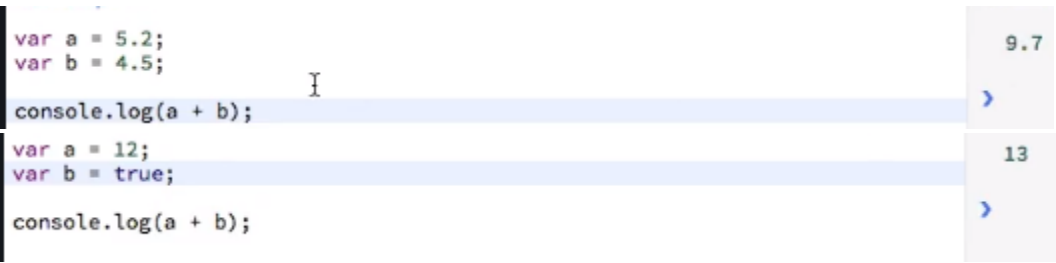


Section 2: Language Basics:


7. Using JavaScript - Inline HTML:


- **Q.1 Why script tag should be right before the closing body tag?**
- The browser goes from top to bottom through your file and builds up your webpage, now imagine you have a JavaScript code here which triggers some long running operation.
- **Ans.** In this case, the **browser would be occupied and couldn't continue loading your webpage below this script tag** here.
- Now that would of course mean that you might not see your webpage, even though technically it could be loaded, even with your script not being finished.
- Therefore by putting your scripts at the end of the page, you make sure that the **HTML document actually gets loaded before the script is executed**, which of course first results in a performance increase and second, it makes sure that if your script is trying to access some HTML elements in your page, those elements are actually there because the browser already ran over the specific code.
- **Q.2 How can you handle this case that a user visits your page which has JavaScript turned off?**
- **Ans.** It of course depends on your page, if your page depends heavily on JavaScript, **you might just want to show him a message telling him that he should turn off JavaScript or leave the page** because you're not able to provide a browsing experience.
- In other cases, it might not really matter but if you want to display such a message or any message, **you can use the noscript tags with any message between them and this message** here will only be displayed, if a user visits the page with JavaScript being turned off.

27. Operators – Addition:

- 
-

- If JavaScript can't add both variables in a mathematical way, it will have a look at the next way it knows, creating strings.

- 

```
var a = true;
var b = ' join';
console.log(a + b);
```
- 


```
var a = [1, 2];
var b = ' join';
console.log(a + b);
```
- JavaScript is really clever and trying hard to make it work if we join something with a string.

- So null is transferred into zero when using it in calculations.



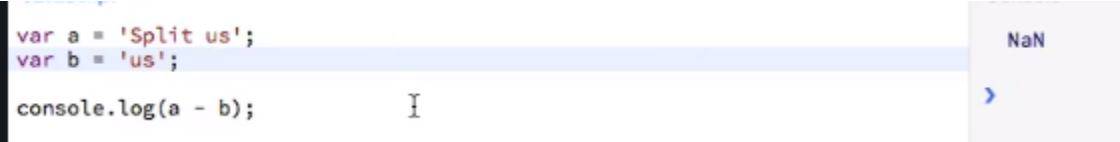
```
JavaScript ▾
var a = 12;
var b = null;
console.log(a + b);
```

- We get Not A Number(NaN) because undefined isn't a number and JavaScript doesn't transform it into zero or anything like that, instead it just recognizes that it can't do this calculation and, in such cases, it will return us not a number(NaN).




```
JavaScript ▾
var a = 12;
var b = undefined;
console.log(a + b);
```

28. Operators – Subtraction:



```
var a = 'Split us';
var b = 'us';
console.log(a - b);
```

- 

```
JavaScript ▾
var a = 12;
var b = '1';
console.log(a - b);
```
- So, when subtracting, JavaScript is not able to construct new strings out of that, therefore it falls back to the other way of subtraction it knows, using numbers and it tries if it is able to cast this string into a number which is of course possible with 1 and therefore, it's actually doing this calculation, even though the second argument, b, isn't a number.

29. Operators - Multiplication & Floating-Point Problems:

```
JavaScript
var a = 1.3;
var b = 2;

console.log(a * b);
```

Console

2.6

-
-

```
JavaScript
var a = 1.3;
var b = 2.2;

console.log(a * b);
```

Console

2.8600000000000003

-

- **That is very strange.** That clearly is not the correct answer, the answer should be 2.86 and all of that stuff shouldn't be there.
- **And that's a bug in JavaScript**, it has its issues with floating point numbers and it has the bug of adding this strange part here, so making a more complex number out of this than it actually is, which of course is a big problem if you were doing the following for example:

```
JavaScript
var a = 1.3;
var b = 2.2;

console.log(a * b);

if (a * b == 2.86) {
  console.log('true');
} else {
  console.log('false');
}
```

Console

2.8600000000000003

"false"

-

- **Now a good fix for this problem** is to put this calculation into parentheses here and use the **'toFixed' method** on it.
- Now this method here takes an argument where you define to how many decimal places you want to basically round or fix there.

```
JavaScript
var a = 1.3;
var b = 2.2;

console.log(a * b);

if ((a * b).toFixed(2) == 2.86) {
  console.log('true');
} else {
  console.log('false');
}
```

Console

2.8600000000000003

"true"

-

- This is a fix you should use when doing calculations with floating point numbers since it might be a source of very annoying and hard to trace down bugs because who would expect for JavaScript to behave this way.

```
JavaScript
var a = 2;
var b = '2.2';
console.log(a * b);
```

Console
4.4

- JavaScript is not able to build strings out of multiplication operations, **so it tries to cast a string into a number instead.**

```
JavaScript
var a = 12;
var b = 'join';
console.log(a * b);
```

Console
NaN

- **Multiply with null:**

```
JavaScript
var a = 12;
var b = null;
console.log(a * b);
```

Console
0

- **Multiply with infinity:**

```
JavaScript
var a = 12;
var b = Infinity;
console.log(a * b);
```

Console
Infinity

30. Operators - Division & Modulus:

```
JavaScript
var a = 12;
var b = '2';
console.log(a / b);
```

Console
6

- JavaScript is not able to create strings out of that, that only works for addition.

