**5 jQuery**

**5.1 jQuery Introduction**

* **What is jQuery?**

jQuery is a lightweight JavaScript library that simplifies the process of writing JavaScript code. It provides easy-to-use methods for tasks like DOM manipulation, event handling, animations, and AJAX interactions.

* **Key Features of jQuery:**

Cross-browser compatibility: jQuery handles browser inconsistencies automatically.

Chaining: You can execute multiple jQuery methods on the same element(s) in a single statement.

Plugins: jQuery allows developers to extend its functionality using plugins.

**5.1.1 Use of jQuery :**

* DOM Manipulation: Changing HTML content, attributes, or CSS styles dynamically.
* Event Handling: Simplifying tasks like click, hover, or keypress events.
* Animations: Adding effects like hide/show, fade, or slide.
* AJAX: Simplifying asynchronous HTTP requests.

**5.1.2 Difference between jQuery and JavaScript :**

|  |  |  |
| --- | --- | --- |
| Aspect | Javascript | jQuery |
| Definition | A programming language for the web. | A library built on top of JavaScript. |
| Ease of Use | Requires more lines of code for complex tasks. | Simplifies complex tasks with fewer lines. |
| Cross browser | Needs manual handling for inconsistencies. | Automatically handles browser compatibility. |
| Performance | Faster for simple tasks as it doesn’t include the library overhead. | Slightly slower due to the library size. |
| Code example | document.getElementById('id').style.color = 'red'; | $('#id').css('color', 'red'); |

**5.1.3 HTML/CSS method of jQuery :**

HTML methods :

1. html()

Get or set the HTML content inside an element.

EX :

// Get HTML content

let content = $('#myDiv').html();

// Set HTML content

$('#myDiv').html('<b>New Content</b>');

1. text()

Get or set the plain text content (without HTML tags) of an element.

EX :

// Get text

let text = $('#myDiv').text();

// Set text

$('#myDiv').text('Hello, World!');

1. val()

Get or set the value of form elements like input, textarea, or select.

EX :

// Get value

let inputValue = $('#myInput').val();

// Set value

$('#myInput').val('New Value');

1. append()

Insert content at the end of the selected element.

EX :

$('#myDiv').append('<p>Appended content</p>');

1. prepend()

Insert content at the beginning of the selected element.

EX :

$('#myDiv').prepend('<p>Prepended content</p>');

1. attr()

Get or set attributes of an element (e.g., src, href, id).

EX :

// Get attribute

let src = $('#myImage').attr('src');

// Set attribute

$('#myImage').attr('src', 'new-image.jpg');

1. remove()

Remove selected elements, including their data and events.

EX :

$('#myDiv').remove();

1. empty()

Remove all child elements and content of the selected element.

EX :

$('#myDiv').empty();

1. clone()

Create a copy of the selected element(s).

EX :

let copy = $('#myDiv').clone();

$('body').append(copy);

1. wrap()

Wrap HTML element(s) around each selected element.

EX :

$('#myDiv').wrap('<div class="wrapper"></div>');

CSS methods :

1. css()

Get or set one or more CSS properties.

EX :

// Get CSS property

let color = $('#myDiv').css('color');

// Set a CSS property

$('#myDiv').css('color', 'red');

// Set multiple properties

$('#myDiv').css({

'background-color': 'yellow',

'font-size': '18px'

});

1. addClass()

Add one or more classes to the selected elements.

EX :

$('#myDiv').addClass('highlight');

1. removeClass()

Remove one or more classes from the selected elements.

EX :

$('#myDiv').removeClass('highlight');

1. toggleCLass()

Add a class if it doesn’t exist or remove it if it does.

EX :

$('#myDiv').toggleClass('highlight');

1. height() and width()

Get or set the height and width of an element.

EX :

// Get height

let height = $('#myDiv').height();

// Set height

$('#myDiv').height(300);

// Get width

let width = $('#myDiv').width();

// Set width

$('#myDiv').width(400);

1. show() and hide()

Show or hide an element.

EX :

$('#myDiv').hide(); // Hides the element

$('#myDiv').show(); // Displays the element

1. toggle()

Toggles the visibility of an element.

