Module -2

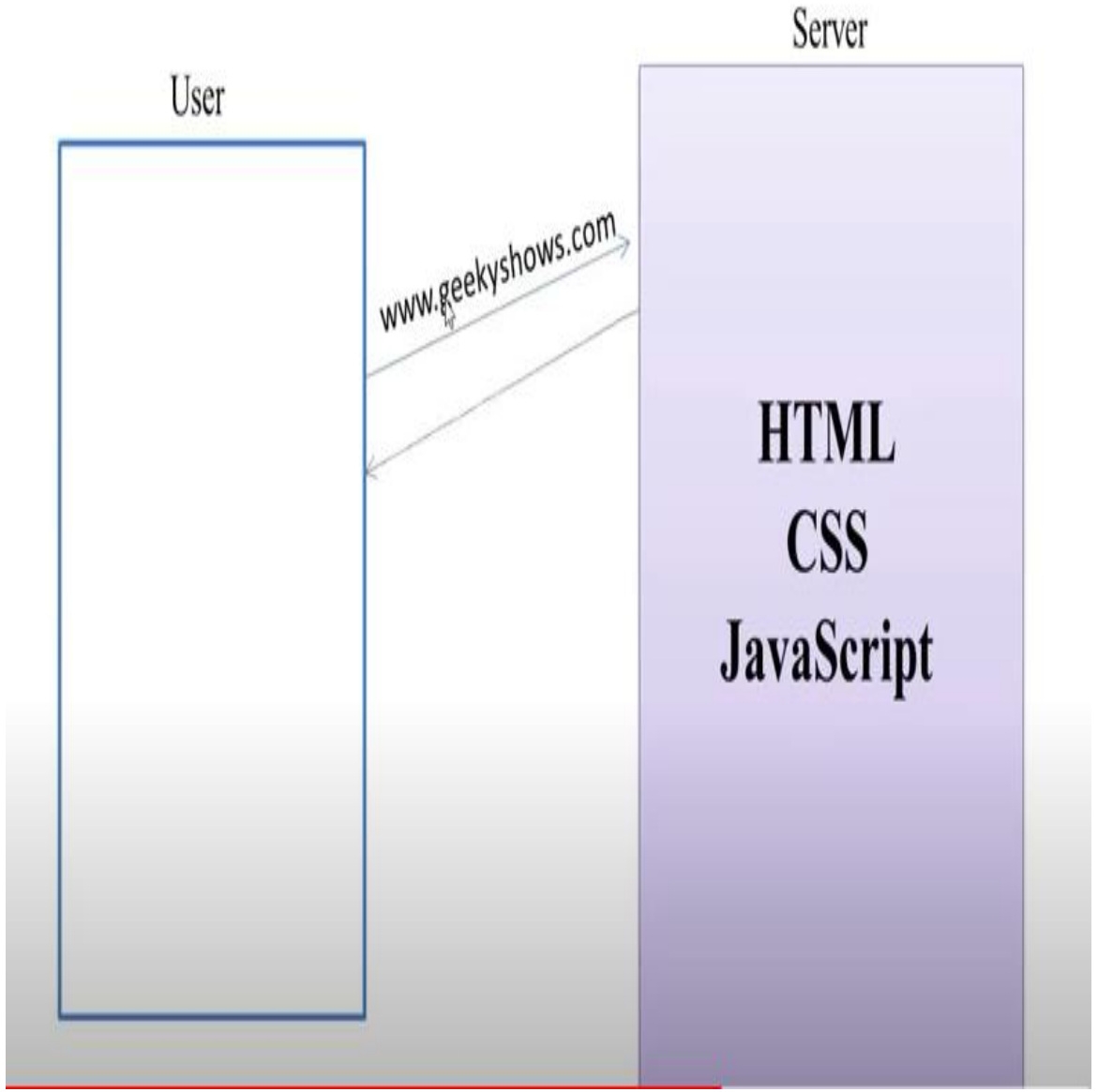
JavaScript

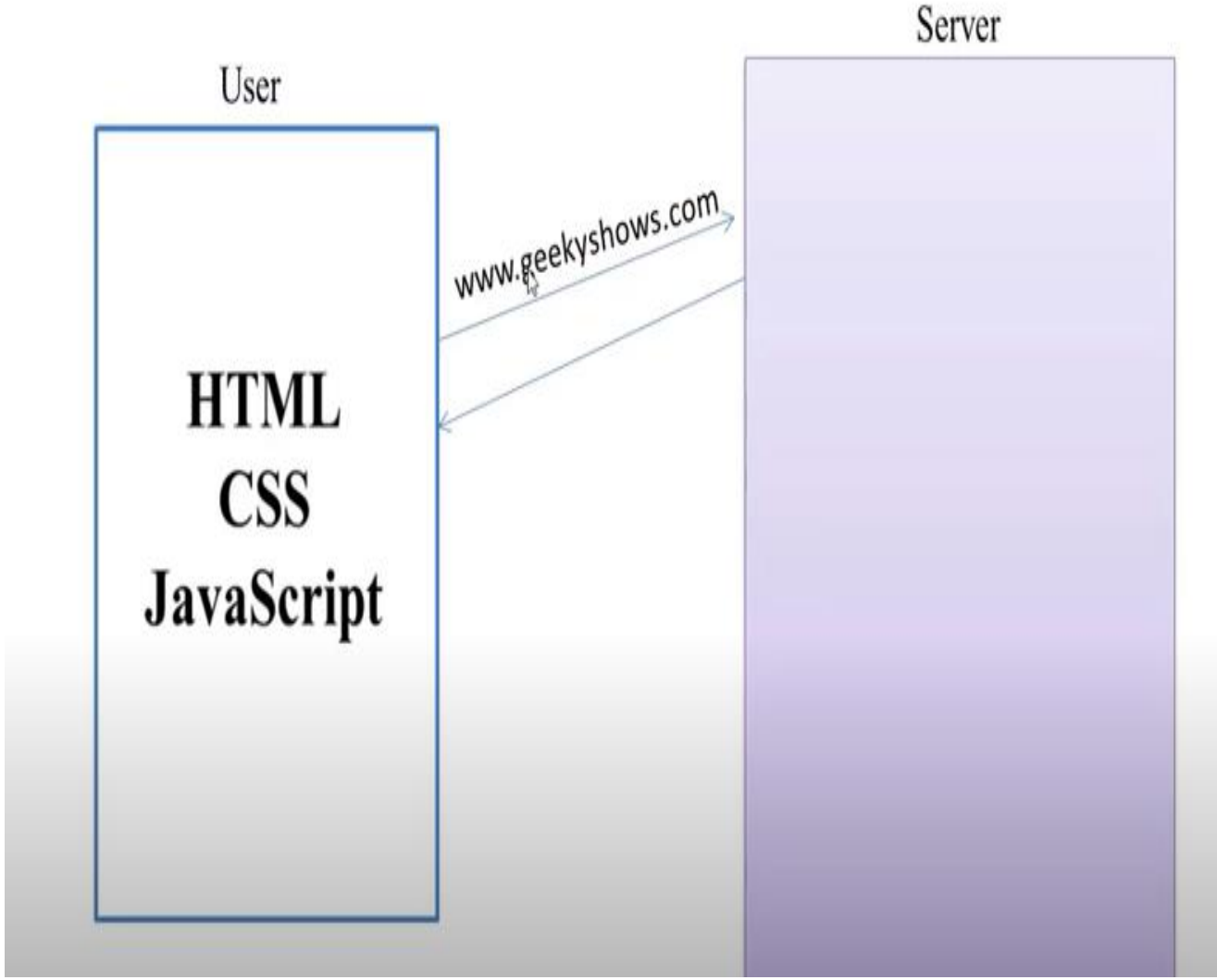
What is JavaScript ?

JavaScript is the programming language of HTML and the Web. It makes web page dynamic. It is an interpreted programming language with object-oriented capabilities.

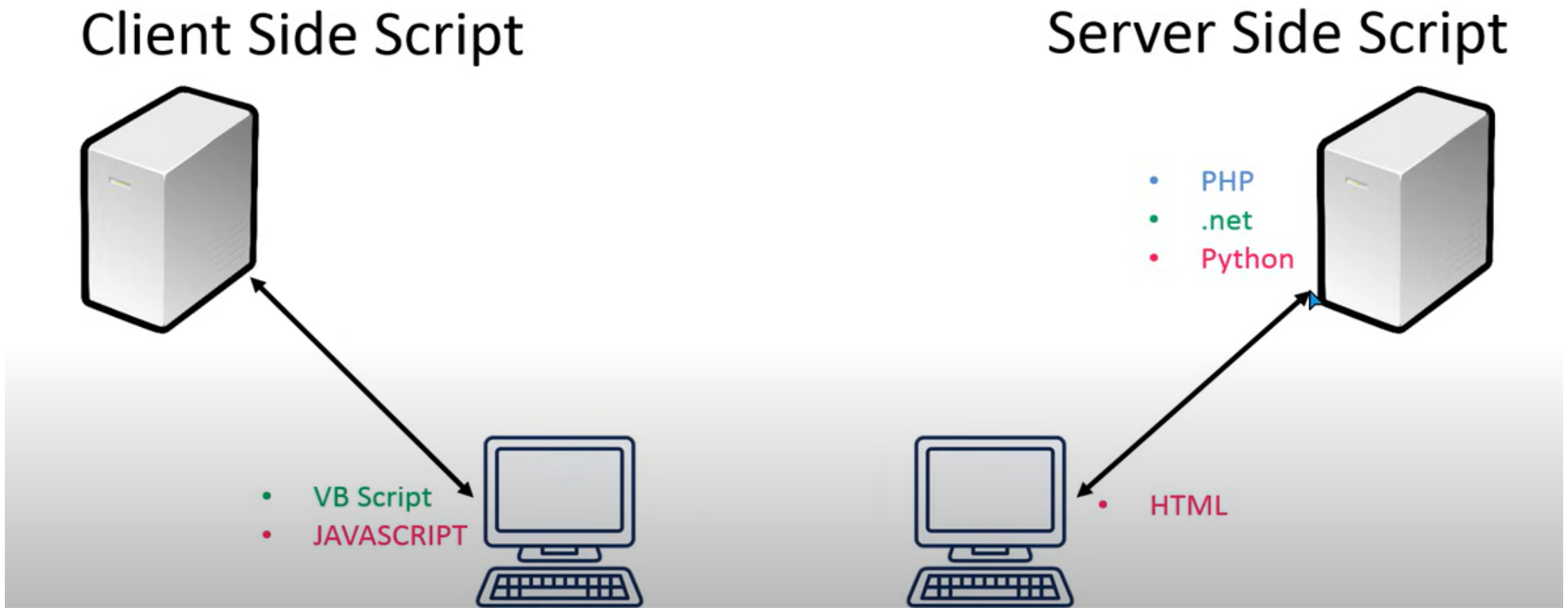
JavaScript History

- 1995 by Brendan Eich (NetScape)
- Mocha
- LiveScript
- JavaScript
- ECMAScript


Server
User
www.geekyshows.com
HTML
CSS
JavaScript


Server
User
www.geekyshows.com
HTML
CSS
JavaScript

Web based Scripting Language



Benefits of learning JavaScript:

Web Development	Desktop App Development	Mobile App Developm
• jQuery • Angular Js • React Js • VueJS • NodeJS	• Electron JS	• Angular Js • React Js • VueJS • React Native • NodeJS

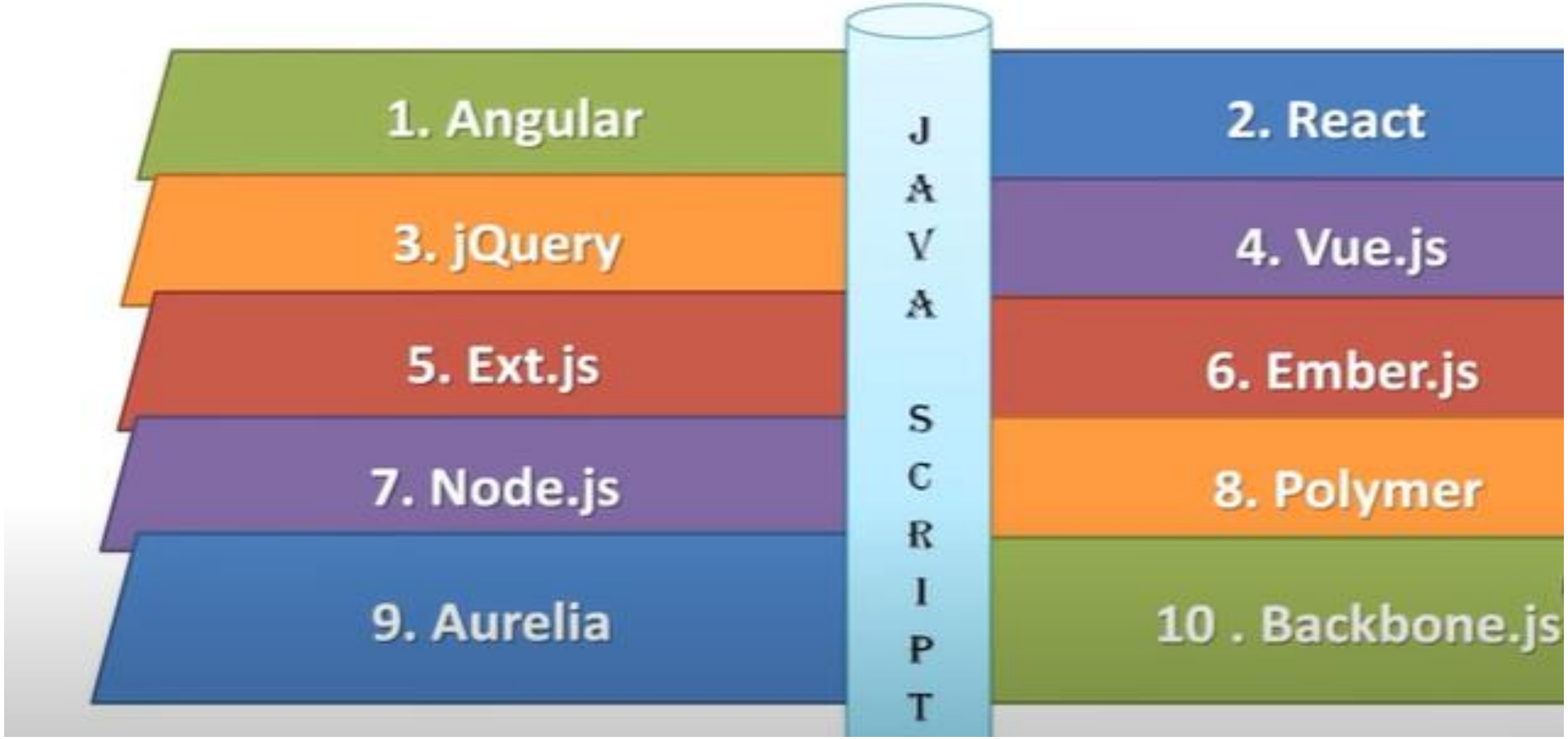
What is JavaScript

1. JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages.
2. JavaScript was developed by Mr. Brendan Eich in 1995 who was working in Netscape.
3. The JavaScript Translator(embedded in the browser) is responsible for translating the JavaScript code for the web browser.
4. It is an interpreted, full-fledge programming languages that enables dynamic interactivity on websites when applied to an HTML document.
5. JavaScript helps to make our webpage more lively and interactive. JavaScript is widely used in mobile application development as well as in game development

6. JavaScript was initially called as Live Script and later on the name is changed to JavaScript. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional websites uses .js to provide several forms of interactivity and simplicity

What makes JavaScript unique ?

JavaScript frameworks and libraries.



Why We Learn JavaScript ?

JavaScript is a Event Based Programming Language

- Click
- Double Click
- Right Click
- Mouse Hover
- Mouse Out
- Drag Drop
- Key Press
- Key Up
- Load
- Unload
- Resize
- Scroll

Uses of JavaScript in Web Development

- Dropdown Menu
- Animated Sliders
- Maps
- Chart - Graphs
- Pop-up window
- Audio Players
- Video Players
- Zoom effect
- Animated Gallery
- Form Validations
- Accordions
- Calendar

How to Link JavaScript File in HTML ?

- **JavaScript can be added to HTML file in two ways:**

- **Internal JS**: We can add JavaScript directly to our HTML file by writing the code inside the <script> tag. **The <script> tag can either be placed inside the <head> or the <body> tag according to the requirement**.

- **External JS**: We can write JavaScript code in another files hand then link this file inside the <head> tag of the HTML file in which we want to add this code.

Syntax:

```
<script>
   // JavaScript Code
</script>
```

- ✓ We can create a separate file to hold the code of JavaScript with the (.js) extension and later incorporate/include it into our HTML document using the **src** attribute of the <script> tag.

- ✓ It becomes very helpful if we want to use the same code in multiple HTML documents.
- ✓ It also saves us from the task of writing the same code over and over again and makes
- ✓ it easier to maintain web pages.

- ✓ Both of the above programs are saved in the same folder, but you can also store JavaScript code in a separate folder, all just you need to provide the address/path of the (.js) file in the src attribute of <script> tag.

Important points

- ✓ JavaScript files are common text files with (.js) extensions such as we created and used in the above program.

- ✓ External JavaScript file only contains JavaScript code and nothing else, even the <script>.... </script>tag are also not used in it.

```
<html>
<head>
<title>External JS</title>
</head>
<body>
<form>
<input type="button" value="Result" onclick="show();"/>
</form>
</body>
</html>
```

```
function show()
{
    alert("Hello World!");
}
```

```
<html>
<head>
<title>External JS</title>
</head>
<body>
<form>
<input type="button" value="Result" onclick="show();"/>
</form>
<script src="hello.js">
</script>
</body>
</html>
```

Characteristics of JavaScript

- JavaScript is a lightweight, interpreted, client-side scripting language.
- Designed for developing network- based applications.
- JavaScript is complementary to Java.
- JavaScript is complementary to and integrated with HTML. It is open source and cross-platform.
- The user input is validated before sending the page to the server. The minimizes the server traffic, which tends to fewer loads on the server.
- There is no need for the user to wait to see if something has forgotten to enter.
- Interactive interfaces can be created, which can give responses to end user actions like mouse or keyboard activities.
- JavaScript can include elements like drag-drop components and sliders to provide a feel of rich interfaces to the users.

