Day 3.3

Javascript 3

There are two ways to write in javascript.

- 1st one is Under script tag
- 2nd is write your javascript code in external file for e.g ‘script_js.js’ and give the address of this file in src attribute of script element

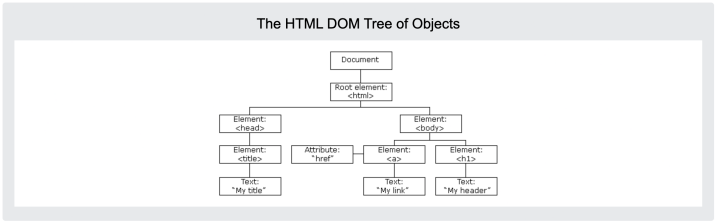
```
<script
src="https://code.jquery.com/jquery-3.5.1.slim.min.js"

></script>
```

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

```
<script>
  function animateBall() {
const ball = document.getElementById("ball1");
let top = 0;
let left = 0;
const id = setInterval(moveBall, 20);

function moveBall() {
    if(top < 400 && left < 100) {
        top = top + parseInt(Math.random()*2);
        left = left + parseInt(Math.random()*2);
        ball.style.top = top + "px";
        ball.style.left = left + "px";
    } else {
        clearInterval(id);
    }
}
}

</script>
```

Event handler

‘onClick’




```
HTML
<button
onClick="someFunction()"
>
MyButton
</button>
```

- When you click on the button created using <button> tag the function named ‘someFunction()’ will be called.
- This function is defined in javascript.

Creating an element of html using javascript

- Whenever you click on the button ‘MyButton’, <p> element of html will be created. But it is not added in the DOM tree (in simple words it is not added in your html).

```
JS
function someFunction() {
    const element = document.createElement("p");
}
```

Adding inner html to this element

- That means p tag can have text or inner html using JS.
- ‘.innerHTML’ is used for adding it.

```
function someFunction() {
    const element = document.createElement("p");
    element.innerHTML = "Some text I am adding in P";

}
```

- Whenever the button is clicked the ‘p’ element is created and the text in double quotes will be added in that html element. Still this ‘p’ element is not added in DOM tree.

Add it to DOM tree

- Now the <p> element with some text inside it will be added in the DOM tree. This is done using the command “document.body.appendChild(element’)”. Which will add it as a child of body element.
- Note that appendChild() will always add the element at the end.
- 1st line creates empty p tag.
- 2nd line adding some text inside that p tag.
- 3rd line is add this new p tag.

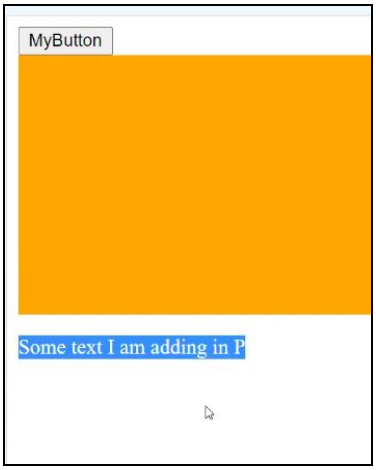
```
function someFunction() {
    const element = document.createElement("p");
    element.innerHTML = "Some text I am adding in P";
    document.body.appendChild(element);

}
```

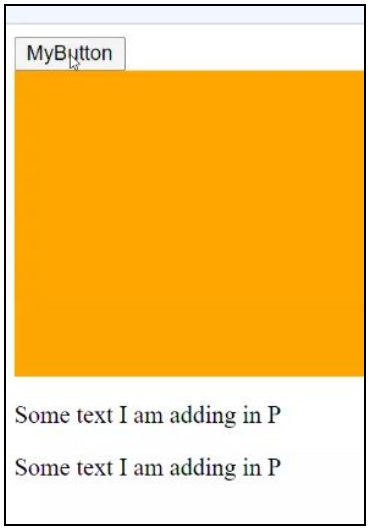
HTML

```
4   MyButton
5   </button>
6   <div id="c1">
7   </div>
```

CSS

```
1   #c1 {
2       height: 200px;
3       background-color: orange;
4   }
```






```
function someFunction() {
    const element = document.createElement("p");
    element.innerHTML = "Some text I am adding in P";
    // document.body.appendChild(element);
    const containingElement = document.body;
    containingElement.appendChild(element);

}
```

- When the button was clicked the text was added at the end not before the div.
- When the button clicked once more it will add one more p and so on.
- This is called dynamically adding the element.

