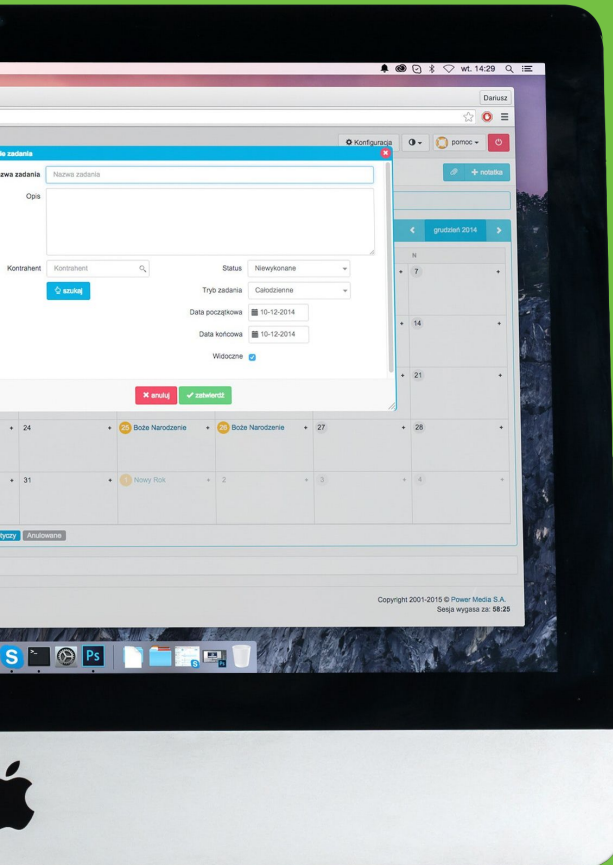


4CS017

Internet Software Architecture



ISA

Lecture Week 3

Functions, DOM and Events

This week's agenda

- Functions
- Arrow Function
- Anonymous Function
- IIFE (Immediately Invoked Function Expression)
- Higher Order Function
 - Map
 - Reduce
 - Filter
- ForEach Iteration
- HTML DOM
- Events

Functions

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- Functions are the main building block of a JavaScript Program.
- A function is a procedure – a set of statements to perform certain task or do a certain calculation.
- A function declaration consists of the **function** keyword, followed by:
 - The name of the function.
 - A list of parameters to the function, enclosed in a parenthesis and separated by a comma (,) in case of multiple parameters.
 - A JavaScript code statement enclosed inside a curly braces ({}).

Functions

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

A function should always return something, be it a value, an object or a Boolean value. If the **“return”** parameter is missed or is empty the function will return a message similar to **“undefined”**.

```
function myFunction(p1, p2){  
    return p1 * p2;  
}  
myFunction(2, 4); //Returns 8
```

Arrow Functions

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- Arrow functions were introduced in ES6.
- It is also known as a "fat arrow" function
- We use arrow function to write shorter function syntax.
- Arrow functions cannot be used as constructors, meaning that they cannot be called with the new keyword to create a new object.

```
hello = () => "Hello World";
```

```
hello = function(){  
    return "Hello World";  
}
```

```
const double = (x) => x * 3;
```

Arrow Functions

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- The Arrow function does not bind the “**this**” keyword.
- A regular function tends to bind “**this**” keyword to its preceding object.

```
const person = {  
  name: "Bishal",  
  greet: () => {  
    console.log(`Hello, my  
name is ${this.name}`);  
  }  
};
```

```
person.greet();
```

```
const person = {  
  name: "Bishal",  
  greet: function(){  
    console.log(`Hello, my  
name is ${this.name}`);  
  }  
};
```

```
person.greet();
```

Understanding Arrow Functions: Example...

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- Lets create a constructor function then create an instance of it.

```
function Counter(){  
    this.num = 0;  
}  
var a = new Counter();
```

- As you should know, in the constructor function's instance. "this" keyword is bound to the object being created, in this case "a".

```
console.log(a.num);  
// Returns 0
```

- That is why we get "0" when we console.log it.

Understanding Arrow Functions: Example...