- **Division with floating points:**

```
File Add library Share HTML CSS JavaScript Console Output
JavaScript
var a = 3.3;
var b = 2.2;
console.log(a / b);
```

Console
1.4999999999999998

```
JavaScript
var a = 3.3;
var b = 2.2;
console.log((a / b).toFixed(2));
```

Console
"1.50"

- **Modulus operator:**

```
JavaScript ▾  
var a = 10;  
var b = 3;  
console.log(a % b);
```

Console
1

- **Division with Infinity:**

```
JavaScript ▾  
var a = 10;  
var b = Infinity;  
console.log(a / b);
```

Console
0

32. Operators - Important Rules:

```
JavaScript ▾  
console.log(NaN == NaN);
```

Console
false

- There is a rule in JavaScript and that rule **simply says not a number(NaN) is not like not a number(NaN). it's a built-in rule**, therefore if you check if both are not equal, you get true.

```
JavaScript ▾  
console.log(0 == null);
```

Console
false

- The reason for this is another rule in JavaScript, null can't be compared is the rule.
- **You can't compare null.** You can check if a variable is null in an if condition, as I said, null is treated like false then, but you can't compare anything to null and that's just another built-in rule.

```
JavaScript ▾  
console.log(null == undefined);
```

Console
true

- Well that's an exception from the rule, null can be compared to undefined.

JavaScript ▾	Console
<code>console.log(0 == undefined);</code>	false

- This is another JavaScript rule, undefined compared to anything is always false except when comparing it to null.
- null can't be compared, except with undefined and undefined will always return false except for when comparing it with null.

33. Operators – Boolean:

JavaScript ▾	Console
<code>console.log((1 == 1) (2 == 3) && (4 == 5));</code>	true
JavaScript ▾	Console
<code>console.log((1 == 1) && (2 == 3) (4 == 5));</code>	false
JavaScript ▾	Console
<code>console.log((1 == 1) && (2 == 3) (4 == 4));</code>	true
JavaScript ▾	Console
<code>console.log(((1 == 1) && (2 == 3)) (4 == 4));</code>	true
JavaScript ▾	Console
<code>var isTrue = true; console.log(!isTrue);</code>	false

35. Operators – Precedence:

JavaScript ▾	Console
<code>var a = 5; var b = 6;</code>	22
<code>console.log((a + b) * 2);</code>	

Table

The following table is ordered from highest (19) to lowest (0) precedence.

Precedence	Operator type	Associativity	Individual operators
19	Grouping	n/a	{ ... }
18	Member Access	left-to-right
	Computed Member Access	left-to-right	... [...]
	new (with argument list)	n/a	new ... (...)
17	Function Call	left-to-right	... (...)
	new (without argument list)	right-to-left	new ...
16	Postfix Increment	n/a	... ++
	Postfix Decrement	n/a	... --
15	Logical NOT	right-to-left	! ...
	Bitwise NOT	right-to-left	~ ...
	Unary Plus	right-to-left	+ ...
	Unary Negation	right-to-left	- ...

Section 8: Built-in Objects & Functions:

80. Timers & Intervals:

- **setInterval:** Now as setTimeout, this is registered on the window object, it's a method of the window object.
- We need to pass function as a first argument and then as a second argument, you provide the interval in milliseconds, for example 500 for half a second.
- Now what this will do is, it will run this function we specify as the first argument, every 500 milliseconds, so twice a second basically and **it won't stop by default.**

JavaScript

```

setInterval(function() {
  console.log('Ping!');
}, 500);

```

Console

```

"Ping!"
"Ping!"
"Ping!"
"Ping!"
"Ping!"
"Ping!"
"Ping!"

```

- Maybe you want to stop it at some time in the future. Can achieve like below:

<pre>JavaScript var interval = setInterval(function() { console.log('Ping!'); }, 500); setTimeout(function() { clearInterval(interval); }, 2500);</pre>	<pre>Console "Ping!" "Ping!" "Ping!" "Ping!" ></pre>
--	---

-

81. Transforming Formats & Values:

- **parseInt():** Available on windows object.
- **String to integer:**

<pre>JavaScript var a = '5'; console.log(parseInt(a));</pre>	<pre>Console 5 ></pre>
--	---------------------------

-

- **Hexadecimal to integer:**

<pre>JavaScript var a = 'FBB123'; console.log(parseInt(a, 16));</pre>	<pre>Console 16494883 ></pre>
---	----------------------------------

-

- **toString():** this is possible on each object in JavaScript.

