**JavaScript(JS)**

**#1 & #2 ) JS Intro. and Why should we learn JavaScript?**

Source: <https://www.youtube.com/watch?v=uDwSnnhl1Ng&list=PLsyeobzWxl7qtP8Lo9TReqUMkiOp446cV&index=1>

Pre-requisite for JavaScript is basic HTML5 & CSS

Tools: VS Code Studio, Chrome or Mozilla Web Browser

VS code studio plugins: Live Server, Prettier

Server: Node JS

1. What is JavaScript?

JS was invented by Brendan Eich in 1995

It is a programming language of the Web. and Java != JavaScript

JS original motive: It was meant to perform “client side scripting” to enhance website’s functionality on the client side.

But overtime it has been diversified & now can be used on server-side as well.

1. What can we do with JS?

We can build WebApps, Real Time Networking Apps, Games, CLI Tools etc. We can use JS to implement backend for desktop apps/mobile apps android etc.

The basic fundamental knowledge of JS is used vastly with various frameworks like follows:

* Node.JS: used for server side web dev. to make dynamic web apps
* React JS: originally developed by Facebook for web & mobile apps
* React Native: for mobile apps iOS/android
* Angular JS: is a MVC based framework created by google & now is open-source
* Electron JS: for desktop based apps
* React VR: can create virtual reality apps
* TensorFlow JS: developed by google, used for AI ML etc.

1. Where does JS code run?

JS was originally designed to run only in web browser which has a JavaScript engine

Following are few of the JavaScript engines:

* Google chrome web browser – v8 engine (open-source)
* Mozilla Firefox – spider monkey

In 2009, Ryan Dahl embedded the v8 engine into a C++ program which is not called as NODE

Node is a C++ program that works with googles v8 engine & hence we can run the JS code outside of a browser i.e. we can pass JS code to Node for execution. Therefore the browser & node provides us a Runtime environment for JS code execution

1. JavaScript vs. ECMAScript?

ECMAScript are a set of specifications whereas JavaScript is a programming language.

ECMA defines guidelines & standards of ECMAScript

We need HTML5, CSS and JavaScript together.

* HTML is to add structure to our web pages <html> <body> <div> <section> tags.
* CSS is used for providing styling to the web pages
* JS is used to program various functionalities or validations to the web page. We can either embed it into the HTML web page or write a separate JS file & import it into the web page.

**#3) What is DOM?**

1. How does the JS work or interact with the HTML web page?

Document Object Model(DOM)

It’s just a different representation of a HTML page. JS used the DOM for the better interpretation of a web page(JS does not work with a HTML web page directly instead works on its DOM representation)

So, using DOM, the JS can easily manipulate the HTML via all the equivalent objects/nodes representation of the HTML tags in a web page, There are 3 types of Node in a DOM

1. Element Node
2. Attribute Node
3. Text Node

**#4) First hello world program in HTML with <script> tag**

We prefer writing large chunks of JavaScript code at the bottom of a web page, because when a browser renders a webpage the JS engine parses the JS code line by line & takes some time.

2 different ways in which we can write JavaScript in a web page.

1. We can write the <script type=’’text/javascript’’></script> tag anywhere in the html file(recommended approach is to write it in the end of a html web page)
2. Write the JavaScript code in an individual file with .js extension & then import this file into the HTML page using the <script> tag

**#5) Environment and Editor Setup**

Download & Install VS Code editor for JS HTML CSS code impl.

Installed & using “Monokai-Contrast” theme for the VS studio

Installed “Live Server” plugin in VS studio for ‘Live reload’ effect of code impl. on web browser(no need to save & then refresh on browser)

**#6) Variables & Datatypes in JS**

What is a variable? : Variable is a named storage used to store value which can variate at runtime based on conditions & info. passed to the program

*Data types in JS:*

*Primitive Types Trivial Types Composite Types*

- Number - null - Objects

- String - undefined - Arrays

- Boolean

Trivial are primitive data types with just a different name

JS is a Loosely Typed programming language & a Dynamically typed scripting language.

Here, we don’t have to define datatypes initially unlike java(**hence it is** **not strict & is loosely typed**) & the variable datatype **dynamically** holds the value which is provided.

var num = 47;

var name = “Roger Stankovic”;

var isAvailable = true;

name = 1213141; // variable ‘name’ earlier was holding a String & now is reused to hold a number making JS a dynamically typed language

Variable creation rules: we can name a variable either by starting with any letter, an underscore(\_) or a dollar sign($). Variable names are case sensitive i.e num, Num, NUM all are different variables in JS.

**#7) Operators in JavaScript**

1. Arithmetic operators: all basic operators as is like +, -, /, \*, %, ++(increment), --(decrement)
2. Comparison operators:

=== equal value **&** equal type

given that var x=5; then the expression (x === 5) is true | (x === “5”) evaluates to false

!== not equal value or not equal type

(x !== 123) true; since not equal value so true | (x !== 5) false; since x is equal to 5, hence false | (x !== “5”) is true; since type of 5(LHS) is number whereas the RHS “5” is a string type

1. Logical/Relational operators: (and) &&, (or)||, (not)!
2. Ternary/Conditional operator: ex: var isAbleToVote = (varAge < 18) ? “Too young” : “Old enough” ;
3. Type operators:

typeof: returns the data type of the variable

instanceof: returns true if an object(LHS) is an instance of the object type (RHS)

1. String operator: ‘+’ operator acts as the concatenation op. for strings “abc”+”xyz” results into “abcxyz” string
2. Bitwise operator: &, |, >>, <<, >>>
3. Assignment operator: ‘=’ operator

**#10) JavaScript control statements**

|-----------------------------------------------|---------------------------------------------|

Conditional control statements Looping/Iterative control statements

if-else & switch for loop, while loop, do while loop, for … in loop

break & continue statement are used for assistance in above constructs

switch case, for loop, while and do-while loop are same as that in java with no conceptual & syntactical difference

**#14) Functions in JavaScript**

A JavaScript function is a block of code designed to perform a particular task upon the function invocation.

var count = 1; // this is a global variable

flag = true; // this is also a global variable

function addOperation(num1, num2) {

var result = num1+num2; // result is a local variable

isAvailable = false; // this is a global variable

}

**“USE STRICT”; Strict mode in JavaScript a more restrictive, secure & restrained way of coding**

By specifying this anywhere inside the script tag we intend to write code in a strict mode which makes JavaScript more secure.

It basically does not allow a variable declarations without a type as below:

use strict;

var count = 12; // this is compiled

counter = 43; // this won’t be compiled & would give an error (bdw. without strict mode this is a global variable)

This helps the developer from by-mistakenly create a global variable by mistyping. If we are not using strict mode then

varcount = 12; // here developer mistakenly creates a global variable as “varcount”; in strict mode this is a compilation error

source: <https://www.w3schools.com/js/js_strict.asp>

deleting a variable (or object) is not allowed.

"use strict";  
var x = 3.14;  
delete x;  // This will cause an error

Deleting a function is not allowed.

function x(param1, param2) {};  
delete x;   // This will cause an error

Duplicating a parameter name is not allowed.

"use strict";  
function x(param1, param1) {};   // This will cause an error

Octal numeric literals and escape characters are not allowed etc. more rules

"use strict";  
var x = 010;             // This will cause an error

var x = "\010";          // This will cause an error

**#16) Arrays in JavaScript**

Arrays in javascript are similar to arrays in java starting with index 0

Following are the 2 ways in which we can define arrays

var names = ["Sam", "Ram", "Molly", "Arjun", 123, false]; // recommended approach

var fruits = new Array("Apple", "Mango", "Banana", true, 451);

A single array can hold different datatypes at a time like above

**#17) Objects in JavaScript**

An Object can be created like follows:

var cellPhone = {

company : "One Plus Org.",

name : "One Plus 6",

color : "Midnight Black",

camera : 48,

ram : 12,

hdd : 128,

operatingSystem : "Andriod lollipop",

displayFeatures : function () {

document.write("<br>Company: "+cellPhone.company+"<br>Name: "+cellPhone.name+"<br>Color: "+cellPhone.color+"<br>Camera: "+cellPhone.camera

+"<br>RAM: "+cellPhone.ram+"<br>HDD: "+cellPhone.hdd+"<br>OS: "+cellPhone.operatingSystem);

}

}

cellPhone.displayFeatures();

**Constructor Function: creates a template**

we create a function having parameters & assign them to class fields. This function ctor can have functions inside them which are called methods.

Following is an ex. Of function constructor & how we create several objects from it & use this template like a class.