Adding p element inside that div

```
function someFunction() {
    const element = document.createElement("p");
    element.innerHTML = "Some text I am adding in P";
    // const containingElement = document.body;
    const containingElement = document.getElementById("c1
    containingElement.appendChild(element);

}
```

- Obtaining the element using ‘document.getElementById(“c1”);’ store the element into a const variable ‘containingElement’.
- Add ‘p’ element to ‘c1’ element (id of div) using ‘containingElement.appendChild(element);’

What if the text in ‘p’ element contains other html tags?

```
function someFunction() {
    const element = document.createElement("div");
    element.innerHTML = "<p><b>element</b>";
    // const containingElement = document.body;
    const containingElement = document.getElementById("c1
    containingElement.appendChild(element);

}
```

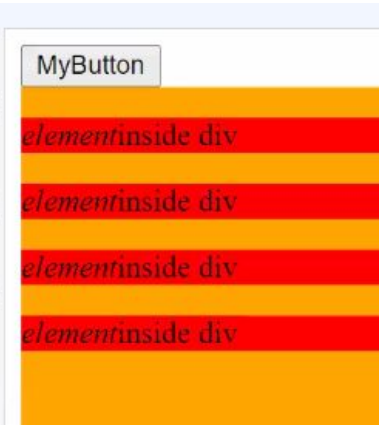
- <b> element will be actually added in the DOM tree.

```
element.innerHTML = "<p><b>element</b>inside div</p>";
```

- The output will be :

- If you verify using inspect(right click on browser and click inspect) you will see the <p> tag inside a div.
- Adding css to internal element that we are creating is done using using ‘element.style.backgroundColor=”red”;’

```
function someFunction() {
    const element = document.createElement("div");
    element.innerHTML = "<p><i>element</i>inside div</p>";
    element.style.backgroundColor = "red";
    // const containingElement = document.body;
    const containingElement = document.getElementById("c1");
    containingElement.appendChild(element);

}
```



- Multiple output is the result of clicking button again and again.
- Observe the background-color used in css and backgroundColor in script. Hyphen is removed and the letter in front of hyphen becomes capital.

Summarizing

- Four things learned.
    - Creating element.
    - Adding innerhtml.
    - Getting containing element.
    - Appending child (Adding it to DOM tree)

In industry this(following) is not preferred

```
element.innerHTML = "<p><i>element</i>inside div</p>";
```

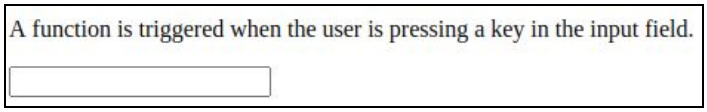
We must do it this way

```
const containingElement = document.createElement("div");
containingElement.appendChild(element);
```

Other Event Listeners

- **'onkeypressed'**
    - Whenever you are pressing any key the function passed as an argument will be called.
    - Example:

```
<input type="text" onkeypress="myFunction()">

<script>
function myFunction() {
  alert("You pressed a key inside the input field");
}
</script>
```

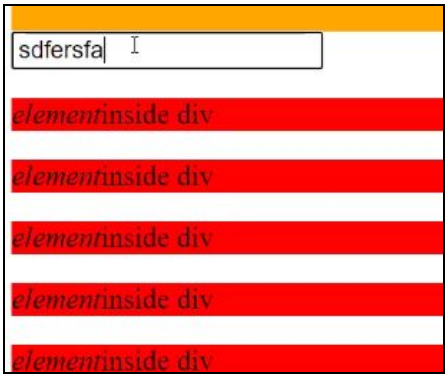


- **‘onkeydown’**
  - It's an event when the user presses a key.
  - When you press any key ‘someFunction()’ is called, in the below example you are writing ‘sdfersfa’ in the input field hence ‘someFunction()’ is called 8 times(browser height was small, that is why showing 5 times only).

```
<div id="c1" >
</div>
<input type="text" name="user" onkeydown="someFunction()"></input>
```