<pre>JavaScript var a = 10; console.log(a.toString());</pre>	<pre>Console "10" ></pre>
--	------------------------------

-

- **toFixed():**

<pre>JavaScript var a = 10.3; console.log(a.toFixed(2));</pre>	<pre>Console "10.30" ></pre>
--	---------------------------------

-

82. String Functions:

- **String.length:**

<pre>JavaScript var string = 'Any text'; console.log(string.length);</pre>	<pre>Console 8 ></pre>
--	---------------------------

-

- ```
JavaScript
var string = 'Any text';
console.log(string[2]);
```

Console

"y"

- **charAt():**

- ```
JavaScript
var string = 'Any text';
console.log(string.charAt(2));
```

Console

"y"

- **concat():**

- ```
JavaScript
var string = 'Any text';
console.log(string.concat(' add a new string'));
```

Console

"Any text add a new string"

- **toUpperCase():**

- ```
JavaScript
var string = 'Any text';
console.log(string.toUpperCase());
```

Console

"ANY TEXT"

- **split():**

- ```
JavaScript
var string = 'Any text';
console.log(string.split(' '));
```

Console

["Any", "text"]

- **trim():**

- ```
JavaScript
var string = 'Any text   ';
console.log(string.trim());
```

Console

"Any text"

83. The Math Object:

- ```
JavaScript
var pi = Math.PI;
console.log(pi);
```

Console

3.141592653589793

- ```
JavaScript
var e = Math.E;
console.log(e);
```

Console

2.718281828459045

- **abs():**

JavaScript ▾

```
var a = -3;
```

```
console.log(Math.abs(a));
```

Console

3



-
- **round():**

JavaScript ▾

```
var a = 1.27;
```

```
console.log(Math.round(a));
```

Console

1



-
- **Ceil():**

JavaScript ▾

```
var a = 1.27;
```

```
console.log(Math.ceil(a));
```

Console

2



-
- **floor():**

JavaScript ▾

```
var a = 1.99;
```

```
console.log(Math.floor(a));
```

Console

1



-
- **exp():**

JavaScript ▾

```
var a = 2;
```

```
console.log(Math.exp(a));
```

Console

7.3890560989306495



-
- **log():**

JavaScript ▾

```
var e = Math.E;
```

```
console.log(Math.log(e));
```

Console

1



-
- **max():**

JavaScript ▾

```
console.log(Math.max(1, 100, 1000));
```

Console

1000



-
- **min():**

JavaScript ▾

```
console.log(Math.min(1, 100, 1000));
```

Console

1



- ```
JavaScript
console.log(Math.random());
```

```
Console
0.0839478669506446
```

## 84. The Date Object:

- ```
JavaScript
var today = new Date();
console.log(today);
```

```
Console
[object Date] { ... }
```
- ```
JavaScript
var today = new Date();
console.log(today.toString());
```

```
Console
"Tue Jun 07 2016 10:40:46 GMT+0200 (CEST)"
```
- ```
JavaScript
var today = new Date(2016, 5, 26);
console.log(today.toString());
```

```
Console
"Sun Jun 26 2016 00:00:00 GMT+0200 (CEST)"
```
- ```
JavaScript
var today = new Date(2016, 11, 32);
console.log(today.toString());
```

```
Console
"Sun Jan 01 2017 00:00:00 GMT+0100 (CET)"
```
- ```
JavaScript
var today = new Date('2016/05/20');
console.log(today.toString());
```

```
Console
"Fri May 20 2016 00:00:00 GMT+0200 (CEST)"
```

- Date.parse** : This is the amount of milliseconds since the first January in 1970, that is where the date starts in JavaScript, 1st January of 1970. So, this is where it starts, and this is the amount of **milliseconds which passed from 1st January of 1970 to this date here.**

- ```
JavaScript
console.log(Date.parse('2016/05/20'));
```

```
Console
1463695200000
```

- getDate():**

- ```
JavaScript
console.log(Date.parse('2016/05/20'));
var today = new Date();
console.log(today.getDate());
```

```
Console
1463695200000
7
```

- **getDay():**

JavaScript	Console
<code>console.log(Date.parse('2016/05/20'));</code>	1463695200000
<code>var today = new Date();</code>	2
<code>console.log(today.getDay());</code>	>

85. Regular Expressions:

JavaScript	Console
<code>var string = 'abc';</code>	[object RegExp] { ... }
<code>var pattern = /ab/;</code>	>
<code>console.log(pattern);</code>	

-
- **/ab/:** This is just a different syntax, but it creates an object, it creates a regular expression and we can use this pattern.

JavaScript	Console
<code>var string = 'abc';</code>	["ab"]
<code>var pattern = /ab/;</code>	>
<code>console.log(pattern.exec(string));</code>	

-
- So, this basically tells us that ab, our pattern here, was found in above array.

JavaScript	Console
<code>var string = 'ac';</code>	null
<code>var pattern = /ab/;</code>	>
<code>console.log(pattern.exec(string));</code>	

-
- Now we get null because we don't find this pattern in this string.

