Full Stack Web Development

UNIT \_ I Introduction to CSS and JavaScript

Web:

Web consists of billions of clients and server connected through wires and wireless networks. The web clients make requests to web server. The web server receives the request, finds the resources and return the response to the client. When a server answers a request, it usually sends some type of content to the client. The client uses web browser to send request to the server. The server often sends response to the browser with a set of instructions written in HTML(HyperText Markup Language). All browsers know how to display HTML page to the client.

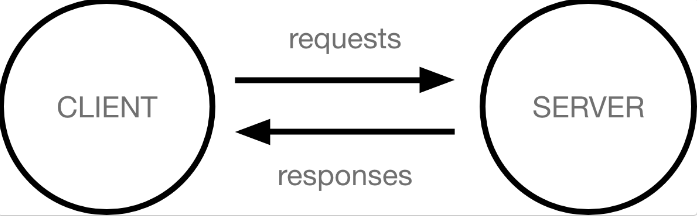
Web Application

A website is a collection of static files(webpages) such as HTML pages, images, graphics etc. A Web application is a web site with dynamic functionality on the server. Google, Facebook, Twitter are examples of web applications.

### HTTP (Hypertext Transfer Protocol)

* HTTP is a protocol that clients and servers use on the web to communicate.
* It is similar to other internet protocols such as SMTP(Simple Mail Transfer Protocol) and FTP(File Transfer Protocol) but there is one fundamental difference.
* HTTP is a stateless protocol i.e HTTP supports only one request per connection. This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
* The client sends an HTTP request and the server answers with an HTML page to the client, using HTTP.

Computers connected to the web are called **clients** and **servers**. A simplified diagram of how they interact might look like this:



* Clients are the typical web user's internet-connected devices (for example, your computer connected to your Wi-Fi, or your phone connected to your mobile network) and web-accessing software available on those devices (usually a web browser like Firefox or Chrome).
* Servers are computers that store webpages, sites, or apps. When a client device wants to access a webpage, a copy of the webpage is downloaded from the server onto the client machine to be displayed in the user's web browser.

HTTP

* API : Programming Interfaces enable software to interact with other software through exposed functionality.
* Client : The client is the initiating party that sends an API request. Often times there will be many clients consuming the same API.
* Server : The server is software or hardware that provides a service by responding to requests across a network.

Main points

* Client-server communication Model
* The HTTP (Hypertext Transfer Protocol) is a communication protocol
* Markup Languages

Protocols

The Internet is essentially a network of computer machines.

In the early days of the Internet, each machine on the network must have an Internet Protocol (IP) address, which is in the form of nnn.nnn.nnn.nnn where nnn must be a number from 0 - 255. Domain names works as easier to remember masks onto IP addresses. The transmission of information is broken into packets and sent along the TCP/IP protocol stack, which has many layers from the hardware layer to the application layer.

Learning about each network protocol layer and how they fit together would be more than a semester long course. The main protocol to focus on for building web applications are Hypertext Transfer Protocol (HTTP), which is on the application layer. Another protocol on the application layer to be aware of is the Simple Mail Transfer Protocol (SMTP) for electronic mail, e-mail for short. A majority of "web development work" when folks talk about front end or back end work falls more into the scope of HTTP rather than SMTP layer. That said, the rest of this guide focuses on HTTP.

HTTP & Client-Server Communication

HTTP or HyperText Transfer Protocol, as the name suggests it is a protocol, it is a set of rules that the server needs to follow to transmit all kinds of files like images, text, audio, video, and other kinds over the world wide web (www).

The internet is made up of clients and servers. So when you are accessing the internet through a web client like browsers like google chrome, Mozilla, IE, etc. When you enter the name of any website you want to visit, you are sending a request to a web server. Suppose, you type amazon.in, you (the browser which is the web client) is requesting the web server and often requesting a bunch of documents from that server.

Maybe it is an HTML, CSS, images, video, JSON. You make a request and the server responds, that is the basic relationship. This request is made using the HTTP protocol. A protocol is just a set of rules or standards that everyone on the internet has agreed to.

This is the client-server request-response model. HTTP is an application layer protocol and usually communicates with the server using the Transmission Control Protocol (TCP).

