JQuery

jQuery Introduction

jQuery is a lightweight, "write less, do more", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

Use of jQuery

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX

Difference between jQuery and JavaScript

JavaScript	jQuery
JavaScript uses long lines of code as an individual has to write the code own-self.	With JQuery, one has to write fewer lines of code than JavaScript. We just need to import the library and use the only specific functions or methods of the library in our code.

In JavaScript, we have to write extra code or move around to have cross-browser compatibility.	JQuery has an inbuilt feature of cross-browser compatibility. We don’t need to worry about writing extra lines of code or moving around in order to make our code compatible with any browser.
JavaScript can be a burden over a developer as it may take a number of lines of lengthy code to attain functionality.	Unlike JavaScript, JQuery is more user-friendly only a few lines of code have to write in order to have its functionality.
Pure JavaScript can be faster for DOM selection/manipulation than jQuery as JavaScript is directly processed by the browser and it curtails the overhead which JQuery actually has.	JQuery is also fast with modern browsers and modern computers. JQuery has to be converted into JavaScript to make it run in a browser.
We can make animations in JavaScript with many lines of code. Animations are mainly done by manipulating the style of an Html page.	In JQuery, we can add animation effects easily with fewer lines of code.

HTML/Css method of jQuery

The following table lists all the methods used to manipulate the HTML and CSS.

Method	**Description**
addClass()	Adds one or more class names to selected elements
after()	Inserts content after selected elements
append()	Inserts content at the end of selected elements
appendTo()	Inserts HTML elements at the end of selected elements
attr()	Sets or returns attributes/values of selected elements
before()	Inserts content before selected elements
clone()	Makes a copy of selected elements
css()	Sets or returns one or more style properties for selected elements
detach()	Removes selected elements (keeps data and events)

empty()	Removes all child nodes and content from selected elements
hasClass()	Checks if any of the selected elements have a specified class name
height()	Sets or returns the height of selected elements
html()	Sets or returns the content of selected elements
innerHeight()	Returns the height of an element (includes padding, but not border)
innerWidth()	Returns the width of an element (includes padding, but not border)
insertAfter()	Inserts HTML elements after selected elements
insertBefore()	Inserts HTML elements before selected elements
offset()	Sets or returns the offset coordinates for selected elements (relative to the document)
offsetParent()	Returns the first positioned parent element
outerHeight()	Returns the height of an element (includes padding and border)

outerWidth()	Returns the width of an element (includes padding and border)
position()	Returns the position (relative to the parent element) of an element
prepend()	Inserts content at the beginning of selected elements
prependTo()	Inserts HTML elements at the beginning of selected elements
prop()	Sets or returns properties/values of selected elements
remove()	Removes the selected elements (including data and events)
removeAttr()	Removes one or more attributes from selected elements
removeClass()	Removes one or more classes from selected elements
removeProp()	Removes a property set by the prop() method
replaceAll()	Replaces selected elements with new HTML elements
replaceWith()	Replaces selected elements with new content
scrollLeft()	Sets or returns the horizontal scrollbar position of selected elements

scrollTop()	Sets or returns the vertical scrollbar position of selected elements
text()	Sets or returns the text content of selected elements
toggleClass()	Toggles between adding/removing one or more classes from selected elements
unwrap()	Removes the parent element of the selected elements
val()	Sets or returns the value attribute of the selected elements (for form elements)
width()	Sets or returns the width of selected elements
wrap()	Wraps HTML element(s) around each selected element
wrapAll()	Wraps HTML element(s) around all selected elements
wrapInner()	Wraps HTML element(s) around the content of each selected element

jQuery selector

Selector	**Example**	**Selects**
*	$("*")	All elements
#id	$("#lastname")	The element with id="lastname"
.class	$(".intro")	All elements with class="intro"
.class,.class	$(".intro,.demo")	All elements with the class "intro" or "demo"
element	$("p")	All <p> elements
el1,*el2*,*el3*	$("h1,div,p")	All <h1>, and <p> elements
:first	$("p:first")	The first <p> element
:last	$("p:last")	The last <p> element
:even	$("tr:even")	All even <tr> elements
:odd	$("tr:odd")	All odd <tr> elements