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- Lets create a `Counter` constructor function using an arrow function

```
const Counter = () =>{  
    this.num = 0;  
}  
var a = new Counter();
```

- So,if you tried to use this arrow function as a constructor function with the `new` keyword,you would get a `TypeError` because this would not be defined as expected.

```
console.log(a.num);  
// TypeError
```

- For this reason, arrow functions are not used as constructor functions in JavaScript.

Understanding Arrow Functions: Example...

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- What if we want to increase the value of "**a.num**" every second?
- We can use the **setInterval()** function. It is a function that calls a function after a set number of milliseconds.

```
function Counter() {  
    this.num = 0;  
    this.timer = setInterval(function() {  
        this.num++;  
        console.log(this.num);  
    }, 1000);  
}  
var b = new Counter();
```

- The above code will return "**NaN**" every second.

Understanding Arrow Functions: Example...

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- As you can see, the result was not as the expected one. So what went wrong?
- Lets clear the annoying interval first with **clearInterval()** function.

```
clearInterval(b.timer);
```

- Our **setInterval()** function is being called on a declared object. It isn't being called with the "**new**" keyword. setInterval is just a normal function.
- The value of **this** in **setInterval** is being bound to the global object or the **window** object.

Understanding Arrow Functions: Example...

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- It was logging in **NaN** every second because it was looking for a property “**num**” in the global object, i.e. **window.num**, which does not exist.
- So how do we fix this? With an arrow function. Remember its property of not binding **this** keyword. This is where it makes things easy.
- With an arrow function, the “**this**” keyword binding keeps its original binding context.

Understanding Arrow Functions: Example

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- Lets rewrite our Counter function.

```
function Counter() {  
  this.num = 0;  
  this.timer = setInterval(() => {  
    this.num++;  
    console.log(this.num);  
  }, 1000);  
}  
var b = new Counter();
```

- As you try this code, you will notice that the console will now log an incremental number every second.



Anonymous Functions

What is an anonymous function?!

Anonymous Function

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- An anonymous function is a function that is declared without any named identifier to refer to it.
- An anonymous function is usually not accessible after its initial creation.
- One common use for anonymous functions is as arguments to other functions.

```
setTimeout(function(){  
    console.log("I am Anonymous!!");  
});
```


Anonymous Function

It is usually defined inline as an argument to another function or assigned to a variable.

Here is another example of anonymous function.

```
var add = function(x,y){  
    return x+y;  
});  
var result=add(4,5);  
console.log(result);
```

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

```
(function () {} ) ();
```

IIFE (Immediately Invoked Function Expression)

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order


Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined.
- It is a design pattern which is also known as a **Self-Executing Anonymous Function**.
- We simply want to call a function in order to get an output and never want to use it again and don't want our program to ever be able to accidentally access it.

```
function anonymous(){  
    console.log("I am  
Anonymous!!");  
})();  
  
anonymous();
```



```
(function(){  
    console.log("I am  
Anonymous!!");  
})();
```





```
[🐮, 🥔, 🐔, 🌽].map(cook) ⇒ [🍔, 🍟, 🍗, 🍲]  
[🍔, 🍟, 🍗, 🍲].filter(isVegetarian) ⇒ [🍟, 🍲]  
[🍔, 🍟, 🍗, 🍲].reduce(eat) ⇒ 🤮
```

Higher Order Functions : Array Map

- 1. Functions
 - 1.1. Arrow Functions
 - 1.2. Anonymous Functions
 - 1.3. Higher Order Functions : Array Map**
 - 1.4. Higher Order Functions: Array Filter
 - 1.5. Higher Order Functions: Array Reduce
 - 1.6. Higher Order Functions: Array ForEach

- The **map()** method creates a new array with the results of calling a function for every array element.
- The **map()** method calls the provided function once for each element in an array, in order.
- This method does not change the original array.
- **map()** does not execute the function for array elements without values.

Higher Order Functions: Array Map Parameters

- 1. Functions
 - 1.1. Arrow Functions
 - 1.2. Anonymous Functions
 - 1.3. Higher Order Functions : Array Map**
 - 1.4. Higher Order Functions: Array Filter
 - 1.5. Higher Order Functions: Array Reduce
 - 1.6. Higher Order Functions: Array ForEach

```
array.map(function(currentValue, index, arr),  
thisValue)
```

- **currentValue**: Required. The value of the current element
index: Optional. The array index of the current element
arr: Optional. The array object the current element belongs to
thisValue: Optional. A value to be passed to the function to be used as its "this" value.

