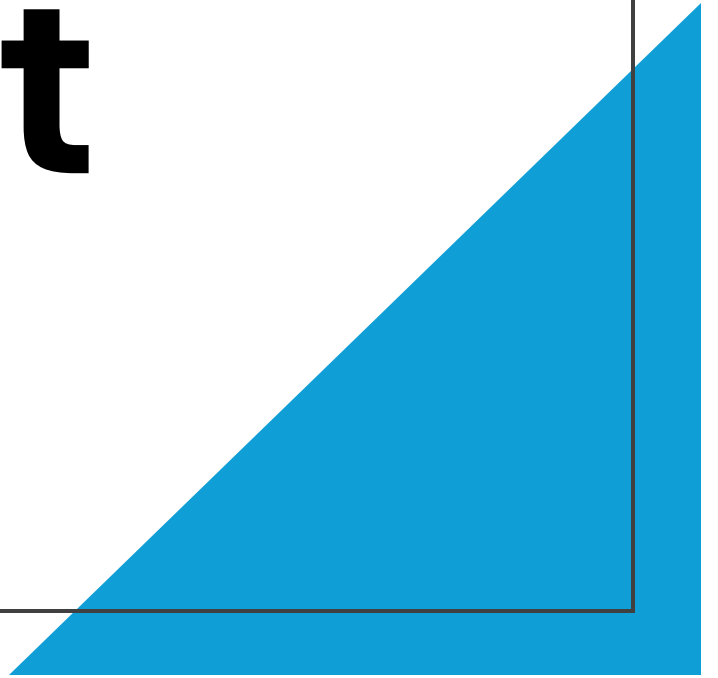


# Full Stack Web Development

by Dr Piyush Bagla



# JavaScript

- JavaScript is the programming language of the web.
- It can update and change both HTML and CSS.
- It can calculate, manipulate and validate data.

# JavaScript

- JavaScript can change HTML Elements.
- JavaScript can change HTML Attributes.
- JavaScript can change CSS.
- JavaScript can Hide HTML Elements.
- JavaScript can show HTML Elements.

# JavaScript

- JavaScript can "display" data in different ways.
- Writing into an HTML element, using **innerHTML**.
  - The **innerHTML** property defines the HTML content
- Writing into the HTML output using **document.write()**.
  - Using **document.write()** after an HTML document is loaded, will **delete all existing HTML**.
  - The **document.write()** method should only be used for testing.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.
  - For debugging purposes, you can call the **console.log()** method in the browser to display data.

# JavaScript can be included in an HTML document in several different ways; the most common are :

- **Inline JavaScript**

```
<button onclick="alert('Hello, world!')">Click me</button>
```

- **Internal JavaScript**

- **Inside <body>**

```
<script>  
    // JavaScript code here  
</script>
```

- **Inside <head>**

```
<script>  
    // JavaScript code here  
</script>
```

- **External JavaScript**

```
<script src="script.js"></script>
```

# Common ways to select HTML element

- `getElementById()`
- `getElementsByClassName()`
- `getElementsByName()`
- `getElementsByTagName()`

# JavaScript Variables

Variables can  
be declared in  
4 ways:

Automatically

```
a = 5;
```

Using **var**

```
var a = 5;
```

Using **let**

```
let a = 5;
```

Using **const**

```
const a = 5;
```



# Block Scope

- Before ES6 (2015), JavaScript did not have **Block Scope**.
  - JavaScript had **Global Scope** and **Function Scope**.
  - ES6 introduced the two new JavaScript keywords: **let** and **const**.
  - These two keywords provided **Block Scope** in JavaScript.
  - Variables declared with the **var** always have **Global Scope or Function Scope**.
  - Variables declared with the **var** keyword can NOT have block scope.
  - Variables declared with **var** inside a { } block can be accessed from outside the block.
-





# Introduced

**var** - pre ES2015

**let** - ES2015 (ES6)

**const** – ES2015 (ES6)

---

# Declaration

```
var a; // allowed
```

```
let b; // allowed
```

```
const c; // not allowed
```

SyntaxError: Missing initializer in const declaration

```
const c = 10; // allowed
```

# Initialization

*Same line Initialization :*

```
var a = 10,  
let b = 20;  
const c = 30;
```

*Later Initialization :*

```
var a;  
a = 10; // allowed  
  
let b;  
b = 20; // allowed  
  
const c;  
c = 30; // not allowed  
SyntaxError: Missing initializer in const declaration
```

# Redeclaration

```
var a = 10;  
var a = 20; //its possible to with var  
  
let b = 10;  
let b = 20; //its not possible with let  
SyntaxError: Identifier 'b' has already been declared  
  
const c = 10;  
const c = 20; //its not possible with const  
SyntaxError: Identifier 'c' has already been declared
```