:first-child	$("p:first-child")	All <p> elements that are the first child of their parent
:first-of-type	$("p:first-of-type")	All <p> elements that are the first <p> element of their parent
:last-child	$("p:last-child")	All <p> elements that are the last child of their parent
:last-of-type	$("p:last-of-type")	All <p> elements that are the last <p> element of their parent
:nth-child(*n*)	$("p:nth-child(2)")	All <p> elements that are the 2nd child of their parent
:nth-last-child(*n*)	$("p:nth-last-child(2)")	All <p> elements that are the 2nd child of their parent, counting from the last child
:nth-of-type(*n*)	$("p:nth-of-type(2)")	All <p> elements that are the 2nd <p> element of their parent
:nth-last-of-type(*n*)	$("p:nth-last-of-type(2)")	All <p> elements that are the 2nd <p> element of their parent, counting from the last child
:only-child	$("p:only-child")	All <p> elements that are the only child of their parent

:only-of-type	$("p:only-of-type")	All <p> elements that are the only child, of its type, of their parent
parent > child	$("div > p")	All <p> elements that are a direct child of a element
parent descendant	$("div p")	All <p> elements that are descendants of a element
element + next	$("div + p")	The <p> element that are next to each elements
element ~ siblings	$("div ~ p")	All <p> elements that appear after the element
:eq(*index*)	$("ul li:eq(3)")	The fourth element in a list (index starts at 0)
:gt(*no*)	$("ul li:gt(3)")	List elements with an index greater than 3
:lt(*no*)	$("ul li:lt(3)")	List elements with an index less than 3
:not(*selector*)	$("input:not(:empty)")	All input elements that are not empty
:header	$(":header")	All header elements <h1>, <h2> ...
:animated	$(":animated")	All animated elements

:focus	$(":focus")	The element that currently has focus
:contains(*text*)	$(":contains('Hello')")	All elements which contains the text "Hello"
:has(*selector*)	$("div:has(p)")	All elements that have a <p> element
:empty	$(":empty")	All elements that are empty
:parent	$(":parent")	All elements that are a parent of another element
:hidden	$("p:hidden")	All hidden <p> elements
:visible	$("table:visible")	All visible tables
:root	$(":root")	The document's root element
:lang(*language*)	$("p:lang(de)")	All <p> elements with a lang attribute value starting with "de"
[*attribute*]	$("[href]")	All elements with a href attribute
[*attribute*=*value*]	$("[href='default.htm']")	All elements with a href attribute value equal to "default.htm"

[*attribute*!=*value*]	$("[href!='default.htm']")	All elements with a href attribute value not equal to "default.htm"
[*attribute*$=*value*]	$("[href$='.jpg']")	All elements with a href attribute value ending with ".jpg"
[*attribute*|=*value*]	$("[title|='Tomorrow']")	All elements with a title attribute value equal to 'Tomorrow', or starting with 'Tomorrow' followed by a hyphen
[*attribute*^=*value*]	$("[title^='Tom']")	All elements with a title attribute value starting with "Tom"
[*attribute*~=*value*]	$("[title~='hello']")	All elements with a title attribute value containing the specific word "hello"
[*attribute**=*value*]	$("[title*='hello']")	All elements with a title attribute value containing the word "hello"
:input	$(":input")	All input elements
:text	$(":text")	All input elements with type="text"
:password	$(":password")	All input elements with type="password"

:radio	$(":radio")	All input elements with type="radio"
:checkbox	$(":checkbox")	All input elements with type="checkbox"
:submit	$(":submit")	All input elements with type="submit"
:reset	$(":reset")	All input elements with type="reset"
:button	$(":button")	All input elements with type="button"
:image	$(":image")	All input elements with type="image"
:file	$(":file")	All input elements with type="file"
:enabled	$(":enabled")	All enabled input elements
:disabled	$(":disabled")	All disabled input elements
:selected	$(":selected")	All selected input elements
:checked	$(":checked")	All checked input elements