Higher Order Functions : Array Map Example

- 1. Functions
 - 1.1. Arrow Functions
 - 1.2. Anonymous Functions
 - 1.3. Higher Order Functions : Array Map**
 - 1.4. Higher Order Functions: Array Filter
 - 1.5. Higher Order Functions: Array Reduce
 - 1.6. Higher Order Functions: Array ForEach

```
> var numbers = [65, 44, 12, 4];  
   var newarray = numbers.map(myFunction)  
  
   function myFunction(num) {  
     return num * 10;  
   }  
  
   console.log(newarray);  
  
▶ (4) [650, 440, 120, 40]
```


Higher Order Functions : Array Filter

1. Functions

1.1. Arrow Functions

1.2. Anonymous Functions

1.3. Higher Order

Functions : Array Map

1.4. Higher Order

Functions: Array Filter

1.5. Higher Order

Functions: Array Reduce

1.6. Higher Order

Functions: Array ForEach

- The **filter()** method creates an array filled with all array elements that passes a test (provided as a function).
- **filter()** does not execute the function for array elements without values.
- **filter()** does not change the original array.

Higher Order Functions : Array Filter Parameters

```
array.filter(function(currentValue, index, arr),  
thisValue)
```

- **currentValue**: Required. The value of the current element
index: Optional. The array index of the current element
arr: Optional. The array object the current element belongs to
thisValue: Optional. A value to be passed to the function to be used as its "this" value.

1. Functions
1.1. Arrow Functions
1.2. Anonymous Functions
1.3. Higher Order
Functions : Array Map
**1.4. Higher Order
Functions: Array Filter**
1.5. Higher Order
Functions: Array Reduce
1.6. Higher Order
Functions: Array ForEach

Higher Order Functions : Array Filter Example

- 1. Functions
- 1.1. Arrow Functions
- 1.2. Anonymous Functions
- 1.3. Higher Order Functions : Array Map
- 1.4. Higher Order Functions: Array Filter**
- 1.5. Higher Order Functions: Array Reduce
- 1.6. Higher Order Functions: Array ForEach

```
> var numbers = [65,44,12,4];  
   var newarray = numbers.filter(myFunction)  
  
   function myFunction(num){  
       return num > 40;  
   }  
   console.log(newarray);  
  
▶ (2) [65, 44]
```

- 1. Functions
 - 1.1. Arrow Functions
 - 1.2. Anonymous Functions
 - 1.3. Higher Order Functions : Array Map
 - 1.4. Higher Order Functions: Array Filter
 - 1.5. Higher Order Functions: Array Reduce**
 - 1.6. Higher Order Functions: Array ForEach

Higher Order Functions : Array Reduce

- The **reduce()** method reduces the array to a single value.
- The **reduce()** method executes a provided function for each value of the array (from left-to-right).
- The return value of the function is stored in an accumulator (result/total).
- **reduce()** does not execute the function for array elements without values.
- this method does not change the original array.

Higher Order Functions : Array Reduce Parameters

- 1. Functions
 - 1.1. Arrow Functions
 - 1.2. Anonymous Functions
 - 1.3. Higher Order Functions : Array Map
 - 1.4. Higher Order Functions: Array Filter
 - 1.5. Higher Order Functions: Array Reduce**
 - 1.6. Higher Order Functions: Array ForEach

```
array.reduce(function(total, currentValue, currentIndex, arr),  
initialValue)
```

- **total:** Required. The initialValue, or the previously returned value of the function
- currentValue:** Required. The value of the current element
- currentIndex:** Optional. The array index of the current element
- arr:** Optional. The array object the current element belongs to
- initialValue:** Optional. A value to be passed to the function as the initial value

Higher Order Functions : Array Reduce Example

- 1. Functions
- 1.1. Arrow Functions
- 1.2. Anonymous Functions
- 1.3. Higher Order Functions : Array Map
- 1.4. Higher Order Functions: Array Filter
- 1.5. Higher Order Functions: Array Reduce**
- 1.6. Higher Order Functions: Array ForEach

```
> var numbers = [175, 50, 25];  
   var val = numbers.reduce(myFunc);  
   function myFunc(total, num) {  
     return total - num;  
   }  
   console.log(val);
```