<script type="text/javascript">

function FootballClub(country, name, org, estdYear, captain, manager) {

this.country = country;

this.name = name;

this.org = org;

this.estdYear = estdYear;

this.captain = captain;

this.manager = manager;

*// method inside a function constructor*

this.displayClubDetails = function() {

document.write("<br>Country: " + this.country +

"<br>Club Name: " + this.name +

"<br>Club ESTD.: " + this.estdYear +

"<br>Manager: " + this.manager +

"<br>Captain Name: " + this.captain +

"<br>Organising committee: " + this.org

);

}

}

var dynamitesFC = new FootballClub("India", "Dynamites S.C", "1987", "Aniket Bharsakale", "P.D.F.A, Pune, India.", "Himanshu Masani");

dynamitesFC.displayClubDetails();

</script>

We can define properties of a class even after and outside the var class template.

* Boolean, String and Numbers are always objects if we define it using ‘new’. Whereas Date, Maths and regular expressions are objects as well.
* While doing DOM manipulations we deal with the object tree of the html web page representation

**typeof() function**: used to find out the datatype of the provided parameter.

var jetName = "Rafale";

var squadName = new String("Golden Arrows");

document.write("<br>JetName is typeOf: '"+**typeof(jetName)**+"' primitive type"); // prints “string”

document.write("<br>SquadName string is typeOf: '"+**typeof(squadName)**+"' type"); // prints “object”

**#19) DOM manipulations in JavaScript**

var elementId = document.getElementById(“elementId”);

In JS, we use this method to get the hold of elementId of the HTML but in an object style & then can perform operations on them.

To get the value entered by user from an input textfield

document.getElementById(“elementId”).value;

To get value of a html label

document.getElementById(“elementId”).innerHTML;

**Radio Buttons**

We need to provide **type** as radio & **name** of the radioButton group i.e some common grouping name

<html>

<head><title>Radio Button Demo</title></head>

<body>

<input **type="radio"** **name="genderGrp"** id="maleOpt" value="Male">Male</input><br>

<input **type="radio" name="genderGrp"** id="femaleOpt" value="Female">Female</input><br><br>

<input type="button" id="btnSubmit" value="Submit" onclick="displayGender()">

</body>

<script type="text/javascript">

function displayGender() {

var maleRadioBtn = document.**getElementById**("maleOpt");

var femaleRadioBtn = document.getElementById("femaleOpt");

if( **maleRadioBtn.checked** == true )

alert("Your gender is: "+**maleRadioBtn.value);**

else if( **femaleRadioBtn.checked** == true )

alert("Your gender is: "+**femaleRadioBtn.value);**

else

alert("Please select a gender!")

}

</script>

</html>

checked and value are 2 methods used to check if values are true/false & get values & operate upon them

Some Example Context for below 3 methods

<p id = “para1” class=”intro”>Paragraph1</p>

<p id = “para2” class=”intro”>Paragraph2</p>

<p id = “para3” class=”section1”>Paragraph1</p>

<p id = “para4” class=”section1”>Paragraph2</p>

**getElementById(uniqueElementId)**

returns a single unique element based on its unique element. Only one html component is fetched

var para1 = document.getElementById(“para1’);

**getElementByTagName(tagName)**

this helps us to access all the elements of a **particular tag** type, ex: paragraph <p>, anchor <a> etc. it returns a collection of all html tags, upon which we can iterate & perform set of operations,

Here, all 4 above paragraphs are fetched as a collection of html elements & then we can perform any operations on them collectively

var htmlCollection = document.getElementByTagName(“p”);

**getElementByClassName(className)**

here, we can fetch all and any html elements based on their associated **class** tag attribute which is regardless of their tags(p, a, div, . . .etc.)

Below all elements with class = “intro” are fetched as a collection of html elements & we can perform a set of operations on them collectively

var htmlElements = document.**getElementByClassName**(“intro”);

Mouseover & Mouseout event are called over an image & we can write functions so as to perform an action over this triggering event

Following is an example snippet for the usage of the same:

<html>

<head> <title>Mouse Event Demo</title> </head>

<body>

**<img id="img1" src="diwali-pics/crackers.jpg" width="300" onmouseover="setNewImg()" onmouseout="setOldImg()">**

</body>

<script type="text/javascript">

function **setOldImg**() {

**document.getElementById("img1").src = "diwali-pics/crackers.jpg";**

}

function **setNewImg**() {

document.getElementById("img1").src = "diwali-pics/ninja.jpg";

}

</script>

</html>

Form Validation & form tag

Following is an example of the Form Validation. Here we have used a button with **type=”submit”** attribute which aligns with the form tag(we used onclick=”validate()” attribute of button tag till now)

* ‘action' attribute of form tag indicates what action/redirection is to be performed when the form is submitted.
* ‘onsubmit’ attribute of form tag allows us to invoke a function & perform the user authentication with the obtained details.

based on the return value from the validation function, if true the form is submitted & redirected & if false the form is not submitted

* we can perform several validations like, trim whitespaces, disallow empty fields, disallow special characters in username field etc.

<html>

<head>

<title>Form validations</title>

<script type="text/javascript">

function authenticateUser() {

var uid = document.getElementById("uname");

var pwd = document.getElementById("pass");

if(uid.value.**trim()** == "" || pwd.value.trim() == "") {

alert("No blank values allowed!");

return false;

} else {

return true;

}

}

</script>

</head>

<body>

**<form onsubmit="return authenticateUser()" action="message.html" >**

<input id="uname" placeholder="Username" type="text"><br><br>

<input id="pass" placeholder="Password" type="password"><br><br>

<button id="btnLogin" **type="submit">**Login</button>

**</form>**

</body>

</html>

**#28 to #31) Regular Expressions in JavaScript**

* regular expressions are treated as objects in javascript
* we can use regular expressions to validate a mobile number or an email id.

Example regex for email validation: **var regExEmail = /^([a-zA-Z0-9\.-]+)@([a-zA-Z0-9]+).([a-z]{2,8})(.[a-z]{2,8})?$/;**

**Ex**. Regex for mobile no. validation: **var regExMobile = /^[7-9]\d{9}$/;**

**#35) Timing functions in JavaScript**

**setTimeout(functionReference, miliseconds):** we use it to basically invoke a function after a specific time interval, this executes the mentioned function only once.

It accepts a reference to a function(i.e a function name & not a function call) which is to be invoked after ‘x’ milliseconds are elapsed.

**var timerId = window.setTimeout(printMessage, 3000);** is from window class, on execution it returns a timer as return value

printMessage is a reference to the printMessage (…) function while 3000 is 3000 miliseconds(3seconds).

This means JS will execute the printMessage() function **once** after 3 sec. are elapsed

**window.clearTimeout(timerId):** This is used to clear the timeout set by the setTimeout function(both work together)

we can get the timerId from the setTimeout() & use it as param in this clearTimeout() function which is essential

**var timerId = window.setInterval(functionRef, ms)**

This is similar to setTimeout(fn, ms) but setInterval allows us to execute a mentioned function multiple times unlike setTimeout()

Here, timerId = window. **setInterval** (printTimerMsg, 3000); after every 1 second printTimerMsg function will be executed

**window.clearInterval(timerId)**

this function is similar to window.clearTimeout() but works well with setInterval()

<html>

<head>

<title>Timing Interval in JavaSCript</title>

<script type="text/javascript">

var secondsCounter = 0;

var timerId = 0;

function start() {

//timerId = window.setTimeout(printTimerMsg, 3000);

timerId = window.setInterval(printTimerMsg, 1000);

}

function stop() {

//window.clearTimeout(timerId);

window.clearInterval(timerId);

}

function printTimerMsg() {

document.getElementById("someTxt").innerHTML = +secondsCounter+" seconds.";

secondsCounter++;

}

</script>

</head>

<body>

<button id="btnStart" onclick="start()">Start</button>

<h1 id="someTxt">Some Text</h1>

<button id="btnStop" onclick="stop()">Stop</button>

</body>

</html>

**#35) Zoom in and Zoom out animation transition example**

<html>

<head>

<title>Basic Animation Part - 2 (Zoom In and Out)</title>

</head>

<body>

<img src="\diwali-pics\ninja.jpg" id="imgId" width="200px" **onmouseover**="increase()" **onmouseout**="decrease()">

</body>

<script type="text/javascript">

var difference = 2;

var imgId = document.getElementById("imgId");

var intervalId = 0;

var width = 200;

function **increase**() {

intervalId = setInterval(**zoomIn**, 20);

}

function **zoomIn**() {

if(width < 500) {

width = width + difference;

imgId.style.width = width;

} else {

clearInterval(intervalId);

}

}

function **decrease**() {

intervalId = setInterval(**zoomOut**, 20);

}

function **zoomOut**() {

if(width > 200) {

width = width - difference;

imgId.style.width = width;

} else {

clearInterval(intervalId);