JS

```
function someFunction() {
    const element = document.createElement("div");
    element.innerHTML = "<p><i>element</i>inside div</p>";
    element.style.backgroundColor = "red";
    // const containingElement = document.body;
    const containingElement = document.body;
    containingElement.appendChild(element);

}
```



- Following is the difference between ‘onKeyDown’, ‘onKeyUp’ and ‘onKeyPressed’

- The `onKeyDown` event is triggered when the user presses a key.
- The `onKeyUp` event is triggered when the user releases a key.
- The `onKeyPress` event is triggered when the user presses & releases a key (`onKeyDown` followed by `onKeyUp`).

‘onLoadhandler’

- This will call as specified function whenever the body of html is loaded.

```
<body onLoad="someFunction()">
```

How to read the value from an html tag

- In the following example we will take input from the user and display the input inside a div after clicking on ‘Read Input’ button.

HTML

```
>
Read Input
</button>
<input type="text" name="user" id="myinput"></input>
```

Getting input element

```
const inputElement = document.getElementById("myinput");
```

Getting value of ‘name’ attribute of input element

```
element.innerHTML = inputElement.name;
```

Getting value of ‘value’ attribute of input element

```
element.innerHTML = inputElement.value;
```

Note: inputElement.class; will not return the class name only this is the exception to the above code. All other attributes can be access except class attribute.

This is an input element defined in html

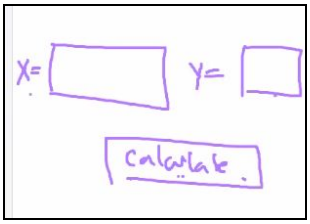
```
<input type="text" name="user" id="myinput" value="ddasd"></input>
```

This is the output in browser

- ‘ddasd’ is was the default value present in the input element. When clicked read input it showed ‘ddasd’. Then we wrote ‘updated ger’ and clicked on ‘Read Input’.
- ‘updated ger’ with red bg came after clicking on the read input button.
- Html automatically updates the latest value you inserted by the user in the value attribute.

Activity

- To create an adder..
- After pressing calculate, x+y should be shown.

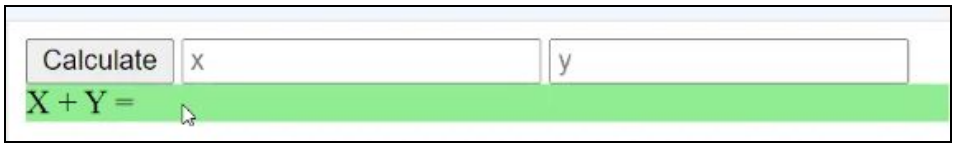



```
<button
onClick="someFunction()"
>
Calculate
</button>
<input type="number" name="x" id="x" placeholder="x">
<input type="number" name="y" id="y" placeholder="y">
```

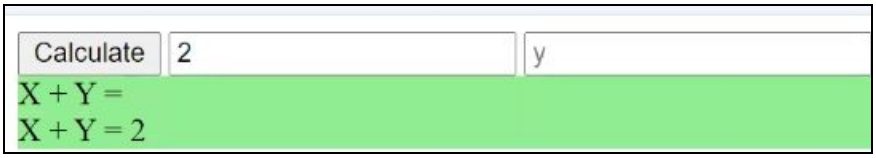
JS

```
function someFunction() {
    const x = document.getElementById("x");
    const y = document.getElementById("y");
    const element = document.createElement("div");

    const xval = x.value;
    const yval = y.value;
    element.innerHTML = "X + Y = " + (xval + yval);
    element.style.backgroundColor = "lightgreen";
    const containingElement = document.body;
    containingElement.appendChild(element);

}
```

After pressing calculate



- We don't give any input and after pressing calculate the output is empty.
- When one of them have data the output is as below



- Now data is given to both input, observe the output, the ans should be 5.

- This explains that even if the input is number, javascript is automatically converting the number in string and displaying it. So this string must be converted to number then perform addition then display the number.