100

Higher Order Functions : Array ForEach

- 1. Functions
 - 1.1. Arrow Functions
 - 1.2. Anonymous Functions
 - 1.3. Higher Order Functions : Array Map
 - 1.4. Higher Order Functions: Array Filter
 - 1.5. Higher Order Functions: Array Reduce
 - 1.6. Higher Order Functions: Array ForEach**

- The **forEach()** method calls a function once for each element in an array, in order.
- The function is not executed for array elements without values.
- The provided function may perform any kind of operation on the elements of the given array.
- The return value of this function is always undefined. This function may or may not change the original array provided as it depends upon the functionality of the argument function.

Higher Order Functions : Array ForEach Parameters

```
array.forEach(function(currentValue, index, arr),  
thisValue)
```

- **currentValue**: Required. The value of the current element
index: Optional. The array index of the current element
arr: Optional. The array object the current element belongs to
thisValue: Optional. A value to be passed to the function to be used as its "this" value.

1. Functions
1.1. Arrow Functions
1.2. Anonymous Functions
1.3. Higher Order
Functions : Array Map
1.4. Higher Order
Functions: Array Filter
1.5. Higher Order
Functions: Array Reduce
**1.6. Higher Order
Functions: Array ForEach**

Higher Order Functions : Array ForEach Example

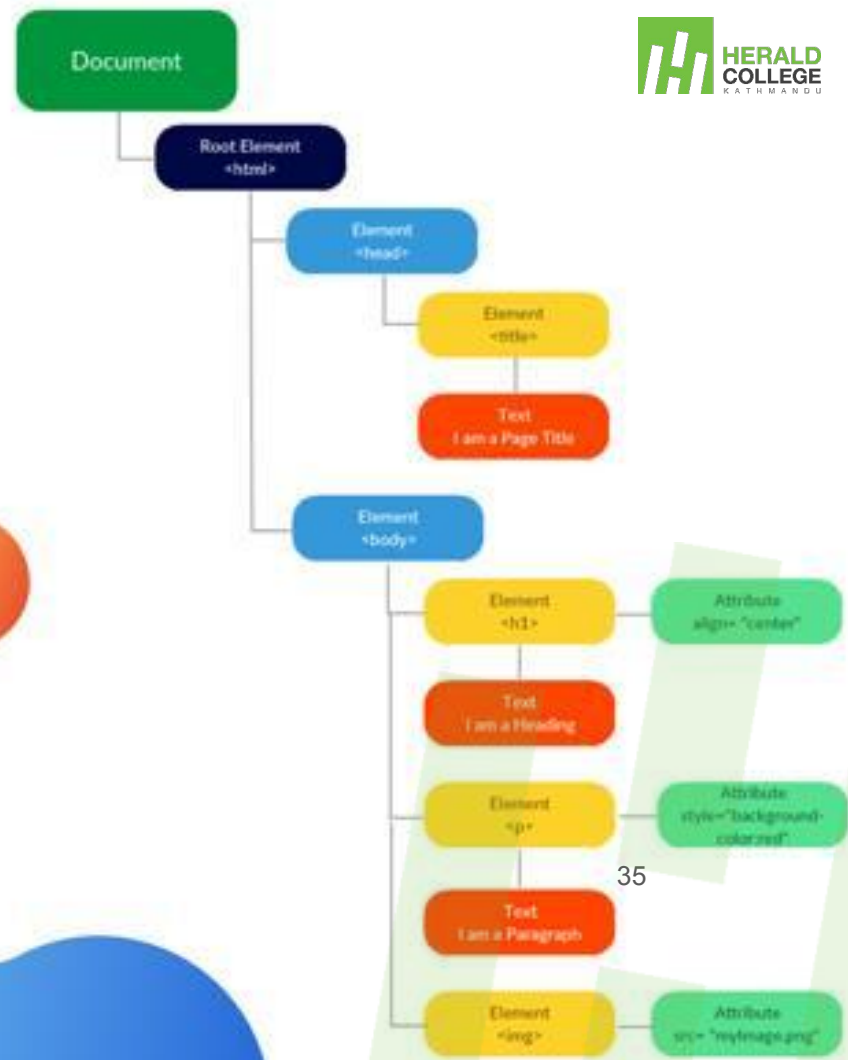
- 1. Functions
- 1.1. Arrow Functions
- 1.2. Anonymous Functions
- 1.3. Higher Order Functions : Array Map
- 1.4. Higher Order Functions: Array Filter
- 1.5. Higher Order Functions: Array Reduce
- 1.6. Higher Order Functions: Array ForEach**

```
> const items = [1, 29, 47];  
   const copy = [];  
  
   items.forEach(function(item){  
     copy.push(item*item);  
   });  
   console.log(copy);  
  
▶ (3) [1, 841, 2209]
```

