Now before we dive into DOM, let’s talk about web browsers.

When you try to open a webpage, you go to a browser's address bar, typing [www.google.com](http://www.google.com). You hit enter and you wait, because something must happen and it does, a webpage with a picture of google and a few elements appears right away. Now how did this happen exactly? Have you ever thought about it?

Imagine the following process. This here is the client and this is the server. The client is a program, which in our case is the browser. The server is just a server. Now first, when you type a URL into the browser’s address bar and you hit enter, the browser will go and ask one system, named DNS, where is google.com located? Without going into details, the DNS translates the name by looking it up in its database and says, google.com is located on this IP address. The browser now knows exactly which server it should be asking in order to receive google's landing page and prepares an HTTP request in the following format.

GET / HTTP/1.1

Host: [www.google.com](http://www.google.com)

{headers}

As we know, computers communicate over the internet via the underlying network protocols. The Domain Name System (DNS), which is a hierarchical and decentralized naming system, is one of those protocols and the HyperText Transfer Protocol (HTTP), which is a protocol for retrieving named hypermedia resources, like chunks of information, such as web pages or pictures, is another one. The HTTP functions as a request-response protocol in this client-server communication and since the request is now prepared, the browser sends it to the server and awaits for it's response. The message travels inside something we call a socket, which is established by the TCP/IP protocol and a port number. The client and the server made a handshake, agreeing that for HTTP this port number is 80. Without getting into further networking details, let's see what happens next.

The server is sitting there on the other side, with his programs and services always listening on a specific port and when an HTTP request arrives, the server passes it to port 80, which is the webserver. The webserver is just program, which listens for requests and answers them. In our case the webserver accepts the incomming HTTP request, handles it and returns an appropriate HTTP response in the following format.

HTTP/1.1 200 OK

{headers}

{body}

Basically, what it says is that it has found the resource that we are looking for and is sending it back inside the request's body in Hyper Text. This response now travels back to the client, your browser and without going into details about what Hyper Text is, we can say that your browser receives plain text. That’s right, it does not receive a website, it receives a text. And this is where it gets interesting, the browser must now understand what this text is, so it reads the headers to properly do so and begins parsing it. Everytime, whenever you open a web page in your browser, the browser sends a request and receives a response with a text and parses it, building up a model of the document's structure that represents a tree of elements, which we call the Document Object Model, or DOM for short.

Now that we finally know why it's called a DOM we can understands its definition and begin explaining the theory behind it. The Document Object Model is an API for HTML and XML documents, where API stands for Application Program Interface. The DOM is the parsed tree-like data representation of all HTML and XML text elements in a document, in the form of objects, where the document root is the main object and each other node is an object representing a part of the document.

But why is it a tree and not a graph? Well because the document has only one entry point, the document root element, which contains branches and its structure looks very much like a tree, where each branch of the tree ends in a node and each node itself is an object. This object represents an XML or HTML element and can also contain children of other nodes, or simply said, another objects, which represents XML or HTML elements.

The DOM is also called an API because it has an interface that we can use. The interface of the DOM gives us access to all elements in the DOM tree, by providing us with the root object or its children objects and this root object or his children is something that a JavaScript program can work with very well. Infact this is the sole reason why JavaScript became the undisputed king of the front-end web development. Despite that the interface is language independent, JavaScript excels in the ways in which it can work with this object and manipulate a document. When we access this DOM tree from JavaScript, it acts as a live data structure that you can manipulate and when it's modified, for example if we add a new child element, a new <div> tag, for example <div>Hello World!</div>, the page on the screen is updated to reflect the changes.

**Document Structure**

HTML documents has the following structure.

- The <!DOCTYPE html> declaration, which defines that this document is an HTML5 document.

- The <html> element, which is the root element of an HTML page

- The <head> element, children of <html>, containing meta information about the HTML page.

- The <body> element, children of <html>, defining the document's body and is a container for all the other visible elements.

- The <div> element, children of <body>, is the element we added right now with JavaScript.