Hypertext Transfer Protocol (HTTP) is the protocol that web browsers and web servers use "under the hood" to communicate with each other over the Internet. It is text based. HTTP is a protocol which allows the fetching of resources, such as HTML documents. After the request is serviced by a server, the connection between client and server across the Internet is disconnected. A new connection must be made for each request. Most protocols are connection oriented, where the connection is kept open over the Internet. HTTP does not however. Before an HTTP request can be made by a client, a new connection must be made to the server.

Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called requests and the messages sent by the server as an answer are called responses. A request is made by an entity called a user-agent, which is typically a web browser however can be a bot or scraper. The server answer with a response. In between can be any number of proxies or caches that can act as gateways.

HTTP is stateless, which means inherently data isn’t saved. HTTP cookies allow use of stateful sessions. This might be used for example with an e-commerce website as you click from page to page.

HTTP Request

Requests consists of the following elements:

An HTTP method, usually a verb like GET, POST or a noun like OPTIONS or HEAD that defines the operation the client wants to perform. Typically, a client wants to fetch a resource (using GET) or post the value of an HTML form (using POST), though more operations may be needed in other cases.

The path of the resource to fetch; the URL of the resource stripped from elements that are obvious from the context, for example without the protocol (http://), the domain (here, developer.mozilla.org), or the TCP port (here, 80).

The version of the HTTP protocol.

Optional headers that convey additional information for the servers.

Or a body, for some methods like POST, similar to those in responses, which contain the resource sent.

HTTP Response

Responses consist of the following elements:

The version of the HTTP protocol they follow.

A status code, indicating if the request was successful, or not, and why.

A status message, a non-authoritative short description of the status code.

HTTP headers, like those for requests.

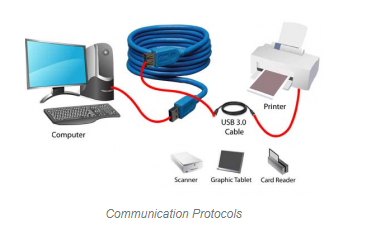
Optionally, a body containing the fetched resource.

# What are Communication Protocols & Their Working

In the digital world, communication protocols create many rules. For instance, on the Internet, the formation of these protocols can be done by groups such as the association of WWW (World Wide Web) or W3C & the IETF (Internet Engineering Task) assists in providing worldwide operations & limits different types of liability as well as vulnerabilities within these technologies. Communication protocol plays a key role when HTTP turns into HTTPS otherwise an extensive use of SSL (Secure Sockets Layer) certificate becomes the standard. Other types of protocols use the data packets within global network routes and sometimes its looks like particle physics. So, technology is advancing day by day, so different communication protocols are used in advanced networks. The Internet of Things (IoT) assures the latest developing communications protocols to connect the wide set of devices to a universal network. This article discusses an overview of communication protocols in embedded systems.

## What are Communication Protocols?

The proper descriptions of digital message formats as well as rules are known communication protocols. The main function of these protocols is to exchange messages from one computer system to another. These are significant in telecommunications systems as they consistently send and receive messages. These protocols cover error detection & correction, signaling, and authentication. They can also explain the semantics, syntax & brings analog & digital communications together.



The implementation of these protocols can be done within hardware as well as software. So communications protocols are available around thousand types which are used all over in analog & digital communications, so computer networks cannot be present without them.

* Protocol: A set of rules and regulations is called a protocol.
* Communication: Exchange of information from one system to another system with a medium is called communication.
* Communication Protocol: A set of rules and regulations that allow two electronic devices to connect to exchange the data with one and another.

#### Why is Communication Protocol Important?

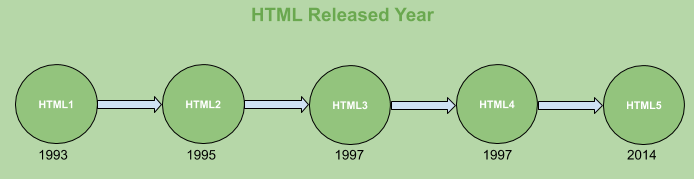
Communication protocols assist varied network devices to converse with each other by transmitting the analog signals, digital signals, different files & process the data from one device to other devices. These types of protocols are applicable in telecommunication & computer networks where suitable rules are executed to transmit information from source to destination. The most vital protocols within networking are TCP (Transmission Control Protocol) & User datagram protocol (UDP).