# Redeclaring **Var**

- Variables defined with **var** **can** be redeclared.
- Redeclaring a variable using the **var** keyword can impose problems.

```
var x = 10;  
// Here x is 10  
{  
  var x = 2;  
  // Here x is 2  
}  
// Here x is 2
```

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

```
var a = 5;  
var a; // var is still 5
```

# Reinitialization

```
var a = 10; //declared once  
a = 20; //reintialized again --> its possible with var
```

```
let b = 10; //declared once  
b = 20; //reintialized again --> its possible with let
```

```
const c = 10; //declared once  
c = 20; //reintialized again --> its NOT possible with let,  
TypeError: Assignment to constant variable.
```

# Scope

## a) Functional Scope: *declare greeting variable without **var***

When we don't declare variables without any var, let and const , variables gets hoisted globally .

```
function wishFoofi() {  
  greeting = "Hello, foofi!"; // hoisted globally  
  console.log(greeting);  
}  
wishFoofi();  
console.log(greeting); // Hello, foofi!
```

### **Output:**

```
Hello, foofi!  
Hello, foofi!
```

# Scope

a) **Functional Scope:** *declare greeting variable with **var** now*

```
function wishFoofi() {  
  var greeting = "Hello, foofi!"; //greeting remained functional scoped  
  console.log(greeting);  
}
```

**Output:**

Hello, foofi!

ReferenceError: greeting is not defined



# Scope

## b) Block Scope

```
{  
  var x = 10;  
}  
console.log(x); // output 10  
  
{  
  let y = 20;  
}  
console.log(y); // output : ReferenceError: y is not defined  
  
{  
  const z = 30;  
}  
console.log(z); // output : ReferenceError: z is not defined
```

# Hoisted

```
console.log(x); // Outputs 'undefined'  
var x = 10; // Assignment remains in its original position
```

***let and const*** : are also Hoisted\*

\* variables with let and const are also **hoisted but at the top of block scope** and they are not assigned with undefined.

Keywords	var	let	const
Eample	var a = 10	let b = 10	const c = 10
Initialization	Can be declared without an initial value	Can be declared without an initial value	Must be assigned an initial value when declared
Re-declaration	Can be redeclared within the same scope	Cannot be redeclared within the same block scope	Cannot be redeclared within the same block scope
Re-initialization	Can be reassigned	Can be reassigned	Cannot be reassigned
Scope	Function-scoped	Block-scoped	Block-scoped
Hoisted	Hoisted to the function/global scope, initialized with undefined	Hoisted to the block scope, not initialized	Hoisted to the block scope, not initialized
Introduced	Available in JavaScript since the beginning-1995	Introduced in ECMAScript 6 (ES6), also known as ECMAScript 2015	Introduced in ECMAScript 6 (ES6), also known as ECMAScript 2015

# JavaScript Datatypes

## In JavaScript:

- **var**, **let**, and **const** are not data types themselves; they are keywords used for variable declaration.
- Data types in JavaScript are the types of values that variables can hold, such as **numbers**, **strings**, **booleans**, **objects**, **arrays**, etc.
- When you write something like **var a = 10;**, a can hold a number value (10 in this case), so we say that a is a variable holding a numeric literal.
- In summary, while in **C** and **C++**, we **explicitly specify the data type of a variable when declaring it**, in **JavaScript**, we declare variables using keywords like **var**, **let**, or **const**, and the data type of a variable is determined by the value it holds.

# JavaScript Data Types

## JavaScript has 8 Datatypes

1. String
2. Number
3. BigInt
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

## The Object Datatype

The object data type can contain:

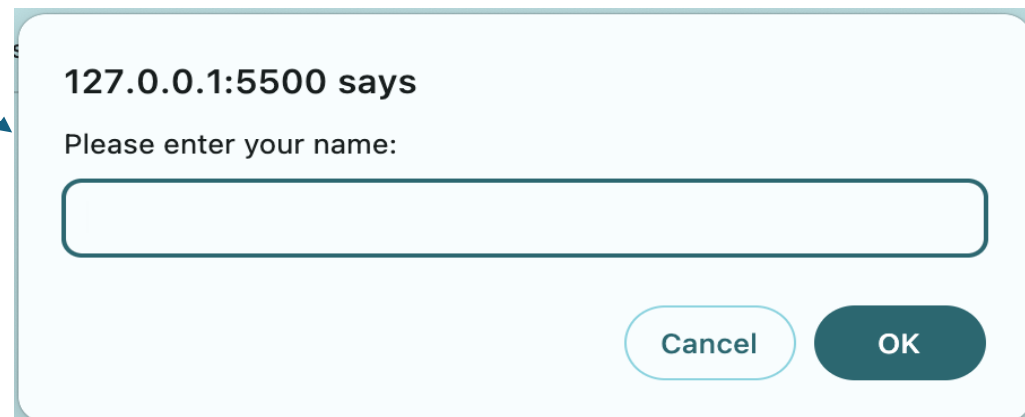
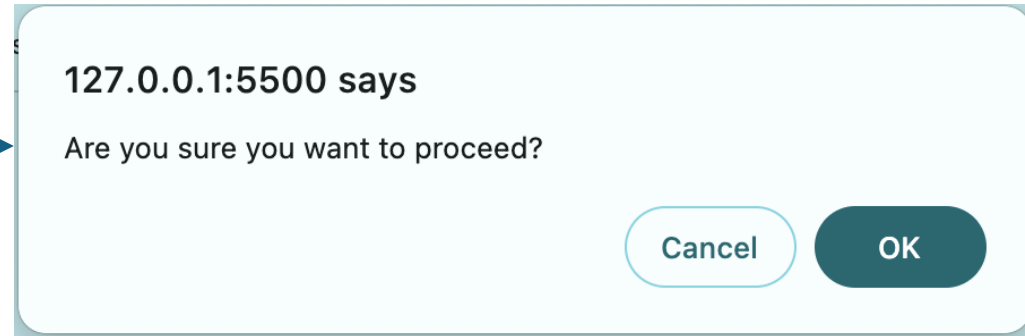
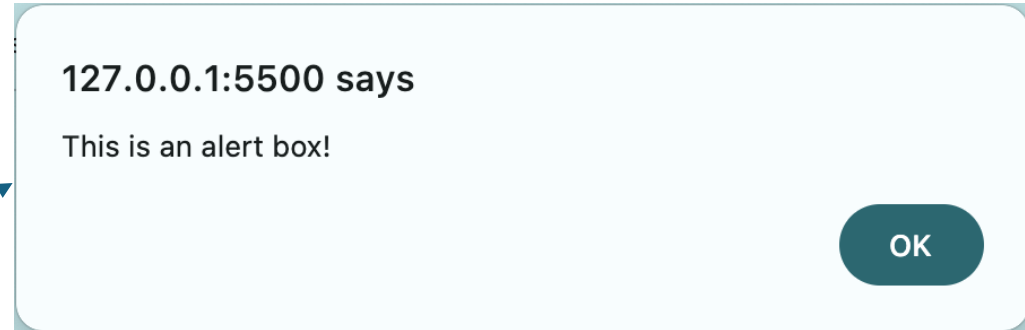
1. Primitive values
2. Other objects
3. Arrays
4. Functions
5. Dates
6. Symbols and more

# Pop-up boxes in JavaScript

1. Alert Box

2. Confirm Box

3. Prompt Box



# JavaScript Form Validation

Forms can be validated using HTML, and JavaScript.

**1.HTML Validation:** HTML5 introduced several built-in validation attributes for form inputs such as required, min, max, pattern, etc.

**2.JavaScript Validation:** You can use JavaScript to implement custom validation logic beyond what HTML provides. This allows you to perform complex validation tasks, such as validating the format of input data, checking for uniqueness, or interacting with external data sources for validation.

**3.CSS Role:** While CSS itself cannot perform form validation, it can be used to enhance the visual feedback of validation errors. For example, you can style invalid form inputs using the **:invalid** and **:valid** pseudo-classes to provide visual cues to users.

# Regular Expression

Regular expressions in JavaScript, also known as **regex** or **regexp**, are a sequence of characters that form a search pattern.

## 1. Creating Regular Expression

Regular expressions in JavaScript can be created using the **RegExp** constructor or by using a regex literal enclosed in forward slashes (/).

```
// Using RegExp constructor
```

```
const regex1 = new RegExp('pattern');
```

```
// Using regex literal
```

```
const regex2 = /pattern/;
```



## 2. Matching Patterns

Regular expressions are used with string methods like **test()** and **match()** to check if a pattern matches a string or to extract substrings that match the pattern.

