**jQuery**

1. **jQuery Introduction**

jQuery is a fast, small, and powerful JavaScript library designed to make it easier to work with HTML documents, handle events, create animations, and perform AJAX interactions.

1. **Use of jQuery:**

* DOM Manipulation: Easily select and change HTML elements and their content.
* Event Handling: Handle events like click, hover, submit, etc. with simple syntax.
* Animations & Effects: Add effects like fade, slide, show/hide, and custom animations.
* AJAX Support : Simplifies AJAX calls to load data without refreshing the page.
* CSS Manipulation: Add/remove/change CSS styles and classes dynamically.

1. **Difference between jQuery and JavaScript**

| **Feature** | **JavaScript** | **jQuery** |
| --- | --- | --- |
| Type | Programming language | JavaScript library |
| Purpose | Write logic, manipulate DOM, handle events, etc. | Simplify JavaScript tasks (DOM, events, AJAX, effects) |
| Cross-browser issues | May need custom code for browser differences | Handles cross-browser compatibility internally |
| Speed | Native, usually faster | Slightly slower (adds overhead of a library) |
| Dependencies | No dependencies | Requires including the jQuery library |

1. **HTML/CSS method of jQuery**

**HTML methods:**

1. text(): Sets or returns the text content of selected elements
2. html(): Sets or returns the content of selected elements
3. val(): Sets or returns the value attribute of the selected elements (for form elements)
4. attr(): Sets or returns attributes/values of selected elements
5. append(): Inserts content at the end of selected elements
6. prepend(): Inserts content at the beginning of selected elements
7. after(): Inserts content after selected elements
8. before(): Inserts content before selected elements
9. remove(): Removes the selected elements (and its child elements)
10. empty(): Removes all child nodes and content from selected elements

**CSS methods:**

1. addClass(): Adds one or more class names to selected elements
2. removeClass(): Removes one or more classes from selected elements
3. toggleClass(): Toggles between adding/removing one or more classes from selected elements
4. css(): Sets or returns one or more style properties for selected elements
5. **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. |
| Positional | :nth-child(n) | Select elements by position in the DOM hierarchy. |

1. **Events of jQuery**
2. **Basic events**
3. .click(): Fired when an element is clicked
4. .dblclick(): Fired when an element is double-clicked
5. .mouseenter(): Mouse enters the element
6. .mouseleave(): Mouse leaves the element
7. .hover(): Shortcut for mouseenter + mouseleave
8. .keydown(): A key is pressed down
9. .keyup(): A key is released
10. .keypress(): A key is pressed
11. .submit(): A form is submitted
12. .change(): Value of input, select, or text area changes
13. .focus(): Element gains focus
14. .blur(): Element loses focus
15. .load(): Page is fully loaded
16. .resize(): Window is resized
17. .scroll(): User scrolls the page
18. **How to fire event programmatically**
19. Using .trigger(): The trigger() method fires an event for selected elements.

Ex: $("#myButton").trigger("click"); // triggers a click event

1. Using Shorthand Methods: Shorthand methods can also be used to trigger events directly.

Ex: $("#myButton").click(); // triggers click event

1. **jQuery Validation**

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

rules: Define validation rules for each field.

messages: Customize error messages for each rule.

Ex:

$('#form').validate({

rules:{

name:{

required: true,

minlength: "10"

},

age:{

required: true,

}

},

messages:{

name:{

required: "Name is requird",

minlength: "Name must be 10 character long"

},

age: "Age is required"

},

errorElement: "span"

});

1. **jQuery Function**
2. **map():** Transforms elements in an array or jQuery object and returns a new array.

Syntax : $.map(arrayOrObject, callback);

1. **grep():** Filters an array based on condition. Returns only the element that pass the test.

Syntax: $.grep(array, function(item, index) {

return condition; // true to keep, false to exclude

});

1. **extend():** Copies properties from one or more objects into a target object. Useful for merging configurations of defaults.

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

1. **each():** The each() function in jQuery is used to loop through elements or items in a collection (like an array or an object).

Syntax: $.each(arrayOrObject, function(index, value) {

// logic here

});

1. **merge():** 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);

1. **Regular expression 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** |
| ^\d+$ | Matches numbers only |
| ^[a-zA-Z]+$ | Matches letters only |
| ^\w+$ | Matches alphanumeric characters |
| ^[^\s]+$ | Matches non-whitespace strings |
| \d{3}-\d{3}-\d{4} | Matches a phone number format (123-456-7890) |

1. **Callback functions**

A callback function is a function passed as an argument to another function, which is then called (executed) later, usually after something has finished.

In jQuery, callbacks are heavily used - especially when performing animations, AJAX requests, or any code that runs asynchronously or after an event.

Ex: $("#box").fadeOut(1000, function() {

// This is the callback

alert("Fade out complete!");

});

1. **Deferred& Promise object**
2. **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 in deferred:**

1. $.Deferred(): Creates a new Deferred object.
2. .resolve(): Marks the Deferred object as successful and triggers the done callbacks.
3. .reject(): Marks the Deferred object as failed and triggers the fail callbacks.
4. .done() : Adds a callback for when the operation is successful.
5. .fail(): Adds a callback for when the operation fails.
6. .always(): Adds a callback that executes after the operation is completed (success/fail).

Ex:

var deferred = $.Deferred();

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); });

1. **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.

**Methods in promise:**

1. .done(): Attaches a callback for when the Promise is resolved (success).
2. .fail(): Attaches a callback for when the Promise is rejected (failure).
3. .always(): Attaches a callback for when the Promise is completed (resolved or rejected).
4. .then(): Attaches success and failure handlers. Can chain additional asynchronous calls.
5. .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."); });

1. **Ajax**
2. **What is Ajax?**

AJAX stands for Asynchronous JavaScript and XML.

It is not a programming language, but a technique used in web development to send and receive data from a server without reloading the web page.

1. **Use of Ajax**
2. Load data without refreshing
3. Form submission
4. Dynamic content update
5. Pagination & filtering
6. **How to send data with Ajax Request?**

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

Ex: $("#ajaxBtn").click(function () {

$.post("https://687f84e0efe65e52008a1051.mockapi.io/dummy/users",

{

name: "Rohit Sharma",

},function(){

$.ajax({ //Instead of ajax you can use get

url: "https://687f84e0efe65e52008a1051.mockapi.io/dummy/users", success: function (result) {

for (x in result) {

$('#div1').append(`<div>${result[x].name}</div>`);

}

// $('#div1').text($.param(result));

}

});

});

});

1. **Difference between get, post, put, delete method**

| **Method** | **Purpose** | **Data Sent In** | **Common Use** |
| --- | --- | --- | --- |
| GET | Retrieve data | URL params | Fetching info |
| POST | Send new data | Request body | Creating records |
| PUT | Update existing data (full) | Request body | Updating records |
| DELETE | Delete data | URL or body | Deleting records |

1. **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 :

{

"name": "Kiran",

"age": 19

}

1. **Serialization & De-serialization**

**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: "Kiran", age: 19 };

let serializedData = JSON.stringify(obj);

console.log(serializedData); // Output: {"name":"Kiran","age":19}

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

Ex:

let jsonString = '{"name":"Kiran","age":19}';

let deserializedData = JSON.parse(jsonString);

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