HTML

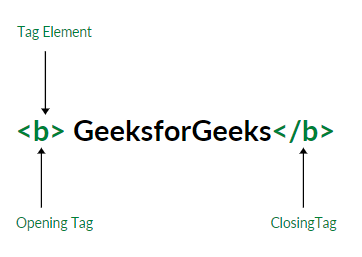
* HTML stands for Hyper Text Markup Language
* HTML is the standard markup language for creating Web pages
* HTML describes the structure of a Web page
* HTML consists of a series of elements
* HTML elements tell the browser how to display the content
* HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

**HTML** stands for HyperText Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

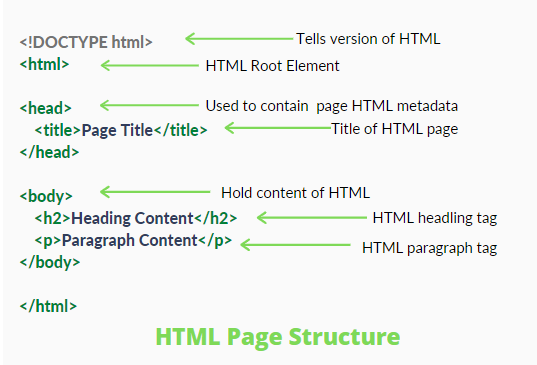
HTML is a markup language used by the browser to manipulate text, images, and other content, in order to display it in the required format. HTML was created by Tim Berners-Lee in 1991. The first-ever version of HTML was HTML 1.0, but the first standard version was HTML 2.0, published in 1995.



**Elements and Tags:** HTML uses predefined [tags](https://www.geeksforgeeks.org/html-html-tag/#:~:text=The%20tag%20in%20HTML,DOCTYPE%3E%20tag.) and [elements](https://www.geeksforgeeks.org/html-elements/) which tell the browser how to properly display the content. Remember to include closing tags. If omitted, the browser applies the effect of the opening tag until the end of the page.



**HTML page structure:**The basic structure of an HTML page is laid out below. It contains the essential building-block elements (i.e. doctype declaration, HTML, head, title, and body elements) upon which all web pages are created.



[<!DOCTYPE html>](https://www.geeksforgeeks.org/html-doctypes/): This is the document type declaration (not technically a tag). It declares a document as being an HTML document. The doctype declaration is not case-sensitive.

[<html>](https://www.geeksforgeeks.org/html-html-tag/): This is called the HTML root element. All other elements are contained within it.

[<head>](https://www.geeksforgeeks.org/html-head-tag/#:~:text=The%20tag%20in%20HTML,head%3E%20element%20can%20be%20omitted.): The head tag contains the “behind the scenes” elements for a webpage. Elements within the head aren’t visible on the front-end of a webpage. HTML elements used inside the <head> element include:

* [<style>](https://www.geeksforgeeks.org/html-style-tag/)
* [<title>](https://www.geeksforgeeks.org/html-title-tag/)
* [<base>](https://www.geeksforgeeks.org/html-base-tag/)
* [<noscript>](https://www.geeksforgeeks.org/html-noscript-tag/)
* [<script>](https://www.geeksforgeeks.org/html-script-tag/)
* [<meta>](https://www.geeksforgeeks.org/html-meta-tag/#:~:text=The%20tag%20in%20HTML,keywords%2C%20document%20author%2C%20etc.)
* [<link>](https://www.geeksforgeeks.org/html-link-tag/)

[<body>](https://www.geeksforgeeks.org/html-body-tag/#:~:text=The%20tag%20in%20HTML,well%20as%20an%20ending%20tag.): The body tag is used to enclose all the visible content of a webpage. In other words, the body content is what the browser will show on the front-end.

An HTML document can be created using any text editor. Save the text file using .html or .htm. Once saved as an HTML document, the file can be opened as a webpage in the browser.

Example: This example illustrates the basic structure of HTML code.

<!DOCTYPE html>

<html>

<head>  <title>Demo Web Page</title> </head>

<body>

    <h1>Hello World.!</h1>

<p>Welcome</p>

</body>

</html>

Output:

Hello World.!

Welcome

Working with Text and Images with CSS

* CSS is the language we use to style an HTML document.
* CSS describes how HTML elements should be displayed.
* This tutorial will teach you CSS from basic to advanced.