EX :

$('#myDiv').toggle(); // Shows if hidden, hides if visible

1. outerHeight() and outerWidth()

Get the height or width of an element, including padding and border.

EX :

let height = $('#myDiv').outerHeight();

let width = $('#myDiv').outerWidth();

1. hasClass()

Check if an element has a specific class.

EX :

if ($('#myDiv').hasClass('highlight')) {

console.log('Element has the class.'); }

**5.1.4 jQuery selector :**

|  |  |  |
| --- | --- | --- |
| Category | Selectors | Purpose |
| Basic | \*, #id, .class | Universal, ID, and class selection. |
| Hierarchy | parent > child | Select direct child elements. |
| Attribute | [attr], [attr=value] | Select based on attributes and their values. |
| Filter | :first, :last | Select first or last elements. |
| Visibility | :visible, :hidden | Select visible or hidden elements. |
| Form | :checked, :text | Select form elements based on state or type. |
| Positional | :nth-child(n) | Select elements by position in the DOM hierarchy. |
| Content | :contains(text) | Select elements containing specific text. |

**5.2 Event of jQuery**

**5.2.1 Basic events :**

1) Mouse Events :

|  |  |  |
| --- | --- | --- |
| Event | Description | Example |
| click() | Triggered when an element is clicked. | $("#btn").click(function() { alert("Button clicked!"); }); |
| dblclick() | Triggered when an element is double-clicked. | $("#btn").dblclick(function() { alert("Double clicked!"); }); |
| mouseenter() | Triggered when the mouse pointer enters an element. | $("#box").mouseenter(function() { console.log("Mouse entered!"); }); |
| mouseleave() | Triggered when the mouse pointer leaves an element. | $("#box").mouseleave(function() { console.log("Mouse left!"); }); |
| hover() | Combines mouseenter and mouseleave. | $("#box").hover(function() { console.log("Hovered!"); }); |
| mousedown() | Triggered when the mouse button is pressed on an element. | $("#btn").mousedown(function() { console.log("Mouse down!"); }); |
| mouseup() | Triggered when the mouse button is released on an element. | $("#btn").mouseup(function() { console.log("Mouse up!"); }); |
| mousemove() | Triggered when the mouse moves over an element. | $("#box").mousemove(function(e) { console.log("X: " + e.pageX + ", Y: " + e.pageY); }); |

2) Keyboard events

|  |  |  |
| --- | --- | --- |
| Event | Description | Example |
| keydown() | Triggered when a key is pressed. | $(document).keydown(function(e) { console.log("Key pressed: " + e.key); }); |
| keyup() | Triggered when a key is released. | $(document).keyup(function() { console.log("Key released!"); }); |
| keypress() | Triggered when a key is pressed (deprecated). | $(document).keypress(function(e) { console.log("Key pressed: " + e.charCode); }); |

3) Form events

|  |  |  |
| --- | --- | --- |
| Event | Description | Example |
| focus() | Triggered when an element gains focus. | $("#input").focus(function() { $(this).css("background", "lightblue"); }); |
| blur() | Triggered when an element loses focus. | $("#input").blur(function() { $(this).css("background", ""); }); |
| change() | Triggered when the value of an element changes. | $("#dropdown").change(function() { alert($(this).val()); }); |
| submit() | Triggered when a form is submitted. | $("form").submit(function(e) { e.preventDefault(); alert("Form submitted!"); }); |
| select() | Triggered when text is selected inside an input element. | $("#text").select(function() { console.log("Text selected!"); }); |

4) Document/Window Events

|  |  |  |
| --- | --- | --- |
| Event | Description | Example |
| ready() | Triggered when the DOM is fully loaded and ready. | $(document).ready(function() { console.log("DOM is ready!"); }); |
| resize() | Triggered when the window is resized. | $(window).resize(function() { console.log("Window resized!"); }); |
| scroll() | Triggered when the user scrolls within an element or window. | $(window).scroll(function() { console.log("Scrolled!"); }); |