Features of JavaScript

- **There are following features of JavaScript:**
- All popular web browsers support JavaScript as they provide built-in execution environments.
- JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
- JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
- It is a light-weighted and interpreted language.
- It is a case-sensitive language.
- JavaScript is supportable in several operating systems including, Windows, macOS, etc.
- It provides good control to the users over the web browsers.

Application of JavaScript

- JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc

Difference between ES5 and ES6

- **ECMAScript 5 (ES5P)**
- ES5 is also known as ECMAScript 2009 as it was released in 2009. It is a function contractors focus on how the objects are instantiated. For ES5 you have to write a function keyword and return, it to be used to define the function, like in normal general JavaScript language.

- **ECMAScript 6 (ES6)**
- ES6 is also known as ECMAScript 2015 as it was released in 2015. Its class allows the developers to instantiate an object using the new operator, using an arrow function, in case it doesn’t need to use the function keyword to define the function, also return keyword can be avoided to fetch the computer value.

Based on	ES5	ES6
Definition	ES5 is the fifth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International)	ES6 is the sixth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International).
Release	It was introduced in 2009.	It was introduced in 2015.
Data-types	ES5 supports primitive data types that are **string, number, boolean, null,** and **undefined**.	In ES6, there are some additions to JavaScript data types. It introduced a new primitive data type **'symbol'** for supporting unique values.
Defining Variables	In ES5, we could only define the variables by using the **var** keyword.	In ES6, there are two new ways to define variables that are **let** and **const**.

Performance	As ES5 is prior to ES6, there is a non-presence of some features, so it has a lower performance than ES6.	Because of new features and the shorthand storage implementation ES6 has a higher performance than ES5.
Support	A wide range of communities supports it.	It also has a lot of community support, but it is lesser than ES5.
Object Manipulation	ES5 is time-consuming than ES6.	Due to destructuring and speed operators, object manipulation can be processed more smoothly in ES6.
Arrow Functions	In ES5, both **function** and **return** keywords are used to define a function.	An arrow function is a new feature introduced in ES6 by which we don't require the **function** keyword to define the function.
Loops	In ES5, there is a use of **for** loop to iterate over elements.	ES6 introduced the concept of **for...of** loop to perform an iteration over the values of the iterable objects.

Conversion of ES6 into ES5

Transpilation:

As of now, there are no browsers that fully support the ES6 features, however, we can convert the ES6 code to the ES5 code by using the transpilation.

There are two major compilers ***Babel*** and ***Traceur***, which are used to convert the ES6 code to ES5 code as part of the build process.

Form Validation

Basic Form validation in JavaScript :-

- A form is also known as web form or HTML form and are used in web – pages.
- In form the user will enter his required details that are further sending it to the server for processing.
- JavaScript Form Validation is a method to validate form values enter by user.

There are various ways to do validation :-

Server side validation:

is performing by web server, after input data is sent to server.

Client side validation:

is performing by web browser, before input data is sent to server.

There are two functions performs in the form validation:-

1. **Basic Validation :**

You have to make sure that all the mandatory fields are filled in. It would require just a loop through each field in the form and check it for data.

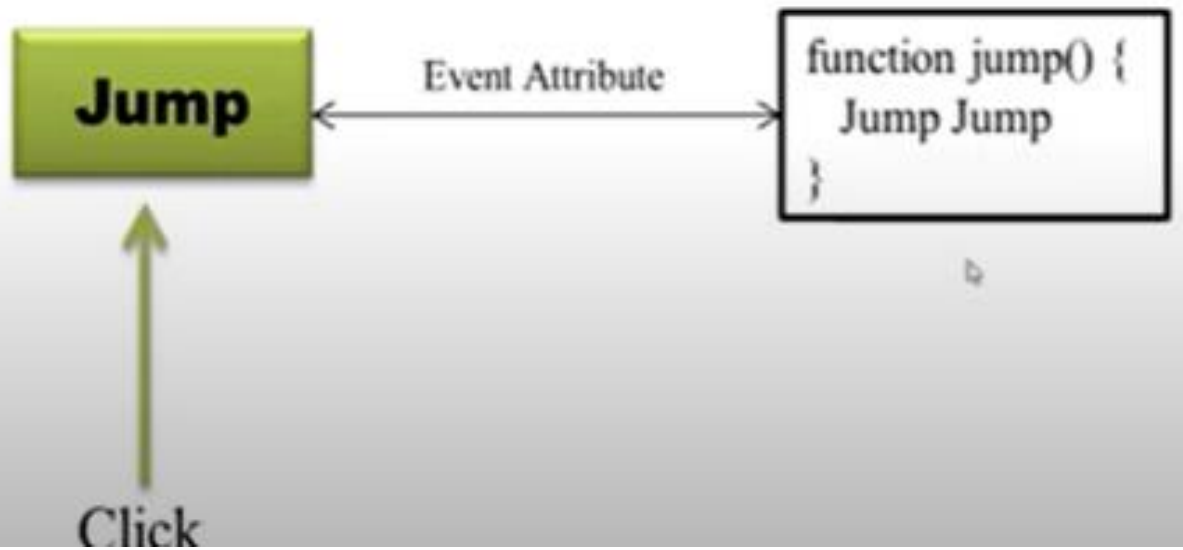
2. **Data Format Validation :**

In this process you have to make sure that user input is clean, safe and useful. The validation includes that the user fill up all the require fields, enter date value in the proper format, numeric field has proper numeric values and so on.

```
<body>
<p>Java Script</p>
<form name = "myform" onsubmit = fun()>
Enter Your Name <input type = "text" name = "stname" required>
Password <input type = "password" name = "passwd" required>

<input type = "submit">


<script>

function fun()
{
var n = document.myform.stname.value
var p = document.myform.passwd.value
document.write("How are you  ",n)

if (p.length<=8)
alert( "password length must be greater than 8.")
}
```

Event Handler

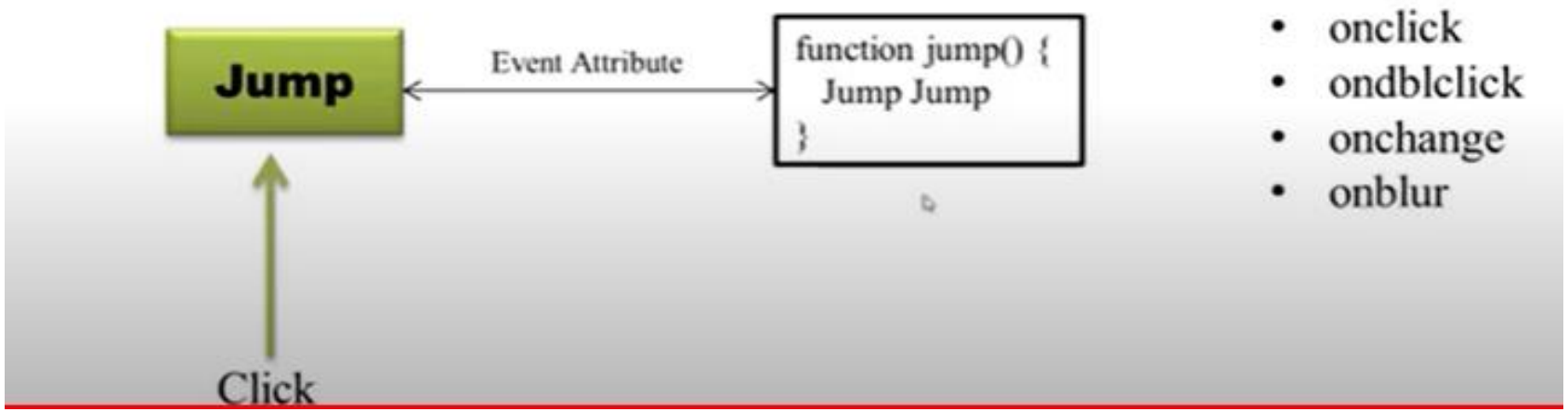
An Event handler is JavaScript code associated with a particular part of the document and a particular event. A handler is executed if and when the given event occurs at the part of the document to which it is associated.



- onclick
- ondblclick
- onchange
- onblur

JavaScript Events

- The change in the state of an object is known as an **Event**. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, js handles the HTML events via **Event Handlers**.
- **For example**, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.



- Common events include mouse clicks, keyboard presses, page loads, and form submissions. Event handlers are JavaScript functions that respond to these events, allowing developers to create interactive web applications.

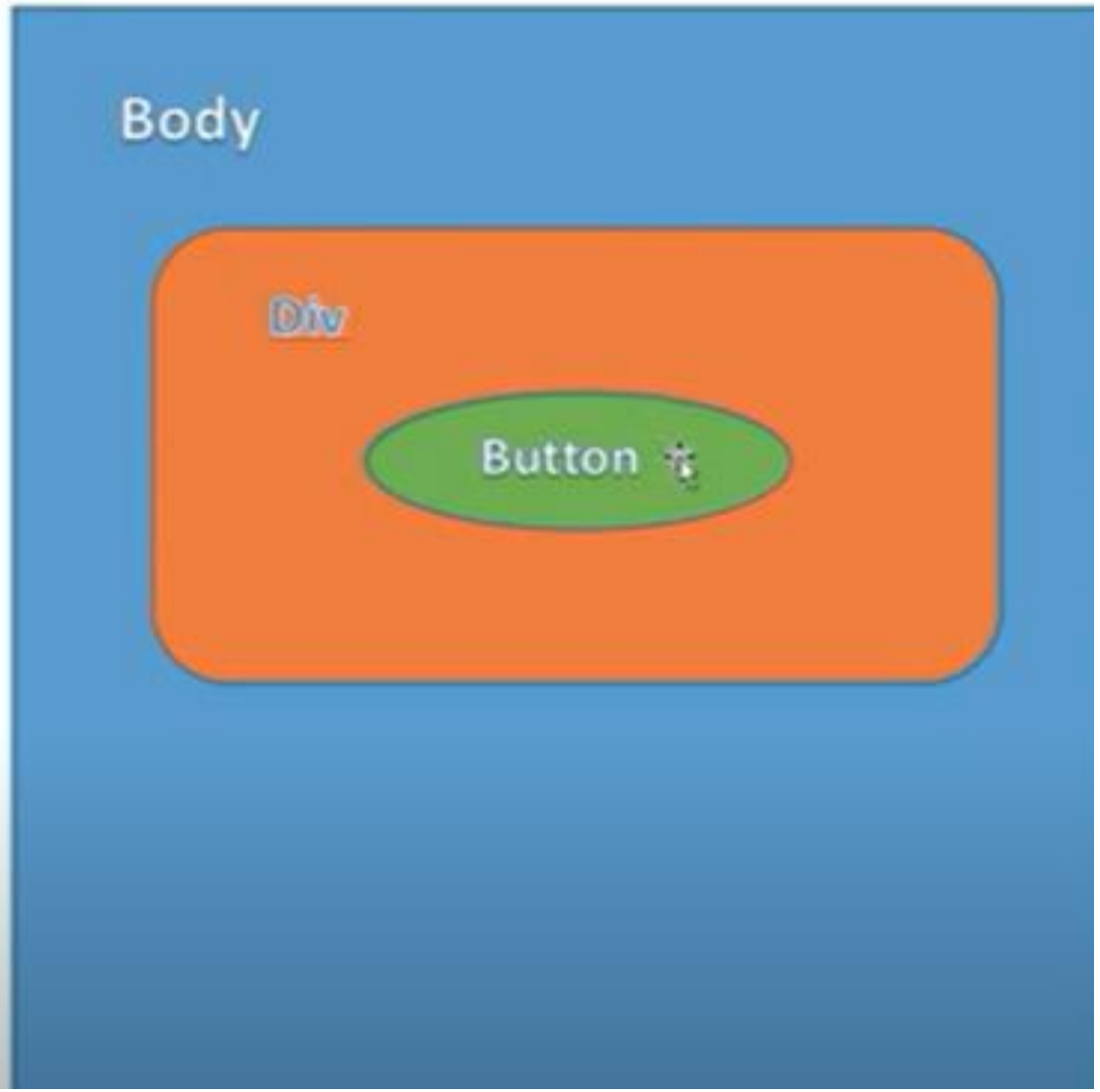
1. Event bubbling

In the event bubbling model, an event starts at the most specific element and then flows upward toward the least specific element (the document or even window).

When you click the button, the click event occurs in the following order:

- button
- div with the id container
- body
- html
- document

```
<body>
  <div id="container">
    <button id='btn'>Click Me!</button>
  </div>
</body>
```

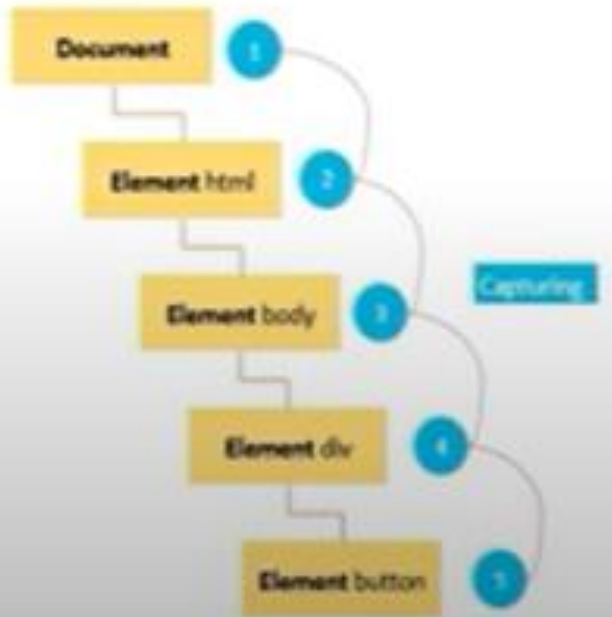


2. Event capturing

In the event capturing model, an event starts at the least specific element and flows downward toward the most specific element.

When you click the button, the click event occurs in the following order:

- document
- html
- body
- div with the id container
- button



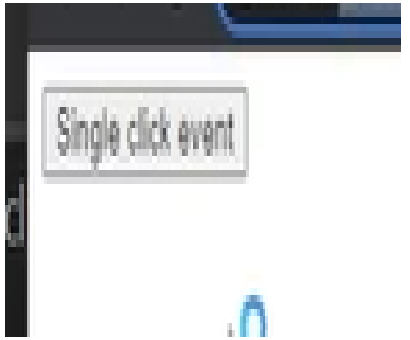
Mouse events:

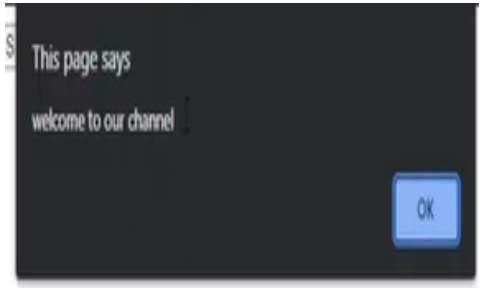
Mouse events:

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Click Event

```html
<!DOCTYPE html>
<html>
    <body>
        <form action="">
            <input type="button" value="Single click event" onclick="clickevent()">
        </form>
        <script>
            function clickevent()
            {
                alert("welcome to our channel");
            }
        </script>
    </body>
</html>
```





MouseOver Event

```
1  <!DOCTYPE html>
2  <html>
3    <body>
4      <p onmouseenter="mousehover()" id="para">welcome to kailash joshi tutorials</p>
5      <script>
6        function mousehover()
7        {
8          document.getElementById('para').setAttribute("style","font-size:50px;color:red");
9        }
10     </script>
11   </body>
12 </html>
```

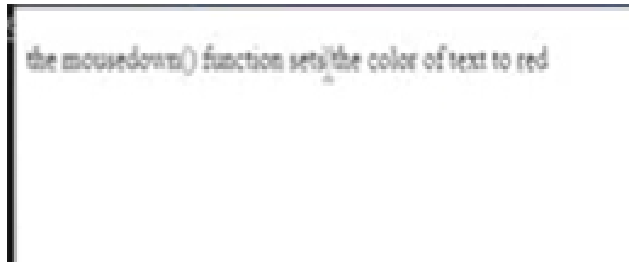
```html
<!DOCTYPE html>
<html>
    <body>
        <p onmouseenter="mousehover()" onmouseout="mouseout()" id="para">welcome to kailash joshi
        <script>
            function mousehover()
            {
                document.getElementById('para').setAttribute("style","font-size:50px;color:red;b
            }
            function mouseout()
            {
                document.getElementById('para').setAttribute("style","none");
            }
        </script>
    </body>
</html>
```

welcome to kailash joshi tutorials

Mouse Up/ Down

mosedownup.html > html > body > script > mouseDown

```
1   <!DOCTYPE html>
2   <html>
3       <body>
4           <p id="para" onmousedown="mouseDown()">
5           the mousedown() function sets the color of text to red
6           </p>
7           <script>
8               function mouseDown()
9               {
10                  document.getElementById('para').style.color="red";
11              }
12          </script>
13      </body>
14  </html>
```



the mousedown() function sets the color of text to red

```html
<!DOCTYPE html>
<html>
    <body>
        <p id="para" onmousedown="mouseDown()" onmouseup="mouseUp()">
        the mousedown() function sets the color of text to red
        <br>
        the mouseup() function sets the color of text to green
        </p>
        <script>
            function mouseDown()
            {
                document.getElementById('para').style.color="red";
            }
            function mouseUp()
            {
                document.getElementById('para').style.color="green";
```

the mousedown() function sets
the color of text to red
the mouseup() function sets the
color of text to green

the mousedown() function sets
the color of text to red
the mouseup() function sets the
color of text to green

Keyboard events:

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events:

Key Up / Down

```html
<!DOCTYPE html>
<html>
    <body>
        <h1>Keyboard events</h1>
        <input type="text" onkeydown="myfunction()" id="first" onkeyup="keyup()">
        <p id="para"></p>
        <script>
            function myfunction()
            {
                document.getElementById('para').innerHTML="you pressed a key";
            }
```