}

}

</script>

</html>

**#34) Fade In and Fade out animation transition example**

<html>

<head>

<title>Basic Animation part-1 - Fade in & Fade out Demo</title>

</head>

<body>

<button id="fadeIn" onclick="fadeIn()" >Fade In</button><br><br><br>

<img id="imgId" src="\diwali-pics\ninja.jpg" width="300px"><br><br><br>

<button id="fadeIn" onclick="fadeOut()">Fade Out</button><br><br>

<button id="stopId" onclick="stop()">Stop Animation</button>

</body>

<script type="text/javascript">

var intervalId = 0;

var opacity = 0;

var imgId = document.getElementById("imgId");

function fadeOut() {

intervalId = window.**setInterval**(**hide**, 200);

}

function **hide**() {

**opacity = Number ( window.getComputedStyle(imgId).getPropertyValue("opacity") );**

if( opacity > 0 ) {

**opacity = opacity - 0.1;**

**imgId.style.opacity = opacity;**

} else

window.clearInterval(intervalId);

console.log("opacity: "+opacity);

}

function **fadeIn**() {

intervalId **= window.setInterval(unHide, 200);**

}

function **unHide**() {

opacity = **Number ( window.getComputedStyle(imgId).getPropertyValue("opacity") );**

if( opacity < 1 ) {

opacity = opacity + 0.1;

imgId.style.opacity = opacity;

} else

window.clearInterval(intervalId);

console.log("opacity: "+opacity);

}

function stop() {

window.clearInterval(intervalId);

}

</script>

</html>

**jQuery**

* It is a JavaScript library which is small, fast and feature rich JS-library.
* The purpose of jQuery is to make it much easier to use JavaScript on our website.
* Write-Less and Do-More
* Cross browser compatible
* Open-source library

jQuery basically reduces the boilerplate code for most of the common tasks by wrapping them into functions which we can invoked in just a single-line

It also simplifies many complex things from JavaScript like AJAX calls & DOM manipulation

jQuery provides following features:

1. DOM and CSS manipulation
2. AJAX
3. HTML event methods
4. Effects & animations
5. jQuery utilities

We can download the jQuery library JS file on local & then use it via import just like: <script src="\jquery\jquery-lib.js"></script>

Other way to use the jQuery is via CDNs (recommended way to use jQuery library)

<script src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossorigin="anonymous"></script>

Here we will be using the jQuery from the CDN server instead of our local which is secure, fast & reliable.

**jQuery Selectors**

jQuery selectors allows us to manipulate the HTML elements via its own set of methods.

Basically we can select the html elements & perform operation on them

These selectors are used to find/select the html elements based on their name, id, class, type, attributes, values of attributes & much more.

They are based on the existing CSS selectors & have their own custom selectors as well.

Following is a demo of selectors in jQuery

<html>

<head>

<title>Selectors in jQuery Demo</title>

<link rel="stylesheet" href="css/style.css">

</head>

<body>

<h2 class="header" **id="lbl1"**>jQuery Selector Demo</h2><br><br>

<button onclick="fadeout()">Fade Out</button>

<h2 **class="subHead"** id="lbl2">Dot operator(.)</h2>

<h2 class="subHead" id="lbl3">Hash operator(#)</h2>

<h2 class="subHead" id="lbl4">Some text</h2>

<br><br>

<button onclick="vanisher()">Fade Out</button>

</body>

<script src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossorigin="anonymous"></script>

<script type="text/javascript">

**// Here, we access an HTML element based on its ID via the hash(#) & then perform an action(fadeToggle) over it**

function fadeout() {

**$("#lbl1").fadeToggle(1000);**

}

**// Here, we use the Dot(.) this helps us to perform an action on group of html elements based on their class attribute**

function vanisher() {

**$(".subHead").fadeToggle();**

}

</script>

</html>

Elements are accessed by #

Classes are accessed by .

There are several selector types with different usage: <https://www.w3schools.com/jquery/jquery_ref_selectors.asp>

**Events in jQuery**

In HTML & simple JavaScript, we used the **onclick** event attribute of a button

<button id="btnFadeOut" **onclick**=”fadeout()”>Fade Out</button>

**Now we sail by the jQuery ways . . . .**

In jQuery, we delete the onclick event attribute from HTML, & here onward, all the events will be handled using jQuery syntax in javascript

$("#btnFadeOut").**click**(**fadeOut**); // fadeout is function reference to fadeout(…) function

// here we created an anonymous function inside the click() function

$("button").**click**( function() {

$("p:odd").fadeToggle(1000);

});

This below line means, only when the entire webpage is loaded in the web browser then only all the JS functionalities will be available for usage

**$(document).ready(function() {**

// all regular JS code with JS functions etc.

function fadeout( ) {

// . . .

}

$("p:odd").fadeToggle(1000);

// selector to fade out entire 'div' tag

**$("div").**fadeToggle(1000);

// selector to fade out the entire div tag, also all other paragraph elements from other div tags

**$("#firstDiv, p, .introDiv").**fadeToggle(1000);

// class selector to hide all elements associated or having this tag

**$(".introDiv").**fadeToggle(1000);

// selector to hide specific elements(say, paragraph tag) inside a div tags (both are same)

**$("div > p")**.fadeToggle(1000);

**$("div p").**fadeToggle(1000);

// selector to fade out very first html element

**$("p:first").**fadeToggle(1000);

// selector to fade out even html elements from list

**$("li:even").**fadeToggle(1000);

// selector to fade out odd html elements from list

**$("li:odd").**fadeToggle(1000);

**});**

More about jQuery events: <https://www.w3schools.com/jquery/jquery_events.asp>

**double click event**

$("#btnVanisher").**dblclick**(fadeOut); // we need to double click on the button with id= btnVanisher

// **mouseenter** event which is equivalent to onmousein

/\*$("#btnVanisher").**mouseenter**( function() {

$("#list1").fadeOut(1000);

});

// **mouseleave** event is equivalent to onmouseout event

$("#btnVanisher").**mouseleave**( function() {

$("#list1").fadeIn(1000);

});\*/

// **hover** event is a combination of mouseenter & mouseleave which accepts 2 functions respectively

$("#btnFadeOut").**hover**(fadeIn, fadeOut);

**Effects in jQuery**

Using the element/hash(#) selector, we select the elements from html webpage & perform animation effects on them like, sliding effect, hide/show or fade in/fade out

Following is the example of these effects in jQuery with anonymous functions usage

<html>

<head>

<title>Effects in jQuery - Part 1</title>

<link rel="stylesheet" href="/14-jquery/css/style.css">

</head>

<body>

<!-- observe here, no onclick attribute provided here-->

<div id="myEffectButtons" class="jQueryEffects">

<button id="btnHide">Hide</button>

<button id="btnShow">Show</button>

<button id="btnToggleShowHide">Show-Hide-Toggle</button>

<button id="btnFadeOut">Fade Out</button>

<button id="btnFadeIn">Fade In</button>

<button id="btnFadeToggle">Fade Toggle</button>

<button id="btnSlideUp">Slide Up</button>

<button id="btnSlideDown">Slide Down</button>

<button id="btnSlideToggle">Slide Toggle</button>

</div>

<h1>Introduction to jQuery</h1>

<p>paragraph 0 - Title</p>

<p>Paragraph 1 - Author</p>

<div id="firstDiv" class="introDiv">

<p>Paragraph 2 - Preface</p>

<p>Paragraph 3 - Synopsis</p>

<p>Paragraph 4 - Thanksgiving</p>

</div><br>

<div id="secondDiv" class="introDiv">

<p>Paragraph 5 - Index & Table of Content</p>

<p>Paragraph 6 - Content</p>

</div><br>

<div id="thirdDiv" class="introDiv">

<p>Paragraph 7 - Story</p>

</div><br>

<ul class="introDiv" id="list1">

<h4>

<li>List0 Item0 - Selectors</li>

<li>List0 Item1 - Effects </li>

<li>List0 Item2 - Animation</li>

<li>List0 Item3 - Events</li>

</h4>

</ul><br>

</body>

<script src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossorigin="anonymous"></script>

<script type="text/javascript">

// More about jQuery events on :https://www.w3schools.com/jquery/jquery\_events.asp

$(document).ready(function() {

// element(buttonId) selector used & upon that element's click event we perform a jQuery "hide()" effect

**$("#btnHide").click( function() {**

**$("#firstDiv").hide();**

**});**

// element(buttonId) selector used & upon that element's click event we perform a jQuery "show()" effect

$("#btnShow").click(function(){

$("#firstDiv").**show**();

});

// toggle() effect is the combination of hide() & show() effect

$("#btnToggleShowHide").click(function(){

$("#firstDiv").**toggle**();

});

$("#btnFadeOut").click(function(){

$("#secondDiv").**fadeOut**();

});

$("#btnFadeIn").click(function(){

$("#secondDiv").**fadeIn**();

});

$("#btnFadeToggle").click(function(){

$("#secondDiv").**fadeToggle**();

});

$("#btnSlideUp").click(function(){

$("#list1").**slideUp**();

});

$("#btnSlideDown").click(function(){

$("#list1").**slideDown**();