Events of jQuery

Method / Property	Description
blur()	Attaches/Triggers the blur event
change()	Attaches/Triggers the change event
click()	Attaches/Triggers the click event
dblclick()	Attaches/Triggers the double click event
event.currentTarget	The current DOM element within the event bubbling phase
event.data	Contains the optional data passed to an event method when the current executing handler is bound
event.delegateTarget	Returns the element where the currently-called jQuery event handler was attached
event.isDefaultPrevented()	Returns whether event.preventDefault() was called for the event object
event.isImmediatePropagationStopped()	Returns whether event.stopImmediatePropagation() was called for the event object

event.isPropagationStopped()	Returns whether event.stopPropagation() was called for the event object
event.namespace	Returns the namespace specified when the event was triggered
event.pageX	Returns the mouse position relative to the left edge of the document
event.pageY	Returns the mouse position relative to the top edge of the document
event.preventDefault()	Prevents the default action of the event
event.relatedTarget	Returns which element being entered or exited on mouse movement
event.result	Contains the last/previous value returned by an event handler triggered by the specified event
event.stopImmediatePropagation()	Prevents other event handlers from being called
event.stopPropagation()	Prevents the event from bubbling up the DOM tree, preventing any parent handlers from being notified of the event
event.target	Returns which DOM element triggered the event

event.timeStamp	Returns the number of milliseconds since January 1, 1970, when the event is triggered
event.type	Returns which event type was triggered
event.which	Returns which keyboard key or mouse button was pressed for the event
focus()	Attaches/Triggers the focus event
focusin()	Attaches an event handler to the focusin event
focusout()	Attaches an event handler to the focusout event
hover()	Attaches two event handlers to the hover event
keydown()	Attaches/Triggers the keydown event
keypress()	Attaches/Triggers the keypress event
keyup()	Attaches/Triggers the keyup event
mousedown()	Attaches/Triggers the mousedown event
mouseenter()	Attaches/Triggers the mouseenter event

mouseleave()	Attaches/Triggers the mouseleave event
mousemove()	Attaches/Triggers the mousemove event
mouseout()	Attaches/Triggers the mouseout event
mouseover()	Attaches/Triggers the mouseover event
mouseup()	Attaches/Triggers the mouseup event
off()	Removes event handlers attached with the on() method
on()	Attaches event handlers to elements
one()	Adds one or more event handlers to selected elements. This handler can only be triggered once per element
$.proxy()	Takes an existing function and returns a new one with a particular context
ready()	Specifies a function to execute when the DOM is fully loaded
resize()	Attaches/Triggers the resize event
scroll()	Attaches/Triggers the scroll event

select()	Attaches/Triggers the select event
submit()	Attaches/Triggers the submit event
trigger()	Triggers all events bound to the selected elements
triggerHandler()	Triggers all functions bound to a specified event for the selected elements

How to fire event programmatically

To fire an event programmatically in jQuery, you can use the `trigger()` method. Here's how you can do it:

// Suppose you have an HTML element with an ID "myButton" and you want to programmatically trigger a click event on it.

// HTML:
// <button id="myButton">Click me</button>

// JavaScript:
```
$(document).ready(function () {
  // Select the element by its ID
  var $myButton = $("#myButton");

  // Programmatically trigger a click event on the element
  $myButton.trigger("click");
});

// Event handler for the click event
$("#myButton").on("click", function () {
```

```
  console.log("Click event triggered!");
});
```

In the code above:

1. We select the HTML element with the ID "myButton" using jQuery and store it in the `$myButton` variable.

2. We then use the `trigger()` method to programmatically trigger a click event on the selected element.

3. When the click event is triggered, the event handler attached to the element responds by logging a message to the console.

This allows you to simulate a user clicking the button, triggering the associated event handler as if it were a natural user interaction.

jQuery Validation

Basic validation

Form Validation is the process of checking whether the data entered into a form by a user meets certain criteria or rules before it's submitted to a server. The main goals of form validation are to ensure data accuracy, completeness, and adherence to specific rules.