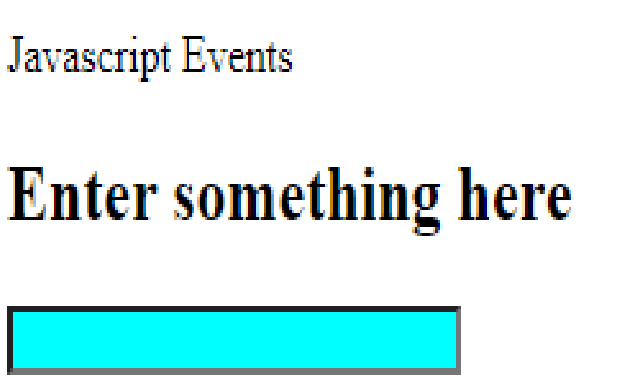
```html
<!DOCTYPE html>
<html>
    <body>
        <h1>Keyboard events</h1>
        <input type="text" onkeydown="myfunction()" id="first" onkeyup="keyup()" onkeypress="keyp
        <p id="para"></p>
        <script>
            function myfunction()
            {
                document.getElementById('para').innerHTML="you pressed a key";
            }
            function keyup()
            {
                let x=document.getElementById('first');
                x.value=x.value.toUpperCase();
            }
            function keypress()
            {
                alert("you press a key in input box");
            }
```

Form events:

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Focus Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>
<!--
        function focusevent()
        {
                document.getElementById("input1").style.background=" aqua";
        }
//-->
</script>
</body>
</html>
```



Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Load event


www.javatpoint.com says
Page successfully loaded
OK
Run
Online Code Editor
<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page successfully loaded');">
<script>
<!--
document.write("The page is loaded successfully");
//-->
</script>
</body>
</html>
Javascript Events
The page is loaded successfully

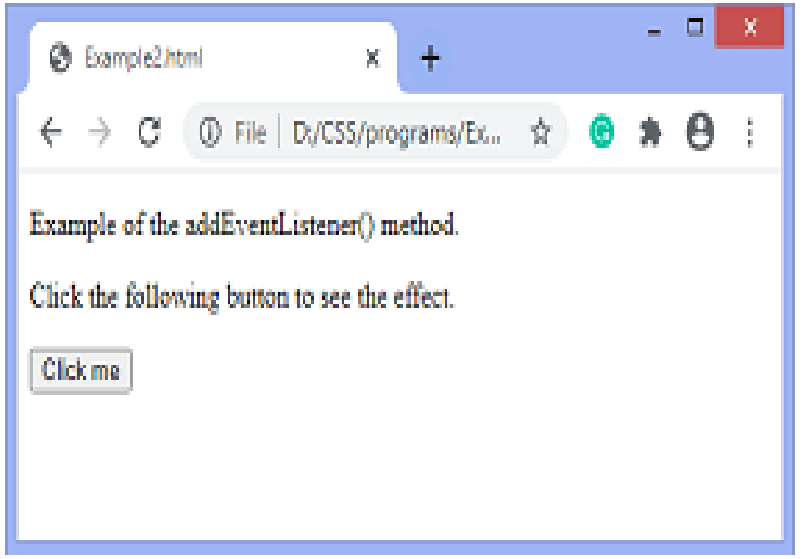
JavaScript addEventListener()

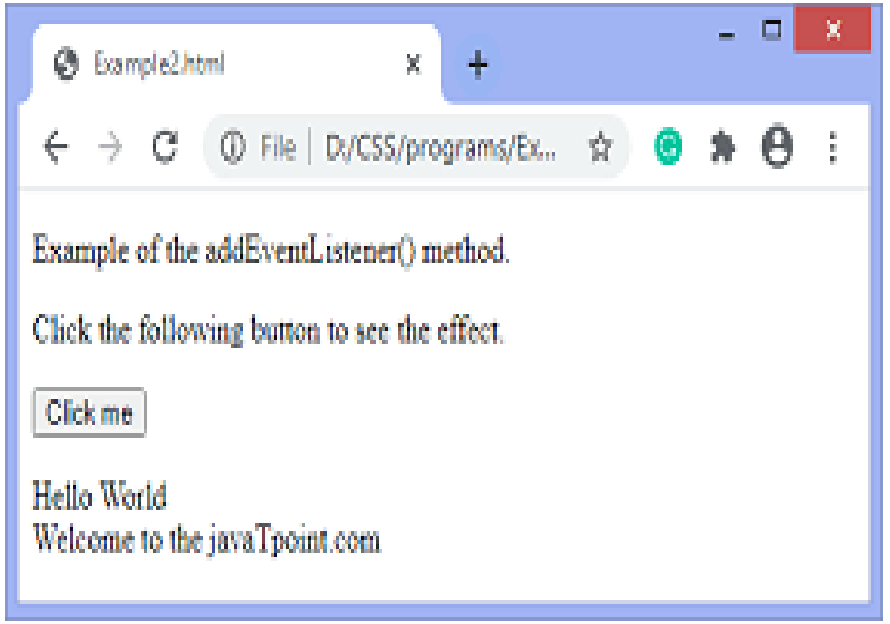
- The **addEventListener()** method is used to attach an event handler to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.
- The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.
- **Syntax**
- **element.addEventListener(event, function, useCapture);**

- Although it has three parameters, the parameters ***event*** and ***function*** are widely used. The third parameter is optional to define. The values of this function are defined as follows.
- Parameter Values
- **event:** It is a required parameter. It can be defined as a string that specifies the event's name.
- **function:** It is also a required parameter. It is a JavaScript function which responds to the event occur.
- **useCapture:** It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is **false**.

It is a simple example of using the addEventListener() method. We have to click the given HTML butt
to see the effect.

```
<!DOCTYPE html>
<html>
<body>
<p> Example of the addEventListener() method. </p>
<p> Click the following button to see the effect. </p>
<button id = "btn"> Click me </button>
<p id = "para"></p>
<script>
document.getElementById("btn").addEventListener("click", fun);
function fun() {
document.getElementById("para").innerHTML = "Hello World" +
<br>" + "Welcome to the  javaTpoint.com";
}
</script>
</body>
</html>
```

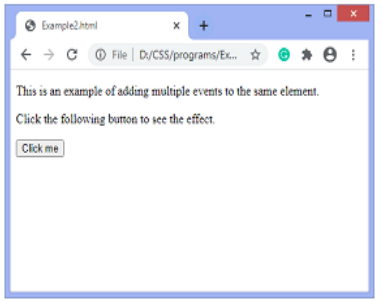




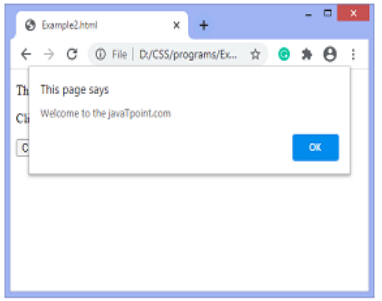
In this example, we are adding multiple events to the same element

```
<!DOCTYPE html>
<html>
<body>
<p> This is an example of adding multiple events to the same element. </p>
<p> Click the following button to see the effect. </p>
<button id = "btn"> Click me </button>
<p id = "para"></p>
<p id = "para1"></p>
<script>
function fun() {
    alert("Welcome to the javaTpoint.com");
}

function fun1() {
   document.getElementById("para").innerHTML =  "This is second function";

}
function fun2() {
   document.getElementById("para1").innerHTML =  "This is third function";
}
var mybtn = document.getElementById("btn");
mybtn.addEventListener("click", fun);
mybtn.addEventListener("click", fun1);
mybtn.addEventListener("click", fun2);
</script>
</body>
```

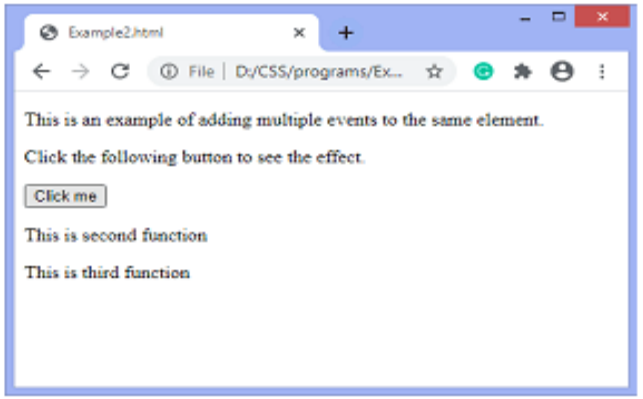
Output



Now, when we click the button, an alert will be displayed. After clicking the given HTML button, the output will be -



When we exit the alert, the output is -



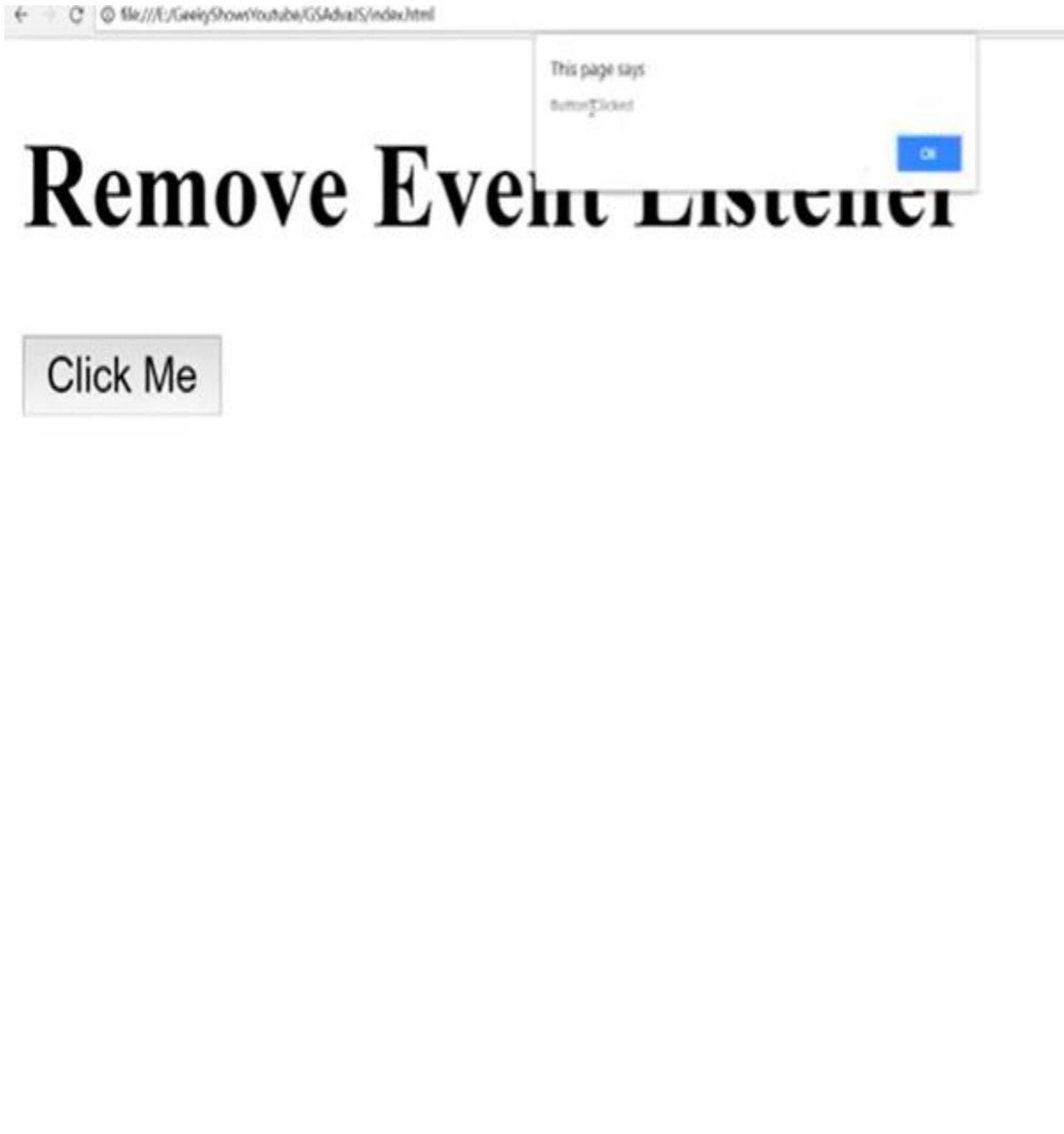
removeEventListener ( )

This method is used to remove handlers, with the same arguments given when the event was added.

Ex: -

```
btn.addEventListener("click", show, false);
btn.addEventListener("click", show, true);

btn.removeEventListener("click", show, true);
```

```html
<head>
    <title>Geeky Shows</title>
</head>
<body>

    <h1>Remove Event Listener</h1>
    <button id="mybtn">Click Me</button>


    <script>
        function show() {
            alert("Button Clicked");
        }
        var btn = document.getElementById("mybtn");
        btn.addEventListener("click", show, false);
        btn.addEventListener("click", show, true);

    </script>
</body>
</html>
```



```html
<head>
    <title>Geeky Shows</title>
</head>
<body>

    <h1>Remove Event Listener</h1>
    <button id="mybtn">Click Me</button>


    <script>
        function show() {
            alert("Button Clicked");
        }
        var btn = document.getElementById("mybtn");
        btn.addEventListener("click", show, false);
        btn.addEventListener("click", show, true);

        btn.removeEventListener("click", show, true);