**Example:**

```
const str = 'Hello, world!';  
const pattern = /hello/i; // Case-insensitive match  
console.log(pattern.test(str)); // Output: true  
console.log(str.match(pattern)); // Output: ["Hello"]
```

### 3. Modifiers

Regular expressions support modifiers that affect how a pattern is matched, such as:

1. i: Case-insensitive match
2. g: Global match (find all matches, not just the first)
3. m: Multiline match

#### Example:

```
const str = 'Apple, apple, APPLE';  
const pattern = /apple/ig; // Case-insensitive, global match  
console.log(str.match(pattern)); // Output: ["Apple", "apple", "APPLE"]
```

#### Replace

```
str.replace(/apple/i, "GEHU");
```

# Regular Expression Patterns

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

Expression	Description
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x y)	Find any of the alternatives separated with

**Metacharacters** are characters with a special meaning:

Metacharacter	Description
\w	Find a word character
\W	Find a non word character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\d	Find a digit

# Regular Expression Patterns

**Quantifiers** define quantities:

Quantifier	Description
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n

# DOM – Document Object Model

- The DOM is a W3C (World Wide Web Consortium) standard.
- The DOM (Document Object Model) is a programming interface provided by web browsers that represents HTML, XML, and XHTML documents as a structured tree of objects.
- It defines the logical structure of documents and how a document is accessed and manipulated.
- When a web page is loaded, the browser creates a **Document Object Model** of the page.

# HTML DOM

```
- Document
  └─ <html>
    └─ <head>
      └─ <meta charset="UTF-8">
      └─ <meta name="viewport" content="width=device-width, initial-scale=1.0">
      └─ <title> ("DOM Example")
    └─ <body>
      └─ <h1 id="title"> ("Hello, World!")
      └─ <p class="info"> ("This is a simple paragraph.")
      └─ <button> ("Click Me")
      └─ <script> (Contains Javascript function)
```

## Note:

**window** object represents the browser window and provides browser-related functionality such as navigating to URLs (**window.location**), managing browser history (**window.history**), setting timeouts (**window.setTimeout**), and more., the **document** object represents the HTML document loaded in that window and allows manipulation of its content.

# Manipulation Using DOM

Below are some of the functionality that JavaScript can perform

- JavaScript Can Change HTML Content
- JavaScript Can Change HTML Attribute Values
- JavaScript Can Change HTML Styles (CSS)
- JavaScript Can Hide HTML Elements
- JavaScript Can Show HTML Elements
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events on the page
- JavaScript can create new HTML events on the page

# JavaScript can create new HTML events on the page

## 1. Creating and Dispatching Standard Events

You can manually trigger built-in events like **click**, **keydown**, or **change** using JavaScript.

```
<body>
  <button id="myButton">Click Me</button>
  <script>
    let button = document.getElementById("myButton");
    // Add a click event listener
    button.addEventListener("click", function () {
      alert("Button was clicked!");
    });
    // Manually trigger a click event
    function triggerClick() {
      let event = new Event("click"); // Create a standard click event
      button.dispatchEvent(event);    // Dispatch the event
    }
    // Simulate a click after 3 seconds
    setTimeout(triggerClick, 3000);
  </script>
</body>
```



# JavaScript can create new HTML events on the page

## 2. Creating and Dispatching Custom Events

- Use the **CustomEvent** constructor to create events with additional data.
- Use **dispatchEvent()** to trigger the event.

### Example: Creating a Custom Event

```
<body>
  <button id="triggerEvent">Trigger Custom Event</button>
  <p id="message"></p>

  <script>
    // Step 1: Create an event listener for the custom event
    document.addEventListener("myCustomEvent", function(event) {
      document.getElementById("message").textContent = event.detail.message;
    });

    // Step 2: Dispatch the custom event when the button is clicked
    document.getElementById("triggerEvent").addEventListener("click", function() {
      let customEvent = new CustomEvent("myCustomEvent", { detail: { message: "Custom event triggered!" } });
      document.dispatchEvent(customEvent);
    });
  </script>
</body>
```

# JQuery (\$)

## What is jQuery?

- Lightweight JavaScript Library.
- Simplifies JavaScript programming.
- **Write Less, Do More.**
- Browser independent.
- Used by Google, Microsoft, IBM, Netflix