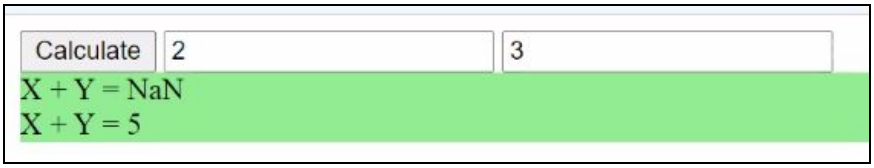
Modifying the function

uParsing the Integer from the string.

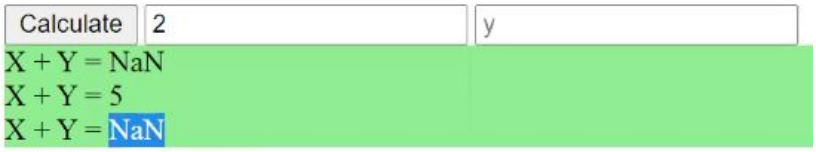
```
const xval = parseInt(x.value);
const yval = parseInt(y.value);
element.innerHTML = "X + Y = " + (xval + yval);
```

No input is given and clicked on calculate, the output is NaN

- If you give an empty string to parseInt() it will return NaN(Not a Number).
- The output is correct if you give numbers in the input field now.



- What if only one input is give, then the output would again be NaN.



- 'xval' got converted to number=2;
- 'yval' got converted to NaN, and 2+NaN=NaN.

Ways to handle this:

```
const xval = x.value.length == 0 ? 0 : parseInt(x.value);
const yval = y.value.length == 0 ? 0 : parseInt(y.value);
```

- This will store ‘0’ in the ‘xval’ and ‘yval’ if the length of the string is 0, else it will store integer parsed using parseInt().

```
function someFunction() {
    const x = document.getElementById("x");
    const y = document.getElementById("y");
    const element = document.createElement("div");
    const xval = x.value.length == 0 ? 0 : parseInt(x.value);
    const yval = y.value.length == 0 ? 0 : parseInt(y.value);
    element.innerHTML = "X + Y = " + (xval + yval);
    element.style.backgroundColor = "lightgreen";
    const containingElement = document.body;
    containingElement.appendChild(element);
}
```

The above solution will work in all cases as long as the value is int. It will fail in the case of double

- Use ‘parseFloat(x.value)’ for decimal input value.
- Cannot use ‘parseDouble(x.value)’. There is only parseFloat nothing like parseDouble.

The Converter function for numbers is ‘Number’.

```
const xval = Number(x.value);
const yval = Number(y.value);
```

JS

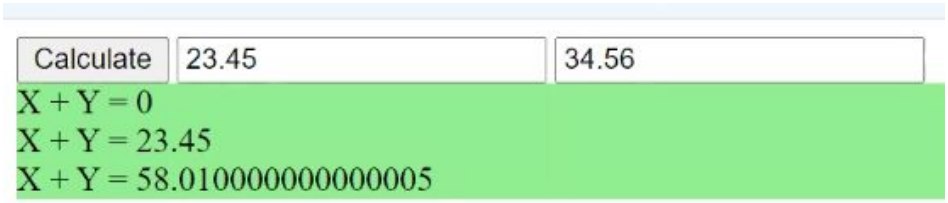
```
1   function someFunction() {
2       const x = document.getElementById("x");
3       const y = document.getElementById("y");
4       const element = document.createElement("div");
5       const xval = Number(x.value);
6       const yval = Number(y.value);
7       element.innerHTML = "X + Y = " + (xval + yval);
8       element.style.backgroundColor = "lightgreen";
9       const containingElement = document.body;
10      containingElement.appendChild(element);
11  }
```