    </script>
```

Promise

A Promise is an object representing the eventual completion or failure of an asynchronous operation.

A JavaScript Promise object contains both the producing code and calls to the consuming code. It can be used to deal with Asynchronous operation in JavaScript.

Promise State:-

Pending - Initial State, Not yet Fulfilled or Rejected

Fulfilled/Resolved – Promise Completed

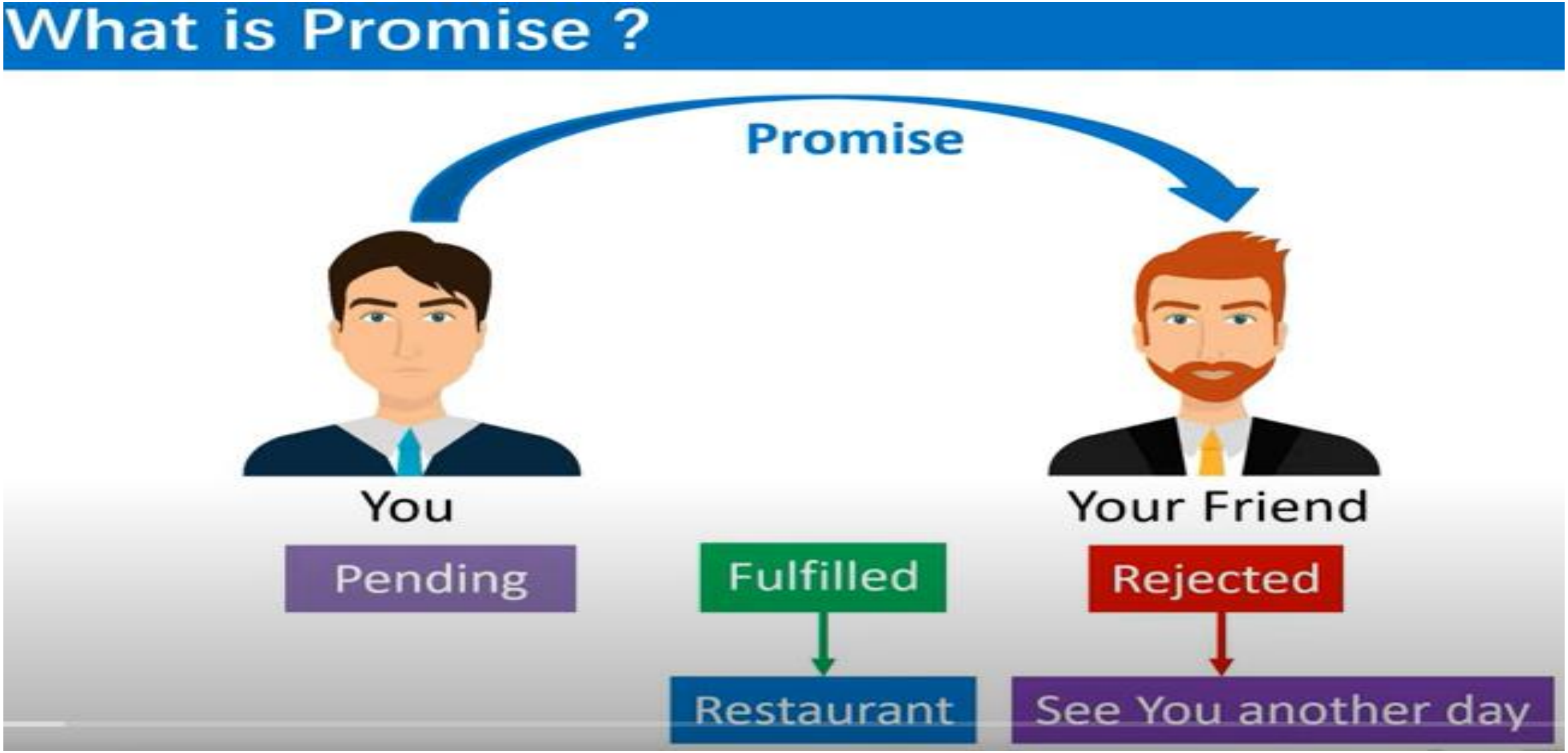
Rejected – Promise Failed

How Does a Promise Work?

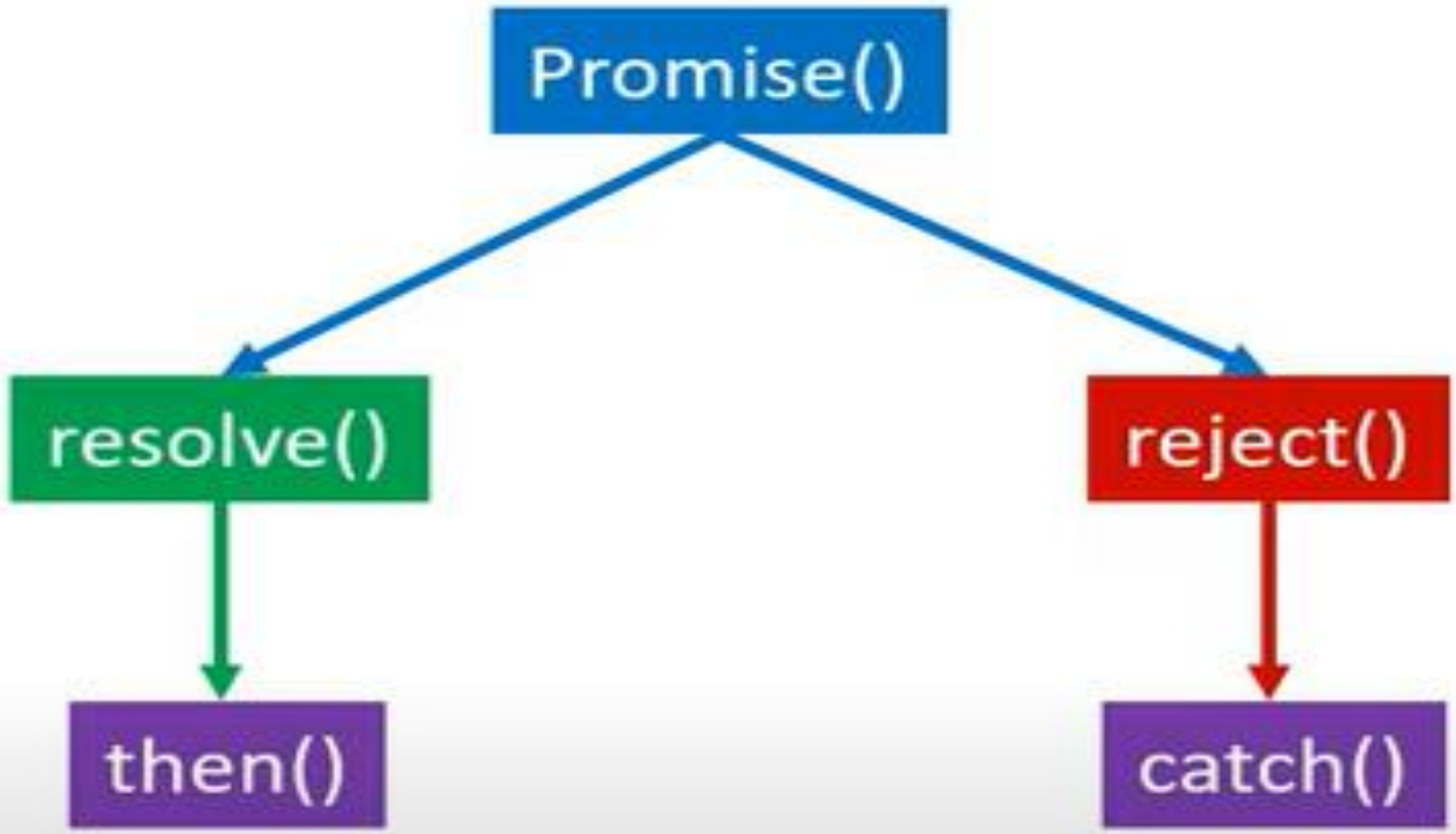
A promise can be in one of three states:

- **Pending:** The promise is waiting for something to finish. For example, waiting for data to load from a website.
- **Fulfilled:** The promise has been completed successfully. The data you were waiting for is now available.
- **Rejected:** The promise has failed. Maybe there was a problem, like the server not responding.

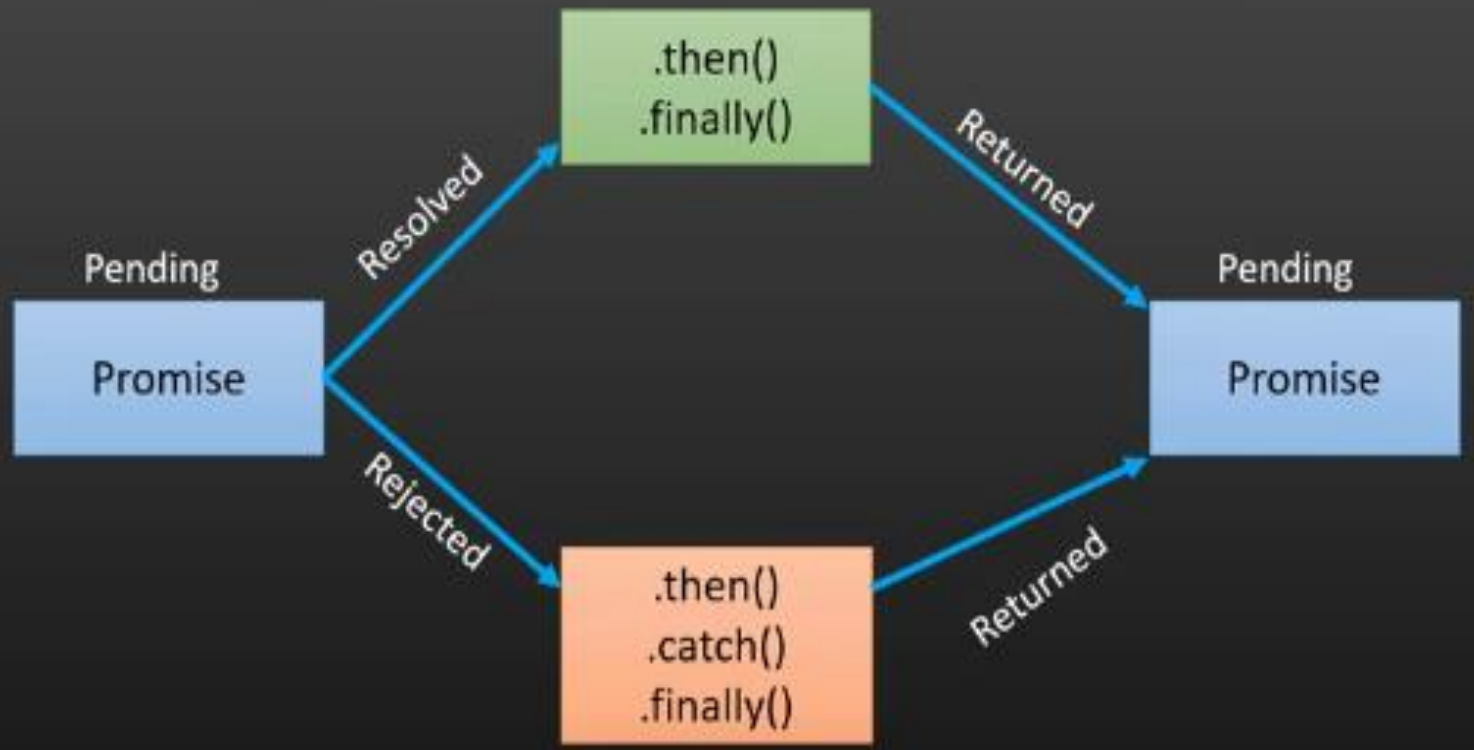
Promise


What is Promise ?
Promise
You
Your Friend
Pending
Fulfilled
Rejected
Restaurant
See You another day

What is Promise ?



How Promise works



- A pending promise can either be Resolved with a value or Rejected with a reason (error).
- When either of these options happens, the associated handlers queued up by a promise's *then* method are called.
- A promise is said to be settled if it is either Resolved or Rejected, but not Pending.

```
let prom = new Promise();

let prom = new Promise(function(){

});

let prom = new Promise(function(resolve, reject){

});
```

```
let prom = new Promise(function(resolve, reject){
    if(condition){
        resolve("Here is a Success");
    else{
        reject("Here is a Faliure");
    }
});

prom.then(onfulfilment);

prom.catch(onRejection);
```

```
let onfulfilment = (result) => {
    console.log(result);
}

let onRejection = (error) => {
    console.log(error);
}
```

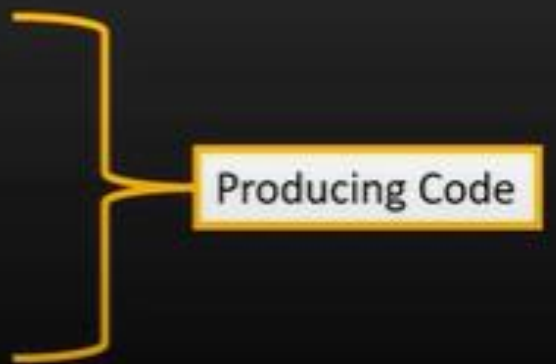
Creating Promise

Promise () – A Promise object is created using the *new* keyword and its constructor. This constructor takes a function, called the "executor function", as its parameter. This function should take two functions as parameters. The first of these functions (resolve) is called when the asynchronous task completes successfully and returns the results of the task as a value. The second (reject) is called when the task fails, and returns the reason for failure, which is typically an error object.

Syntax:- Promise (executor)

Syntax:-

```
const promiseObj = new Promise (  (resolve, reject) =>{
        Do asynchronous operation
        resolve(value)
        reject(Error)
} )
```



A JavaScript Promise object contains both the producing code and calls to the consuming code.

Function Returning a Promise

```
function myFunction(){
        return new Promise(  (resolve, reject) => {


        } )
}
```

then( ) Method

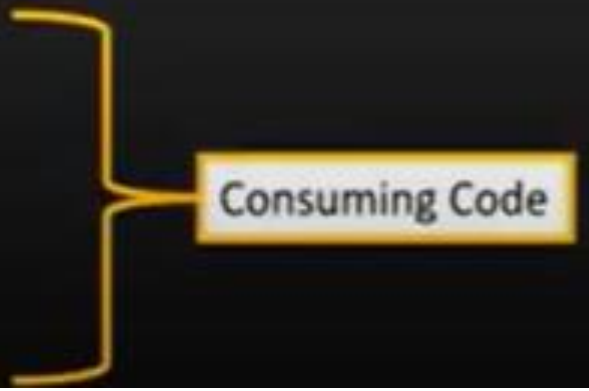
The then() method returns a Promise. It takes up to two arguments: callback functions for the success and failure cases of the Promise. As then method returns a Promise so we can do method chaining.

Syntax:- then(onResolved, onRejected)

onResolved - A Function called if the Promise is fulfilled. This function has one argument, the fulfillment value.

onRejected - A Function called if the Promise is rejected. This function has one argument, the rejection reason.

```
promiseObj.then(value => {
  console.log(value); },
error => {
  console.log(error); });
```



Promise

```
const promiseObj = new Promise((resolve, reject)=>{
        let req = true
        if (req == true){
            resolve("Request Success");
        } else {
            reject("Request Rejected");
        }
});

promiseObj.then(
        (value) => {console.log(value);},
        (error) => {console.log(error);},
);
```

```
const promiseObj = new Promise((resolve, reject)=>{
        let req = true
        if (req == true){
            resolve("Request Success");
        } else {
            reject("Request Rejected");
        }
}).then(
        (value) => {console.log(value);},
        (error) => {console.log(error);},
);
```

Chaining

The then method returns a Promise which allows for method chaining. If the function passed as handler to then returns a Promise, an equivalent Promise will be exposed to the subsequent then in the method chain.

```
const promiseObj = new Promise((resolve, reject)=>{
        let num = 10
        resolve(num);
}).then(
        (value) => {console.log(value);
        return value + 10;
}).then (
        (value) => { console.log(value);
});
```

catch () Method

The *catch()* method returns a Promise and deals with rejected cases only. It behaves the same as calling *then(undefined, onRejected)*. In fact, calling *catch(onRejected)* internally calls *then(undefined, onRejected)*.

The catch method is used for error handling in promise composition. Since it returns a Promise, it can be chained in the same way as its sister method, then()

Syntax:- catch(callback)

Where the callback is a function called when the Promise is rejected. This function has one argument *error* – The rejection error.

catch () Method

```
const promiseObj = new Promise((resolve, reject)=>{
        let req = true
        if (req == true){
            resolve("Request Success");
        } else {
            reject("Request Rejected");
        }
}).then(
        (value) => {console.log(value);
}).catch(
        (error)=>{ console.log(error);
});
```

finally () Method

The finally() method returns a Promise. When the promise is settled, i.e either fulfilled or rejected, the specified callback function is executed. This provides a way for code to be run whether the promise was fulfilled successfully or rejected once the Promise has been dealt with.

This helps to avoid duplicating code in both the promise's then() and catch() handlers.

Syntax:- finally(callback)

finally () Meth

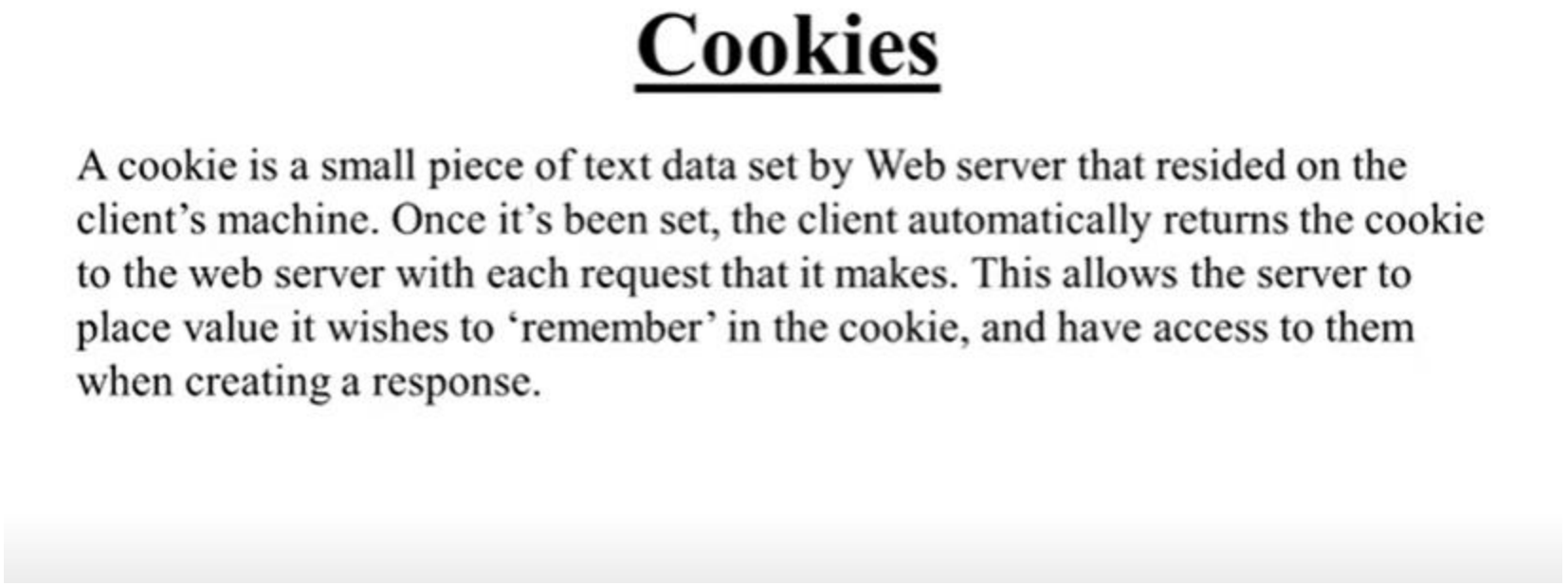
```
const promiseObj = new Promise((resolve, reject)=>{
        let req = true
        if (req === true){
            resolve("Request Success");
        } else {
            reject("Request Rejected");
        }
}).then(
        (value) => { console.log(value);
}).catch(
        (error)=>{ console.log(error);
}).finally(
        () =>{ console.log("Cleaned");
});
```

Chaining

```
const promiseObj = new Promise((resolve, reject)=>{
        let num = 10
        resolve(num);
}).then(
        (value) => { console.log(value);
        return value + 10
}).then (
        (value) => { console.log(value);
        return value;
}).then(
        () => { throw new Error("Error Found");
}).catch(
        (error) => { console.log(error);
});
```

JavaScript Cookies

- A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.
- A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.



Cookies

A cookie is a small piece of text data set by Web server that resided on the client's machine. Once it's been set, the client automatically returns the cookie to the web server with each request that it makes. This allows the server to place value it wishes to 'remember' in the cookie, and have access to them when creating a response.

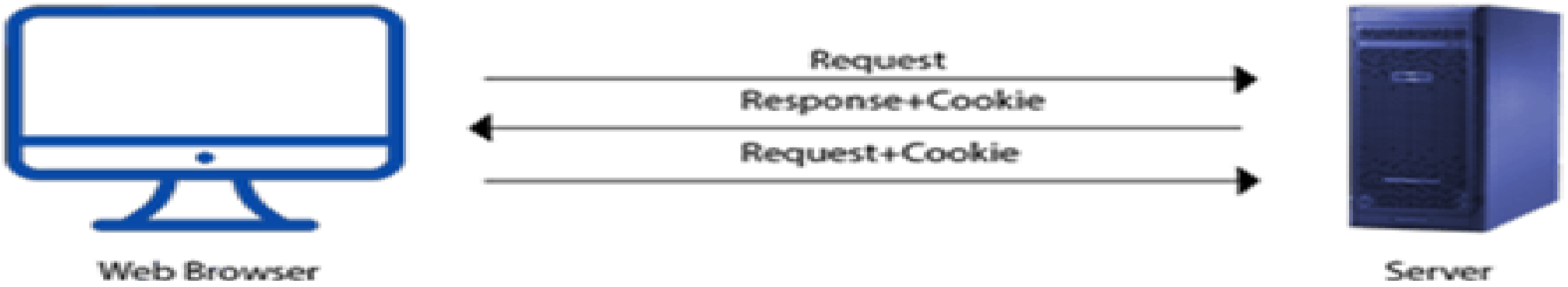
Type of Cookies

- Session Cookies – Cookies that are set without the expires field are called session cookies. It is destroyed when the user quits the browser.

- Persistent Cookies – The browser keeps it up until their expiration date is reached.

How Cookies Works?

1. When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.
2. So, to recognize the old user, we need to add the cookie with the response from the server.
3. browser at the client-side.
4. Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.



- **How to create a Cookie in JavaScript?**
- In JavaScript, we can create, read, update and delete a cookie by using **document.cookie** property.
- The following syntax is used to create a cookie:

```
document.cookie="name=value";
```

You can see Cookies in Google Chrome by following chrome://settings/content/cookies

```
<!DOCTYPE html>
<html>

<head>
    <title>Geeky Shows</title>
</head>

<body>
    <h1>Cookies</h1>
    <script>
        document.cookie = "username=sem";
        alert(document.cookie);

    </script>
</body>

</html>
```

Creating Cookie in JavaScript

- Cookies are created by a web server and sent to the user's browser as part of the HTTP response headers.
- Creating cookies in JavaScript involves using the document.cookie object to set key-value pairs and additional parameters. To create a cookie, assign a string containing the desired cookie information to document.cookie. This string **can include attributes like expiration date, domain, and path.**

Syntax: -

```
document.cookie = "name=value";
document.cookie = "name=value; expires=date; domain=domain; path=path; secure";
document.cookie = "name=value; max-age=inSecond; domain=domain; path=path; secure";
```

Ex:-

```
document.cookie = "username=geeky";
document.cookie = "username=geeky; expires=Monday, 3-Sep-2018 09:00:00 UTC";
document.cookie = "username=geeky; max-age="+60*60*24*10;
```

Note - name-value pair must not contain any whitespace character, Commas or semicolons.
Ex: - username=geeky shows;

Optional Cookies Attribute:-

max-age

expires

domain

path

secure

Whenever you omit the optional cookie fields, the browser fills them in automatically with reasonable defaults.