- Required Fields:
- Data Format Validation:
- Length or Size Constraints:
- Numeric and Range Validation:
- Consistency Checks:
- Custom Validation Rules:
- Preventing Form Submission:

validation with jQuery validator

Form Validation ensures user-submitted data is accurate and adheres to rules before it's sent to the server.

jQuery Validation Plugin simplifies this process:

1. Include jQuery and Plugin: Include jQuery and the jQuery Validation plugin in your HTML.

2. Create HTML Form: Build your form with validation attributes like `required`, `minlength`, etc.

3. Initialize Validation: In JavaScript, use `.validate()` to define validation rules and error messages for form fields.

4. Validation Rules: Use built-in rules (`required`, `email`, etc.) to specify how fields should be validated.

5. Display Errors: jQuery Validation shows error messages next to fields when validation fails.

6. Prevent Submission: It prevents form submission on validation errors.

Regular expression in jQuery

A regular expression is a sequence of characters that forms a search pattern

The search pattern can be used for text search and text replace operations.

Syntax : /pattern/modifiers;

Modifiers : Modifiers are used to perform case-insensitive and global searches:

g : Perform a global match (find all matches rather than stopping after the first match)
i : Perform case-insensitive matching
m : Perform multiline matching

Brackets: Brackets are used to find a range of characters:

[abc] : Find any character between the brackets
[^abc] : Find any character NOT between the brackets
[0-9] : Find any character between the brackets (any digit)
[^0-9] : Find any character NOT between the brackets (any non-digit)
(x|y) : Find any of the alternatives specified

1. Test Method (`.test()`):
- Checks if a string contains the pattern.
- Returns `true` if the pattern is found, otherwise `false`.

```
var regex = /pattern/;
var str = "Some text with the pattern.";
console.log(regex.test(str)); // true
```

2. Match Method (`.match()`):
- Searches for a pattern in a string.
- Returns an array of matched substrings or `null` if no match is found.

```
var regex = /pattern/;
var str = "Some text with the pattern.";
console.log(str.match(regex)); // ["pattern"]
```

3. Search Method (`.search()`):
- Searches for a pattern in a string.
- Returns the index of the first match or -1 if no match is found.

```
var regex = /pattern/;
var str = "Some text with the pattern.";
console.log(str.search(regex)); // 17 (the index of the first character of the match)
```

4. Replace Method (`.replace()`):
- Replaces matched patterns with a specified replacement.

```
var regex = /pattern/g; // "g" flag to replace all occurrences
var str = "Some text with the pattern. Another pattern here.";
var replaced = str.replace(regex, "replacement");
console.log(replaced);
// "Some text with the replacement. Another replacement here."
```

Jquery Functions

1. `$.map()`:

- `$.map()` is used to apply a function to each item in an array or object and create a new array with the results.
- It's commonly used for transforming data in an array-like structure.

- Example:

```
var numbers = [1, 2, 3];
var squaredNumbers = $.map(numbers, function (n) {
  return n * n;
});
// squaredNumbers will be [1, 4, 9]
```

2. `$.grep()`:

- `$.grep()` is used to filter elements in an array or object based on a given condition.
- It returns a new array containing only the elements that meet the specified criteria.

- Example:

```
var numbers = [1, 2, 3, 4, 5];
var evens = $.grep(numbers, function (n) {
  return n % 2 === 0;
});
// evens will be [2, 4]
```

3. `$.extend()`:

- `$.extend()` is used to merge the contents of two or more objects into the target object.
- It's often used to combine settings or options from multiple sources into a single object.

- Example:

```
var defaultOptions = { color: 'blue', size: 'medium' };
var userOptions = { size: 'large', weight: 'heavy' };
var mergedOptions = $.extend({}, defaultOptions, userOptions);
// mergedOptions will be { color: 'blue', size: 'large', weight: 'heavy' }
```

4. `$.each()`:

- `$.each()` is used to iterate over elements in an array or object.
- It's similar to a `for` loop but provides a more concise way to iterate.