JavaScript	Console
<code>var string = 'abc';</code>	null
<code>var pattern = /ac/;</code>	>
<code>console.log(pattern.exec(string));</code>	

-

- **test():**

```
JavaScript
var string = 'abab';
var pattern = /ab/;
console.log(pattern.test(string));
```

Console
true

- **match():**

```
JavaScript
var string = 'abab';
var pattern = /ab/;
console.log(string.match(pattern));
```

Console
["ab"]

Section 9: Working with the Window and Document Object Model (DOM):

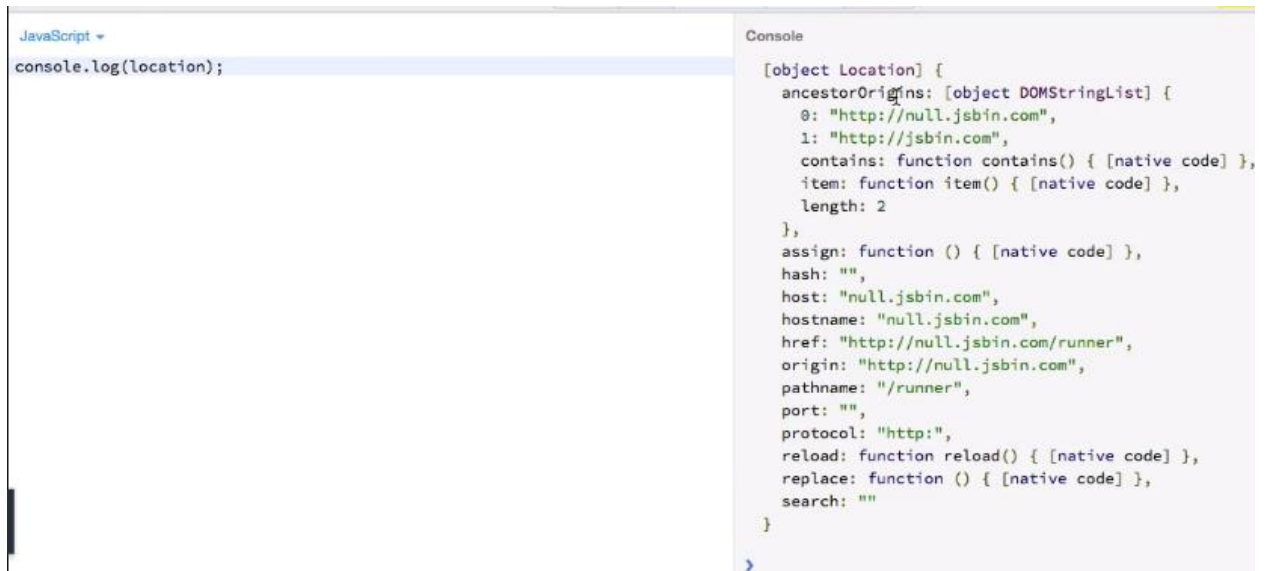
87. Introduction:

The screenshot shows the MDN website article for the Document Object Model (DOM). The page is titled "Document Object Model (DOM)" and includes a sidebar with "SEE ALSO" and "Interfaces" sections. The main content area describes the DOM as a programming interface for HTML, XML, and SVG documents. A yellow box labeled "Window" is positioned above the browser window, and a purple box labeled "Location" is positioned to the right of the browser window. A yellow box labeled "Document (DOM)" is positioned at the bottom right of the page.

88. The Window Object:

```
window.open('http://www.google.com');
```

89. The Location Object:



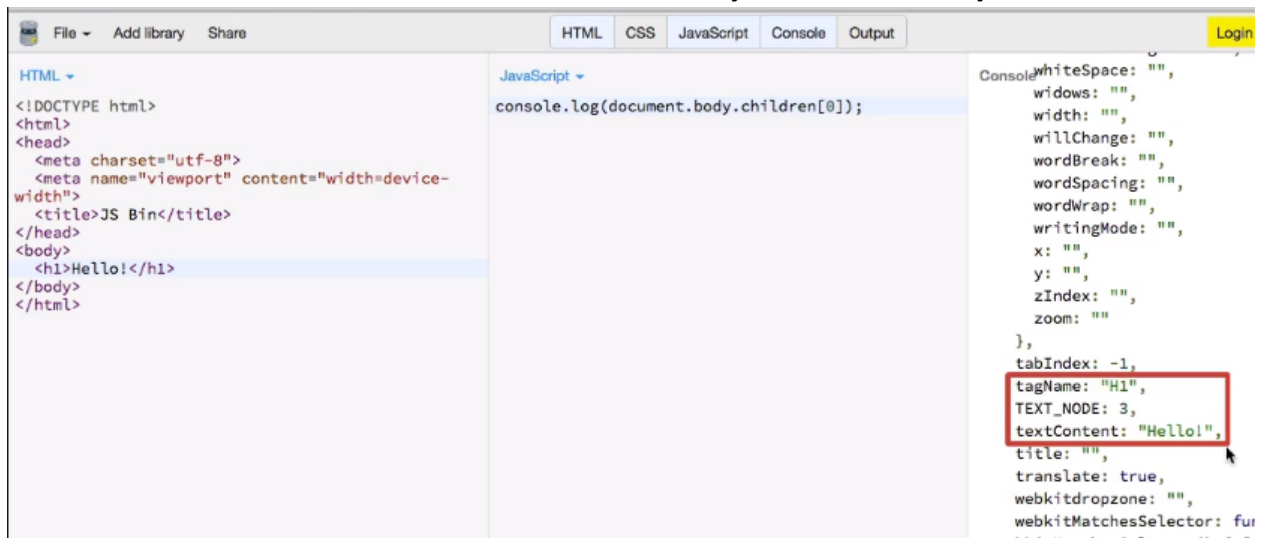
The screenshot shows a web browser's developer console. On the left, the JavaScript tab is active, displaying the code `console.log(location);`. On the right, the Console tab shows the output of this code, which is the `location` object. The object is a `Location` object with various properties and methods. The `ancestorOrigins` property is an `DOMStringList` object containing two origins: `"http://null.jsbin.com"` and `"http://jsbin.com"`. Other properties include `hash`, `host`, `hostname`, `href`, `origin`, `pathname`, `port`, `protocol`, `reload`, `replace`, and `search`.

```
JavaScript
console.log(location);

Console
[object Location] {
  ancestorOrigins: [object DOMStringList] {
    0: "http://null.jsbin.com",
    1: "http://jsbin.com",
    contains: function contains() { [native code] },
    item: function item() { [native code] },
    length: 2
  },
  assign: function () { [native code] },
  hash: "",
  host: "null.jsbin.com",
  hostname: "null.jsbin.com",
  href: "http://null.jsbin.com/runner",
  origin: "http://null.jsbin.com",
  pathname: "/runner",
  port: "",
  protocol: "http:",
  reload: function reload() { [native code] },
  replace: function () { [native code] },
  search: ""
}
```