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

**C**ascading **S**tyle **S**heets, fondly referred to as **CSS**, is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes up each web page. It describes how a webpage should look: it prescribes colors, fonts, spacing, and much more. In short, you can make your website look however you want. CSS lets developers and designers define how it behaves, including how elements are positioned in the browser. while html uses tags, css uses rulesets.  
CSS is easy to learn and understand, but it provides powerful control over the presentation of an HTML document.

* CSS saves time: You can write CSS once and reuse the same sheet in multiple HTML pages.
* Easy Maintenance: To make a global change simply change the style, and all elements in all the webpages will be updated automatically.
* Search Engines: CSS is considered a clean coding technique, which means search engines won’t have to struggle to “read” its content.
* Superior styles to HTML: CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
* Offline Browsing: CSS can store web applications locally with the help of an offline cache. Using this we can view offline websites.

**CSS Syntax:**  
A CSS comprises style rules that are interpreted by the browser and then applied to the corresponding elements in your document.  
A style rule set consists of a selector and declaration block.

Selector -- h1

Declaration -- {color:blue;font size:12px;}

* The selector points to the HTML element you want to style.
* The declaration block contains one or more declarations separated by semicolons.
* Each declaration includes a CSS property name and a value, separated by a colon.  
  For Example:  
  –; color is property and blue is value.  
  –; font-size is property and 12px is value.
* A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.

**Example :**  
In the following example all p elements will be center-aligned, with a blue text color:  
 p {

    color: blue;

    text-align: center;

}

**CSS Selectors**  
CSS selectors are used to “find” (or select) HTML elements based on their element name, id, class, attribute, and more.

**1.** **THE UNIVERSAL SELECTORS:**Rather than selecting elements of a specific type, the universal selector quite simply matches the name of any element type

\* {

color: #000000;

}

**2.** **THE ELEMENT SELECTOR:**The element selector selects elements based on the element name. You can select all p elements on a page like this (in this case, all p elements will be center-aligned, with a red text color) :

p {

text-align: center;

color: red;

}

**3. THE DESCENDANT SELECTOR:**Suppose you want to apply a style rule to a particular element only when it lies inside a particular element. As given in the following example, the style rule will apply to the em element only when it lies inside the ul tag.

ul em {

color: #000000;

}

**4. THE ID SELECTOR :**

* The id selector uses the id attribute of an HTML element to select a specific element.
* The id of an element should be unique within a page, so the id selector is used to select one unique element!
* To select an element with a specific id, write a hash (#) character, followed by the id of the element.
* The style rule below will be applied to the HTML element with id=”para1″:

**5. THE CLASS SELECTORS :**

* The class selector selects elements with a specific class attribute.
* To select elements with a specific class, write a period (.) character, followed by the name of the class.
* In the example below, all HTML elements with class=”center” will be red and center-aligned:

<p class="center large">This paragraph refers to two classes.</p>

Example

<style>

main {

width: 200px;

height: 200px;

padding: 10px;

background: beige;

}

h1 {

font-family: cursive;

color: olivedrab;

border-bottom: 1px dotted darkgreen;

}

p {

font-family: sans-serif;

color: orange;

}

</style>

CSS Images

Rounded Images

Use the border-radius property to create rounded images:

img {  
  border-radius: 8px;  
}

img {  
  border-radius: 50%;  
}

Responsive Images

img {  
  max-width: 100%;  
  height: auto;  
}

Center an Image

img {  
  display: block;  
  margin-left: auto;  
  margin-right: auto;  
  width: 50%;  
}

|  |
| --- |
| CSS Flexbox |

The **flexbox**or flexible box model in CSS is a one-dimensional layout model that has flexible and efficient layouts with distributed spaces among items to control their alignment structure ie., it is a layout model that provides an easy and clean way to arrange items within a container. Flexbox can be useful for creating small-scales layouts & is responsive and mobile-friendly.

The Flexible Box Module, usually referred to as flexbox, was designed as a one-dimensional layout model, and as a method that could offer space distribution between items in an interface and powerful alignment capabilities. This article gives an outline of the main features of flexbox, which we will be exploring in more detail in the rest of these guides.

When we describe flexbox as being one dimensional we are describing the fact that flexbox deals with layout in one dimension at a time — either as a row or as a column. This can be contrasted with the two-dimensional model of CSS Grid Layout, which controls columns and rows together.