```html
<!DOCTYPE html>
<html>

<head>
    <title>Geeky Shows</title>
</head>

<body>
    <h1>Cookies</h1>

    <script>
        document.cookie = "username=geeky";
    </script>
</body>

</html>
```

```html
<!DOCTYPE html>
<html>

<head>
    <title>Geeky Shows</title>
</head>

<body>
    <h1>Cookies</h1>

    <script>
        document.cookie = "username=geeky";
        alert(document.cookie);
    </script>
</body>

</html>
```

max-age

It is used to create persistent cookie. It is supported by all modern browsers except IE.

Type of cookies: -

- Session Cookies – Cookies that are set without the expires/max-age field are called session cookies. It is destroyed when the user quits the browser.
- Persistent Cookies – The browser keeps it up until their expiration date is reached.

Ex:-

```
document.cookie = “username=geeky; max-age=" + 60 * 60 * 24 * 10;
document.cookie = “username=geeky; max-age=" + 60 * 60 * 24 * 10 + “; path=/”;
```

```
<!DOCTYPE html>
<html>

<head>
    <title>Geeky Shows</title>
</head>

<body>
    <h1>Cookies</h1>

    <script>
        document.cookie = "username=geeky; max-age=" + 60 * 60 * 24 * 10 + ";
        path=/";
        alert(document.cookie);
    </script>
</body>

</html>
```

expires

It is used to create persistent cookie.

Type of cookies: -

- Session Cookies – Cookies that are set without the expires/max-age field are called session cookies. It is destroyed when the user quits the browser.
- Persistent Cookies – The browser keeps it up until their expiration date is reached.

Ex:- document.cookie = “username=geeky; expires=Monday, 3-Sep-2018 09:00:00 UTC”;

```html
<!DOCTYPE html>
<html>

<head>
    <title>Geeky Shows</title>
</head>

<body>
    <h1>Cookies</h1>

    <script>
        document.cookie = "username=geeky; expires=Mon, 3-Sep-2018 09:00:00 UTC";
        alert(document.cookie);
    </script>
</body>

</html>
```

domain

It specifies the domain for which the cookie is valid. If not specified, this defaults to the host portion of the current document location. If a domain is specified, subdomains are always included.

Ex: - document.cookie = “username=geeky; domain=geekyshows.com”;

note.geekyshows.com

code.geekyshows.com

Ex: - document.cookie = “username=geeky; domain=code.geekyshows.com”;

path

Path can be / (root) or /mydir (directory). If not specified, defaults to the current path of the current document location, as well as its descendants.

Ex: - document.cookie = “username=geeky; path=/”;

Ex: - document.cookie = “username=geeky; path=/home”;

secure

Cookie to only be transmitted over secure protocol as https. Before Chrome 52, this flag could appear with cookies from http domains.

Ex: - document.cookie = “username=geeky; secure”;

Replace/Append Cookies

When we assign a new cookie value to document.cookie, the current cookie are not replaced. The new cookie is parsed and its name-value pair is appended to the list. The exception is when you assign a new cookie with the same name (and same domain and path, if they exist) as a cookie that already exists. In this case the old value is replaced with the new.

- Ex: - document.cookie = "username=geeky"
  document.cookie = "username=shows"

  } Replace

- Ex: - document.cookie = "username=geeky"
  document.cookie= "userid=geek1"

  } Append

Deleting Cookies

A cookie is deleted by setting a cookie with the same name (and domain and path, if they were set) with an expiration date in the past and if using max-age then must set a negative value.

Ex: -

```
document.cookie = “username=geeky; expires=Monday, 3-Sep-2018 09:00:00 UTC”;
document.cookie = “username=geeky; expires=Thu, 01-Jan-1970 00:00:01 GMT”;

document.cookie = “username=geeky; max-age=" + 60 * 60 * 24 * 10;
document.cookie = “username=geeky; max-age=-60";

document.cookie = “username”;
document.cookie = “username; expires=Thu, 01-Jan-1970 00:00:01 GMT”;
```

Updating Cookies

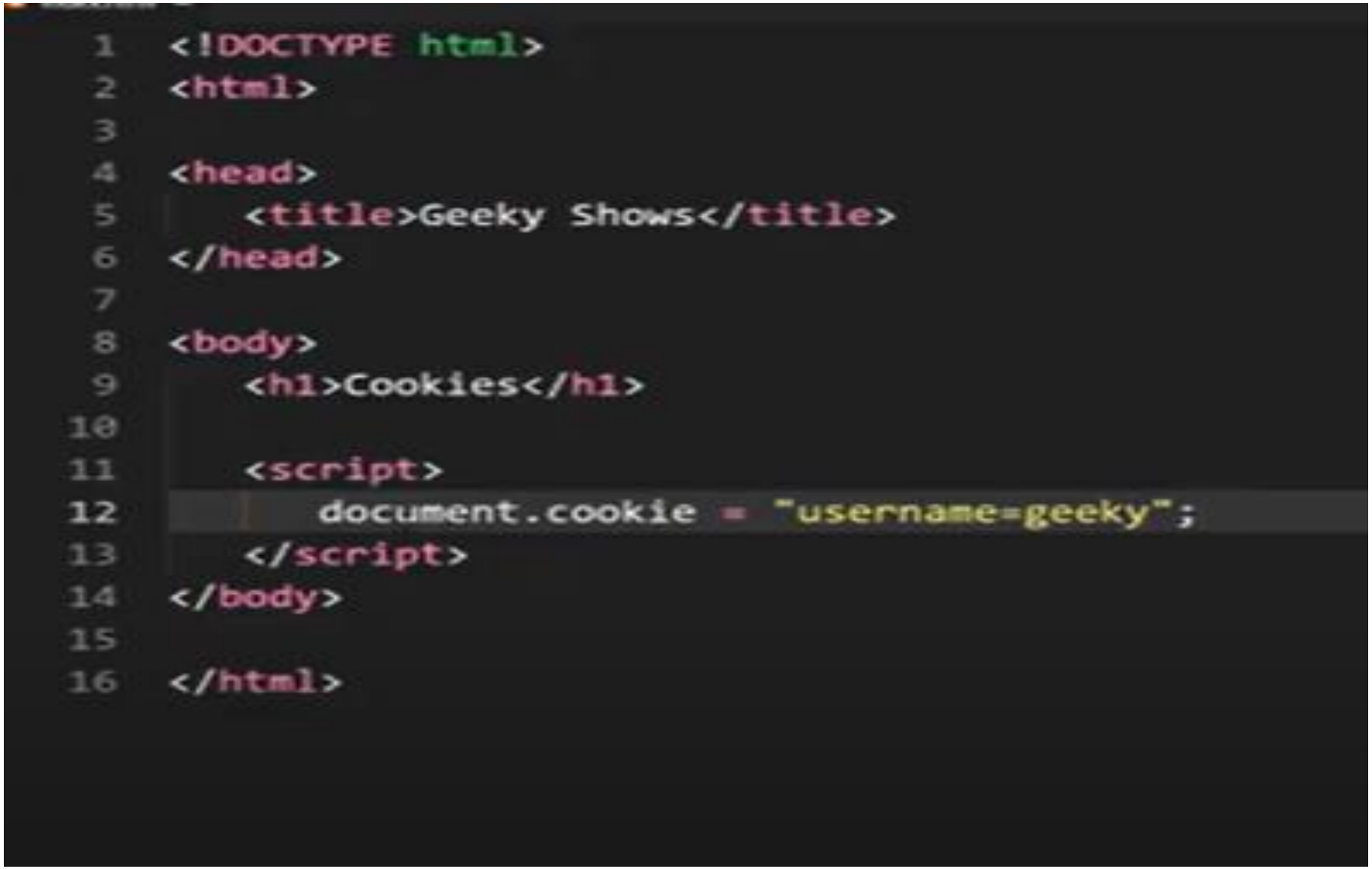
A cookie is possible to update by setting new value to a cookie with the same name.

Ex: -

```
document.cookie = “username=geeky”;
document.cookie = “username=shows”;
```

Cookies Security Issues

- Can misuse Client Details
- Can track User
- Client Can Delete Cookies
- Client can Manipulate Cookies


1 <!DOCTYPE html>
2 <html>
3
4 <head>
5 <title>Geeky Shows</title>
6 </head>
7
8 <body>
9 <h1>Cookies</h1>
10
11 <script>
12 document.cookie = "username=geeky";
13 </script>
14 </body>
15
16 </html>

Generators and Iterators in JavaScript

- **Generators**
- The Generators are a special type of function in JavaScript that can be paused and resumed during their execution. They are defined using the asterisk (*) after the function keyword. The Generators use the yield keyword to yield control back to the caller while preserving their execution context. The Generators are useful for creating iterators, asynchronous code, and handling sequences of data without loading all the data into the memory at once.

```
function* GFG() {
    yield 10;
    yield 20;
    yield 30;
}
const generator = GFG();
console.log(generator.next().value);
console.log(generator.next().value);
console.log(generator.next().value);
```

Output

```
10
20
30
```

Iterators

- The Iterators are objects with a special structure in JavaScript. They must have a next() method that returns an object with the value and done properties. The value property represents the next value in the sequence and the done property indicates whether there are more values to be iterated. The Iterators are commonly used for iterating over data structures like arrays, maps, and sets.

```
const colors = ['red', 'green', 'blue'];
const GFG = colors[Symbol.iterator]();
console.log(GFG.next());
console.log(GFG.next());
console.log(GFG.next());
console.log(GFG.next());
```

Output

```
{ value: 'red', done: false }
{ value: 'green', done: false }
{ value: 'blue', done: false }
{ value: undefined, done: true }
```

Difference between Generators and Iterators:

Generators	Iterators
Functions with the asterisk (*) and yield keyword.	Object that provides a sequence of values.
Often used for asynchronous operations.	Typically used for looping through collections.
It can Pause and resume during the execution.	The Sequentially iterate through data elements.
The Explicitly define and yield data.	The Automatically iterate through existing the data structures.
Every generator is an iterator.	Every iterator is not a generator.

Browser Object Model

- The **Browser Object Model** (BOM) is used to interact with the browser.
- The default object of browser is window means you can call all the functions of window by specifying window or directly. For example

```
window.alert("hello javatpoint");
```

is same as:

```
alert("hello javatpoint");
```

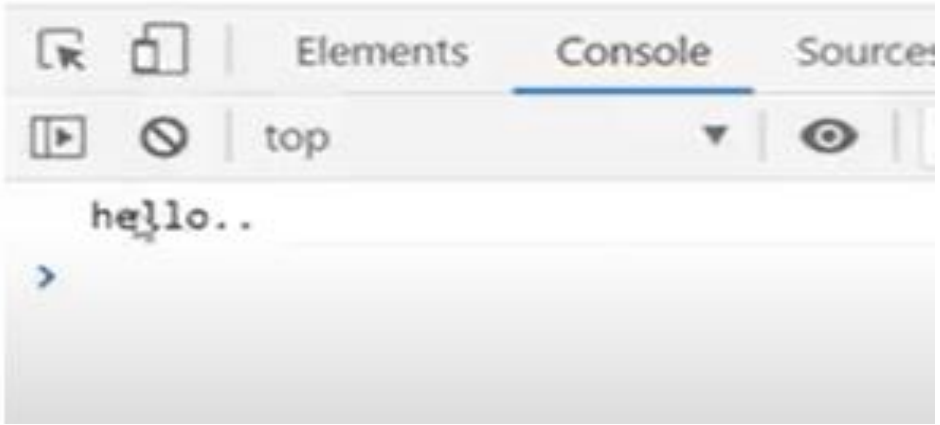
You can use a lot of properties (other objects) defined underneath the window object like document, history, screen, navigator, location, innerHeight, innerWidth,

All global JavaScript objects, functions, and variables with the var keyword automatically become members of the window object.

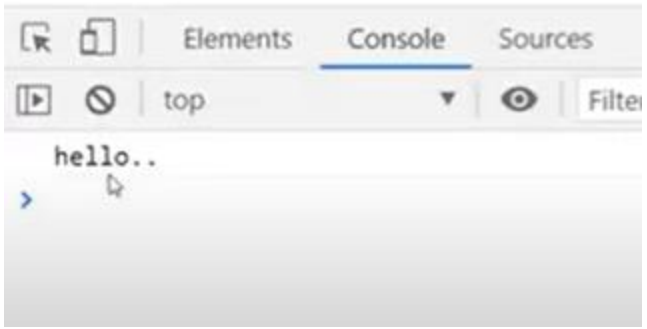
Global variables are properties of the window object.

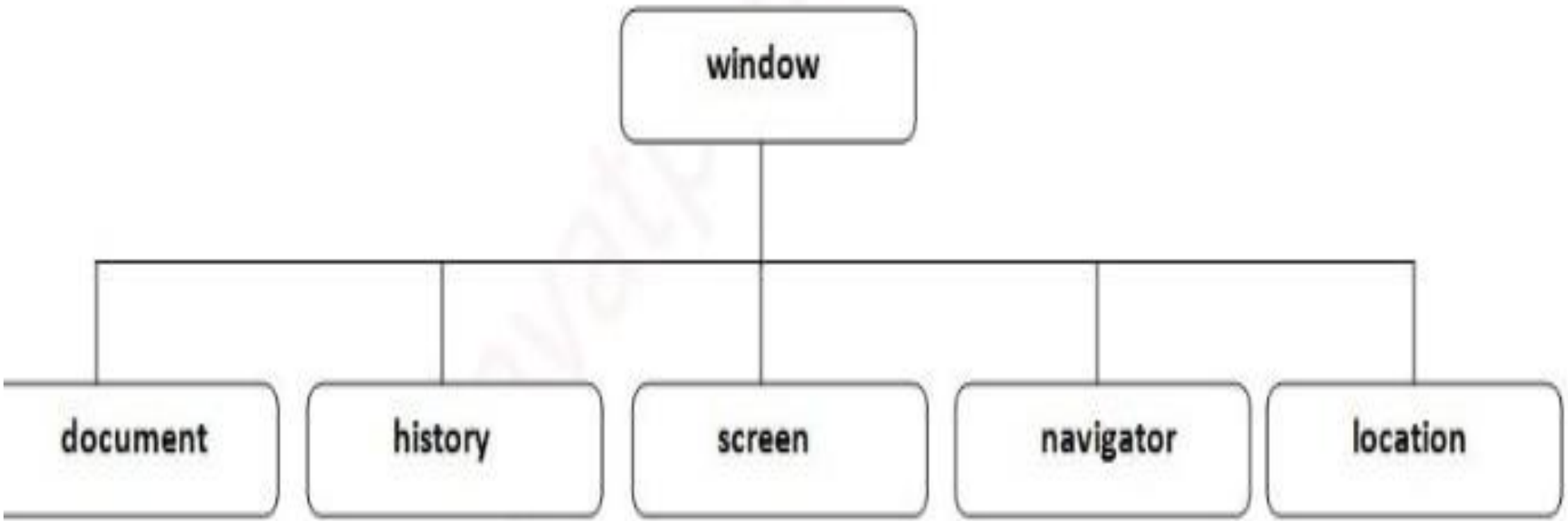
Global functions are methods of the window object.