RELAX
REFRESH
RECHARGE

be back in 10 minutes...

HTML DOM



Understanding DOM...

- DOM stands for **Document Object Model**.
- It is the data representation of the objects that comprise the structure and content of a document on the web.
- It is a programming interface for valid HTML and well structured XML documents.
- It defines the logical structure of documents and the way a document is accessed and manipulated.
- DOM is a way to represent the webpage in the structured hierarchical way.

2. HTML DOM

2.1. Understanding DOM

Understanding DOM...

- DOM can be thought of as a Tree of different html elements, its properties, methods and events.
- It is called an “Object Model”, because the document is modeled using objects. The model not only includes the structure but also the behavior of a document.
- JavaScript uses this Object Model to interact, manipulate and communicate with the structure of an html page.

2. HTML DOM

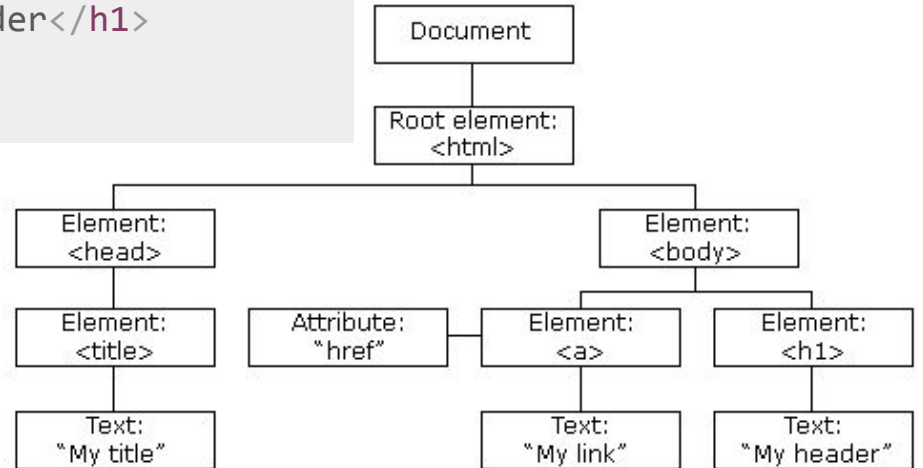
2.1. Understanding DOM


Understanding DOM

2. HTML DOM

2.1. Understanding DOM

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Title</title>
  </head>
  <body>
    <a href="#">My Link</a>
    <h1>My Header</h1>
  </body>
</html>
```




Tools · [chrisdavidelli](#) | Sign out
Document

[WEB TECHNOLOGIES](#) · [MOZILLA DOCS](#) · [DEVELOPER TOOLS](#) · [FEEDBACK](#)

[LANGUAGES](#)

[EDIT](#)

[MDN > Learn web development](#)

Learn web development

SEE ALSO

Complete beginners start here!

- Getting started with the Web

HTML — Structuring the Web

- Introduction to HTML

Media and embedding

CSS — Styling the Web

- Introduction to CSS
- Styling web
- Styling boxes
- CSS layout

JavaScript — Dynamic client-side scripting

- JavaScript first steps

Welcome to the MDN Learning Area. This set of articles aims to provide complete beginners to web development with all they need to start coding simple websites.

The aim of this area of MDN is not to take you from "beginner" to "expert" but to take you from "beginner" to "comfortable". From there you should be able to start making your way, learning from the rest of MDN, and other intermediate to advanced resources that assume a lot of previous knowledge.

If you are a complete beginner, web development can be challenging — we will hold your hand and provide enough detail for you to feel comfortable and learn the topics properly. You should feel at home whether you are a student learning web development for your own or as part of a class, a teacher looking for class materials, a hobbyist, or someone who just wants to understand more about how web technologies work.

Important: the content in the Learning Area is being added to regularly. If you have questions regarding topics you'd like to see covered or find interesting, use the [Contact us](#) section below for information on how to get in touch.

VIRTUAL DOM



Virtual DOM...

3. Virtual DOM

- It is a virtual tree which is kept in the browser's memory and updated in runtime.
- It is an object which is very similar to the JavaScript Object.
- Virtual DOM is simply a representation of JavaScript DOM object.
- It has the same methods, properties and events like the original DOM.
- It is simple a concept used by many JavaScript frameworks and libraries, one of which is React JS.

Virtual DOM

3. Virtual DOM

- While the DOM re-renders every node even when there is a small change in it. Virtual DOM looks for the node diff.
- The VDOM looks for the node diff and only changes the required DOM nodes, resulting in better performance and fast load of the page.