90. The Document Object and How to Interact with It:

- Document object holds the complete DOM, the complete HTML code and a bunch of other properties, methods and events.
- Ex: **document.URL**, **document.title**, **document.body**, **document.body.children**



The screenshot shows a web browser's developer console. On the left, the HTML tab is active, displaying the HTML code of the page. The code includes a `<h1>Hello!</h1>` tag. In the middle, the JavaScript tab is active, displaying the code `console.log(document.body.children[0]);`. On the right, the Console tab shows the output of this code, which is the first child of the `document.body`, which is the `h1` element. The `h1` element is a `HTMLH1Element` object with properties like `tagName`, `TEXT_NODE`, `textContent`, `title`, `translate`, `webkitdropzone`, and `webkitMatchesSelector`. The `textContent` property is highlighted with a red box, showing the value `"Hello!"`.

```
HTML
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <h1>Hello!</h1>
</body>
</html>

JavaScript
console.log(document.body.children[0]);

Console
HTMLH1Element {
  whiteSpace: "",
  widows: "",
  width: "",
  willChange: "",
  wordBreak: "",
  wordSpacing: "",
  wordWrap: "",
  writingMode: "",
  x: "",
  y: "",
  zIndex: "",
  zoom: "",
  tabIndex: -1,
  tagName: "H1",
  TEXT_NODE: 3,
  textContent: "Hello!",
  title: "",
  translate: true,
  webkitdropzone: "",
  webkitMatchesSelector: function () { [native code] }
}
```

HTML

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
</head>
<body>
  <h1>Hello!</h1>
</body>
</html>

```

JavaScript

```

console.log(document.body.children[0].textContent);

```

Console

```

"Hello!"

```

HTML

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
</head>
<body>
  <h1>Hello!</h1>
</body>
</html>

```

JavaScript

```

document.body.children[0].textContent = 'Something else!';

```

Console

```


```

Output

Something else!

HTML

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
</head>
<body>
  <h1>Hello!</h1>
</body>
</html>

```

JavaScript

```

document.body.children[0].style.backgroundColor = 'red';

```

Console

```

textDecoration: "",
textIndent: "",
textOrientation: "",
textOverflow: "",
textRendering: "",
textShadow: "",
textTransform: "",
top: "",
touchAction: "",
transform: "",
transformOrigin: "",

```

Output

321px

Hello!

91. Traversing the DOM:

HTML

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
</head>
<body>
  <ul>
    <li><a href="#">Link</a></li>
    <li><a href="#">Link</a></li>
    <li><a href="#">Link</a></li>
  </ul>
</body>
</html>

```

JavaScript

```

console.log(document.body.children[0].children[0]);

```

Console

```

width: "",
willChange: "",
wordBreak: "",
wordSpacing: "",
wordWrap: "",
writingMode: "",
x: "",
y: "",
zIndex: "",
zoom: ""
},
tabIndex: -1,
tagName: "LI",
TEXT_NODE: 3,
textContent: "Link",
title: "",
translate: true,

```

HTML	JavaScript	Console
<pre><!DOCTYPE html> <html> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device-width"> <title>JS Bin</title> </head> <body> Link Link Link </body> </html></pre>	<pre>console.log(document.body.firstChild);</pre>	<pre> window.runnerWindow.proxyConsole.log } catch (error) { throw error; } // sourceURL=wupiseto.js </script></body>, previousElementSibling: null, previousSibling: null, PROCESSING_INSTRUCTION_NODE: 7, remove: function remove() { [native code] }, removeChild: function removeChild() { [native code] }, removeEventListener: function removeEventListener() { [native code] }, replaceChild: function replaceChild() { [native code] }, replaceData: function replaceData() { [native code] }, splitText: function splitText() { [native code] }, substringData: function substringData() { [native code] }, TEXT_NODE: 3, textContent: " ", wholeText: " "</pre>

HTML	JavaScript	Console
<pre><!DOCTYPE html> <html> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device-width"> <title>JS Bin</title> </head> <body> Link Link Link </body> </html></pre>	<pre>console.log(document.body.firstChild);</pre>	<pre> wordSpacing: "", wordWrap: "", writingMode: "", x: "", y: "", zIndex: "", zoom: "" }, tabIndex: -1, tagName: "UL", TEXT_NODE: 3, textContent: " Link Link Link ", title: "", translate: true, type: "", webkitdropzone: "", webkitMatchesSelector</pre>