```
<script>
    var x = "hello..";
    console.log(x);
</script>
```




```
<script>
    var x = "hello..";
    console.log(window.x);
</script>

</body>
</html>
```




window
document
history
screen
navigator
location

Browser Object Model Types

Types	Description
Window Object Model	Represents the browser window and serves as the main tool for interaction within the Browser Object Model (BOM).
History Object Model	Enables navigation control by keeping track of the user's browsing history in the Browser Object Model (BOM).
Screen Object Model	Offers information about the user's screen, like its size, within the Browser Object Model (BOM).
Navigator Object Model	Provides details about the user's browser and system characteristics within the Browser Object Model (BOM).
Location Object Model	Manages the current web address (URL) and allows changes within the Browser

Window Object

- The ‘window object in JavaScript represents your web browser’s window. Everything you create in JavaScript, like variables and functions, is automatically part of this window.
- Global variables are like items in the window, and global functions are tools you can use from anywhere. The “document” object, which deals with HTML content, is also a special tool in the window. So, think of the “window” as a big container holding all your JavaScript stuff.

The HTML DOM Objects can be treated as the window object’s property

```
window.document.getElementById("header");
```

which is the same as

```
document.getElementById("header");
```

Window Size

You can use two properties to find out the dimensions of the browser window, and both provide the measurements in pixels:

Window Size	Description
window.innerHeight	This gives you the vertical size or height of the browser window in pixels.
window.innerWidth	This gives you the horizontal size or width of the browser window in pixels.

Other Window Methods

Window Methods	Descriptions
window.open()	This method is used to open a new tab or window with the specified URL and name.
window.close()	This method is used for closing a certain window or tab of the browser that was previously opened by the window.open( ) method.
window.moveTo()	This method is used in the window to move the window from the left and top coordinates.
window.resizeTo()	This method is used to resize a window to the specified width and height

JavaScript History Object

- The **JavaScript history object** represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.
- The history object is the window property, so it can be accessed by:

```
window.history
```

Or,

```
history
```

- Property of JavaScript history object

No.	Property	Description
1	length	returns the length of the history URLs.

Methods of JavaScript history object

There are only 3 methods of history object.

No.	Method	Description
1	forward()	loads the next page.
2	back()	loads the previous page.
3	go()	loads the given page number.

JavaScript Navigator Object

- The **JavaScript navigator object** is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.
- **The navigator object is the window property, so it can be accessed by:**

```
window.navigator
```

Or,

```
navigator
```

- Property of JavaScript navigator object

No.	Property	Description	
1	appName		returns the name
2	appVersion	returns the version	
3	appCodeName	returns the code name	
4	cookieEnabled	returns true if cookie is enabled otherwise false	
5	userAgent	returns the user agent	
6	language		returns the language. It is supported in Netscape and Firefox only.
7	userLanguage	returns the user language. It is supported in IE only.	

8	plugins		returns the plugins. It is supported in Netscape and Firefox only.
9	systemLanguage		returns the system language. It is supported in IE only.
10		mimeTypes[]	returns the array of mime type. It is supported in Netscape and Firefox only.

11		platform	returns the platform e.g. Win32.
12	online	returns true if browser is online otherwise false.	

Methods of JavaScript navigator object

The methods of navigator object are given below.

No.	Method	Description
1	javaEnabled()	checks if java is enabled.
2	taintEnabled()	checks if taint is enabled. It is deprecated since JavaScript 1.2.

JavaScript Screen Object

- The **JavaScript screen object** holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.
- The navigator object is the window property, so it can be accessed by:

```
window.screen
```

Or,

```
screen
```

- Property of JavaScript Screen Object

No.	Property	Description
1	width	returns the width of the screen
2	height	returns the height of the screen
3	availWidth	returns the available width
4	availHeight	returns the available height
5	colorDepth	returns the color depth
6	pixelDepth	returns the pixel depth.

Variables

A variable is an identifier or a name which is used to refer a value. A variable is written with a combination of letters, numbers and special characters _ (underscore) and $ (dollar) with the first letter being an alphabet.

Ex: c, fact, b33, total_amount etc.

Rules

- Variable can contain combination of letters, digits, underscores (_), and dollar signs ($).
- Must begin with a letter A-Z or a-z or underscore or dollar sign
- A variable name cannot start with a number
- Must not contain any space characters
- JavaScript is case-sensitive
- Can’t use reserved keywords

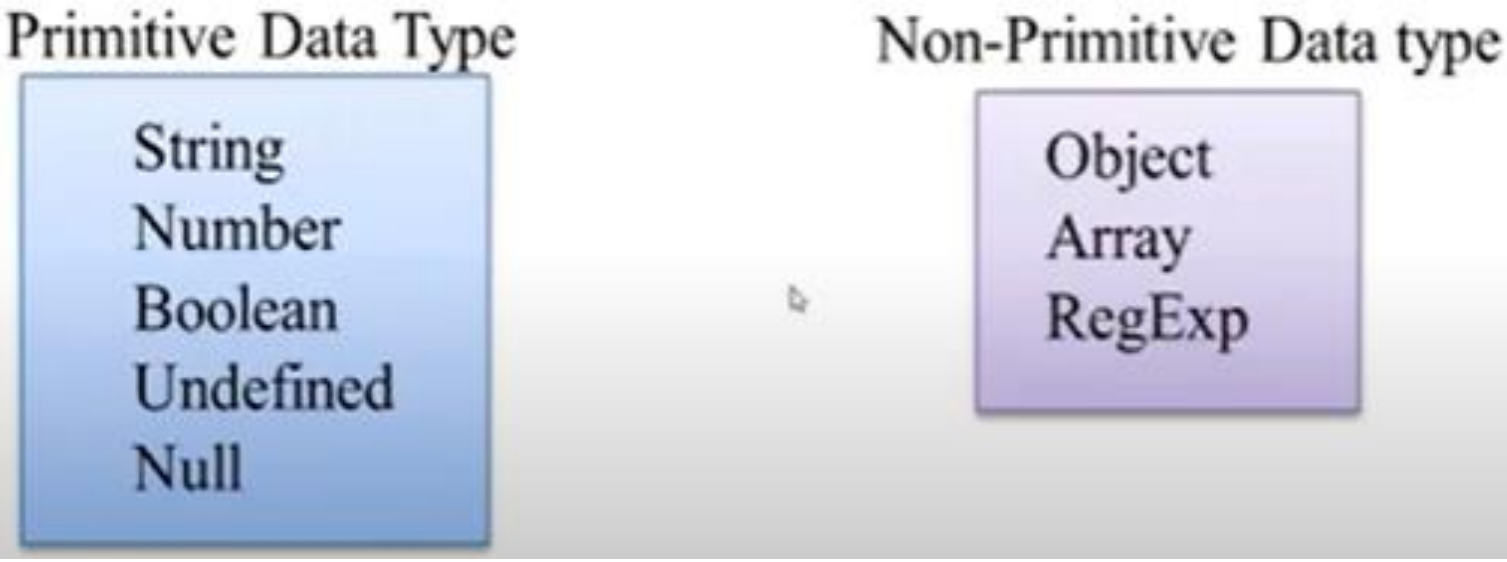
Keywords or Reserved Words

var	delete	for	let	break
super	void	case	do	static
function	new	switch	while	interface
catch	else	if	package	finally
this	with	class	enum	default
implements	private	throw	yield	typeof
const	export	import	protected	return
true	continue	extends	in	instanceof
public	try	debugger	false	

Data Type

In JavaScript we do not need to specify type of the variable because it is dynamically used by JavaScript engine.

We can use **var** data type. It can hold any type of data like String, Number, Boolean, etc.



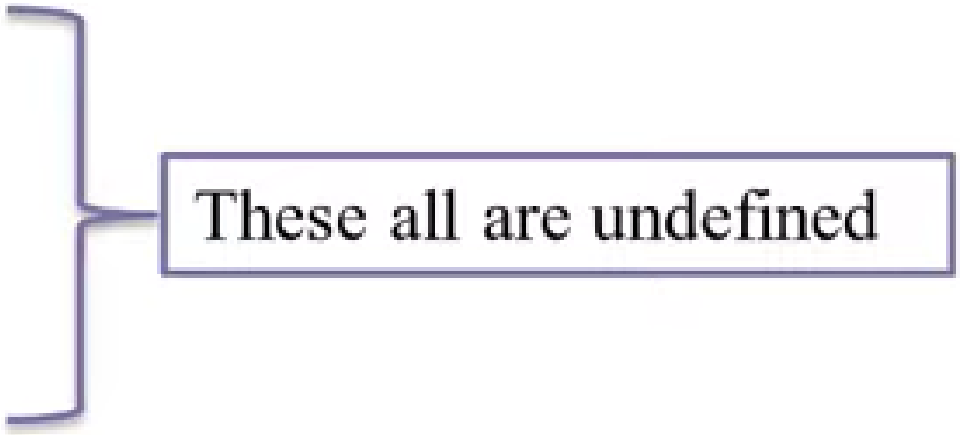
Declaring Variable

- Variable can contain combination of letters, digits, underscores (_), and dollar signs ($).
- Must begin with a letter A-Z or a-z or underscore or dollar sign
- A variable name cannot start with a number
- Must not contain any space characters
- JavaScript is case-sensitive
- Can't use reserved keywords

name	58show
Cab125	Steel City
New_delhi	var
Rup$	

Declaring Variable

```
var roll;
var name;
var price;
```



A variable declared without a value will have the value undefined.

Initializing Variable

```
var roll;
roll = 101;
```

```
var roll = 101;
```

```
roll = 101;
```

```
var name;
name = "geeky shows";
```

```
var name = "geeky shows";
```

```
name = "geeky shows";
```

```
var price;
price = 125.36;
```

```
var price = 125.36;
```

```
price = 125.36;
```

- Strings are written inside double or single quotes.
- Numbers are written without quotes.
- If you put a number in quotes, it will be treated as a text string.

Initializing Variable

var ans = true;

var result = false;

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>

    <body>
    <script>
        var name = "Geeky Shows";
        document.write(name);

    </script>
```

Initializing Variable

```
var x = 10, y = 20, c = 30;
var fname = “Geeky”, lname = “Shows”;
var name = “Geeky Shows”, roll = 101;


var name = “Geeky Shows”,
      roll = 101,
      address = “Steel City”;
```

Re-Declaring Variable

If you re-declare a JavaScript variable, it will not lose its value.

```
var name = “Geeky Shows”;
var name;
document.write(name);
```

- The statements are executed, one by one, in the same order as they are written.
- JavaScript programs (and JavaScript statements) are often called JavaScript code.
- Semicolons separate JavaScript statements.
- Ending statements with semicolon is not required, but highly recommended.
- JavaScript ignores multiple spaces.
- Use line Break (Enter Key).

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>

    <body>
    <script>
        var x = 10, y = 20, c = 30;
        document.write(x);
        document.write(y);
        document.write(c);

    </script>
```

102030

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>

    <body>
    <script>
        var wifi;
        document.write(wifi);


    </script>
</body>
</html>
```

undefined

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>

    <body>
    <script>
        var wifi = "JIO";
        var wifi;
        document.write(wifi);


    </script>
</body>
```

JIO

var, let and const

var - The scope of a variable declared with var is its current execution context, which is either the enclosing function or, for variables declared outside any function, global.

let - let allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used.

const - This declaration creates a constant whose scope can be either global or local to the block in which it is declared. Global constants do not become properties of the window object, unlike var variables. An initializer for a constant is required; that is, you must specify its value in the same statement in which it's declared which can't be changed later.

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>
            var a = 10;
            let b = 20;
            const c = 30;
            document.write(a + "<br>");
            document.write(b + "<br>");
            document.write(c + "<br>");
        </script>
    </body>
</html>
```

Differences between var , let, and const

var	let	const
The scope of a *var* variable is functional or global scope.	The scope of a *let* variable is block scope.	The scope of a *const* variable is block scope.
It can be updated and re-declared in the same scope.	It can be updated but cannot be re-declared in the same scope.	It can neither be updated or re-declared in any scope.
It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.
It can be accessed without initialization as its default value is “undefined”.	It cannot be accessed without initialization otherwise it will give ‘referenceError’.	It cannot be accessed without initialization, as it cannot be declared without initialization.
These variables are hoisted.	These variables are hoisted but stay in the temporal dead zone untill the initialization.	These variables are hoisted but stays in the temporal dead zone until the initialization.

Single Line Comment

Single line comments start with //.

Text between // and the end of the line will be ignored by JavaScript.

```
// you can assign any type of value
var imvalue = 101;

var imvalue = 101;        // assign any type of value
```

Multi Line Comment

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored by JavaScript.

Ex: -

```
/* Comment Here */
```

Adding // in front of a code line changes the code lines from an executable line to a comment.

```
var imvalue = 101;
// var imvalue = 101;
```

JavaScript Operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

Arithmetic Operators

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1
+ +	Increment	A = 10; A+ +	11
− −	Decrement	A = 10; A− −	9

Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === "5"	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == "5"	True

Logical Operators

Operator	Meaning	Example	Result
&&	Logical and	(5<2)&&(5>3)	False
||	Logical or	(5<2)||(5>3)	True
!	Logical not	!(5<2)	True

&&

Operand 1	Operand 2	Result
True	True	True
True	False	False
False	True	False
False	False	False

||

Operand 1	Operand 2	Result
True	True	True
True	False	True
False	True	True
False	False	False

!

Operand	Result
False	True
True	False

Assignment Operators

Operator	Example	Equivalent Expression
=	m = 10	m = 10
+=	m += 10	m = m + 10
−=	m −= 10	m = m − 10
*=	m *= 10	m = m * 10
/ =	m / =	m = m/10
% =	m % = 10	m = m%10
<<=	a <<= b	a = a << b
>>=	a >>= b	a = a >> b
>>>=	a >>>= b	a = a >>> b
& =	a & = b	a = a & b
^ =	a ^ = b	a = a ^ b
| =	a | = b	a = a | b

Getting input from user

prompt( ) – The browser provides a built-in function which can be used to get input from the user, named prompt. The prompt( ) method displays a dialog box that prompts the visitor for input.

Once the prompt function obtains input from the user, it returns that input.

Syntax: - prompt(text, defaultText)

Ex:- prompt("Enter Your Name: ", "name");

prompt("Enter Your Roll No. : ");

```html
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>

    <body>
    <script>
        var a = prompt("Enter Value: ");
        document.write(a);
    </script>
</body>
</html>
```

10

Conditional Statements in JavaScript

- JavaScript conditional statements allow you to execute specific blocks of code based on conditions. If the condition is met, a particular block of code will run; otherwise, another block of code will execute based on the condition.

Conditional Statement	Description
if statement	Executes a block of code if a specified condition is true.
else statement	Executes a block of code if the same condition of the preceding if statement is false.
else if statement	Adds more conditions to the if statement, allowing for multiple alternative conditions to be tested.
switch statement	Evaluates an expression, then executes the case statement that matches the expression's value.
ternary operator	Provides a concise way to write if-else statements in a single line.
Nested if else statement	Allows for multiple conditions to be checked in a hierarchical manner.

If Statement

It is used to execute an instruction or block of instructions only if a condition is fulfilled.

Syntax: -

```
if (condition)
    {
        block of statements;
    }
```

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        var a = 10;
        if(a == 10)
        document.write("You Entered: "+ a);

    </script>
</body>
</html>
```

If else Statement

if… else statement is used when a different sequence of instructions is to be executed depending on the logical value(True/False) of the condition evaluated.

Syntax: -

```
if(condition)
  {
     Statement_1_Block;
  }
else
  {
     Statement_2_Block;
  }
```

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        var a = 10;
        if(a == 10)
        document.write("Name: Rahul");
        else
        document.write("Wrong value");

    </script>
```

Else If Statement

To show a multi-way decision based on several conditions, we use else if statement.

Syntax: -

```
If(condition_1)
  {
    statements_1_Block;
  }
else if(condition_2)
  {
     statement_2_Blocks;
  }
else if(condition_n)
  {
    Statements_n_Block;
  }
else
    statements_x;
```

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        var result = 17;
        if(result <= 30)
        document.write("Fail");
        else if (result <= 40 )
        document.write("Pass");
        else if (result <= 60)
        document.write("Good");
        else
        document.write("Very Good");

    </script>
```

Switch Statement

Check several possible constant values for an expression.

Syntax: -

```
switch(expression){
        case expression 1:
                block of statements 1;
                break;
        case expression2;:
                block of statements 2;
                break;
        .
        .
        default:
                default block of instructions;
```

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        var day = 5;
        switch(day){
            case 1:
            document.write("Sun");
            break;
            case 2:
            document.write("Mon");
            break;
            case 3:
            document.write("Tue");
            break;
            default:
            document.write("Wrong");
```

JavaScript Loops

- The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

- **There are four types of loops in JavaScript.**

1. **for loop**
2. **while loop**
3. **do-while loop**
4. **for-in loop**

For Loop

The for loop is frequently used, usually where the loop will be traversed a fixed number of times.

Syntax:

```
for (initialization;    test condition;      increment or decrement)
{
        block of statements;
}
```

```
for (initialization; test condition;    increment  or decrement)
  {
     block of statements;
  }
```

```
for(i = 0;   i < 5;    i++)
  {
     document.write(i);
  }
```

```
var i = 0;
for( ;   i < 5;    i++)
  {
      document.write(i);
  }
```

```html
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        for(i=0; i<5; i++)
        {
            document.write(i);
        }
    </script>
</body>
</html>
```

```html
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        var i=0;
        for(; i<5; i++)
        {
            document.write(i + "<br>");
        }
        document.write("<br> Rest of the Code");
    </script>
</body>
</html>
```

Nested For Loop

For loop inside for loop.

Syntax:

```
for (initialization;    test condition;      increment or decrement)
{
        block of statements;
        for (initialization;    test condition;      increment or decrement)
        {
                block of statements;
        }
}
```

```
for(i = 0;   i < 3;    i++)
   {
       document.write(i);
       for(j = 0;   j < 5;    j++)
       {
          document.write(j);
       }
    }
```

While loop

The while loop keeps repeating an action until an associated condition returns false.

Syntax:

```
while (test condition)
  {
      body of the loop;
      increment/decrement ;
  }
```

```
var i = 0;
while (i < 5)
  {
     document.write(i);
     i++ ;
  }
```

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        var i = 0;
        while(i<5)
        {
            document.write(i);
            i++;
            document.write("<br>");
        }

    </script>
</body>
</html>
```

Do While Loop

The do while loop is similar to while loop, but the condition is checked after the loop body is executed. This ensure that the loop body is run at least once.

Syntax :

```
do
  {
     statements;
  } while(test condition);
```

```
var i = 0;
do
 {
    document.write(i);
    i++ ;
 } while (i < 5);
```

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        var i = 0;
        do
        {
            document.write(i);
            i++;
            document.write("<br>");
        } while(i<5);
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        for(i=1; i<=10; i++)
        {
            if(i==8)
            {
                break;  // stop loop
            }
            document.write(i);
            document.write("<br>");
        }
    </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        for(i=1; i<=5; i++)
        {
            if(i==3)
            {
                continue;    // skip loop
            }
            document.write(i);
            document.write("<br>");
        }
    </script>
</body>
</html>
```

JavaScript Functions

- JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don’t need to write many lines of code each time to perform a common task.

Type of Function

- Library or Built-in functions
  Ex: - valueOf( ) , write( ), alert( ) etc

- User-defined functions

Creating and Calling a Function

Creating a Function

Syntax: -

```
function function_name( )
{
    Block of statement;
}
```

Body of Function

Ex: -

```
function display( )
{
    document.write("Geekyshows");
}
```

Calling a Function

Syntax: -

```
function_name( );
```

Ex: -

```
display( );
```

Rules

- Function name only starts with a letter, an underscore (_).
- Function name cannot start with a number.
- Do not use reserved keywords. e.g. else, if etc.
- Function names are case-sensitive in JavaScript.

```
<head><title>Geeky Shows</title>
</head>
<body>
    <script>
        // Creating Function
        function display()
        {
            document.write("GeekyShows");
        }

        // Calling Function
        display();

    </script>
</body>
```

```
<head><title>Geeky Shows</title>
</head>
<body>
    <script>
        // How function call works
        document.write("First Line <br>");
        document.write("GeekyShows <br>");
        function display()
        {
            document.write("Inside Function <br>");
        }
        document.write("Last Line <br>");
        display();
        document.write("End Line <br>");
    </script>
</body>
```

```
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>
            // How function call works
            document.write("First Line <br>");
            display();
            document.write("GeekyShows <br>");
            function display()
            {
                document.write("Inside Function <br>");
            }
            document.write("Last Line <br>");
        </script>
    </body>
```

Function with Parameters

Syntax: -

```
function function_name (parameter1, parameter2, ….)
  {
     Block of statement;
  }
```

Ex: -

```
function display(name)
  {
    document.write(name);
  }
```

- JavaScript function definitions do not specify data types for parameters.
- JavaScript functions do not perform type checking on the passed arguments.
- JavaScript functions do not check the number of arguments received.