## Features:

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

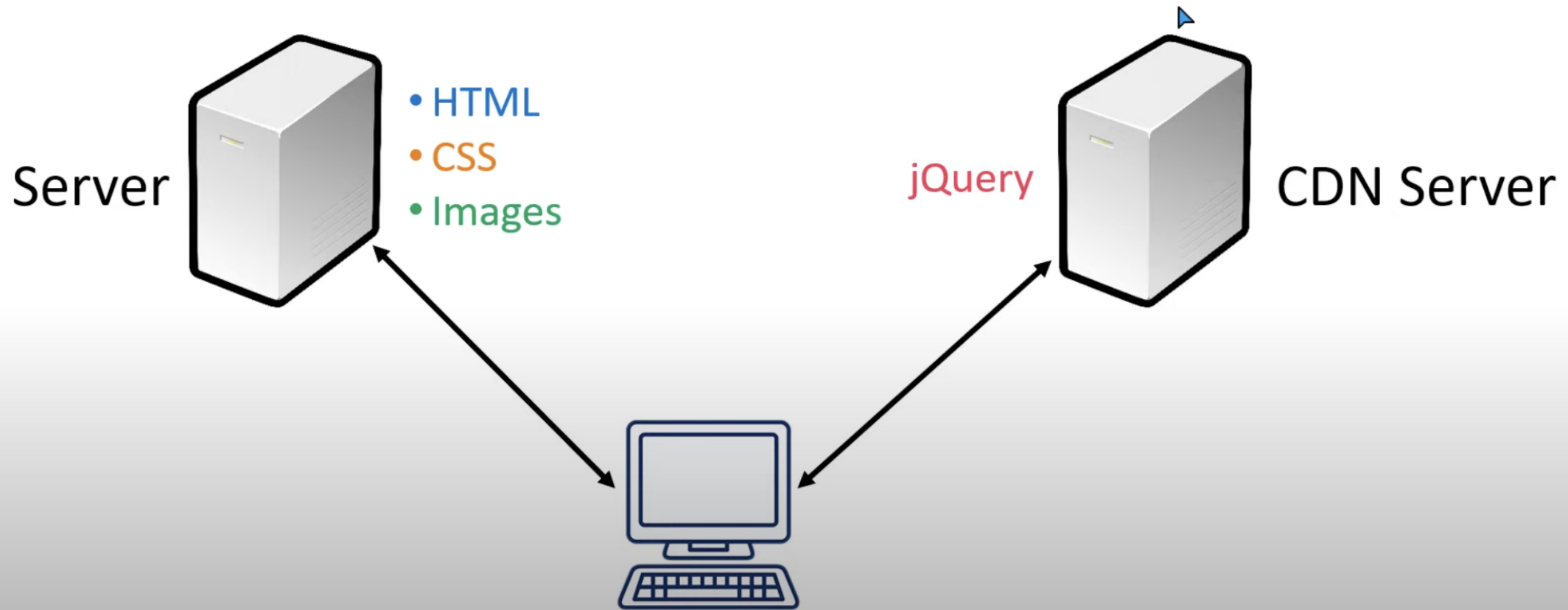


# Where to get JQuery?

There are two ways to include JQuery on your own website.

- [Download](#) the JQuery library.
    - Production Version – minify and compressed
    - Development Version – uncompressed and human readable
  - Include JQuery from a [CDN](#) say Google.
-

# Content Delivery Network



# Syntax

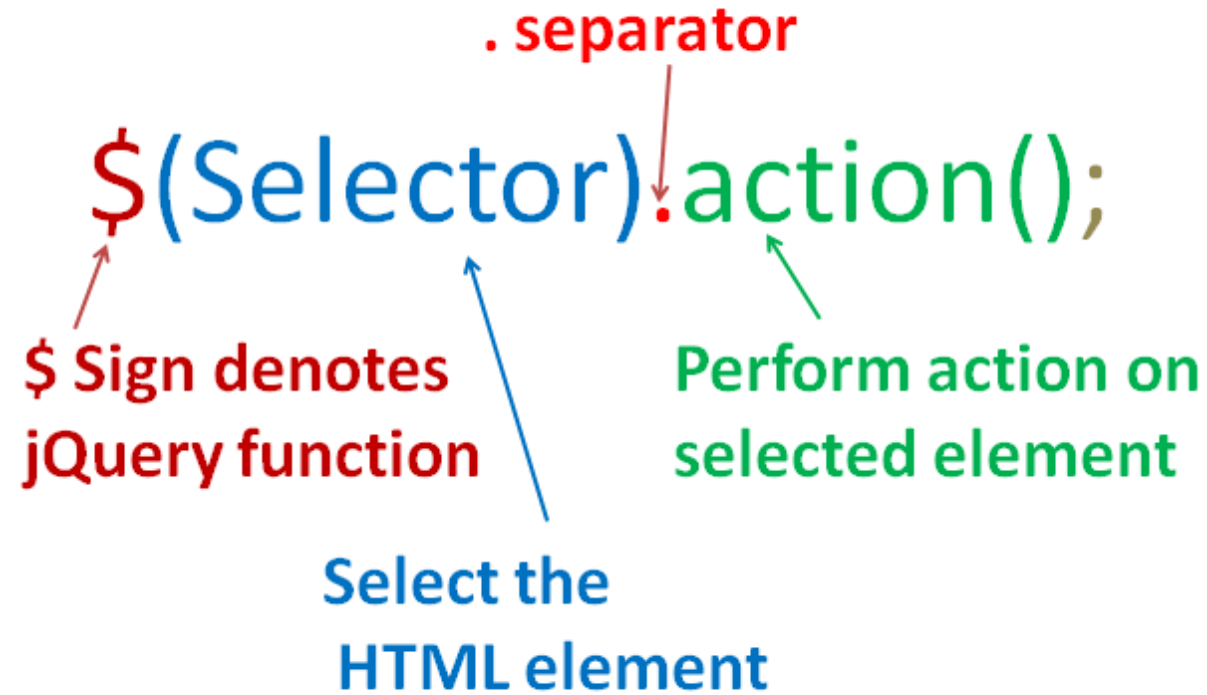
**\$**(Selector)**.**action();