- Example:

```
var fruits = ['apple', 'banana', 'cherry'];
$.each(fruits, function (index, fruit) {
```

```
    console.log('Index ' + index + ': ' + fruit);
  });
  // Output:
  // Index 0: apple
  // Index 1: banana
  // Index 2: cherry
```

5. `$.merge()`:

- `$.merge()` is used to merge the contents of two arrays into the first array.
- It's often used to combine arrays.

- Example:

```
  var arr1 = [1, 2, 3];
  var arr2 = [4, 5, 6];
  $.merge(arr1, arr2);
  // arr1 will be [1, 2, 3, 4, 5, 6]
```

Callback functions

Callback functions are functions passed as arguments to other functions and executed after a particular operation completes. They are used for handling asynchronous tasks, responding to events, controlling flow, error handling, and creating higher-order functions. Callbacks are essential in JavaScript and jQuery for managing asynchronous and event-driven code.

```
function greet(name, callback) {
  console.log("Hello, " + name + "!");
  callback(); // Execute the callback function
}

function sayGoodbye() {
  console.log("Goodbye!");
}

greet("Alice", sayGoodbye);
```

Deferred& Promise object

The Deferred and Promise objects are essential concepts in JavaScript for managing asynchronous operations. They are not specific to jQuery but are often used in jQuery's AJAX and animation methods. These objects provide a way to handle asynchronous tasks, coordinate multiple asynchronous operations, and control their outcomes.

Promise:

- A Promise represents a value (or the eventual result of an operation) that may not be available yet but will be resolved at some point in the future. It can be in one of three states: pending, fulfilled, or rejected.

- Promises have two main methods: `then()` and `catch()`. The `then()` method is used to specify what should happen when a Promise is resolved successfully, and the `catch()` method is used to specify what should happen when a Promise is rejected.

- Promises can be chained together, allowing you to perform a series of asynchronous operations sequentially.

Example:

```
const promise = new Promise((resolve, reject) => {
  // Simulate an asynchronous operation
  setTimeout(() => {
    const result = Math.random();
    if (result > 0.5) {
      resolve(result); // Resolve the promise with a value
    } else {
      reject(new Error("Operation failed")); // Reject the promise with an error
    }
  }, 1000);
});

promise
  .then((value) => {
    console.log("Success:", value);
  })
```

```
  .catch((error) => {
    console.error("Error:", error.message);
  });
```

Deferred:

- A Deferred is an object that encapsulates a Promise and allows you to manually resolve or reject it. It provides greater control over the Promise's state.

- You can create a Deferred object using `$.Deferred()` in jQuery.

- Deferreds are commonly used when you want to perform tasks asynchronously and manually notify when they are complete.

Example (using jQuery's Deferred):

```
function asyncTask() {
  const deferred = $.Deferred();

  setTimeout(() => {
    const result = Math.random();
    if (result > 0.5) {
      deferred.resolve(result); // Resolve the deferred with a value
    } else {
      deferred.reject("Task failed"); // Reject the deferred with an error message
    }
  }, 1000);

  return deferred.promise();
}

asyncTask()
  .then((value) => {
    console.log("Success:", value);
  })
  .fail((error) => {
    console.error("Error:", error);
  });
```

Ajax

What is Ajax?

AJAX = Asynchronous JavaScript and XML.

In short; AJAX is about loading data in the background and display it on the webpage, without reloading the whole page.

Use of Ajax

1. Dynamic Content Loading: Ajax allows you to load new content into a web page without the need for a full page refresh. For example, you can load additional comments on a blog post or new messages in a chat application.

2. Infinite Scrolling: Instead of paginating content, you can use Ajax to continuously load more items as the user scrolls down a page. Social media feeds and product catalogs often use this technique.

3. Real-Time Updates: Ajax is essential for building real-time applications, such as chat applications, online gaming, or collaborative document editing. It enables the server to push updates to the client as soon as they are available.

4. Shopping Cart Updates: E-commerce websites use Ajax to update the shopping cart contents without reloading the entire page. Users can add or remove items from the cart with minimal disruption.

5. Polling for Updates: Applications that need to periodically check for updates, such as checking for new email in a webmail client, use Ajax to fetch new data from the server.

