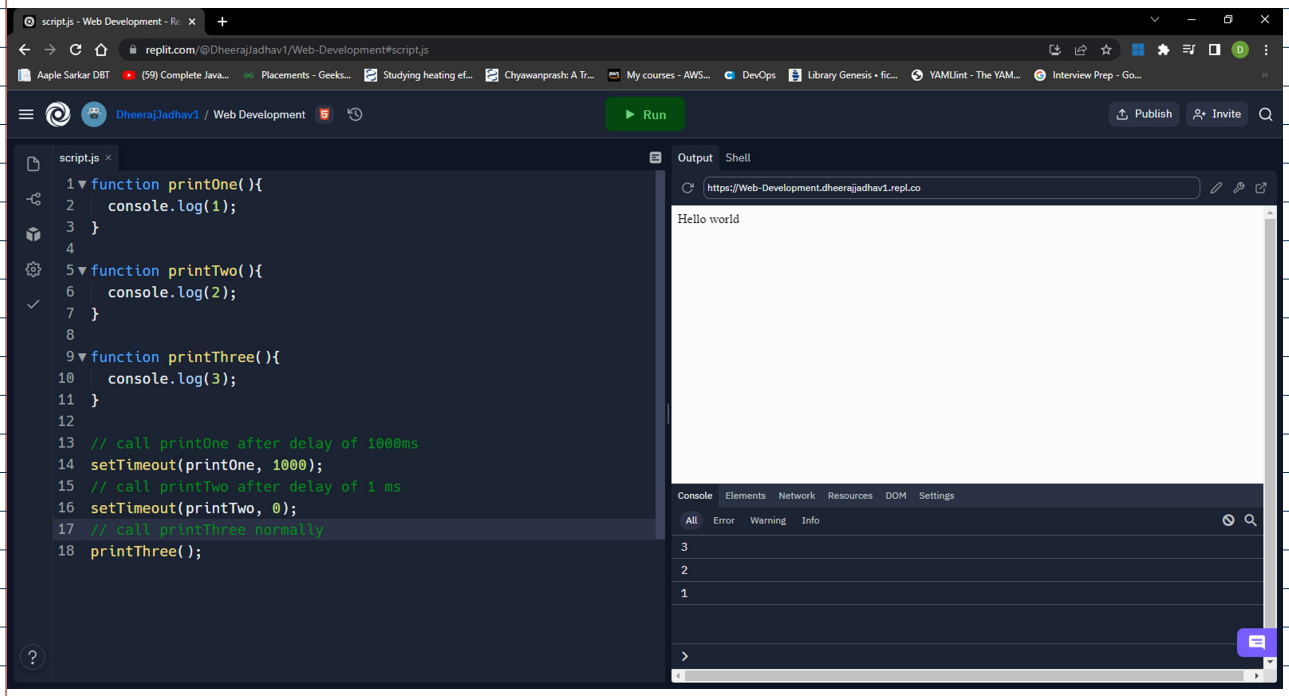
The reason for this is that HTML loads from top to bottom. The head loads first, then the body, and then everything inside the body. If we put our JavaScript links in the head section, the entire JavaScript file will load before loading any of the HTML, which could cause a few problems.

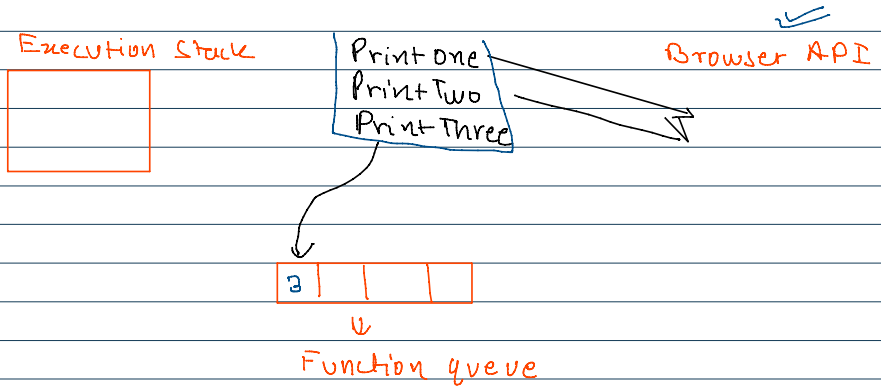
1.If you have code in your JavaScript that alters HTML as soon as the JavaScript file loads, there won't actually be any HTML elements available for it to affect yet, so it will seem as though the JavaScript code isn't working, and you may get errors.
2.If you have a lot of JavaScript, it can visibly slow the loading of your page because it loads all of the JavaScript before it loads any of the HTML. When you place your JavaScript links at the bottom of your HTML body, it gives the HTML time to load before any of the JavaScript loads, which can prevent errors, and speed up website response time.

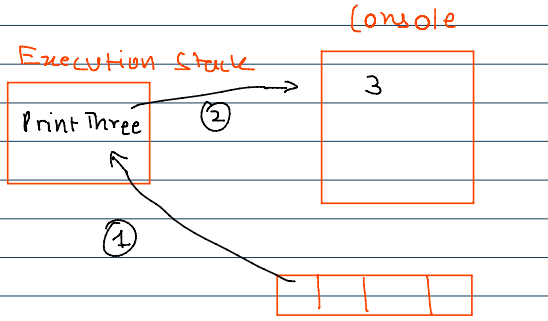
One more thing: While it is best to include your Javascript at the end of your HTML , putting your Javascript in the of your HTML doesn't ALWAYS cause errors. When using jQuery, it is common to put all of your code inside a "document ready" function:

$("document").ready(function(){ // your code here });

This function basically says, don't run any of the code inside until the document is ready, or fully loaded. This will prevent any errors, but it can still slow down the loading time of your HTML, which is why it is still best to include the script after all of the HTML.