5) other events method/property

|  |  |  |
| --- | --- | --- |
| Event | Description | Example |
| on() | Attaches one or more event handlers to elements. | $("#btn").on("click", function() { alert("Clicked!"); }); |
| off() | Removes event handlers attached with on(). | $("#btn").off("click"); |
| one() | Attaches an event handler triggered only once per element. | $("#btn").one("click", function() { alert("Clicked only once!"); }); |
| trigger() | Programmatically triggers the specified event. | $("#btn").trigger("click"); |
| event.preventDefault() | Prevents the default behavior of the event. | $("#link").click(function(e) { e.preventDefault(); alert("Default prevented!"); }); |
| event.stopPropagation() | Stops the event from bubbling up the DOM hierarchy. | $("#child").click(function(e) { e.stopPropagation(); }); |

**5.2.2 How to fire event programmatically :**

**1. Using trigger()**

The trigger() method fires an event for selected elements. It also executes all event handlers attached to the element for the specified event.

Syntax : $(selector).trigger(eventType)

Ex : $("#btn").trigger("click");

**2. Using Shorthand Methods**

Shorthand methods can also be used to trigger events directly.

Ex : $("#btnToClick").click(); // Fires the click event

**5.3 jQuery Validation**

**validation with jQuery validator**

The jQuery Validation Plugin simplifies form validation by providing pre-built methods for common use cases like required fields, email format, and more.

CDN for Plugin :   
https://cdnjs.cloudflare.com/ajax/libs/jquery-validate/1.21.0/jquery.validate.min.js

**Logic :**

rules: Define validation rules for each field.

messages: Customize error messages for each rule.

submitHandler: A function that runs after successful validation, handling form submission.

EX :

$("#validatorForm").validate({

rules: {

username: {

required: true,

minlength: 3,

},

},

messages: {

username: {

required: "Username is required",

minlength: "Username must be at least 3 characters",

},

},

submitHandler: function (form) {

alert("Form successfully validated!");

form.submit();

},

});

**5.4 Jquery Function map(),grep(),extends,each, merge etc.**

**jQuery map() Function :**

The map() function in jQuery is used to transform or modify elements in an array or object and return a new array. It applies a callback function to each item in the collection, processes it, and includes the result in the new array.

**Key Features**

* Transforms Data: Allows you to modify or transform data in a collection.
* Skips Undefined Results: Automatically excludes undefined values from the final array.
* Returns a New Array: The original array remains unchanged.

**Syntax** : $.map(arrayOrObject, callback);

**Parameters**

1. **arrayOrObject**:  
   The array or object to iterate over.
2. **callbackFunction**:  
   A function applied to each element in the collection.  
   It takes three arguments:

* **value**: The current value being processed.
* **indexOrKey**: The index (for arrays) or key (for objects).

The function should return the transformed value to include it in the new array.

**Examples :**

**Example 1: Basic Array Transformation**

let numbers = [1, 2, 3, 4];

let doubled = $.map(numbers, function(value) {

return value \* 2;

});

console.log(doubled); // Output: [2, 4, 6, 8]

**Example 2: Skipping undefined Values**

let numbers = [1, 2, 3, 4];

let evenNumbers = $.map(numbers, function(value) {

return value % 2 === 0 ? value : undefined;

});

console.log(evenNumbers); // Output: [2, 4]

**Example 3: Transforming Objects**

let names = { 1: "keyur", 2: "parth", 3: "priyanshu" };

let upperCaseNames = $.map(names, function(value) {

return value.toUpperCase();

});

console.log(upperCaseNames); // Output: ["KEYUR", "PARTH", "PRIYANSHU"]

**Example 4: Using Index**

let numbers = [10, 20, 30];

let result = $.map(numbers, function(value, index) {

return `Index ${index}: ${value}`;

});

console.log(result); // Output: ["Index 0: 10", "Index 1: 20", "Index 2: 30"]

**Example 5: Flattening Arrays**

let nested = [[1, 2], [3, 4], [5, 6]];

let flat = $.map(nested, function(value) {

return value;

});

console.log(flat); // Output: [1, 2, 3, 4, 5, 6]

**Difference Between map() in jQuery and JavaScript**

* **jQuery map()**:

Works on both arrays and objects.