**\$ Sign denotes  
jQuery function**

**Select the  
HTML element**

**. separator**

**Perform action on  
selected element**

A diagram illustrating the jQuery syntax. The text '\$(Selector).action();' is shown with color-coded parts: '\$' is red, '(Selector)' is blue, '.' is red, 'action()' is green, and ';' is brown. Four arrows point to specific parts with labels: a red arrow from '\$ Sign denotes jQuery function' to the '\$' symbol; a blue arrow from 'Select the HTML element' to the '(Selector)' text; a red arrow from '. separator' to the '.' symbol; and a green arrow from 'Perform action on selected element' to the 'action()' text.

## JavaScript

`document.getElementById("id");`

## jQuery

`$("#id");`

# Syntax

```
$(document).ready(function(){  
    // jQuery methods go here...  
});
```

**OR**

```
$(function(){  
    // jQuery methods go here...  
});
```



# Get Methods

- `text()`
  - `html()`
  - `attr()`
  - `val()`
-



# Set Methods

- `text()`
  - `html()`
  - `attr()`
  - `val()`
-





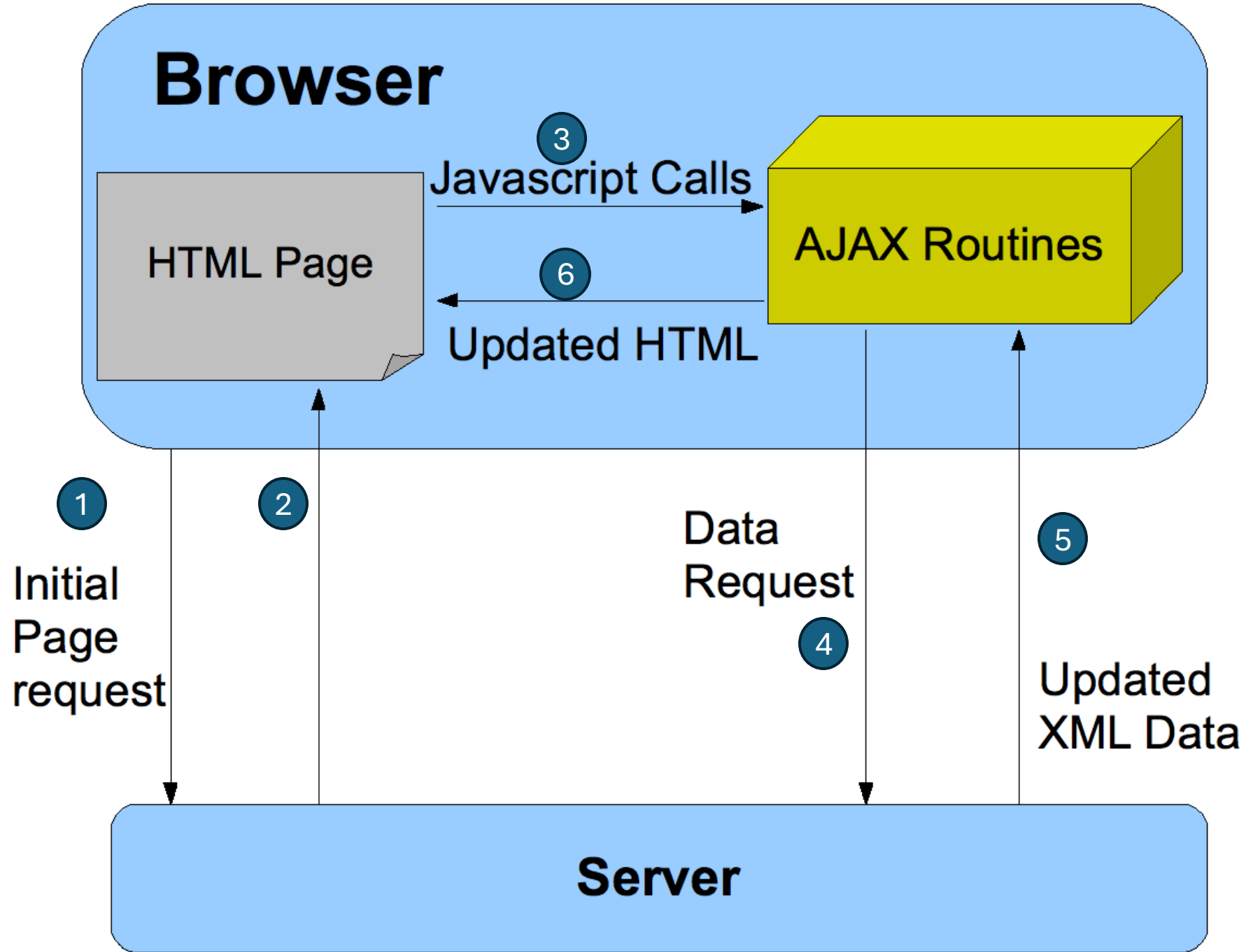
# AJAX

- **A**synchronous **J**avaScript **A**nd **X**ML.
  - AJAX is the art of exchanging data with a server and updating parts of a web page - without reloading the whole page.
  - Examples of applications using AJAX are Gmail, Google Maps, YouTube, and Facebook tabs.
  - With the jQuery AJAX methods, you can request text, HTML, XML, or JSON from a remote server using both HTTP Get and HTTP Post - And you can load the external data directly into the selected HTML elements of your web page!
-



## Note:

Writing regular AJAX code can be complex due to variations in syntax across different browsers. This requires additional code to handle browser compatibility issues. However, jQuery simplifies AJAX implementation by providing a unified syntax, allowing developers to write AJAX functionality with just one line of code.



# Ways to Implement AJAX in JavaScript

There are several ways to implement AJAX in JavaScript. Below are the main methods:

- 1 Using XMLHttpRequest (Oldest Method)
- 2 Using fetch() API
- 3 Using async/await with fetch()
- 4 Using Axios (Third-Party Library)
- 5 **Using jQuery AJAX**

# load()

The load() method in jQuery is a simple way to **fetch data from a server and insert it into an element** without refreshing the page.

## Syntax

```
$(selector).load(url, data, callback);