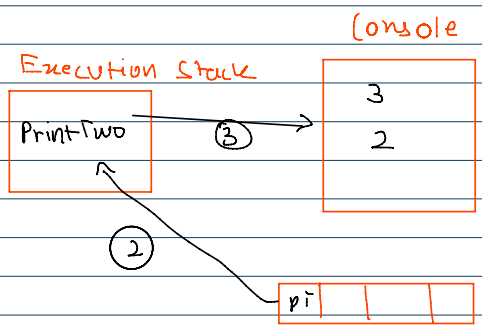




Browser API

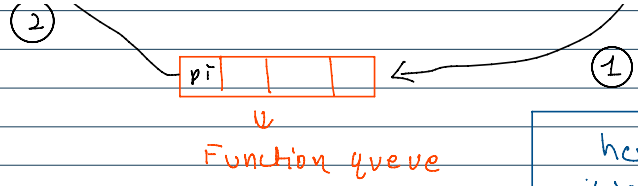
Print One
Print Two

here as PrintOne & PrintTwo are delayed, they will be sent to Browser API, while they getting ready to execute, other function will get executed

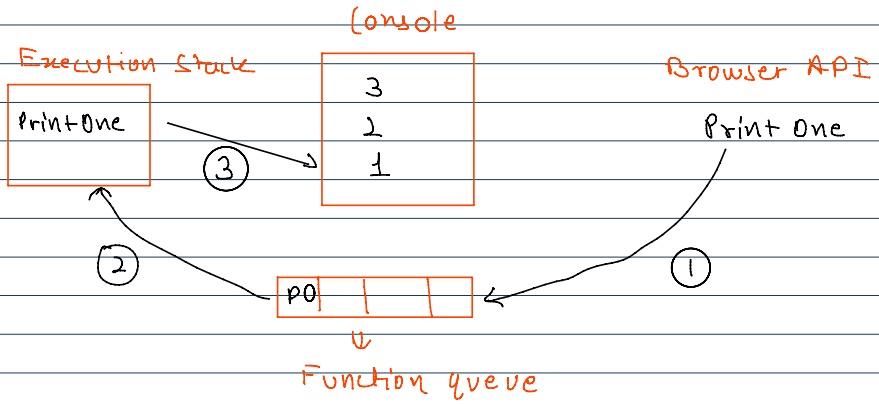


Browser API

Print One
Print Two



here as delay Time for Printtwo is less it will get ready quickly and will be sent to function queue, now as execution stack is empty, it will be sent to execution stack & will be executed



At last printOne will be sent to function queue, and when the execution stack gets clear, it will be sent to execution stack and will get executed.

★ ➢ **Execution Stack in JavaScript**

https://hemanta.io/understanding-execution-stack-in-javascript/

★ ➢ **Callback functions in JavaScript**

Callbacks are a great way to handle something after something else has been completed. By something here we mean a function execution. If we want to execute a function right after the return of some other function, then callbacks can be used.

JavaScript functions have the type of Objects. So, much like any other objects (String, Arrays etc.), They can be passed as an argument to any other function while calling.

JavaScript code to show the working of callback:
Code #1:

```
<script>

   // add() function is called with arguments a, b
   // and callback, callback will be executed just
   // after ending of add() function
   function add(a, b , callback){
   document.write(`The sum of ${a} and ${b} is ${a+b}.` +"<br>");
   callback();
   }

   // disp() function is called just
   // after the ending of add() function
   function disp(){
   document.write('This must be printed after addition');
```

```
    }

    // Calling add() function
    add(5,6,disp);

</script>
```

Output:

```
The sum of 5 and 6 is 11.
This must be printed after addition
```

Explanation:

Here are the two functions – add(a, b, callback) and disp(). Here add() is called with the disp() function i.e. passed in as the third argument to the add function along with two numbers.

As a result, the add() is invoked with 1, 2 and the disp() which is the callback. The add() prints the addition of the two numbers and as soon as that is done, the callback function is fired! Consequently, we see whatever is inside the disp() as the output below the addition output.

Code #2:

An alternate way to implement above code is shown below with anonymous functions being passed.

```
<script>

    // add() function is called with arguments a, b
    // and callback, callback will be executed just
    // after ending of add() function
    function add(a, b , callback){
    document.write(`The sum of ${a} and ${b} is ${a+b}.` +"<br>");
    callback();
    }

    // add() function is called with arguments given below
    add(5,6,function disp(){
    document.write('This must be printed after addition.');
    });

</script>
```

Output:

```
The sum of 5 and 6 is 11.
This must be printed after addition.
```

Callbacks are primarily used while handling asynchronous operations like – making an API request to the Google Maps, fetching/writing some data from/into a file, registering event listeners and related stuff. All the operations mentioned uses callbacks. This way once the data/error from the asynchronous operation is returned, the callbacks are used to do something with that inside our code.

From <https://www.geeksforgeeks.org/javascript-callbacks/>

https://www.freecodecamp.org/news/javascript-callback-functions-what-are-callbacks-in-js-and-how-to-use-them/

a

Ajax = asynchronous javascript and xml