});

$("#btnSlideToggle").click(function(){

$("#list1").**slideToggle**();

});

});

</script>

</html>

**Callback functions and Chaining functions in jQuery**

<script src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossorigin="anonymous"></script>

<script type="text/javascript">

$(document).ready(function() {

// **CALLBACK FUNCTION**

// on btnAnimate's click event we want to perform an sliding action effect on #thirdDiv

$("#btnCallbackEffect").click(function(){

// Here, the function reference 'printMsg' is a callback function

$("#thirdDiv").slideToggle(2000, **printMsg**);

});

// my callback function for slideToggle function; all effects functions have a callback function which is optional for us

**function printMsg(){**

alert("Effect executed!");

}

// **CHAINING OF FUNCTIONS**

$("#btnChainedEffect").click(function(){

$("#list1").**slideToggle**(1500).**fadeToggle**(1500).**fadeIn**().**fadeOut**().**slideUp**().**slideDown**();

});

});

</script>

**DOM Manipulations in jQuery**

**text( )**: this method prints the text content as it is even though if it contains any html tags in it.

when using it as empty text( ) it acts as the getter method, it will the text from the paragraph with id=para1

text(“string”), this acts as the setter method & it overwrites the para1 with the provided text

**html( )**: this method absorbs & applies all the valid html tags inline to the text present

when we get/set text using this method, the html tags are not visible and get applied at runtime instead

**attr( )**: this method is used to fetch the values of attributes of an html element

**css( )**: this method is used to fetch the css style related info like below

<html>

<head><title>DOM Manipulations in jQuery</title></head>

<body>

<h3>DOM Manipulations in jQuery</h3>

<button id="btnText">text(...)</button>

<button id="btnHtml">html(...)</button>

<button id="btnAttr1">GET-VALUE-attr(...)</button>

<button id="btnAttr2">SET-VALUE-attr(...)</button>

<button id="btnCss1">GET-CSS</button>

<button id="btnCss2">SET-CSS</button>

<div id="div1" class="myDivs">

<p id="p1" class="p1ClassValue" >Paragraph-1</p>

<p id="p2" class="p1ClassValue" >Paragraph-2</p>

</div>

</body>

<!-- CDNs jQuery import -->

<script src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossorigin="anonymous"></script>

<script type="text/javascript">

$(document).ready(function(){

**// text(): simply gets the text content for the given html content**

$("#btnText").click(function(){

var p1Text = **$('#p1').text();**

alert("We just fetch the text in plain text format from p1 paragraph :"+p1Text);

});

**// html(...) - acts as a Setter method & overwrites the existing content of this html element**

$("#btnHtml").click(function(){

**$('#p1').html("New content of the P1 paragraph is set");**

alert("We just changed the P1 paragraph content!");

});

**// using attr() to get the existing attribute values of html elements**

$("#btnAttr1").click(function(){

var classVal = **$('#p1').attr("class");**

alert( "GET p1's 'class' value = "+ classVal );

});

**// using attr() to set new value to the attributes of the html elements**

$("#btnAttr2").click(function(){

**$('#p1').attr("class", "new\_p2ClassValue"); // set new value to class**

var classVal = **$('#p1').attr("class"); // get value**

alert( "The NEW 'class' value to p1 is = "+ classVal );

});

**// using css() to get the existing css style values**

$("#btnCss1").click(function(){

var fontFamily = **$("#p2").css("font-family");**

var fontSize = $("#p2").css("font-size");

var color = $("#p2").css("color");

alert("Paragraph2 Details = FontFamily: "+fontFamily+" -- fontSize: "+fontSize+" -- Color: "+color);

});

**// using the css() to set new values at runtime**

$("#btnCss2").click(function(){

**$("#p2").css("font-family", "Algerian");** // setting new values to existing css attributes

$("#p2").css("font-size", "40px");

$("#p2").css("color", "red");

});

});

</script>

</html>

Following are few more methods that are used commonly to manipulate DOM via jQuery

**append( ) | prepend( ) | after( ) | before( ) | remove( ) | empty( )**

Following is a working example of all methods mentioned above

<html>

<head>

<title>DOM Manipulations in jQuery - PART 2</title>

<link rel="stylesheet" href="css/style.css">

</head>

**<!--**

**append: inserts content at the end of the selected element**

**after: inserts content after the selected element**

**prepend: inserts content at the beginning of the selected element**

**before: inserts content before the selected element**

**remove: removes or deletes the html tag from webpage**

**empty: clears the content inside the provided element**

**-->**

<body>

<h3>DOM Manipulations in jQuery</h3>

<button id="btnAppend">Apend</button>

<button id="btnPrepend">Prepend</button>

<button id="btnAfter">After</button>

<button id="btnBefore">Before</button>

<button id="btnEmpty">Empty</button>

<button id="btnRemoveDiv">Remove Div</button>

<button id="btnRemovePara">Remove Para</button>

<button id="btnRemovePara13">Remove Para 1&3</button>

<div id="div1" class="myDivs">

<p id="p1" class="p1ClassValue" > --- Paragraph-1 from 1st Division --- </p>

<p id="p2" class="p2ClassValue" > --- Paragraph-2 from 1st Division --- </p>

<p id="p3" class="p3ClassValue" > --- Paragraph-3 from 1st Division --- </p>

</div>

</body>

<script src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossorigin="anonymous"></script>

<script type="text/javascript">

**// append: adds a new para. inside the div but at the end of the selected para. tag**

$("#btnAppend").click(function(){

$("#div1").**append**("<p>\*\*\* Paragraph-2 from 1st Division \*\*\*</p>");

});

**// prepend: adds a new para. at the beggining of the selected element(div tag)**

$("#btnPrepend").click(function(){

$("#div1").**prepend**("<p>\*\*\* Paragraph-2 from 1st Division \*\*\*</p>");

});

**// after: adds a new para. after the selected element(div tag) | outside div tag**

$("#btnAfter").click(function(){

$("#div1").**after**("<p>\*\*\* Paragraph-2 from 1st Division \*\*\*</p>");

});

**// before: adds a new para. before the selected element(div tag) | outside div tag**

$("#btnBefore").click(function(){

$("#div1").**before**("<p>\*\*\* Paragraph-2 from 1st Division \*\*\*</p>");

});

**// remove: removes/deletes the mentioned element from webpage**

$("#btnRemoveDiv").click(function(){

$("#div1").**remove**();

});

**// remove: removes te para with class: p2ClassValue**

$("#btnRemovePara").click(function(){

$("p").**remove(".p2ClassValue");** // using dot(.) selector

alert("Paragraph 2 of 1st Div. will be removed");

});

**// remove: removes te para with class: p2ClassValue**

$("#btnRemovePara13").click(function(){

$("p").**remove("#p1, #p3")**; // using dot(.) selector

alert("Paragraph 1 & 3 of 1st Div. will be removed now");

});

**// empty: clears ALL the internal html tags of the specified element**

$("#btnEmpty").click(function(){

$("#p1**"**)**.empty( );**

});

</script>

</html>

**addClass( ) | removeClass( ) | toggleClass( )**

<html>

<head>

<title>DOM Manipulations in jQuery - PART 3</title>

<link rel="stylesheet" href="css/style.css">

</head>

**<!--**

**addClass: it can dynmically adds one or more css styles to webpage**

**removeClass: it can dynmically remove one or more css styles to webpage**

**toggleClass: dynamically adds a css style class if its not applies & removes it if its already applied**

**-->**

<body>

<h3>DOM Manipulations in jQuery</h3>

<button id="btnAddClass">Add Class</button>

<button id="btnRemoveClass">Remove Class</button>

<button id="btnToggleClass">Toggle Class</button>

<div id="div1" class="myDivs">

<p id="p1" class="p1ClassValue"> --- Paragraph-1 from 1st Division --- </p>

<p id="p2" class="p2ClassValue"> --- Paragraph-2 from 1st Division --- </p>

<p id="p3" class="p3ClassValue"> --- Paragraph-3 from 1st Division --- </p>

</div>

</body>

<script src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossorigin="anonymous"></script>

<script type="text/javascript">

$(document).ready(function(){

**// addClass: this adds the css style class dynamically upon the click event**

$("#btnAddClass").click(function(){

$("#div1").**addClass**("newDivClass newParaClass");

});

**// removeClass: this will remove the dynamically added css style class at runtime**

$("#btnRemoveClass").click(function(){

$("#div1").**removeClass**("newDivClass newParaClass");

});

**// toggleClass: this will dynamically add the css style class at runtime if absent & will remove it if it’s already applied**

$("#btnToggleClass").click(function(){

$("#div1").**toggleClass**("newDivClass newParaClass");

});

});

</script>

</html>

**CSS**

.myDivs {

border: 2px solid red;

margin: 10px;

padding: 10px;

}

.**newDivClass** {

border-radius: 10px;

background-color: yellow;

}

.**newParaClass** {

font-weight: bold;

color: blueviolet;

}

**jQuery UI**

It is a powerful javascript library built on top of jQuery JavaScript Library. It is free and open-source.

jQuery is the core lib. but the **jQuery UI**(user interface) is build using jQuery core i.e on top of it.