```

## Parameters:

- **url** → The file or API endpoint to load data from.
- **data** (Optional) → Data to send to the server (for POST requests).
- **callback** (Optional) → Function to execute after the request is complete.

# get()

```
$.get(URL, callback);
```

## Parameters:

- The required URL parameter specifies the URL you wish to request.
- The optional callback parameter is the name of a function to be executed if the request succeeds.

The callback function can have different parameters:

- **data** - holds the content of the page requested,
- **status** - holds the status of the request.

# get()

- Used to **fetch data** (HTML, JSON, or plain text) from a URL.
- **Does NOT insert data automatically**; you must use .html() or .text().
- Cannot directly fetch a specific section of an HTML file or a server response (e.g., from server.php).

```
<div id="content">Click the button to load data here...</div>
<button id="btn">Load Data</button>

<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
<script>
  $(document).ready(function () {
    $("#btn").click(function () {
      $.get("data.html", function(response) {
        $("#content").html(response); // Insert manually
      });
    });
  });
</script>
```

## ◆ Key Differences Between `$.load()` and `$.get()`

Feature	<code>\$.load()</code>	<code>\$.get()</code>
Purpose	Load HTML content directly into an element	Fetch data (HTML, JSON, or text) from a server
Auto Inserts Data?	✓ Yes	✗ No, must use <code>.html()</code>
Can Fetch JSON?	✗ No	✓ Yes
Can Load Part of HTML?	✓ Yes ( <code>\$.load("data.html #sectionID")</code> )	✗ No
Can Send Data?	✗ No	✓ Yes ( <code>\$.get("server.php", {name: "John"})</code> )



# post()

\$.post() is used to **send data** to the server using the HTTP POST method. Unlike \$.get(), it is mainly used when submitting forms, sending user input, or updating data on the server.

```
$.post(URL, data, callback);
```

## Parameters:

- The required URL parameter specifies the URL you wish to request.
- *Data* (optional)– *send additional data to the server along with the request.*
- The optional callback parameter is the name of a function to be executed if the request succeeds.

The callback function can have different parameters:

- **data** - holds the content of the page requested,
- **status** - holds the status of the request.