Skips undefined results automatically.

* **JavaScript map()**:

Works only on arrays.

Includes undefined values in the result if explicitly returned.

**jQuery grep() function :**

The grep() function in jQuery is used to filter an array based on a specific condition. It goes through each item in an array and keeps only the elements that meet the given condition.

Think of it as a way to "pick and keep" items from an array that match your criteria.

**Syntax :**

$.grep(array, function(element, index) {

// Return true to keep the element

// Return false to exclude it

}, [invert]);

**Parameters**

1. **array**: The array you want to filter.
2. **function(element, index)**: A callback function that determines whether to keep the current item.

* element: The current item in the array.
* index: The position of the current item in the array (optional).
* Return **true** to include the item, or **false** to exclude it.

1. **invert** (optional):

* If false (default), it keeps elements where the callback returns true.
* If true, it keeps elements where the callback returns false (inverts the result).

Example :

**Example 1: Basic Usage**

let numbers = [1, 2, 3, 4, 5, 6];

let evenNumbers = $.grep(numbers, function(value) {

return value % 2 === 0; // Keep even numbers

});

console.log(evenNumbers); // Output: [2, 4, 6]

**Example 2: Using invert**

let numbers = [1, 2, 3, 4, 5, 6];

let oddNumbers = $.grep(numbers, function(value) {

return value % 2 === 0; // This checks for even numbers

}, true); // Invert the result to exclude even numbers

console.log(oddNumbers); // Output: [1, 3, 5]

**jQuery extend() function :**

The $.extend() function in jQuery is used to merge objects or copy properties from one object to another. It allows you to combine two or more objects into one.

**Syntax :** $.extend(target, object1, object2, ...);

**Parameters:**

1. **target**: The object to which properties will be added or updated.
2. **object1, object2, ...**: Objects whose properties will be copied to the target.

**Example 1: Combine Two Objects**

let obj1 = { a: 1, b: 2 };

let obj2 = { b: 3, c: 4 };

let result = $.extend({}, obj1, obj2);

console.log(result);

// Output: { a: 1, b: 3, c: 4 }

**Example 2: Merge Objects into an Existing Object**

let settings = { theme: "dark", fontSize: "12px" };

let userPreferences = { fontSize: "16px", color: "blue" };

$.extend(settings, userPreferences);

console.log(settings);

// Output: { theme: "dark", fontSize: "16px", color: "blue" }

**jQuery each() function :**

The each() function in jQuery is used to loop through elements or items in a collection (like an array, an object, or a set of HTML elements). It allows you to perform a specific action on each item in the collection.

**Syntax :**

1 $.each(collection, function(index, value) { //code });

2 $(selector).each(function(index, element) { // code });

**Parameters:**

1. **collection**: An array or object to loop through.
2. **function(index, value)**: A callback function that runs for each item in the collection:

* **index**: The position of the current item in the collection.
* **value**: The value of the current item (for arrays/objects).
* **element**: The current HTML element (for jQuery selectors).

**Example 1: Loop Through an Object**

let person = { name: "Keyur", age: 20, city: "Morbi" };

$.each(person, function(key, value) {

console.log(key + ": " + value);

});

// Output:

// name: Keyur

// age: 20

// city: Morbi

**Example 2: Loop Through HTML Elements**

$("li").each(function(index, element) {

console.log("Item " + index + ": " + $(element).text());

});

**Key Difference Between $.each() and $(selector).each():**

* **$.each()**: Works for arrays and objects.
* **$(selector).each()**: Works for HTML elements selected using jQuery.

**jQuery merge() function :**

The merge() function in jQuery is used to combine two arrays into a single array. It does not remove duplicates; instead, it adds the second array’s elements to the end of the first array.

**Syntax :** $.merge(array1, array2);

**Parameters:**

1. **array1**: The first array. This array will have elements from the second array added to it.
2. **array2**: The second array. Its elements will be appended to the first array.

**Returns:**

* A new array that combines both input arrays.