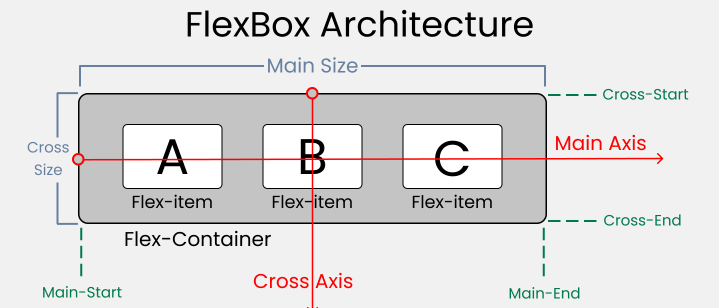
When working with flexbox you need to think in terms of two axes — the main axis and the cross axis. The main axis is defined by the flex-direction property, and the cross axis runs perpendicular to it. Everything we do with flexbox refers back to these axes, so it is worth understanding how they work from the outset.

The main axis is defined by flex-direction, which has four possible values:

* row
* row-reverse
* column
* column-reverse

# Flexbox Architecture

So how does Flexbox architecture work? The **flex-items**[Contents] are distributed along the **Main Axis**and**Cross Axis.**And, depending on the **flex-direction** property, the layout position changes between rows and columns.



# JavaScript

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

* JavaScript is the world's most popular programming language.
* JavaScript is the programming language of the Web.
* JavaScript is easy to learn.
* JavaScript is a lightweight, interpreted programming language.
* Designed for creating network-centric applications.
* Complementary to and integrated with Java.
* Complementary to and integrated with HTML.
* Open and cross-platform

**JavaScript** is a lightweight, interpreted **programming** language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. **JavaScript** is very easy to implement because it is integrated with HTML. It is open and cross-platform.

**JavaScript** is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Javascript:

* Javascript is the most popular **programming language** in the world and that makes it a programmer’s great choice. Once you learnt Javascript, it helps you developing great front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.JS etc.
* Javascript is everywhere, it comes installed on every modern web browser and so to learn Javascript you really do not need any special environment setup. For example Chrome, Mozilla Firefox , Safari and every browser you know as of today, supports Javascript.
* Javascript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.
* JavaScript usage has now extended to mobile app development, desktop app development, and game development. This opens many opportunities for you as Javascript Programmer.
* Due to high demand, there is tons of job growth and high pay for those who know JavaScript. You can navigate over to different job sites to see what having JavaScript skills looks like in the job market.
* Great thing about Javascript is that you will find tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.

Sample JavaScript programing

<html>

<body>

<script language = "javascript" type = "text/javascript">

<!--

document.write("Hello World!")

//-->

</script>

</body>

</html>

Applications of Javascript Programming

As mentioned before, **Javascript** is one of the most widely used **programming languages** (Front-end as well as Back-end). It has it's presence in almost every area of software development. I'm going to list few of them here:

* **Client side validation** - This is really important to verify any user input before submitting it to the server and Javascript plays an important role in validting those inputs at front-end itself.
* **Manipulating HTML Pages** - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.
* **User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.
* **Back-end Data Loading** - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.
* **Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.
* **Server Applications** - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

Advantages of JavaScript

The merits of using JavaScript are −

* **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
* **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
* **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
* **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features −

* Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
* JavaScript cannot be used for networking applications because there is no such support available.
* JavaScript doesn't have any multi-threading or multiprocessor capabilities.

JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types −

* **Numbers,** eg. 123, 120.50 etc.
* **Strings** of text e.g. "This text string" etc.
* **Boolean** e.g. true or false.

JavaScript Variables

* Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.
* Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.
* <script type = "text/javascript">
* <!--
* var money;
* var name;
* //-->
* </script>

You can also declare multiple variables with the same **var** keyword as follows −

* <script type = "text/javascript">
* <!--
* var money, name;
* //-->
* </script>

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

* <script type = "text/javascript">
* <!--
* var name = "Ali";
* var money;
* money = 2000.50;
* //-->
* </script>
* JavaScript Variables
* Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.
* Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.
* <script type = "text/javascript">
* <!--
* var money;
* var name;
* //-->
* </script>
* You can also declare multiple variables with the same **var** keyword as follows −
* <script type = "text/javascript">
* <!--
* var money, name;
* //-->
* </script>

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

* <script type = "text/javascript">
* <!--
* var name = "Ali";
* var money;
* money = 2000.50;
* //-->
* </script>

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

* **Global Variables** − A global variable has global scope which means it can be defined anywhere in your JavaScript code.
* **Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

# JavaScript - Functions

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like **alert()** and **write()** in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

## Function

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

### Syntax

The basic syntax is shown here.