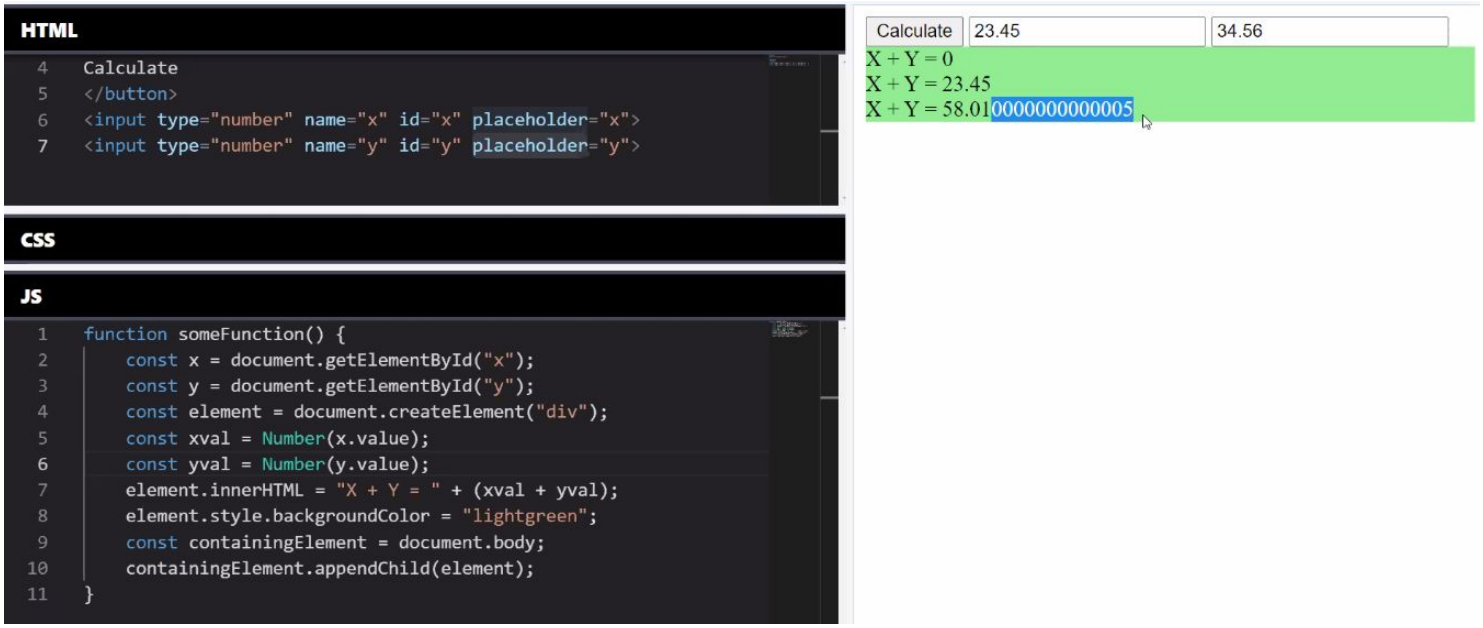
above solution will work in all cases

- Number(x.value) will convert to zero if the string is empty. It will convert to Integer if the string has an integer value. And It will convert to decimal if the string has a decimal point value.
- If no input

- If decimal point
- For single input

- For both the inputs



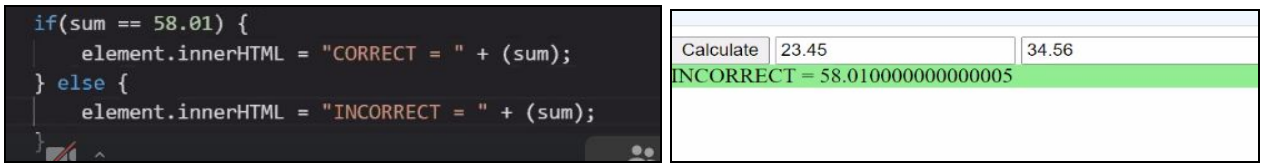


This long number output is a problem. The calculations are not precise for floating point numbers. It's a major issue. These mistakes happens every where, ruby, c/c++, python, etc. Now if you are working on a banking application then you are slowly reducing the persons money or increasing it.

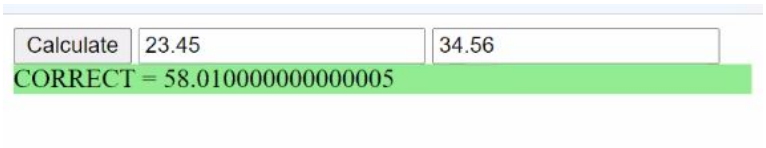
.toFixed(2);
This converts it to proper number but again converts it to string which is of no use which is not a number now.