HTML	JavaScript	Console
<pre><!DOCTYPE html> <html> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device-width"> <title>JS Bin</title> </head> <body> Link1 Link2 Link3 </body> </html></pre>	<pre>console.log(document.body.firstChild.firstChild.nextElementSibling);</pre>	<pre> width: "", willChange: "", wordBreak: "", wordSpacing: "", wordWrap: "", writingMode: "", x: "", y: "", zIndex: "", zoom: "" }, tabIndex: -1, tagName: "LI", TEXT_NODE: 3, textContent: "Link2", title: "", translate: true, type: "", value: 0, webkitdropzone: "",</pre>



92. Selecting Elements:

- As mentioned before, above way of doing is a bit difficult or you have these very long statements and you have to kind of go to your HTML code and count how many levels you have to go deep into to get the correct element, so that's not really the most convenient way of accessing your elements.
- So, a quicker way than this way here is to use some methods the document offers you, for example:



93. Selecting Elements with the Query Selector:

- querySelector**: allows me to select by ID, by class name, by tag name but all in one single method. It gives the first element which matches the query.

```

console.log(document.querySelector('h1'));

```

```

console.log(document.querySelector('.simple'));

```

```

console.log(document.querySelector('#easy'));

```

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <h1>Header!</h1>
  <ul>
    <li><a href="#">Link1</a></li>
    <li class="simple"><a href="#">Link2</a></li>
    <li class="simple"><a href="#">Link3</a></li>
  </ul>
</body>
</html>

```

```

console.log(document.querySelector('h1'));

```

```

console.log(document.querySelector('h1'));

```

```

console.log(document.querySelector('h1'));

```

```

webkitMatchesSelect
function
webkitMatchesSelect
{ [native code]
},
webkitRequestFullsc
function
webkitRequestFullsc
{ [native code]
},
webkitRequestFullSc
function
webkitRequestFullSc

```

94. Selecting Elements – Exercises:

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <h1 id="easy">Header!</h1>
  <ul>
    <li><a href="#">Link1</a></li>
    <li class="simple"><a href="#">Link2</a></li>
    <li class="simple"><a href="#">Link3</a></li>
  </ul>
</body>
</html>

```

```

document.querySelectorAll('.simple')
[1].style.backgroundColor = 'red';

```

```

Output 427px
Header!
• Link1
• Link2
• Link3

```

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <h1 id="easy">Header!</h1>
  <ul>
    <li><a href="#">Link1</a></li>
    <li class="simple"><a href="#">Link2</a></li>
    <li class="simple"><a href="#">Link3</a></li>
  </ul>
</body>
</html>

```

```

document.querySelector('.simple').firstElementChild.textContent = 'Hello';

```

```

Output 427px
Header!
• Link1
• Hello
• Link3

```

95. Creating and Inserting Elements:

```
HTML
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <h1 id="easy">Header!</h1>
  <ul>
    <li><a href="#">Link1</a></li>
    <li class="simple"><a href="#">Link2</a></li>
    <li><a href="#" class="simple">Link3</a></li>
  </ul>
</body>
</html>
```

```
JavaScript
var p = document.createElement('p');
p.textContent = 'A new paragraph!';
p.style.fontSize = '17px';

var a = document.querySelector('a');
a.appendChild(p);
```

Output

Header!

- Link1
- Link2
- Link3

- You see its part of the anchor tag and not really appended after it and that is because the append child will add it to the current element, so inside of it, so, it appended it after the text, inside of this anchor tag.
- However, we wanted it to add it after this link, not inside of it

```
HTML
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <h1 id="easy">Header!</h1>
  <ul>
    <li><a href="#">Link1</a></li>
    <li class="simple"><a href="#">Link2</a></li>
    <li><a href="#" class="simple">Link3</a></li>
  </ul>
</body>
</html>
```

```
JavaScript
var p = document.createElement('p');
p.textContent = 'A new paragraph!';
p.style.fontSize = '17px';

var li = document.querySelector('li');
li.appendChild(p);
```

Output

Header!

- Link1
- Link2
- Link3

A new paragraph!

```
HTML
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <h1 id="easy">Header!</h1>
  <ul>
    <li><a href="#">Link1</a></li>
    <li class="simple"><a href="#">Link2</a></li>
    <li><a href="#" class="simple">Link3</a></li>
  </ul>
</body>
</html>
```

```
JavaScript
var p = document.createElement('p');
p.textContent = 'A new paragraph!';
p.style.fontSize = '17px';