**Example : Merging Two Arrays**

let arr1 = [1, 2, 3];

let arr2 = [4, 5, 6];

let mergedArray = $.merge(arr1, arr2);

console.log(mergedArray);

// Output: [1, 2, 3, 4, 5, 6]

**Difference Between concat() and merge() in JavaScript and jQuery:**

* **concat() (JavaScript)**: Creates a new array by joining two arrays without modifying them.
* **merge() (jQuery)**: Works similarly to concat() but is a jQuery-specific method.

**5.5 Regular Expressions in jQuery**

Regular expressions (regex) are patterns used to match character combinations in strings. While jQuery itself does not have specific methods for working with regex, it leverages JavaScript's RegExp object and string methods like match(), test(), replace(), and others to perform regex operations.

|  |  |  |
| --- | --- | --- |
| Pattern | Description | Example |
| ^\d+$ | Matches numbers only | "123" (valid) |
| ^[a-zA-Z]+$ | Matches letters only | "hello" (valid) |
| ^\w+$ | Matches alphanumeric characters | "hello123" (valid) |
| ^[^\s]+$ | Matches non-whitespace strings | "NoSpaces" (valid) |
| \bword\b | Matches a specific word | "word" in "word bag" |
| \d{3}-\d{3}-\d{4} | Matches a phone number format (123-456-7890) | "123-456-7890" (valid) |

Ex :

let email = $("#email").val();

let regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/; // Regex for email validation

if (regex.test(email)) {

alert("Valid email address!");

} else {

alert("Invalid email address!");

}

**5.6 Callback Functions in jQuery**

A **callback function** in jQuery is a function passed as an argument to another function, executed only after the parent function completes its operation.

**Why Use Callback Functions?**

* To avoid **asynchronous issues** where code execution continues before a previous function finishes.
* To manage **sequential execution** of code.
* To ensure **dependent code** runs only after the necessary conditions are met.

Ex :

$("#box").hide(1000, function () {

alert("Box is now hidden!");

});

* The hide() method hides the element over 1000 milliseconds. After hiding, the callback function triggers and displays the alert.

**Callback Functions vs Regular Functions**

* **Callback Functions**: Passed as arguments to other functions and executed at a specific time or after a specific condition.
* **Regular Functions**: Invoked independently when called.

**5.7 Deferred & Promise object**

**Deferred** and **Promise** are powerful tools in jQuery for managing asynchronous operations like AJAX calls, animations, or other time-dependent tasks. They help streamline how you write and manage callback functions by providing a structured and predictable way to handle success, failure, and completion of these operations.

**1. Deferred Object**

The **Deferred** object, introduced by jQuery, allows you to create and manage your own asynchronous operations. It provides methods to control the state of the operation (resolve, reject, or notify) and attach multiple callbacks for success (done), failure (fail), and progress (progress).

**Methods of Deferred**

Method Description

$.Deferred() Creates a new Deferred object.

.resolve() Marks the Deferred object as successful and triggers the done callbacks.

.reject() Marks the Deferred object as failed and triggers the fail callbacks.

.done() Adds a callback for when the operation is successful.

.fail() Adds a callback for when the operation fails.

.always() Adds a callback that executes after the operation is completed (success/fail).

EX :

var deferred = $.Deferred();

// Simulate an asynchronous operation

setTimeout(function() {

if (Math.random() > 0.5) {

deferred.resolve("Success!"); // Mark as successful

} else {

deferred.reject("Failed!"); // Mark as failed

}

}, 1000);

// Attach callbacks

deferred

.done(function(message) { console.log("Resolved: " + message); })

.fail(function(message) { console.log("Rejected: " + message); });

**2. Promise Object**

A **Promise** is a streamlined, read-only subset of the Deferred object. While a Deferred object allows you to control the state of the operation, a Promise object is used only to consume the outcome of an operation. You cannot resolve or reject a Promise directly, making it useful for ensuring predictable behavior.

**Key Promise Methods**

Method Description

.done() Attaches a callback for when the Promise is resolved (success).

.fail() Attaches a callback for when the Promise is rejected (failure).

.always() Attaches a callback for when the Promise is completed (resolved or rejected).

.then() Attaches success and failure handlers. Can chain additional asynchronous calls.

.catch() Attaches a failure handler. Equivalent to .fail() but used for chaining.