Basically it is a set of plugins for jQuery that adds new functionalities to the jQuery core library.

Hence when using **core jQuery** we should use the **jQuery UI** as well.

**We can use the jQuery U.I from: https://jqueryui.com/**

Download the stable version & import the necessary files into your project

**V.IMP NOTE:** the jQuery core <script> import statement needs to be on top followed by the jQuery U.I <script> import statement

<!-- imported **jQuery Core** Library MAIN/PARENT -->

<script src="/jquery/**jquery-lib.js**" type="text/javascript"></script>

<!-- imported **jQuery U.I Library** SUBMAIN/CHILD -->

<script src="/jquery/jquery-ui/jquery-ui.js" type="text/javascript"></script>

What jQuery U.I provides us extra?

1. Interactions like draggable, droppable, resizable, selectable, sortable
2. Widgets – Accordion, Autocomplete, Button, Checkboxradio, DatePicker etc
3. More animation effects
4. Utilities
5. Menu and Navigation bars etc

**Date Picker in jQuery U.I**

Following is a working example of date picker in jQuery UI

<html>

<head>

<title>Intro. to jQuery U.I</title>

<!-- **Importing the "jQuery UI"** into our webpage -->

<link rel="stylesheet" href="/jquery/jquery-ui/**jquery-ui.theme.css**"/>

<link rel="stylesheet" href="/jquery/jquery-ui/**jquery-ui.structure.css**"/>

<link rel="stylesheet" href="/jquery/jquery-ui/**jquery-ui.theme.css**"/>

<link rel="stylesheet" href="/jquery/css/style.css"/>

</head>

<body>

<div id="myDiv">

<h2>Intro. to jQuery U.I</h2>

<label>Hotel Check-In Date: </label>

**<input** id="**myDatePicker**" placeholder="Select Date">

<button id="submitDate" >Submit Date</button>

</div>

</body>

<!-- **imported jQuery Core Library** MAIN/PARENT -->

<script src="/jquery/jquery-lib.js" type="text/javascript"></script>

<!-- imported **jQuery U.I Library** SUBMAIN/CHILD -->

<script src="/jquery/jquery-ui/jquery-ui.js" type="text/javascript"></script>

<script type="text/javascript">

$(document).ready(function(){

// For More on Date Picker: **https://api.jqueryui.com/datepicker/**

**// Here, we have added a Date Picker to the Input field & added few more attributes**

**$("#myDatePicker").datepicker({**

**numberOfMonths:1,**

**changeYear:true,**

**changeMonth:true,**

**showWeek:true,**

**weekHeader: "Wk No.",**

**showOtherMonths: true,**

**minDate: new Date(2017, 11, 15),**

**maxDate: new Date(2021, 11, 15)**

**});**

**// to get the selected date from the Input field**

$("#submitDate").click(function(){

var jsDate = **$("#myDatePicker").datepicker('getDate');**

if ( jsDate !== null ) {

console.log("InstanceOf: "+ (jsDate instanceof Date));

console.log("Date: "+ jsDate.getDate());

console.log("Month: "+jsDate.getMonth());

console.log("Year: "+jsDate.getFullYear());

}

**// NOTE: indexing of months start from 0 i.e 0=Jan and 11=Dec**

**alert("Day: "+jsDate.getDate()+" -- Month: "+(jsDate.getMonth()+1)+" -- Year: "+jsDate.getFullYear());**

});

});

</script>

</html>

**Accordion widget in jQuery UI**

Following is a working example with comments

<html>

<head>

<title>Accordion Widget Demo</title>

<!-- Importing the "jQuery UI" into our webpage -->

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.structure.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="/jquery/css/style.css"/>

</head>

<body>

<div id="indexDiv">

<h2>Selectors in jQuery</h2>

<p>Dot(.) selector is used to select html elements by their class attribute.<br>

Hash(#) selector is for selecting unique html elements by their ID.</p>

<h2>Animation Effects in jQuery</h2>

<p>Fade Toggle effect is a combination of Fade In & Fade Out effect.<br>

Similarly, Slide Toggle is a combo of slideUp & slideDown effect.<br>

Whereas the toggle() effect is th combo of show() & hide() effect.</p>

<h2>DOM Manipulation using jQuery</h2>

<p>We can use following set of methods to manipulate a webpage using jQuery.<br>

- html()<br>

- attr()<br>

- css()<br>

- text()<br>

- append()<br>

- prepend()<br>

- after()<br>

- before()<br>

- remove()<br>

- empty()<br>

- addClass()<br>

- removeClass()<br>

- toggleClass()<br>

</p>

<h2>Events in jQuery</h2>

<p>Following are few events in jQuery<br>

> dblclick(functionRef)<br>

> mouseenter(functionRef)<br>

> mouseleave(functionRef)<br>

> hover(jQueryEffect1, jQueryEffect2)<br>

</p>

<h2>Introduction to jQuery U.I</h2>

<p>Download jQuery U.I from: <a href="https://jqueryui.com/">jQuery UI Official</a> <br>

Following are the custom items that one can use from JQUERY USER INTERFACE<br>

> Date Picker<br>

> Accordion widget<br>

> Custom message pop-up<br>

> Auto-complete widget<br>

</p>

</div>

</body>

<!-- imported **jQuery Core** **Library** MAIN/PARENT -->

<script src="/jquery/**jquery-lib.js**" type="text/javascript"></script>

<!-- imported **jQuery U.I Library** SUBMAIN/CHILD -->

<script src="/jquery/jquery-ui/**jquery-ui.js**" type="text/javascript"></script>

<script type="text/javascript">

$(document).ready(function(){

// below is the Accordion widget applied on the parent div tag & all the attributes used to make it more usable & look neat

**$("#indexDiv").accordion({**

collapsible: true, // the selected item will collapse

event: "click", // these are different ways or effects to open the blocks ex: "mouseover"

animate: 400, // the time required to open the blocks

active: 1, // indexing is from 0

heightStyle: true, // making the blocks height dynamic in nature & not static

icons: {

header:"ui-icon-plusthick", // the in-active/collapsed tab

activeHeader:"ui-icon-minusthick" // the active tab | icons source: **https://api.jqueryui.com/theming/icons/**

}

**});**

});

</script>

</html>

**Custom Message pop-up in jQuery-UI**

<html>

<head>

<title>Accordion Widget Demo</title>

<!-- Importing the "jQuery UI" into our webpage -->

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.structure.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="css/style.css"/>

</head>

<body>

<h2>Message Box in jQuery UI</h2>

<div id="firstDiv" class="myDivs">

<h3>Welcome to jQuery & JavaScript learning</h3>

<br>

<button id="btnDisplayMessageBox">Display Message Box</button>

</div>

<br>

<div id="msgBox" title="Default Message Box Title" class="myDivs">

<h3>This is a Dummy Title</h3>

<img src="/diwali-pics/jqueryui-pic.png" width="100px" >

<br><br>

<button>Close</button>

</div>

</body>

<!-- imported jQuery Core Library MAIN/PARENT -->

<script src="/jquery/jquery-lib.js" type="text/javascript"></script>

<!-- imported jQuery U.I Library SUBMAIN/CHILD -->

<script src="/jquery/jquery-ui/jquery-ui.js" type="text/javascript"></script>

<script type="text/javascript">

$(document).ready(function(){

$("#btnDisplayMessageBox").click(function(){

// This provides the custom message dialog from jQuery UI

**$("#msgBox").dialog({**

title: "Custom Message Box Title", // Header of the alert pop-up

draggable: true, // defines if you can drag the pop-up or not

resizable: true, // allows us to resize the pop-up

height: 200,

width: 280,

modal: true, // blocks the background access & makes the pop-up as a mandatory

// below are the **custom buttons** with **custom icons** & text(name-value) pairs

buttons: [

{

text: "Close",

icon: "ui-icon-close", // icons source: https://api.jqueryui.com/theming/icons/

click: function() {

$(this).dialog("close");

}

},

{

text: "OK",

icon: "ui-icon-check",

click: function() {

alert("OK is OKAY too! :p")

}

}

]

**});**

});

});

</script>

</html>

**Auto-Complete feature in jQuery UI**

Working example of auto-complete feature using jQuery-UI

<html>

<head>

<title>Accordion Widget Demo</title>

<!-- Importing the "jQuery UI" into our webpage -->

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.structure.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="css/style.css"/>