6. Interactive Maps: Maps and location-based applications can use Ajax to update the map and display information as the user interacts with it.

7. Data Retrieval: Ajax can be used to retrieve data from external sources or APIs without reloading the entire page. This is common in weather widgets, stock market trackers, and news tickers.

How to send data with Ajax Request?

In jQuery, you can use the `$.ajax()` method or one of its shorthand methods (e.g., `$.get()`, `$.post()`, `$.getJSON()`) to send data with an Ajax request. Here, I'll demonstrate how to send data using `$.ajax()`, which provides more control and flexibility:

```
// Sending a POST request with data using $.ajax()
$.ajax({
  url: "https://example.com/api/endpoint",
  type: "POST",
  contentType: "application/json",
  data: JSON.stringify({ key1: "value1", key2: "value2" }),
  success: function(response) {
    // Handle the successful response here.
    console.log(response);
  },
  error: function(xhr, status, error) {
    // Handle errors here.
    console.error("Request failed with status: " + status);
  }
});
```

In this example:

1. `url` specifies the URL to which you want to send the request.
2. `type` defines the HTTP method, such as "POST."
3. `contentType` specifies the content type of the data being sent.
4. `data` contains the data you want to send, typically as a JSON string.
5. `success` is a callback function that handles the successful response.
6. `error` is a callback function that handles errors if the request fails.

Difference between get, post, put, delete method

GET:

- **Purpose**: Used to request data from a specified resource.
- **Idempotent**: Yes (multiple identical requests have the same effect as a single request).
- **Cacheable**: Yes (responses can be cached).
- **Data in URL**: Parameters are included in the URL.
- **Use Cases**: Retrieving information, fetching a webpage, searching, or reading data.

POST:

- **Purpose**: Used to submit data to be processed to a specified resource.
- **Idempotent**: No (each request creates a new resource or updates an existing one).
- **Cacheable**: No (responses should not be cached).
- **Data in URL**: Data is sent in the request body.
- **Use Cases**: Submitting a form, creating a new resource, updating a resource with non-idempotent operations.

PUT:

- **Purpose**: Used to update or replace a current resource with new data.
- **Idempotent**: Yes (if you repeat the same request, it will have the same effect as the first request).

- **Cacheable**: Yes (responses can be cached).
- **Data in URL**: Data is sent in the request body.
- **Use Cases**: Updating an existing resource with complete new data.

DELETE:

- **Purpose**: Used to delete a specified resource.
- **Idempotent**: Yes (multiple identical requests have the same effect as a single request).
- **Cacheable**: Yes (responses can be cached).
- **Data in URL**: Data is usually included in the URL.
- **Use Cases**: Deleting a resource, such as a record or file.

JSON data

JSON data is written as name/value pairs, just like JavaScript object properties.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"name":"Jeet"

JSON objects are written inside curly braces.

Just like in JavaScript, objects can contain multiple name/value pairs:

{"firstName":"Jeet", "lastName":"Sorathiya"}

JSON arrays are written inside square brackets.

Just like in JavaScript, an array can contain objects:

```
"students":[
  {"name":"Jeet"},
  {"name":"Aryan"},
  {"name":"Prince"}
]
```

Serialization & De-serialization

Serialization :

Serialization is the process of converting complex data structures or objects into a format that can be easily stored, transmitted, or shared.

```
var data = { key1: 'value1', key2: 'value2' };
var jsonData = JSON.stringify(data);

$.ajax({
   type: 'POST',
   url: 'https://example.com/api',
   data: jsonData,
   contentType: 'application/json',
   success: function(response) { // Handle the response here }});
```

Deserialization :

Deserialization is the reverse process of serialization. It involves taking the serialized data and reconstructing the original data structure or object.

```
$.ajax({
   type: 'GET',
   url: 'https://example.com/api/data',
   dataType: 'json',
   success: function(data) {
      // 'data' is automatically deserialized from JSON to a JavaScript object
      console.log(data.key1);
      console.log(data.key2);
   }
});
```