# post()

```
<div id="content">Click the button to load data here...</div>  
<button id="btn">Load Data</button>
```

```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>  
<script>  
  $(document).ready(function () {  
    $("#btn").click(function () {  
      $.post("server.php", { key: "value" }, function(response) {  
        $("#content").html(response); // Insert response manually  
      });  
    });  
  });  
</script>
```

## ◆ load() vs get() vs post() in jQuery AJAX

Feature	.load()	\$.get()	\$.post()
Use Case	Load HTML content	Fetch data from a server	Send data to a server
Method Type	GET (only)	GET	POST
Sends Data?	✗ No	✗ No (or via URL)	✓ Yes (in request body)
Receives Data?	✓ Yes (HTML only)	✓ Yes (HTML, JSON, etc.)	✓ Yes (HTML, JSON, etc.)
Response Handling	Auto-inserts response into the element	Manual insertion needed	Manual insertion needed
Supports Specific Element?	✓ Yes ( <code>.load("page.html #sectionID")</code> )	✗ No	✗ No
Typical Use	Quickly load HTML into an element	Fetch API data, read files	Submit forms, update DB
Caching?	✓ Yes (May Cache)	✓ Yes (May Cache)	✗ No (More Secure)

# XML



XML stands for  
**eXtensible Markup**  
Language



XML is a markup  
language much like  
HTML



XML was designed  
to store and  
transport data



XML was designed  
to be self-  
descriptive



XML is a W3C  
Recommendation

## The Difference Between XML and HTML

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on structure of a webpage
- XML tags are not predefined like HTML tags.

# XML is Extensible

- We can add our own tags
- Most XML applications will work as expected even if new data is added (or removed).

```
<note>  
  <to>ABC</to>  
  <from type='abc'>XYZ</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>
```

```
<note>  
  <date>2015-09-01</date>  
  <hour>08:30</hour>  
  <to>ABC</to>  
  <from>XYZ</from>  
  <body>Don't forget me this weekend!</body>  
</note>
```

# XML Syntax Rules

- XML Documents Must Have a Root Element
- The XML Prolog

```
<?xml version="1.0" encoding="UTF-8"?>
```

The XML prolog is optional. If it exists, it must come first in the document.

- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Attribute Values Must Always be Quoted
- White-space is Preserved in XML

# XML Syntax Rules

## Entity References

Some characters have a special meaning in XML.

- If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>salary < 1000</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

```
<message>salary &lt; 1000</message>
```

# Well Formed vs Valid XML Documents

- ❖ An XML document with correct syntax is called "Well Formed".
- ❖ A "well formed" XML document is not the same as a "valid" XML document.
- ❖ A "valid" XML document must be well formed. In addition, it must conform to a document type definition.

There are two different document type definitions that can be used with XML:

- DTD - The original Document Type Definition
- XML Schema - An XML-based alternative to DTD



# DTD

A DTD defines the structure and the legal elements and attributes of an XML document.

Note.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

# XML Schema

An XML Schema describes the structure of an XML document, just like a DTD.

```
<xs:element name="note">

  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

</xs:element>
```

# Why XML Schemas are More Powerful than DTD?

- XML Schemas are written in XML
- XML Schemas are extensible to additions
- XML Schemas support data types
- It is easier to convert data between different data types
- XML Schemas support namespaces
- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schemas with the XML DOM
- You can transform your Schemas with XSLT

# JSON

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. It's widely used for exchanging data between a server and a client, often in web applications.

```
{
  "users": [
    {
      "id": 1,
      "name": "John Doe",
      "email": "john.doe@example.com",
      "hobbies": ["reading", "traveling"]
    },
    {
      "id": 2,
      "name": "Jane Smith",
      "email": "jane.smith@example.com",
      "hobbies": ["photography", "painting"]
    }
  ]
}
```

# XML vs JSON

Feature	XML (Extensible Markup Language)	JSON (JavaScript Object Notation)
Purpose	Designed to <b>store and transport data</b>	Designed to <b>store and exchange data in a lightweight format</b>
Syntax	Uses <b>tags</b> ( <code>&lt;tag&gt;...&lt;/tag&gt;</code> ) like HTML	Uses <b>key-value pairs</b> ( <code>"key": "value"</code> )
Data Size	<b>Larger</b> (more verbose with opening & closing tags)	<b>Smaller</b> (more compact, no redundant tags)
Readability	Less readable due to tags	Easier to read due to simple structure
Parsing	Requires an <b>XML parser (DOM, SAX, etc.)</b>	Can be parsed directly in <b>JavaScript</b> using <code>JSON.parse()</code>
Data Types	Everything is stored as a <b>string</b>	Supports multiple data types ( <code>string</code> , <code>number</code> , <code>boolean</code> , <code>array</code> , <code>null</code> )
Namespaces	<b>Supports namespaces</b> to avoid conflicts	<b>No built-in namespace support</b>
Extensibility	<b>More flexible</b> (can define a schema using DTD/XSD)	<b>Limited structure</b> , but sufficient for most cases
Support	Widely used in <b>enterprise applications, databases, web services (SOAP)</b>	Preferred for <b>web APIs, JavaScript-based apps (REST, GraphQL)</b>
Usage	Used in <b>config files, web services, RSS feeds</b>	Used in <b>APIs, JavaScript apps, NoSQL databases (MongoDB)</b>