var li = document.querySelector('li');
var a = li.firstChild;
li.insertBefore(p, a);
```

Output

Header!

- A new paragraph!
- Link1
- Link2
- Link3

96. Deleting Elements:

HTML	JavaScript	Output
<pre><!DOCTYPE html> <html> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device- width"> <title>JS Bin</title> </head> <body> <h1 id="easy">Header!</h1> Link1 <li class="simple">Link2 Link3 </body> </html></pre>	<pre>var a = document.querySelectorAll('a')[1]; a.parentElement.removeChild(a);</pre>	Header! • Link1 • • Link3

- That's the only version that is supported in older browsers. With newer browsers this is not safe to be used unless you use some polyfills to add the functionality, you may just use the remove method which is of course much shorter.

HTML	JavaScript	Output 427px
<pre><!DOCTYPE html> <html> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device- width"> <title>JS Bin</title> </head> <body> <h1 id="easy">Header!</h1> Link1 <li class="simple">Link2 Link3 </body> </html></pre>	<pre>var a = document.querySelectorAll('a')[1]; a.remove();</pre>	Header! • Link1 • • Link3

- this works but again, not in all browsers.

HTML	JavaScript	Output 427px
<pre><!DOCTYPE html> <html> <head> <meta charset="utf-8"> <meta name="viewport" content="width=device- width"> <title>JS Bin</title> </head> <body> <h1 id="easy">Header!</h1> Link1 <li class="simple">Link2 Link3 </body> </html></pre>	<pre>var a = document.querySelectorAll('a')[1]; a.parentElement.removeChild(a);</pre>	Header! • Link1 • • Link3

97. Elements and Nodes:

- Now you also heard me say that you could use parent node instead of element.
- So, what's the difference here?

- Well basically, all the DOM things you see for example in the developer tools, in Chrome, are nodes and the DOM consists of nodes, it's built up of nodes and some nodes happen to be elements as well. So, all the nodes created from HTML elements.

99. Dialogs:

- alert(), confirm(), prompt().

100. DOM Properties & Methods:

- Node: <https://developer.mozilla.org/en-US/docs/Web/API/Node>

Section 10: Events:

103. The Event Object:

- This is a general event. JavaScript offers and then you have multiple specific events which kind of implement this top-level event and therefore all events in JavaScript share certain properties and methods

104. Event Handlers:

The screenshot illustrates the process of adding and overwriting event handlers in JavaScript. The top part shows a single handler for a button click, logging 'Clicked!'. The bottom part shows the same button with a new handler assigned, which logs 'Also clicked!'. This demonstrates how the previous handler is overwritten.

- So, the first event handler here is simply overwritten, it's no longer there and that is a problem because oftentimes, you'll need multiple listeners and you don't only want to have one handler.

- So, while this approach works fine as long as you only have one thing you want to execute, it certainly has its problems. Will address in next lecture.

105. Event Listeners:

The screenshot shows a code editor with the following JavaScript code:

```
JavaScript
var btn =
document.querySelector('button');

btn.addEventListener('click',
listener1);

function listener1() {
  console.log('Listener 1');
}

function listener2() {
  console.log('Listener 2');
}
```

The console output shows three instances of "Listener 1". The output panel on the right contains a "Click Me" button.

- Adding multiple listener:

The screenshot shows a code editor with the following JavaScript code:

```
JavaScript
var btn =
document.querySelector('button');

btn.addEventListener('click',
listener1);
btn.addEventListener('click',
listener2);

function listener1() {
  console.log('Listener 1');
}

function listener2() {
  console.log('Listener 2');
}
```

The console output shows "Listener 1" followed by "Listener 2". The output panel on the right contains a "Click Me" button.

- Removing one listener after some time:

The screenshot shows a code editor with the following JavaScript code:

```
JavaScript
var btn =
document.querySelector('button');

btn.addEventListener('click',
listener1);
btn.addEventListener('click',
listener2);

setTimeout(function() {
  btn.removeEventListener('click',
listener1);
}, 2000);

function listener1() {
  console.log('Listener 1');
}

function listener2() {
  console.log('Listener 2');
}
```

The console output shows "Listener 1" followed by multiple instances of "Listener 2". The output panel on the right contains a "Click Me" button and a "Run" button.

106. Event Behavior:

- Adding event listener on div(yellow) element:



- 'event' object passed in eventListener:

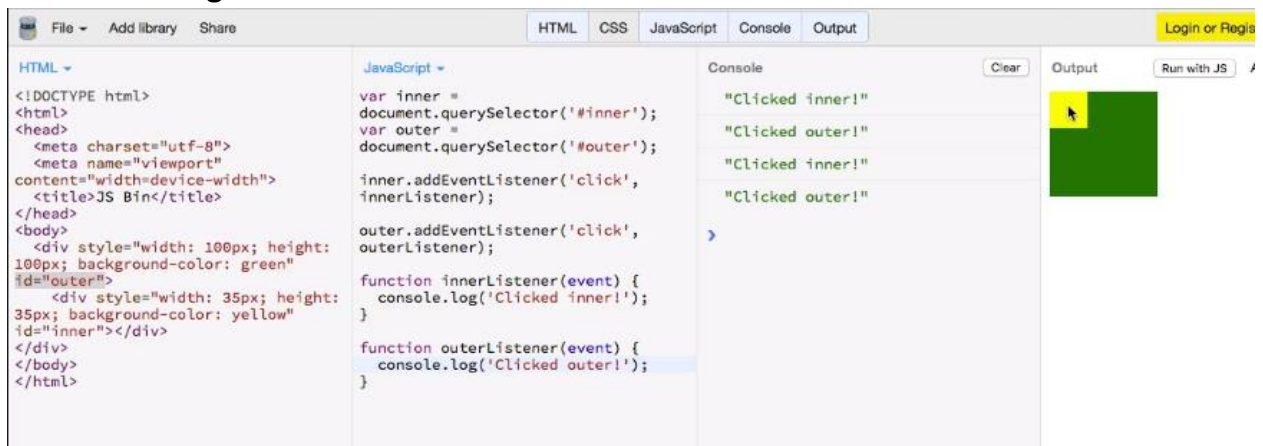
```
inner.addEventListener('click',
innerListener);

outer.addEventListener('click',
outerListener);

function innerListener(event) {
  console.log('Clicked inner!');
}

function innerListener(event) {
  console.log('Clicked inner!');
}
```

- Now one interesting question is where does this event argument here come from?
- JavaScript passes this event argument or this event object automatically to your listeners and this object will depend on the type of event.
- So, there are mouse click events, mouse enter events, different types of events to which you can listen, and all those types will have their own properties and methods
- **Event Bubbling:**



- I get inner and outer. Now you probably expected only to see inner and if I click the green one, you only see outer but why do we see both if I click the yellow one?
- The reason is that events in JavaScript by default propagate, which means if I click on the yellow one here, I also of course kind of click on the green one since the green one holds the yellow one.
- And the event propagates up from the yellow div which is the deepest nested element possible on which I click up to the other elements holding this clicked element, so the green one in this case. Therefore, both event listeners are fired since I technically click on both
- But what if I don't want to trigger the green event listener, I only want to trigger the yellow one.
- We have to use stopPropagation. By calling this method here, I make sure that the event doesn't bubble up.