Default value given

Proof that its incorrect. Its a condition of equality.



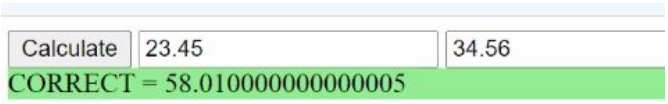
2 ways to fix this issue

- First way
  - Define very small value for e.g. EPSILON. Applying the following if condition and you can check now. This is a standard approach.
  -
```
const EPSILON = 0.000000001;
```
  -
```
const sum = xval + yval;
if(Math.abs(sum - 58.01) < EPSILON) {
    element.innerHTML = "CORRECT = " + (sum);
} else {
    element.innerHTML = "INCORRECT = " + (sum);
}
```
  - 

  -

- 2ndapproach

```
if(sum.toFixed(2) === "58.01") {
    element.innerHTML = "CORRECT = " + (sum);
} else {
    element.innerHTML = "INCORRECT = " + (sum);
}
```



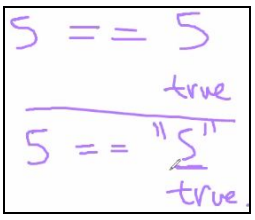
2nd approach is specific to javascript it will not apply to other language.

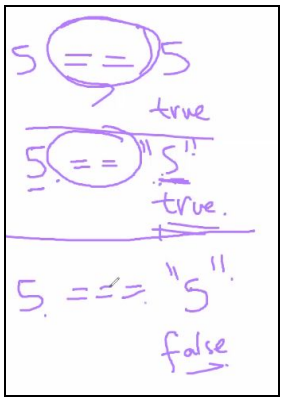
```
// truncation
// Number(sum.toFixed(2))
```

For truncating a number to certain number of decimal places

Problems discussion

- 5==5 will return true and 5==”5” will also return true in javascript which is a problem.





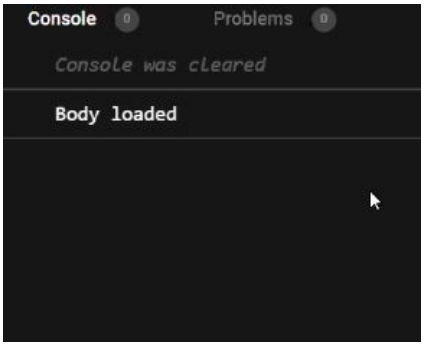
- It can be solved using ‘===’ (triple equal to) operator.
- 3rd approach checks the type as well as value that is why it is giving correct answer false.
- Important points:
  - Use const instead of var.
  - Use === operator instead of ==. always do comparisons using ===