</head>

<body>

<h2>Auto-Complete Widget UI Demo</h2>

<div id="firstDiv" class="myDivs">

<label><b>Enter your favorite City: </b></label>

<input type="text" **id="txtCityBox"**>

</div>

</body>

<!-- imported **jQuery Core Library** MAIN/PARENT -->

<script src="/jquery/**jquery-lib.js**" type="text/javascript"></script>

<!-- imported **jQuery U.I Library** SUBMAIN/CHILD -->

<script src="/jquery/jquery-ui**/jquery-ui.js**" type="text/javascript"></script>

<script type="text/javascript">

$(document).ready(function(){

**// This can be some JSON/REST API we might be fetching data from Front-end & using it dynamically here**

var cities = ["Pune", "Mumbai", "Goa", "Tawang", "Mysore", "Jaipur", "Jamshedpur", "New Delhi",

"Srinagar", "Imphal", "Agartala", "Mysore", "Panjim", "New York", "New Jersey"];

**$("#txtCityBox").autocomplete(**

{

// source: cities // any way is correct

source: ["Pune", "Mumbai", "Goa", "Tawang", "Mysore", "Jaipur", "Jamshedpur", "New Delhi",

"Srinagar", "Imphal", "Agartala", "Mysore", "Panjim", "New York", "New Jersey"]

}, {

autofocus: true, // the 1st item in the list is under focus

delay: 0, // time required to fetch the items

minLength: 2 // minimum 2 letter input required for populating the auto-complete list

**});**

});

</script>

</html>

**Interaction in jQuery U.I**

**Source: https://api.jqueryui.com/category/interactions/**

**Draggable Interaction**

<html>

<head>

<title>Accordion Widget Demo</title>

<!-- Importing the "jQuery UI" into our webpage -->

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.structure.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="/jquery/css/style.css"/>

</head>

<body>

<h1>Interactions in jQuery UI</h2><br>

<h2>Draggable interaction</h2>

<div style="border: 2px solid red;">

<div class="**cube**">

<h3>CUBE-1</h3>

</div>

<div class="**cube**">

<h3>CUBE-2</h3>

</div>

<div class="**cube**">

<h3>CUBE-3</h3>

</div>

</div>

</body>

<!-- imported jQuery Core Library MAIN/PARENT -->

<script src="/jquery/jquery-lib.js" type="text/javascript"></script>

<!-- imported jQuery U.I Library SUBMAIN/CHILD -->

<script src="/jquery/jquery-ui/jquery-ui.js" type="text/javascript"></script>

<script type="text/javascript">

$(document).ready(function(){

**// source: https://api.jqueryui.com/draggable/**

**$(".cube").draggable({**

axis: "x",

distane: 50,

cursor: "grabbing",

opacity: 0.5,

containment: "parent",

//grid: [300, 300],

snap: true,

snapTolerenace: 100

**});**

});

</script>

</html>

**Sortable Interaction**

<html>

<head>

<title>Accordion Widget Demo</title>

<!-- Importing the "jQuery UI" into our webpage -->

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.structure.css"/>

<link rel="stylesheet" href="/jquery/jquery-ui/jquery-ui.theme.css"/>

<link rel="stylesheet" href="/jquery/css/style.css"/>

</head>

<body>

<h1>Interactions in jQuery UI</h2><br>

<h2>Sortable interaction</h2>

<div id="**geometry**">

<div class="cube">

<h3>CUBE-1</h3>

</div>

<div class="cube">

<h3>CUBE-2</h3>

</div>

<div class="cube">

<h3>CUBE-3</h3>

</div>

</div>

</body>

<!-- imported jQuery Core Library MAIN/PARENT -->

<script src="/jquery/jquery-lib.js" type="text/javascript"></script>

<!-- imported jQuery U.I Library SUBMAIN/CHILD -->

<script src="/jquery/jquery-ui/jquery-ui.js" type="text/javascript"></script>

<script type="text/javascript">

$(document).ready(function(){

**// here, we need to apply the sortable on the parent tag & not on the desired tag(i.e we need to apply the sortable on desired tag’s parent specifically)**

**// source: https://api.jqueryui.com/sortable/**

**$("#geometry").sortable({**

cursor: "grabbing",

opacity: 0.5,

containment: "parent",

//delay: 0,

axis: "x"

**});**

});

</script>

</html>

**Function scope & Global scope**

Variables declared in global scope are available in functions/loops & in other blocks as well.

But variables declared inside the functions or loops or blocks will be available to that block itself & not outside that block

**var**: is simply a variable whose value can change after its declaration overtime.

var health = 100;

health = health - 10;

var health = 100; // this line does not give error: we can re-declare/re-define the same variable later too

**const** is constant value however, the value once defined cannot change. (similar to final in java)

const planets = 9;

planets = 10; // **error**: since it’s a const the value cannot be changed

const planets = 22; // **error**: cannot re-declare the same variable name; need to change name or delete this

**let**: acts like a variable & allows to change the value overtime

let level = 4;

level = level + 1; // valid

let level = 12; // **error**: does not allow to re-declare a new variable with a used variable name

=== operator checks the data type & the value for the given variable

**“this” keyword in JavaScript**

“this” keyword in javaScript is almost similar to the “this” keyword in Java

‘this’ refers to the context of the current object in action

In regular/usual javascript, the ‘this’ keyword refers to

Following is an example to demonstrate how this behaves differently with regular JS function & arrow function

// 'this' in regular function and 'this' in arrow function

class Person {

constructor(name) {

this.name = name;

}

printNameArrowFunction() {

setTimeout(() => {

console.log("Arrow: "+this.name);

}, 100);

}

printNameRegularFunction() {

setTimeout(function(){

console.log("Function: "+this.name);

}, 100);

}

}

let person = new Person("Roger");

person.printNameArrowFunction();

person.printNameRegularFunction();

console.log(this.name);

/\*

"Standard JS function syntax defines 'this' based on from where the function is invoked.""

the method execution "person.printNameRegularFunction();" is from GLOBAL scope & hence in regular JS function the 'this' gets redefined

We should not use the 'this' keyword inside a Standard JS function since 'this' gets redefined in that context & there is a different behaviour

Whereas in Arrow function, the 'this' does not get/is not redefined. It does not matter from where a function is invoked.

For arrow function, be it any scope the this works just as expected unlike that in some regular js function/block scoped scenario

\*/

Little more firewood to the fuel

// ----------- OLD JS - this keyword -----------

var person = {

firstName: "Roger",

lastName: "Stankovic",

age: 24,

printAge: function() {

console.log("Age: "+this.age);

that = this; // we need to rejuvenate the this reference again & reuse it

// here, 'this' is redefined | its scope gets redefined to printAge() block level

var fullName = function() {

console.log("Old JS Workaround - "+that.firstName+" "+that.lastName);

console.log("Old JS 'this' issue - "+this.firstName+" "+this.lastName);

};

fullName();

}

};

person.printAge();

// ----------- this keyword - ES-6 New -----------

console.log(`ES-6 - 'this' KEYWORD USAGE`);

const human = {

fName: "Joachim",

lName: "Lowe",

ageInYears: 57,

countryOrigin: "Germany",

occupation: "Football Club Manager",

isCurrentlyEmployed: true,

isActive: true,

**// regular old JS function with an arrow function inside it - VALID**

**printDetails: function( ){**

**printFullName = ( ) => {**

**// inside an arrow function, the scope of 'this' keyword is not re-defined & hence it behaves as expected**

**console.log(`ArrowFunction \nFullName: ${this.fName} ${this.lName}`); // valid & yields the fName and lName as is**

**}**

**printFullName();**

**console.log(`Age: ${this.ageInYears} Country Origin: ${this.countryOrigin} `);**

**}**

}

human.printDetails();

**Arrow functions in JavaScript ES-6**

// regular function with 2 or more params

function sum(a, b ) {

return a + b;

}

// optimized arrow function with 2 params

const sum2 = (a, b) => a + b

// regular function with 1 param

function isPositive(number) {

return number >= 0;

}

// optimized arrow function with 1 param

let isPositive2 = number => number >= 0;

// regular function with 0 param

function randomNumber () {

return Math.random;

}

// optimized arrow function with 0 param

const randomNumber2 = () => Math.random;

// anonymous function which is a callback function

document.addEventListener('click', function(){

console.log("Click");

});

// arrow function which is a callback function

document.addEventListener('click', () => console.log("Click"));

**Variables**

// Old way

// global variable | global scope

var counter = 10;

counter = 1213;

console.log("counter: "+counter);

// we are able to re-declare same variable name again which is **bad**

var counter = 12;

console.log("counter: "+counter);

// block scope

function printName(name) {

var randomNum = Math.random(); // local scope

console.log("Hi "+name+". You hold bitcoins quantity of: "+randomNum);

}

printName("Aniket");

// console.log("counter: "+randomNum); // error in console due to scope issue for ‘randomNum’