```
function innerListener(event) {
  event.stopPropagation();
  console.log('Clicked inner!');
}
```

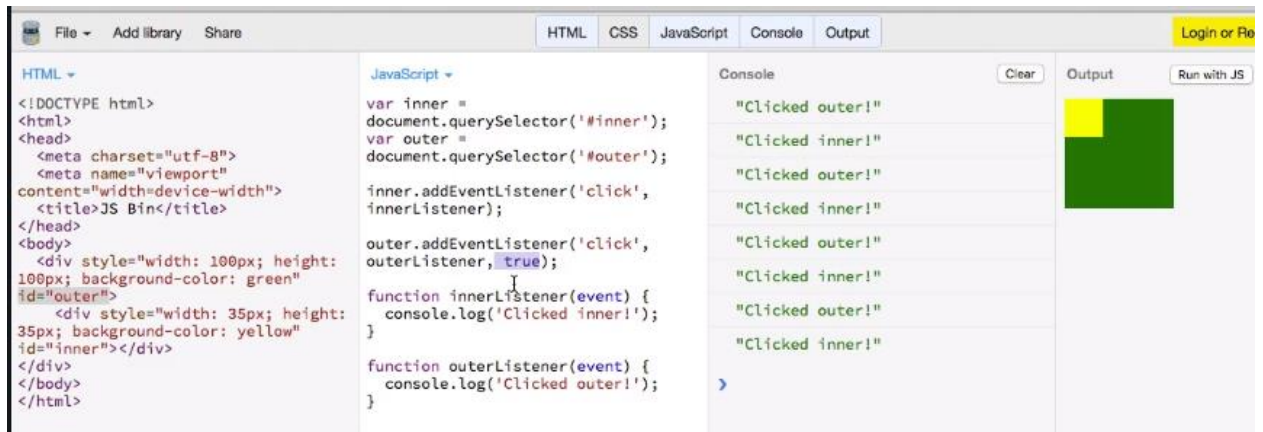
-

107. Event Object Properties:

- **event.target:** gives the element on which event triggered.
- **clientX:** gives x position.

108. Changing Propagation Order:

- What if you actually wanted to trigger the outer one first because even if you click on the yellow one, you might for some reason first want to check if the outer one should handle this event.
- So, you need to reverse the order so that you're not starting in the inner element but instead in the outer one?



- You can easily do this by adding another parameter to the event listeners here.
- So here to the outer listener, to the outer event, you can add true for example
- and if I now run this code, you'll see that clicked outer happens first.
- So, by adding this third parameter and setting it to true, you're telling JavaScript that this event listener should have the priority here and the parent should handle the event first before the child.

Section 11: JavaScript and Http Requests (AJAX):

111. Setup and Sending a GET Request:



- **Parsed response string through JSON.parse():** got array of objects.

JavaScript

```

var http = new XMLHttpRequest();
var url = 'http://jsonplaceholder.typicode.com/posts';
var method = 'GET';

http.open(method, url);
http.onreadystatechange = function() {
  if (http.readyState === XMLHttpRequest.DONE && http.status === 200) {
    console.log(JSON.parse(http.responseText));
  } else if (http.readyState === XMLHttpRequest.DONE && http.status !== 200) {
    console.log('Error!');
  }
};
http.send();

```

Console

```

[object Object] {
  title: "quas fugiat ut perspiciatis vero provident",
  userId: 10
}, [object Object] {
  body: "doloremque ex facilis sit sint culpa
soluta assumenda eligendi non ut eius
sequi ducimus vel quasi
veritatis est dolores",
  id: 98,
  title: "laboriosam dolor voluptates",
  userId: 10
}, [object Object] {
  body: "quo deleniti praesentium dicta non quod
aut est molestias
molestias et officia quis nihil
itaque dolorem quia",
  id: 99,
  title: "temporibus sit alias delectus eligendi possimus magni",
  userId: 10
}, [object Object] {
  body: "cupiditate quo est a modi nesciunt soluta
ipsa voluptas error itaque dicta in
autem qui minus magnam et distinctio eum
accusamus ratione error aut",
  id: 100,
  title: "at nam consequatur ea labore ea harum",
  userId: 10
}

```

- **112. POST Request:**

JavaScript

```

var http = new XMLHttpRequest();
var url = 'http://jsonplaceholder.typicode.com/posts';
var method = 'POST';

var data = 'title=Post%20Title&body=Body';

http.open(method, url);
http.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
http.onreadystatechange = function() {
  if (http.readyState === XMLHttpRequest.DONE && http.status === 201) {
    console.log(JSON.parse(http.responseText));
  } else if (http.readyState === XMLHttpRequest.DONE && http.status !== 201) {
    console.log('Error!');
  }
};
http.send(data);

```

Console

```

[object Object] {
  body: "Body",
  id: 101,
  title: "Post Title"
}

```