Special inbuilt operation

BOM-Body object Model

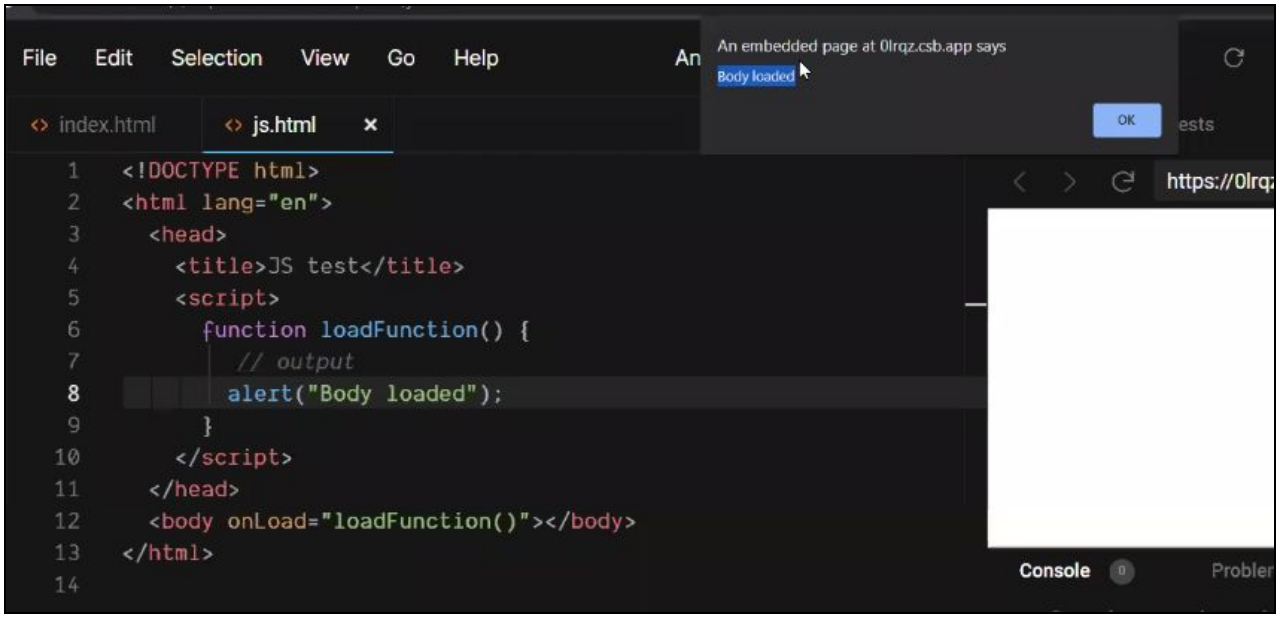
- Bom is available when u run JS in browser, these are inbuilt functions which relate to html. Otherwise not. So if you try to write document.getElementById(“”); this will fail in backend programming.
- In node js(For backend programming) we can’t use it.
- BOM functions/operations.
    - Load function
    - Alert function
    - Prompt function
    - Confirm function

Load function(onLoad=loadFunction())

- loadFunction() will be called as soon as the body of the webpage is loaded. onLoad attribute of body is used to assign the function to call when the webpage loads up the body.
- For e.g. when browser load the webpage loadFunction() is called and “Body loaded” prints.
- 

Alert Function(alert())

- Observe the prompt with message Body loaded. This prompt came because of alert() function, It stops the execution of program gives alert to the user with a message. Its a basic way to alert the user. This only available in the browser.



- Output is true or false.

document.getElementByClassName("class_name");

- This will get all elements which have the class="class_name".

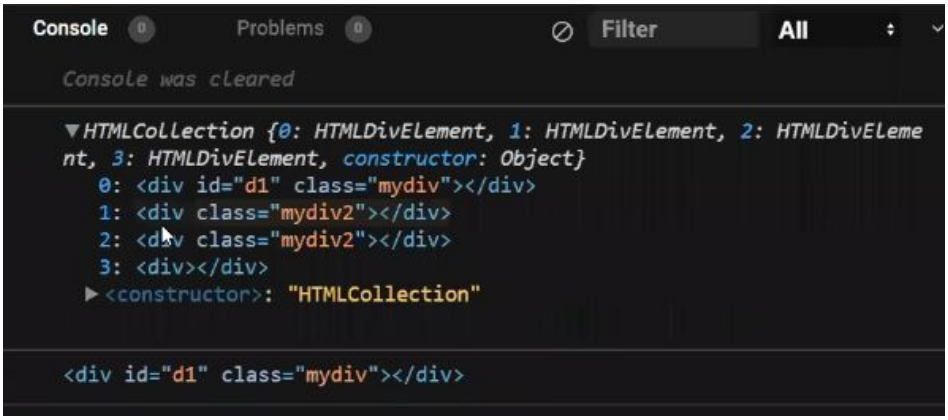
document.getElementById("id_name");

- This will get only one element, as id_name for each element is always unique. It is just showing id is better than class name because it will select a single element as id is unique.

document.getElementsByTagName("div");