```
document.querySelector();  
document.querySelectorAll();
```

JavaScript Selectors...

4. JavaScript Selectors

- JavaScript uses the CSS syntax to select and manipulate HTML elements in a DOM.
- Selectors are used to "find" (select) HTML elements based on their tag name, id, classes, types, attributes, values of attributes and much more.
- In JavaScript, we can select elements in multiple ways as follows:
 - **getElementById(), getElementsByClassName(), getElementsByTagName(), querySelector(), querySelectorAll()**
- We also need to tell JavaScript from where to select these elements. In most cases it is always the "**document**" object.

JavaScript Selectors...

4. JavaScript Selectors

- **getElementById()** and **querySelector()** returns (selects) a single DOM.
- Where as other selectors either returns an **HTML collection** or an **Array of Node Lists**, depending upon which selector type you use.
- Depending upon which value the selectors return, we can manipulate the HTML DOM.
- **querySelector()** and **querySelectorAll()** are the most preferred JS selector to use while programming.



EVENTS

Understanding Events in JavaScript...

4. JavaScript Events

- HTML events are things that happens to an HTML element.
- JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.
- When the page loads, it is called an event. When the user clicks a button, that click too is an event.
- We can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Exploring JavaScript Events with Event Handlers...

4. JavaScript Events

- Events in JavaScript are handled by an **Event Handlers**.
- They are JavaScript code that are not added inside the **<script>** tags, but rather, inside the html tags, that execute JavaScript when something happens, such as pressing a button, moving your mouse over a link, submitting a form etc.

```
<a href="http://google.com"  
onClick="alert('hello!')">Google</a>
```

- Here, **onClick** is the JavaScript event handler.
- While this is a good example of getting to know how events work, JavaScript also has a method called "**Event Listener**". Lets Explore.

Exploring JavaScript Events with Event Handlers...

4. JavaScript Events

- An **event listener** is a method that attaches an **event handler** to a specific element.
- This way we do not always have to rely on writing our codes in HTML itself.
- The best way to work around with events is using an event listener method.
- To use an event listener, we must first select an element on which to attach the event.

Exploring JavaScript Events with Event Handlers

- Now that we know how to select an element, let's look at the following example:

```
<a href="#" id="myBtn">Show  
Alert</a>
```

```
document.getElementById("myBtn").addEventListener("click", () =>  
alert("hello!"));
```

- We created an **<a>** element and assigned an **ID** to it. We then selected that element using the **getElementById()** method in JavaScript. Finally we attached an event handler (click) to alert "hello!".
- Now, whenever a user clicks the button, it will show an alert with the message "hello!".

4. JavaScript Events

JavaScript Events: Most used event handlers

- There are numerous event handlers for you to explore. Below are listed some of the most used handlers for you too look into:
 - onClick
 - ondblclick
 - onkeypress
 - onkeyup
 - onmouseover
 - onmouseout
 - onsubmit

4. JavaScript Events



Before you come for Lab, Research!!

- JavaScript Functions
- Higher Order Functions
- [HTML DOM Events](#)
- [HTML DOM Elements](#)

Learn more

**well that's the
end**

Goodbye 20