Call Function with Parameter

Syntax: -

```
function function_name (para1, para2, ….)
 {
    Block of statement;
 }
```

Syntax:-

```
function_name(argument1, argument2);
```

```
function display(name)
 {
     document.write(name);
 }
```

Ex: -

```
display("Geekyshows");
```

```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>
            function display(name)
            {
                document.write(name);
            }
            display(120);
        </script>
    </body>
</html>
```

120

Function Argument Missing

If a function is called with missing arguments, the missing values are set to undefined.

```
function add (a, b, c)
  {
      document.write("A: " + a + "B: " + b + "C: "+ c);
  }

  add (10, 20);
```

```html
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>
            // Function Argument Missing
            function add(a, b, c)
            {
              document.write("A: " + a + " B: " + b + " C: " + c);
            }
            add(10, 20, 30);
        </script>
    </body>
</html>
```

A: 10 B: 20 C: 30

```html
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>
            // Function Argument Missing
            function add(a, b, c)
            {
              document.write("A: " + a + " B: " + b + " C: " + c);
            }
            add(10, 20);
        </script>
    </body>
</html>
```

A: 10 B: 20 C: undefined

Arrow Function

An arrow function expression (previously, and now incorrectly known as fat arrow function) has a shorter syntax compared to function expressions. Arrow functions are always anonymous.

Syntax: -

```
( ) => { statements};
```

```
var myfun = function show( ) {
    document.write("GeekyShows");
};
```

```
var myfun = ( ) =>{document.write("GeekyShows");};
```

```
html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>

            // Arrow Function
            var myfun = () => {
                document.write("GeekyShows");
                };
            myfun();
        </script>
    </body>
/html>
```

```html
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>

            // Arrow Function with parameter
            var myfun = (a) => {
                document.write(a);
                };
            myfun(10);
        </script>
    </body>
</html>
```

```html
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>

            // Arrow Function with parameter
            var myfun = (a, b) => {
                document.write(a + " " + b);
                };
            myfun(10, 20);
        </script>
    </body>
</html>
```

Arrow Function with Default Parameters

```html
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>

            // Arrow Function with parameter
            var myfun = (a, b=20) => {
                document.write(a + " " + b);
                };
            myfun(10, 20);
        </script>
    </body>
</html>
```

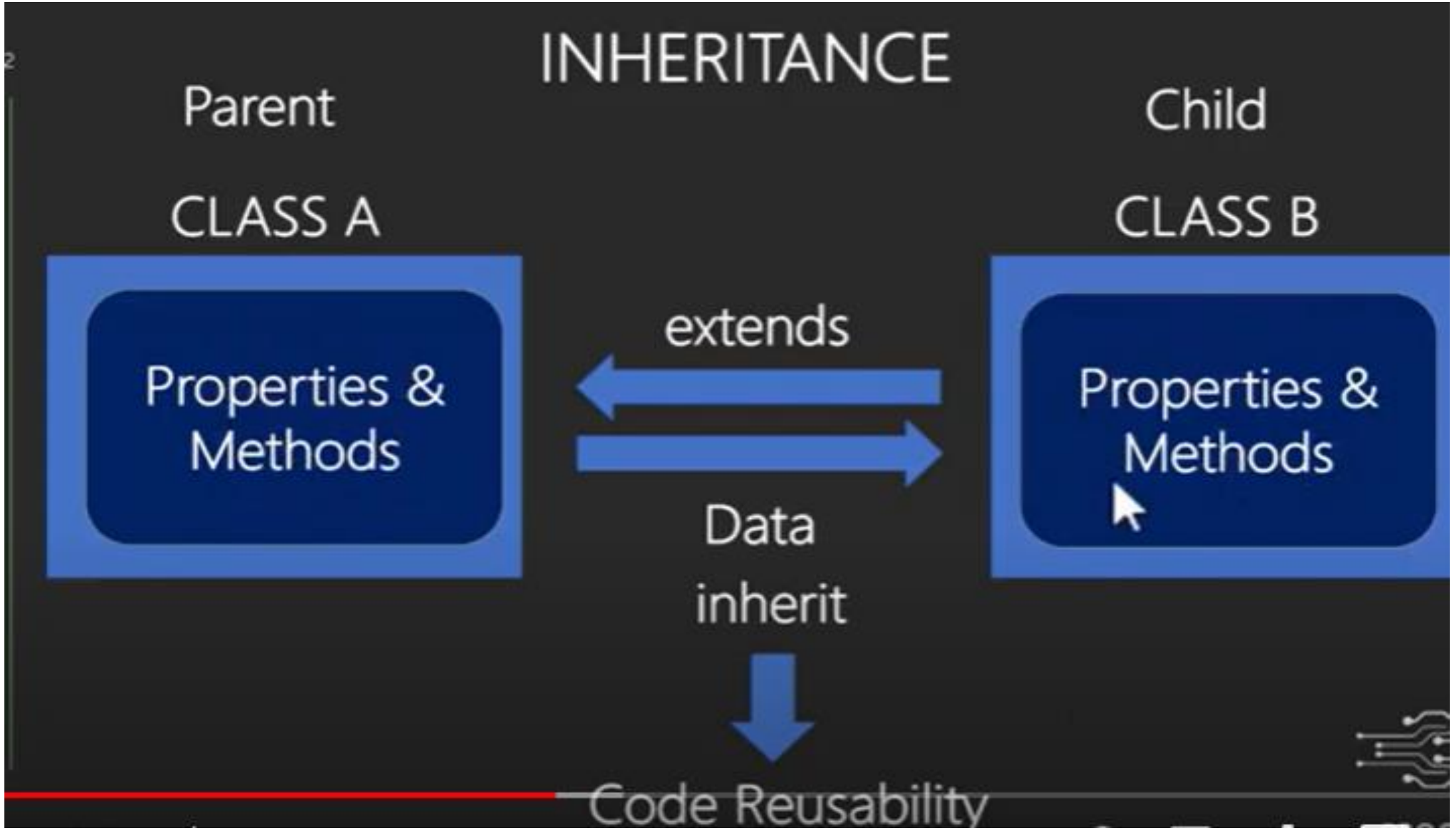
```html
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
        <script>

            // Arrow Function with parameter
            var myfun = (a, b=20) => {
                document.write(a + " " + b);
                };
            myfun(10);
        </script>
    </body>
</html>
```

Advantages of Arrow Functions

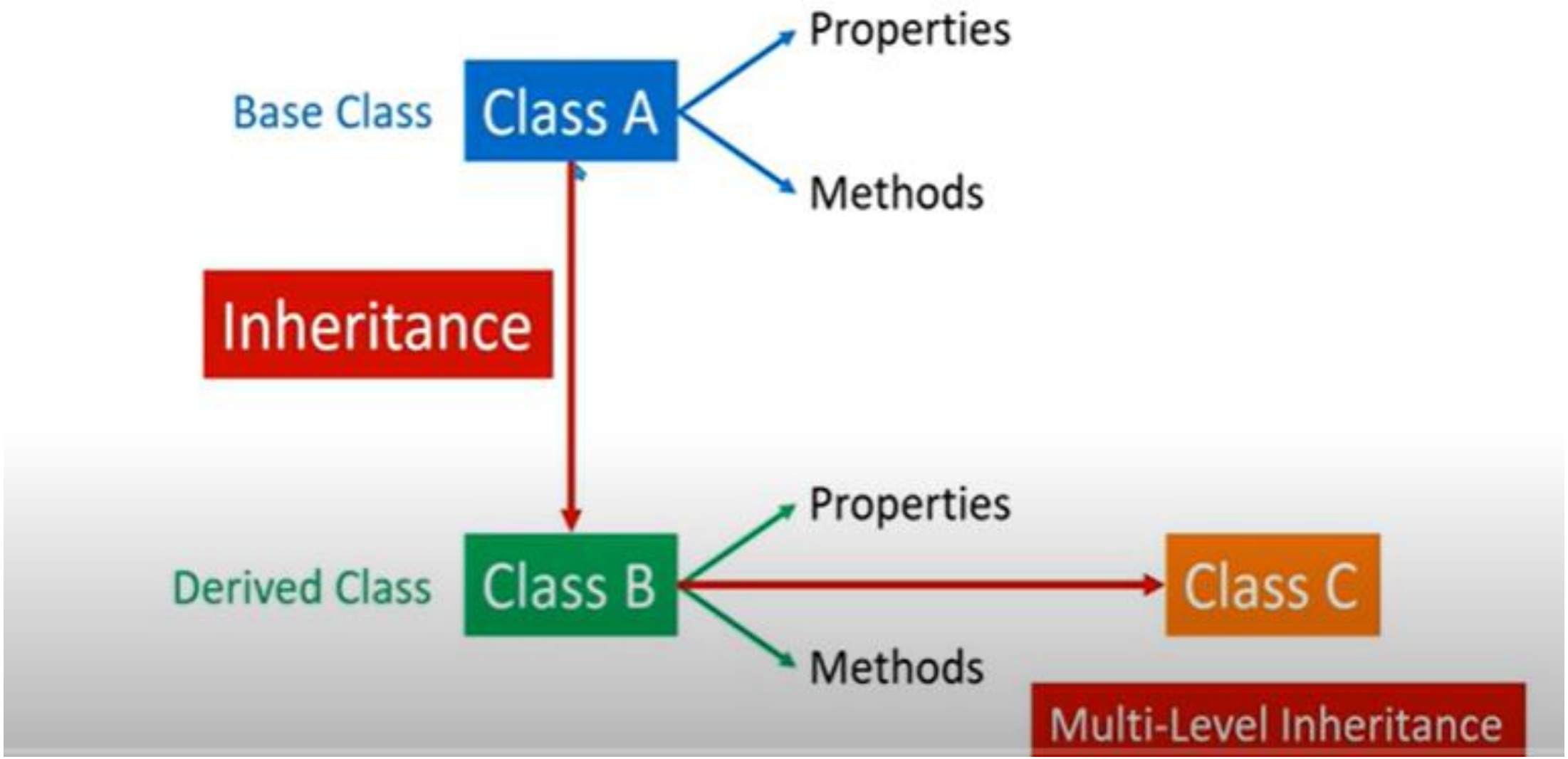
- Arrow functions reduce the size of the code.
- The return statement and function brackets are optional for single-line functions.
- It increases the readability of the code.
- Arrow functions provide a lexical this binding. It means, they inherit the value of "this" from the enclosing scope. This feature can be advantageous when dealing with event listeners or callback functions where the value of "this" can be uncertain.

Limitations of Arrow Functions

- Arrow functions do not have the prototype property.
- Arrow functions cannot be used with the new keyword.
- Arrow functions cannot be used as constructors.
- These functions are anonymous and it is hard to debug the code.
- Arrow functions cannot be used as generator functions that use the yield keyword to return multiple values over time.


INHERITANCE
Parent
Child
CLASS A
CLASS B
Properties & Methods
extends
Properties & Methods
Data
inherit
Code Reusability

What is Inheritance ?

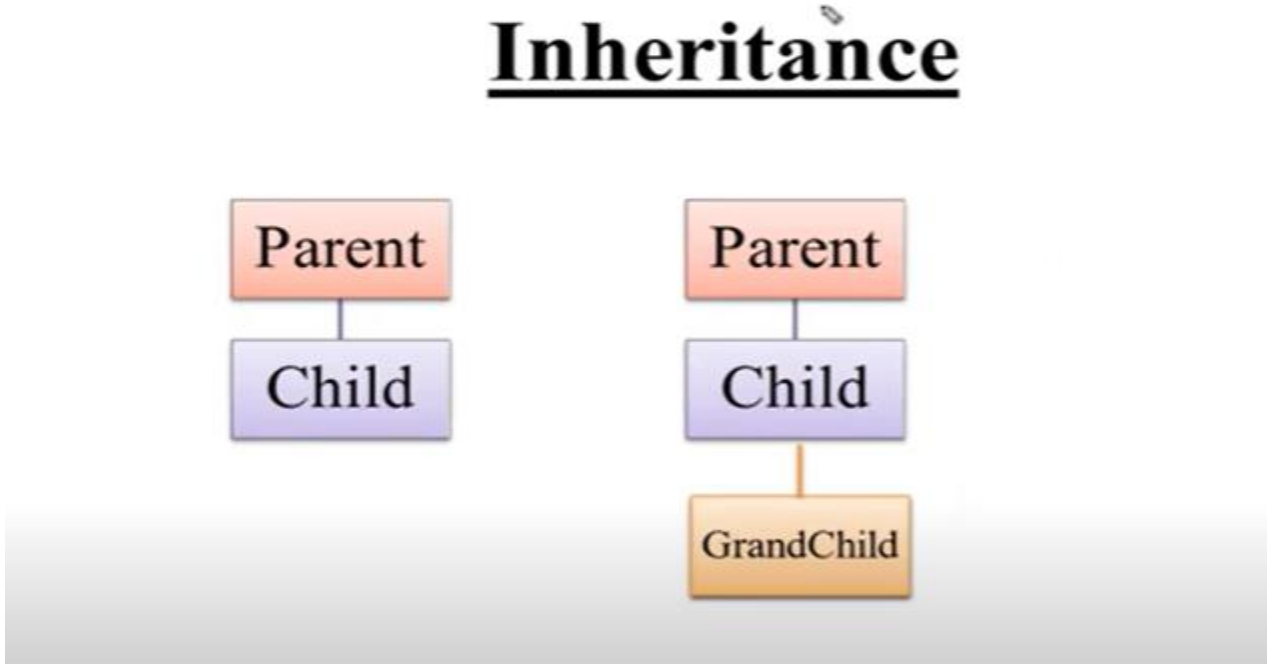


Inheritance in JavaScript

```
class fruits{
    // Properties & Methods
}

class vegetables extends fruits{
    // Properties & Methods
}

let f = new fruits();

let v = new vegetables();
```

JavaScript Inheritance

- The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes. It provides flexibility to the child class to reuse the methods and variables of a parent class.
- The JavaScript **extends** keyword is used to create a child class on the basis of a parent class. It facilitates child class to acquire all the properties and behavior of its parent class.



Class Inheritance

The **extends** keyword is used in class declarations or class expressions to create a class which is a child of another class.

The extends keyword can be used to subclass custom classes as well as built-in objects.

```
class Father {
}


class Son extends Father {
}
```

Class Inheritance

- **Inherit Built-in Object**
  - **Date**
  - **String**
  - **Array**

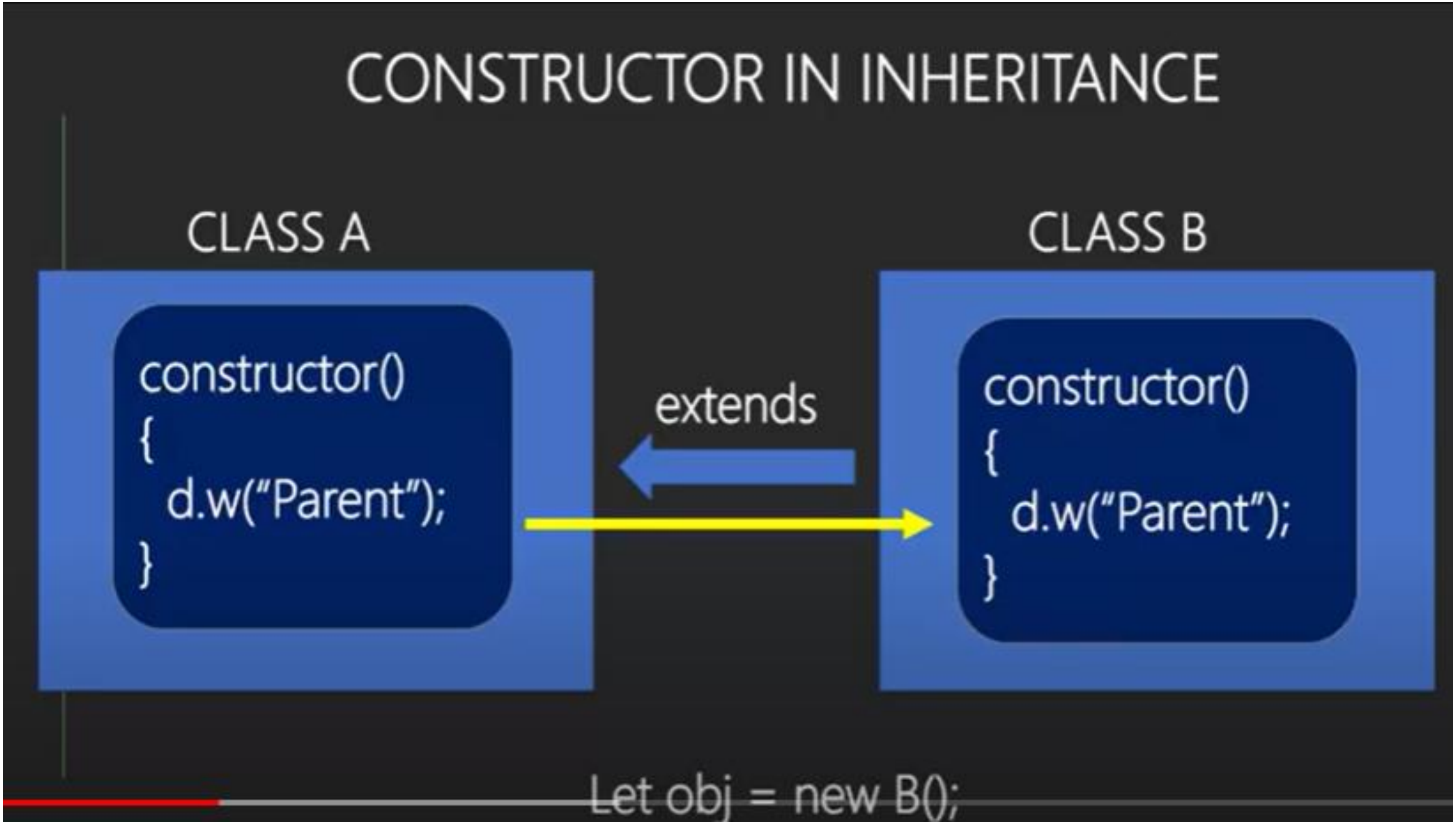
```
class myDate extends Date {


}
```

```
</head>
<body>
    <script>
        class Father {
            showFMoney(){
                return "Father Money <br>";
            }
        }
        class Son extends Father {
            showSMoney(){
                return "Son Money";
            }
        }
        var s = new Son();
        document.write(s.showFMoney());
```