**EX :**

var promise = $.ajax({

url: "example.com/data",

method: "GET"

});

// Attach callbacks

promise

.done(function(data) { console.log("Data received: ", data); })

.fail(function() { console.log("Error occurred."); });

**Key Differences Between Deferred and Promise**

|  |  |  |
| --- | --- | --- |
| Aspect | Deferred | Promise |
| State Control | Allows you to manually resolve, reject | State is controlled internally (read-only). |
| Use Case | Useful for creating custom asynchronous operations. | Used for consuming existing async operations. |
| Example use Case | Writing custom logic for async tasks. | Handling AJAX, animations, or timers. |

**5.8 Ajax**

**5.8.1 What is Ajax?**

AJAX (Asynchronous JavaScript and XML) is a technique used to update parts of a web page without reloading the entire page. It allows web applications to send and retrieve data asynchronously from the server in the background.

**How AJAX Works:**

1. The client-side script sends an asynchronous request to the server.
2. The server processes the request and sends a response (e.g., JSON, XML, or HTML).
3. The client-side script receives the response and updates the webpage dynamically.

**5.8.2 Uses of AJAX**

1. Dynamic Content Loading: AJAX is used to load new data into specific sections of a webpage without reloading the entire page (e.g., news feed updates).
2. Form Validation and Submission: Validates and submits forms to the server without page reloads.
3. Data Filtering: Filters and updates content dynamically, such as search results or product filters.
4. Real-Time Updates: Enables real-time functionalities like chat applications and notifications.
5. Improved Performance: Reduces server load by sending only required data rather than the whole page.

**5.8.3 How to Send Data with AJAX Requests**

AJAX requests can be made using the $.ajax() method in jQuery, along with shorthand methods like $.get() and $.post()

**Using $.ajax()**

The $.ajax() method provides flexibility in configuring AJAX requests.

**EX :**

$.ajax({

url: "server\_endpoint", // Server URL

type: "POST", // HTTP Method

data: { name: "keyur", age: 20 }, // Data to send to the server

dataType: "json", // Expected response format (json, xml, html, etc.)

success: function(response) {

console.log("Data received:", response); // Handle the response

},

error: function(xhr, status, error) {

console.error("Error occurred:", error); // Handle errors

}

});

**Shorthand Methods**

**1. $.get():** Sends a GET request to retrieve data from the server.

$.get("server\_endpoint", { id: 1 }, function(data) {

console.log("Response:", data);

});

**2. $.post():** Sends a POST request to submit data to the server.

$.post("server\_endpoint", { name: "Alice" }, function(response) {

console.log("Response:", response);

});

**5.8.4 Difference Between GET, POST, PUT, DELETE Methods**

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **Purpose** | **Usage** |
| GET | Retrieves data. | Used to fetch data from the server without modifying it |
| POST | Sends data to create a resource. | Used to create a new resource |
| PUT | Updates an existing resource. | Used to modify existing data on the server |
| DELETE | Deletes a resource. | Used to remove a resource from the server |

**5.8.5 JSON Data**

JSON (JavaScript Object Notation) is a text-based format for storing and exchanging structured data. It is lightweight, human-readable, and widely used in web development for client-server communication.

**EX :**

{

"id": 107,

"name": "KEYUR",

"skills": ["JavaScript", "AJAX", "jQuery"],

"isActive": true

}

**Advantages of JSON**

1. Lightweight and easy to parse.
2. Language-independent and widely supported.
3. Readable format for both humans and machines.

**5.8.6 Serialization & Deserialization**

**Serialization**

Serialization is the process of converting a JavaScript object or array into a string format (e.g., JSON or XML) for transmission or storage.

**EX :**

let obj = { name: "Keyur", age: 20 };

let serializedData = JSON.stringify(obj);

console.log(serializedData); // Output: {"name":"Keyur","age":20}

**Deserialization**

Deserialization is the process of converting a serialized string back into a usable JavaScript object or array.

**EX :**

let jsonString = '{"name":"Keyur","age":20}';

let deserializedData = JSON.parse(jsonString);

console.log(deserializedData.name); // Output: Keyur