<script type = "text/javascript">

<!--

function functionname(parameter-list) {

statements

}

//-->

</script>

### Example

Try the following example. It defines a function called sayHello that takes no parameters −

<script type = "text/javascript">

<!--

function sayHello() {

alert("Hello there");

}

//-->

</script>

## Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

<html>

<head>

<script type = "text/javascript">

function sayHello() {

document.write ("Hello there!");

}

</script>

</head>

<body>

<p>Click the following button to call the function</p>

<form>

<input type = "button" onclick = "sayHello()" value = "Say Hello">

</form>

<p>Use different text in write method and then try...</p>

</body>

</html>

## Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

### Example

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

<html>

<head>

<script type = "text/javascript">

function sayHello(name, age) {

document.write (name + " is " + age + " years old.");

}

</script>

</head>

<body>

<p>Click the following button to call the function</p>

<form>

<input type = "button" onclick = "sayHello('Zara', 7)" value = "Say Hello">

</form>

<p>Use different parameters inside the function and then try...</p>

</body>

</html>

Event

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Please go through this small tutorial for a better understanding [HTML Event Reference](https://www.tutorialspoint.com/html/html_events_ref.htm). Here we will see a few examples to understand a relation between Event and JavaScript −

## onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

### Example

Try the following example.

<html>

<head>

<script type = "text/javascript">

<!--

function sayHello() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<p>Click the following button and see result</p>

<form>

<input type = "button" onclick = "sayHello()" value = "Say Hello" />

</form>

</body>

</html>

## onsubmit Event Type

**onsubmit** is an event that occurs when you try to submit a form. You can put your form validation against this event type.

### Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

<html>

<head>

<script type = "text/javascript">

<!--

function validation() {

all validation goes here

.........

return either true or false

}

//-->

</script>

</head>

<body>

<form method = "POST" action = "t.cgi" onsubmit = "return validate()">

.......