// here, var bleeds out of block scope & is available outside for loop too which is bad

for(var i=0; i<5; i++) {

console.log("counter"+i);

}

console.log("counter outside forloop: "+i);

// ES-6

for(let j=0; j<5; j++) {

console.log("es6-let: "+j);

}

// console.log("counter outside forloop: "+j); // gives error: ’j’ is undefined

**Template Literals/String concatenation in ES-6**

//Old way: we need to escape special characters like ‘ or “ and also need to manage white spaces manually & also need to append with + sign

var name = "Aniket";

console.log("My name is **\"**"+name+"**\"**. It**\'**s been great to work with **\"**Edwin**\**" so far.");

// **New Way**: Here the white spaces are managed along with the escape characters being resolved & no + to use. Instead use the ${variable}

**console.log(`Hello, My name is "${name}" and It's been an amazing experience of learning JavaScript with "Edwin".`)**

**Object Literals**

// Object Literals - Old Way

function getBookDetails (name, author, price, version) {

return {

name: name,

author: author,

price: price,

version: version

}

}

console.log(getBookDetails("Rich Dad Poor Dad", "Robert Kiyosaki", "$15", "oldJS-Vanila"));

// Object Literals - NEW ES-6 Way : Here we don’t have to repeat the params to be returned

function getBookDetails (name, author, price, version) {

return {

**name,**

**author,**

**price,**

**version**

}

}

// we dont need to repeat the parameter assignments like old way

console.log(getBookDetails ("Finance Management", "Some Nerd Geeky Guy", "$115", "ES-6"));

**Object Deconstruction**

// Old JS way

var personalInformation = {

name: "Aniket",

age: 23,

gender: "Male",

country: "India"

}

**var name = personalInformation.name;**

console.log("\nOld JS way\nName: "+name);

// ES-6 New Way : Object deconstruction

let userPersonalInformation = {

fullName: "Aniket Bharsakale",

age: 23,

gender: "Male",

country: "India",

contactNo: 10237107403284,

isPassportAvailable: true,

isEmployed: false

}

**let fullUserInfo = userPersonalInformation; // gets the entire Object as is**

**let {contactNo, age, isEmployed} = userPersonalInformation; // gets only the specified fields**

**const {fullName} = userPersonalInformation // to get single field also, we need the curly braces**

console.log(`\nES-6 New Way\nName: ${fullName} \nContact Number: ${contactNo}\n`);

age = age + 1 // able to change the age since it’s a ‘let’

isEmployed = true;

console.log(`Age: ${age}\nisEmployed: ${isEmployed}`);

**Arrow Functions**

var a = 10; var b = 5;

// OLD-1: regular function with 2 or more params

function sum(a, b ) {

return a + b;

}

// NEW-1: arrow function with 2 params

const sum2 = (a, b) => a + b

// OLD-2: regular function with 1 param

function isPositive(number) {

return number >= 0;

}

// NEW-2: optimized arrow function with 1 param

let isPositive2 = number => number >= 0;

// OLD-3: regular function with 0 param

function randomNumber () {

return Math.random;

}

// NEW-3: optimized arrow function with 0 param

const randomNumber2 = () => Math.random;

// OLD-4: anonymous function which is a callback function

document.addEventListener('click', function(){

console.log("Click");

});

// NEW-4: arrow function which is a callback function

document.addEventListener('click', () => console.log("Click"));

**Default Parameters**

// Old way

function multiply(a, b) {

**var x = a || 1;** // If nothing is passed then default value is 1 for a & b

**var y = b || 1;**

return x \* y;

}

console.log("14\*10 = "+multiply(14,10)); // prints 140

console.log("No params passed to multiply( ) = "+**multiply( )** ); // prints 1

// New Way - ES6 – Default Parameters syntax

**const sum = (a = 1, b = 1) => {**

**return a + b;**

**}**

console.log("ES6 Regular SUM = "+**sum(10, 10)**);

console.log("ES6 Default SUM = "+**sum( ));**

**Array Map and Filter in JS – ES6**

// Regular Array

let **fruitList** = ["apples", "papayas", "mangoes", "bananas", "oranges"];

console.log("Original Fruits List:");

// **New ForEach Loop** - here we can iterate on each product in linear style

**fruitList.forEach( ( product, index ) => {**

**console.log(`${index} - ${product}`);**

**});**

console.log("Fruits without index(Single Param Arrow Function):");

fruitList.**forEach**( product => console.log(`${product}`) );

// **NEW Map function** - Get list, we can update this data before storing it in map

let updatedFruitList = **fruitList.map**( ( product, index ) => {

if(index % 2 == 0)

return "fresh "+product;

else

return product;

});

console.log("Updated Fruits List:");

updatedFruitList.**forEach**((updatedProduct, index) => {

console.log(index+" - "+updatedProduct);

});

// **NEW Filter in ES6** - FILTER: GET FRESH FRUITS ONLY

const freshFruitsFilteredList = **updatedFruitList.filter**( ( product) => {

if(product.startsWith("fresh"))

return product;

});

console.log(`FILTERED FRUITS LIST IS BELOW:`);

freshFruitsFilteredList.**forEach**( filteredItem => {

console.log(filteredItem);

});

**Constructor Function**

It’s a blueprint/template/function which when invoked creates an object.

// ----------- Old JS way - Constructor Functions -----------

function Person(name, age, country) {

this.name = name;

this.age = age;

this.country = country;

}

// a method of Person class

Person.prototype.printName = function(version) {

console.log("Name: "+this.name+" from "+version+" version");

};

var Ed = new Person("Ed", 23, "Denmark");

Ed.printName("OLD JS VANILLA");

// function Employee -> (extends) funciton Person

function Employee(empId, designation, contactNum, name, age, country) { // add fields of Person here

Person.call(this, name, age, country); // this entends the Person inside Employee

this.empId;

this.designation;

this.contactNum;

// we need to copy paste the fields of Person to Employee param list

}

// this inherits all the functions from Person to Employee

Employee.prototype = Object.create(Person.prototype);

const emp = new Employee("P14S63XZ", "Manager", 537021987074, "Roger", 23, "Ireland");

emp.printName("OLD JS VANILLA");

**// ----------- ES6 NEW WAY -----------**

**class SoccerClub {**

**constructor (numberOfPlayers, division, version)** {

this.numberOfPlayers = numberOfPlayers;

this.division = division;

this.version = version;

}

printClubDetails( ) {

console.log(`A club can have ${this.numberOfPlayers} players in the ${this.division} division of ${this.version} version.`);

}

}

// Object creation by New ES6 way

const soccerClub = new SoccerClub("30", "Second", "ES6 NEW JS");

soccerClub.printClubDetails();

**class DynamitesSC extends SoccerClub** {

**constructor(clubName, numberOfPlayers, division, version) {**

**super(numberOfPlayers, division, version);**

this.clubName = clubName;

}

printDynamitesDetails(version) {

console.log(`Clubname: ${this.clubName} from ${this.version} version.`);

}

}

// Object creation by New ES6 way

**const dynamitesSC = new DynamitesSC("Dynamites S.C.", 30, "Second", "ES6 NEW JS");**

dynamitesSC.printDynamitesDetails( );

dynamitesSC.printClubDetails( );

**Promise**

It’s basically a promise just like our real life, its either **resolve**d/completed or it fails/gets **reject**ed

NOTE(Keywords) Whenever you work hard to keep your promise, and if the **promise** is completed **then** only it is **resolve**d.

Else if **promise** is broken then we provide a **reject** response respectively to the promise keeper.

Promise accepts a callback function as its parameter while time of promise creation.

A callback is used to initialize a promise.

This callback function is passed two arguments: a **resolve callback** used to resolve the promise with a *value* or the *result* of another promise, and a **reject callback** used to reject the promise with a provided *reason* or *error*.

Using Promise we don’t have to write our own defined callback functions.