- This will get all div tag elements present. Observe the appearance of all the divs in the console. element[0] is selecting 1st element out of the array of div elements.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JS test</title>
    <script>
      function loadFunction() {
        const element = document.getElementsByTagName("div");
        console.log(element);
        console.log(element[0]);
      }
    </script>
  </head>
  <body onLoad="loadFunction()">
    <div id="d1" class="mydiv"></div>
    <div class="mydiv2"></div>
    <div class="mydiv2"></div>
    <div></div>
  </body>
</html>
```

- 

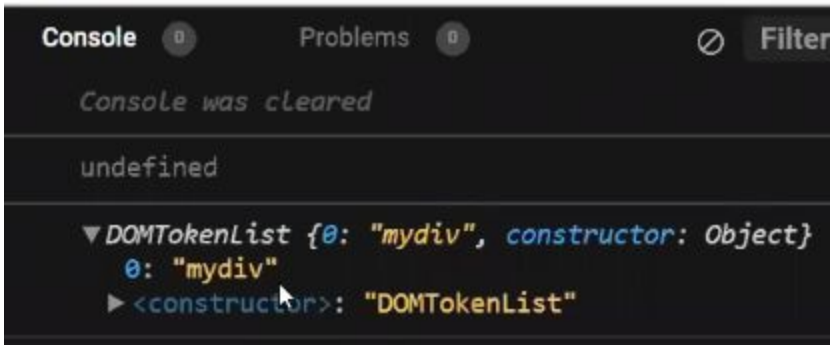
Setting a Class to given element
OR
Adding a Class to given element

```
function changeClass() {
    const element = document.getElementById("d1");
    console.log(element.class);
}
```

This will give not you default class. The proper term is classList

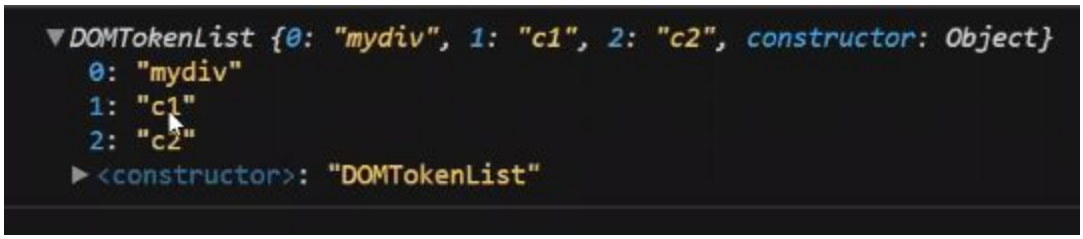
```
<script>
    function changeClass() {
        const element = document.getElementById("d1");
        console.log(element.classList);
    }
</script>
```

This gives you array of classes available for element with d1 id.



If we have other classes like below we can see the array is getting

```
<div id="d1" class="mydiv c1 c2"></div>
```



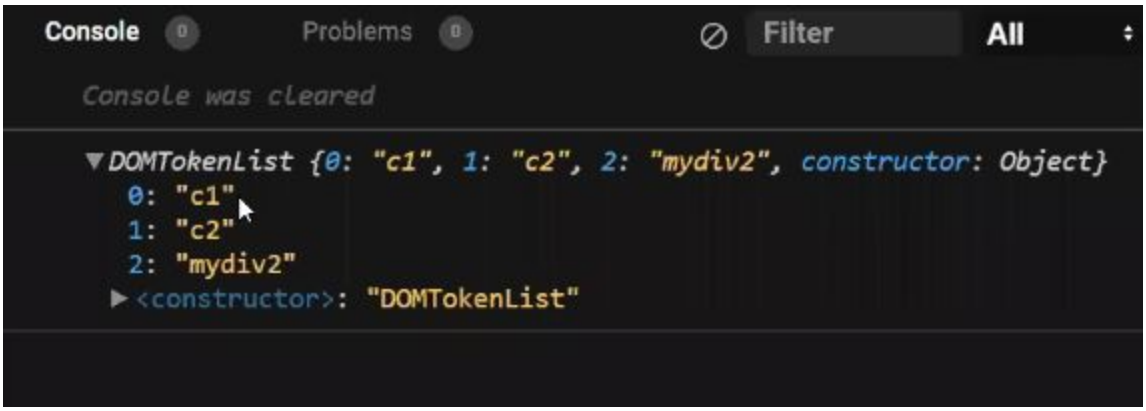
So this is how we access the class value.

Modifying the classes.

2 functions

1) element.classList.add(“mydiv2”)
2) element.classList.add(“mydiv”)

```
<script>
  function changeClass() {
    const element = document.getElementById("d1");
    // add class
    element.classList.add("mydiv2");
    element.classList.remove("mydiv");
    console.log(element.classList);
  }
</script>
```



saved