<input type = "submit" value = "Submit" />

</form>

</body>

</html>

HTML 5 Standard Events

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Value** | **Description** |
| Offline | script | Triggers when the document goes offline |
| Onabort | script | Triggers on an abort event |
| onafterprint | script | Triggers after the document is printed |
| onbeforeonload | script | Triggers before the document loads |
| onbeforeprint | script | Triggers before the document is printed |
| onblur | script | Triggers when the window loses focus |
| oncanplay | script | Triggers when media can start play, but might has to stop for buffering |
| oncanplaythrough | script | Triggers when media can be played to the end, without stopping for buffering |
| onchange | script | Triggers when an element changes |
| onclick | script | Triggers on a mouse click |
| oncontextmenu | script | Triggers when a context menu is triggered |
| ondblclick | script | Triggers on a mouse double-click |
| ondrag | script | Triggers when an element is dragged |
| ondragend | script | Triggers at the end of a drag operation |
| ondragenter | script | Triggers when an element has been dragged to a valid drop target |
| ondragleave | script | Triggers when an element is being dragged over a valid drop target |
| ondragover | script | Triggers at the start of a drag operation |
| ondragstart | script | Triggers at the start of a drag operation |
| ondrop | script | Triggers when dragged element is being dropped |
| ondurationchange | script | Triggers when the length of the media is changed |
| onemptied | script | Triggers when a media resource element suddenly becomes empty. |
| onended | script | Triggers when media has reach the end |
| onerror | script | Triggers when an error occur |
| onfocus | script | Triggers when the window gets focus |
| onformchange | script | Triggers when a form changes |
| onforminput | script | Triggers when a form gets user input |
| onhaschange | script | Triggers when the document has change |
| oninput | script | Triggers when an element gets user input |
| oninvalid | script | Triggers when an element is invalid |
| onkeydown | script | Triggers when a key is pressed |
| onkeypress | script | Triggers when a key is pressed and released |
| onkeyup | script | Triggers when a key is released |
| onload | script | Triggers when the document loads |
| onloadeddata | script | Triggers when media data is loaded |
| onloadedmetadata | script | Triggers when the duration and other media data of a media element is loaded |
| onloadstart | script | Triggers when the browser starts to load the media data |
| onmessage | script | Triggers when the message is triggered |
| onmousedown | script | Triggers when a mouse button is pressed |
| onmousemove | script | Triggers when the mouse pointer moves |
| onmouseout | script | Triggers when the mouse pointer moves out of an element |
| onmouseover | script | Triggers when the mouse pointer moves over an element |
| onmouseup | script | Triggers when a mouse button is released |
| onmousewheel | script | Triggers when the mouse wheel is being rotated |
| onoffline | script | Triggers when the document goes offline |
| onoine | script | Triggers when the document comes online |
| ononline | script | Triggers when the document comes online |
| onpagehide | script | Triggers when the window is hidden |
| onpageshow | script | Triggers when the window becomes visible |
| onpause | script | Triggers when media data is paused |
| onplay | script | Triggers when media data is going to start playing |
| onplaying | script | Triggers when media data has start playing |
| onpopstate | script | Triggers when the window's history changes |
| onprogress | script | Triggers when the browser is fetching the media data |
| onratechange | script | Triggers when the media data's playing rate has changed |
| onreadystatechange | script | Triggers when the ready-state changes |
| onredo | script | Triggers when the document performs a redo |
| onresize | script | Triggers when the window is resized |
| onscroll | script | Triggers when an element's scrollbar is being scrolled |
| onseeked | script | Triggers when a media element's seeking attribute is no longer true, and the seeking has ended |
| onseeking | script | Triggers when a media element's seeking attribute is true, and the seeking has begun |
| onselect | script | Triggers when an element is selected |
| onstalled | script | Triggers when there is an error in fetching media data |
| onstorage | script | Triggers when a document loads |
| onsubmit | script | Triggers when a form is submitted |
| onsuspend | script | Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched |
| ontimeupdate | script | Triggers when media changes its playing position |
| onundo | script | Triggers when a document performs an undo |

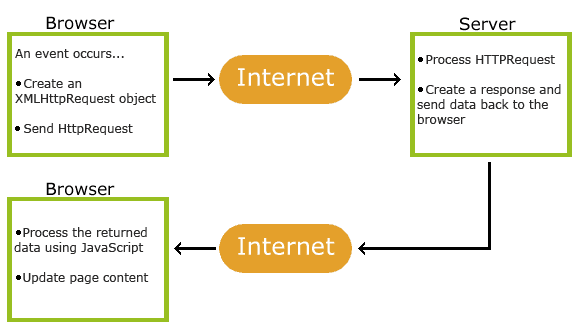
AJAX

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language.

AJAX just uses a combination of:

* A browser built-in XMLHttpRequest object (to request data from a web server)
* JavaScript and HTML DOM (to display or use the data)



AJAX is a developer's dream, because you can:

* Update a web page without reloading the page
* Request data from a server - after the page has loaded
* Receive data from a server - after the page has loaded
* Send data to a server - in the background

Two commonly used methods for a request-response between a client and server are: GET and POST.

* **GET** - Requests data from a specified resource
* **POST** - Submits data to be processed to a specified resource

GET is basically used for just getting (retrieving) some data from the server. **Note:** The GET method may return cached data.

POST can also be used to get some data from the server. However, the POST method NEVER caches data, and is often used to send data along with the request.

Get method:

The $.get() method requests data from the server with an HTTP GET request.

**Syntax:**

$.get(*URL,callback*);

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 following example uses the $.get() method to retrieve data from a file on the server:

### Example

$("button").click(function(){  
  $.get("demo\_test.asp", function(data, status){  
    alert("Data: " + data + "\nStatus: " + status);  
  });  
});

Post method:

The $.post() method requests data from the server using an HTTP POST request.

**Syntax:**

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

The required URL parameter specifies the URL you wish to request.

The optional data parameter specifies some data to send along with the request.

The optional callback parameter is the name of a function to be executed if the request succeeds.

The following example uses the $.post() method to send some data along with the request:

### Example

$("button").click(function(){  
  $.post("demo\_test\_post.asp",  
  {  
    name: "Donald Duck",  
    city: "Duckburg"  
  },  
  function(data, status){  
    alert("Data: " + data + "\nStatus: " + status);  
  });  
});