Father Money

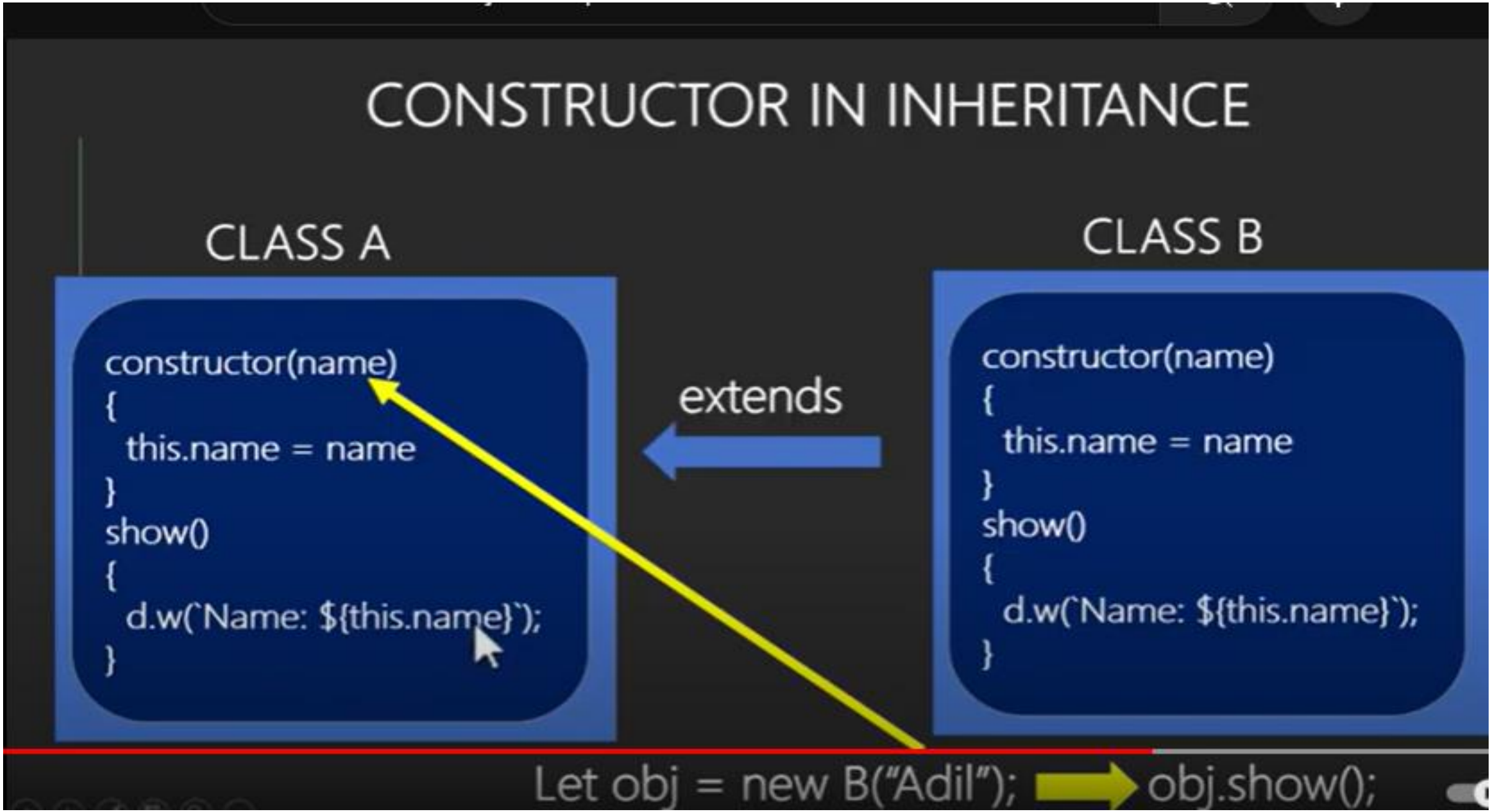
Uses of Inheritance

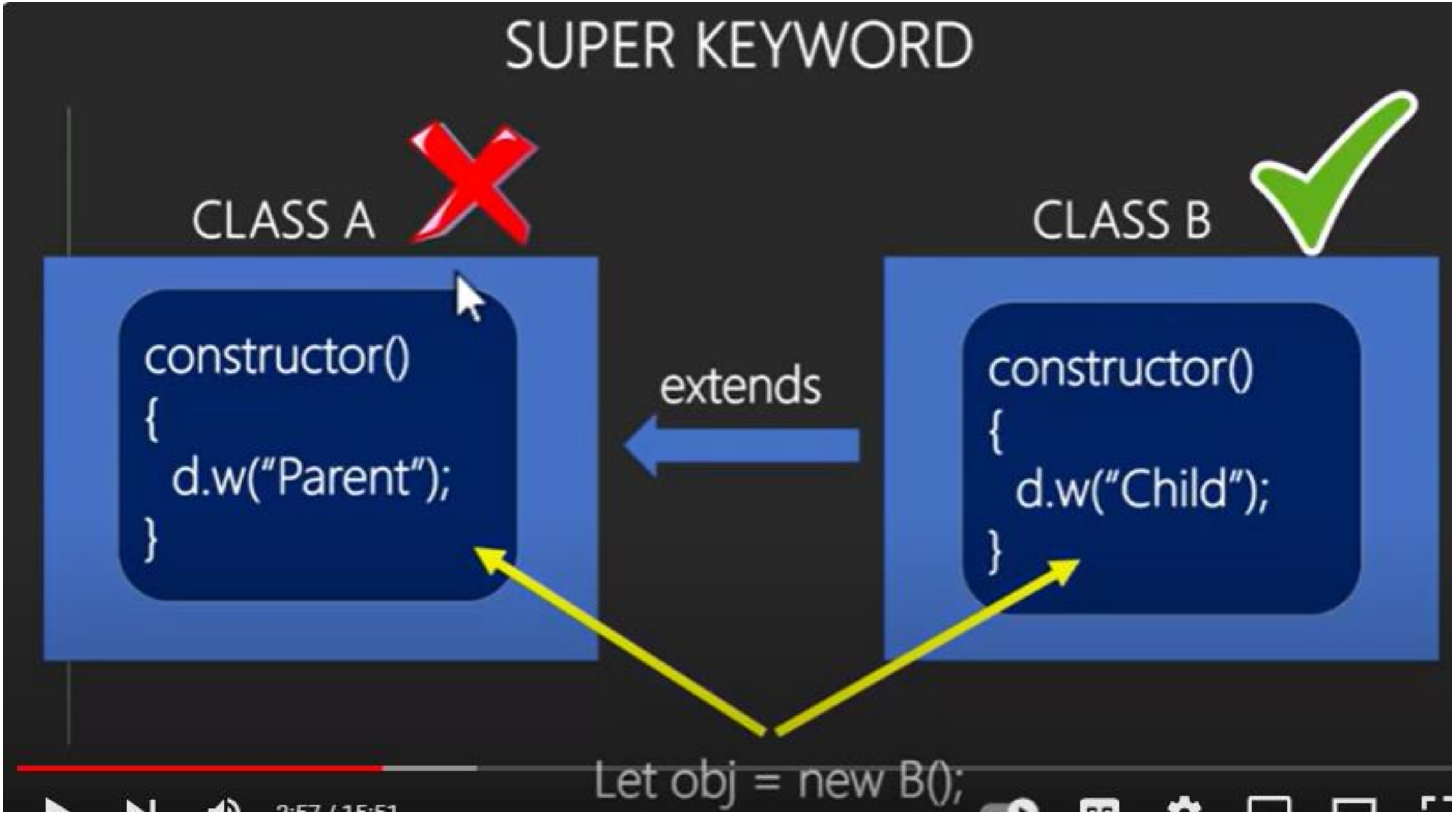
- Since a child class can inherit all the functionalities of the parent's class, this allows code reusability.
- Once a functionality is developed, you can simply inherit it. This allows for cleaner code and easier to maintain.
- Since you can also add your own functionalities in the child class, you can inherit only the useful functionalities and define other required features.

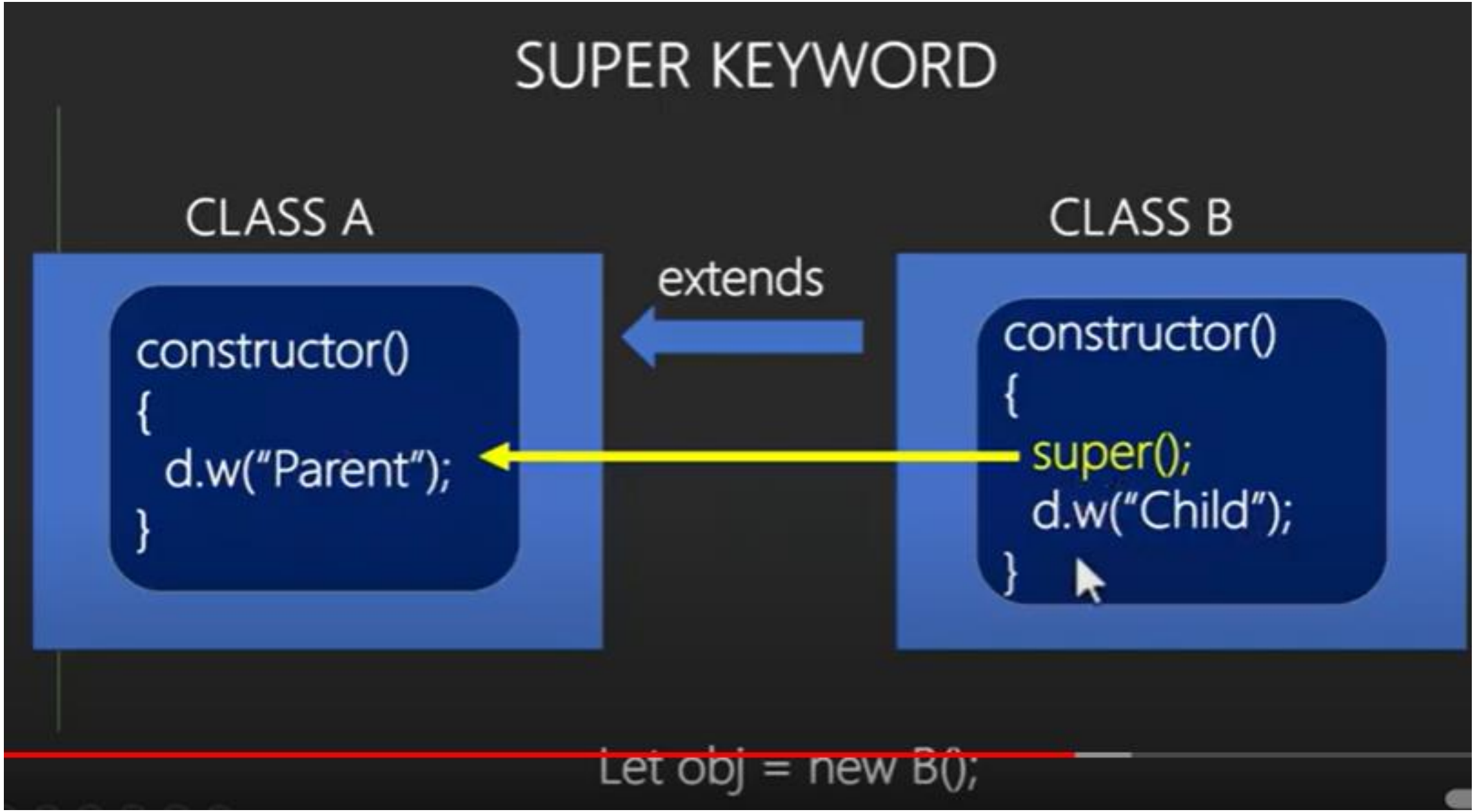

CONSTRUCTOR IN INHERITANCE
CLASS A
CLASS B
constructor()
{
d.w("Parent");
}
extends
constructor()
{
d.w("Parent");
}
Let obj = new B();

```
class A
    {
        constructor()
        {
            document.write(`Parent Class Constructor..`);
        }
    }

class B extends A
{

}

    let obj = new B();
```

Parent Class Constructor..


CONSTRUCTOR IN INHERITANCE
CLASS A
constructor(name)
{
this.name = name
}
show()
{
d.w(`Name: ${this.name}`);
}
extends
CLASS B
constructor(name)
{
this.name = name
}
show()
{
d.w(`Name: ${this.name}`);
}
Let obj = new B("Adil");
obj.show();


SUPER KEYWORD
CLASS A
CLASS B
constructor()
{
d.w("Parent");
}
extends
constructor()
{
d.w("Child");
}
Let obj = new B();


SUPER KEYWORD
CLASS A
CLASS B
extends
constructor()
{
d.w("Parent");
}
constructor()
{
super();
d.w("Child");
}
Let obj = new B();

Super

Super ( ) is used to initialize parent class constructor. If there is a constructor present in subclass, it needs to first call super() before using "this". A constructor can use the super keyword to call the constructor of a parent class.

```
Class Father {
        constructor (money) {
            this.Fmoney = money;
        }
}
Class Son extends Father {
        constructor (money){
          super(money);
        }
}
```

```
class A
    {
        constructor()
        {
            document.write(`Parent Constructor`);
        }
    }

class B extends A
    {
        constructor()
        {
            document.write(`Child Constructor`);
        }
    }

    let obj = new B();
```

Child Constructor

```
class A
    {
        constructor()
        {
            document.write(`Parent Constructor <br>`);
        }
    }

class B extends A
    {
        constructor()
        {
            super();
            document.write(`Child Constructor <br>`);
        }
    }

    let obj = new B();
```

Parent Constructor
Child Constructor

JAVASCRIPT SUPER() KEYWORD

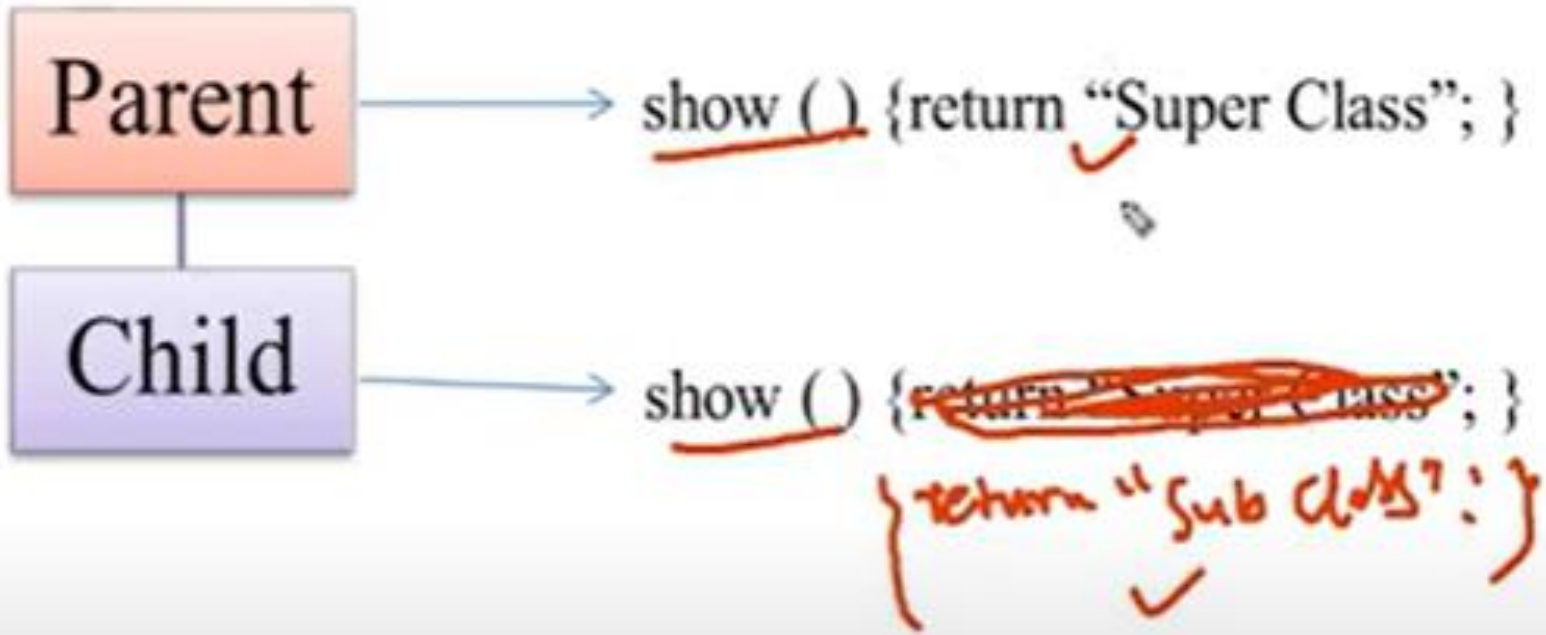
- The super keyword used inside a child class denotes its parent class.
- The super() method refers to the parent class.
- Super keyword is always used in child class.
- Super means Parent

JAVASCRIPT SUPER() KEYWORD

- By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

Method Overriding

Same function name with different implementation.



```
</head>
<body>
    <script>
        class Father {
            show(){
                return "Super Class <br>";
            }
        }

        class Son extends Father {
            show(){
                return "Sub Class";
            }
        }
```

```
}
var s = new Son();
document.write(s.show());
```

Sub Class