So, resolve( ) works with then( ) and usually reject( ) works with catch( ) construct

Syntax: Promise( function( resolve, reject ) );

Create a promise, then provide it a resolver when promise succeeds & provide it a reject construct when the promise fails

The resolve & reject are nothing but mediators or method invocations like a trigger event; when pass resolve(data) when fail reject(error/data)

Now, we work on the then( ) part of resolve and the catch( ) part of the reject

// --------- ES6 - PROMISE -----------

const dbPromise = new Promise((resolve, reject) => {

// All code written here is Async

setTimeout(() => {

**resolve({userName: 'Roger31', password: 'a12f41sd4131svcxa14ds14'}**);

**//reject(new Error("Database layer exception"));**

}, 500);

});

// if promise is success then we execute **then( )** construct with data received from **resolve( )** above

dbPromise.**then**(**data** => {

console.log(**data**);

})

// if the then() fails due to error then display the error or whatever

.**catch**(**error** => {

console.error(**error**);

});

// --------- ES6 - PROMISE -----------

const dbPromise = new Promise((resolve, reject) => {

// All code written here is Async

setTimeout(() => {

resolve({userName: 'Roger31', password: 'a12f41sd4131svcxa14ds14'});

//reject(new Error("Database layer exception"));

}, 500);

});

// async promise if success then work on data received

dbPromise.then(data => {

console.log(data);

}) // if the then() fails due to error then display the error or whatever

.catch(error => {

console.error(error);

});

// --------- PROMISE Example-2 ---------

let promise = new Promise((resolve, reject) => {

let a = 11 + 1;

if(a == 2) {

resolve("Success! . . GreatWork!");

} else {

reject("Failure. . .Shame on you!");

}

});

promise.then((msg1) => {

console.log(msg1);

})

.catch(msg2 => {

console.error(msg2);

});

// --------- Promise Example-3 ---------

const switch2Promise = new Promise((resolve, reject) => {

resolve("Switch 2 completed!");

});

const frenchPromise = new Promise((resolve, reject) => {

resolve("French Language learning resumed!");

});

const forces = new Promise((resolve, reject) => {

resolve("Greate! joined forces!");

});

// when we want multiple promise to execute & write a common then & catch

Promise.all([

switch2Promise, frenchPromise, forces

]).then((message) => {

console.log(message);

}).catch((error) => {

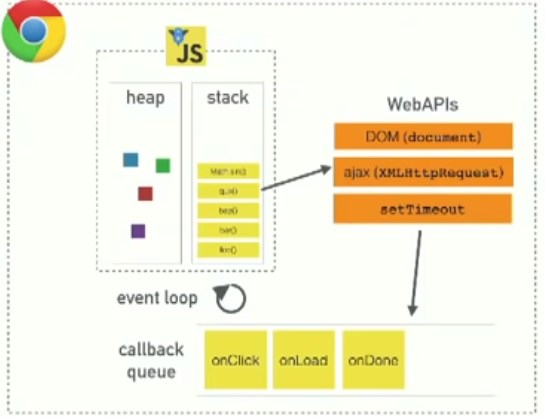
console.error(error);

});

**Async Callbacks & The Call Stack**

Source : Philip Roberts: <https://www.youtube.com/watch?v=8aGhZQkoFbQ&t=1014s>

What is JS ? – A single threaded non-blocking asynchronous concurrent language. It has a call stack, heap, an event loop, a callback queue & some other api’s



**Fig 1.1:** Above is a broader picture/overview of the chrome’s v& engine’s internal workings

As we all know JS is single threaded i.e it has a single call stack which means it can execute one piece of code at a time only

Following is a small scenario which does not has a async block of code & runs synchronously & linearly.

function **multiply**(a, b) { **return a \* b;** }

function **square**(n) { return **multiply(n, n);** }

function **printSquare**(n){

var result = **square(n);**

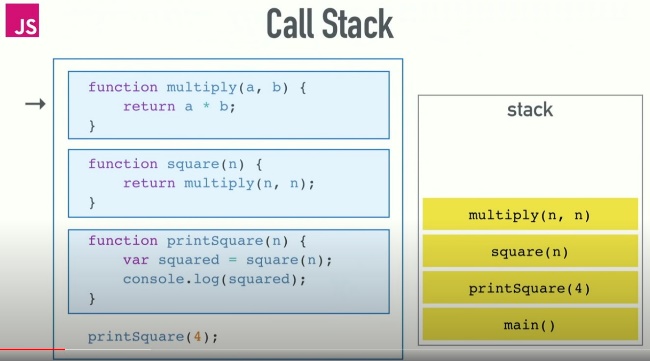
console.log(result);

}

// method call which results in the method chaining & pushing functions calls on call stack

**printSquare**(4);

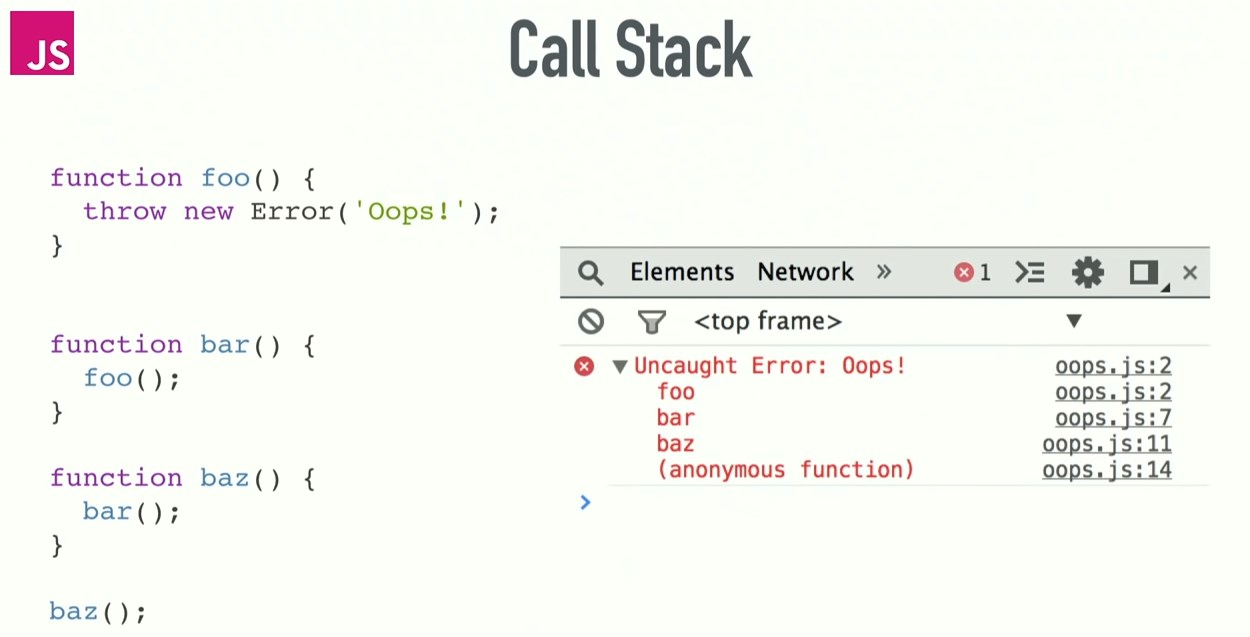
// As soon as printSquare(n) is invoked, the call stack gets loaded with its resp. chained methods



**Fig 1.2**

& as the multiply(a,b0 executes it returns & so do all in a linear style without blocking wherein main() is the last anonymous function to return

Also, when you encounter an exception following is an example wherein we can see the call stack trace which displays the function calls on stack



**Fig.1.3**

What is the problem with synchronous way of coding or single threaded stuff?

The problem is the browser since our JS code runs on the browser, when synch code executes & say a network i/o call might be taking too long to complete

So until that time the entire browser gets blocked since its waiting for the result to move on. This is a bad thing for us.  
Solution? **Asynchronous Callbacks!**

They are non-blocking functions which run in browser. Like we run the some code then the async code then all gets executed, then the asynch code returns back with results & then we render it later

For instance, setInterval(), setTimeout() or forEach() they all accept a callback function. But not necessarily all callback functions are asynchronous in nature.

Like this below snippet, where forEach( ) has an anonymous function which is not an asynchronous callback function.

Wherein, setTimeout() is an asynchronous function, it also accepts a callback function as 1st param along with delay miliseconds & 2nd param

console.log('start');

const items = [1,2,3,4];

**// Synchronous code block**

console.log(`Synchronous Block: Print items via forEach loop: `);

items.**forEach**( item => console.log('fe-synch:'+item) );

**// Asynchronous code block**

console.log(`Asynchronous Block: Print items via forEach loop: `);

**setTimeout( ( )** => {

**items.forEach**( item => console.log('fe-asynch-'+item) );

**}, 100);**

console.log('end');

**Concurrency & Event Loop**

If we refer to the Fig.1.1 you’ll observe the event loop is like a mediator/interface between the call stack & call back queue.

The Call stack holds the function call frames which are pushed & popped accordingly. The asynch calls are handled by the WebApis.

WebApis are not a part of V8 engine but are provided by the browser itself. It looks after the async callback functions & does the timer scheduling for them.

As soon as the async functions are completed, they are returned back to the callback queue & not the call stack.

The only job of Event Loop is to view the call stack if its empty & call back queue If it has any callback functions.

As soon as the call stack completes its entire execution, callBackQueue pushes the callbackfunction into the call stack for immediate completion.

CallStack executes or renders the async function or its results & then the execution is done.

This is how the asyn callback functions work in JS where we see call stack, WebApis, event loop and callBackQueue’s mechanism

**TODO**

Object oriented programming – ES5 Prototypes & ES6 Classes

Async JS – AJAX, Fetch, Promise, Async/Await

Error handling | Local & Session Storage

Regular Expressions

Programming Scope

Arrow Functions

Map | Set | iterators