**HOW BROWSER WORKS?**

When the web browser fetches data from an internet connected server, it uses a piece of software called a rendering engine to translate that data into text and images. This data is written in HYPER TEXT MARKUP LANGUAGE (HTML) and web browsers read this code to create what we see, hear and experience on the internet.

[HYPERLINKS](https://developer.mozilla.org/docs/Glossary/Hyperlink) allow users to follow a path to other pages or sites on the web. Every webpage, image and video have its own unique [UNIFORM RESOURCE LOCATOR](https://wikipedia.org/wiki/URL) (URL), which is also known as a web address. When a browser visits a server for data, the web address tells the browser where to look for each item that is described in the html, which then tells the browser where it goes on the web page.

**THE BROWSER'S HIGH-LEVEL STRUCTURE**

1. **The user interface**: this includes the address bar, back/forward button, bookmarking menu, etc. Every part of the browser display except the window where you sees the requested page.
2. **The browser engine**: marshals’ actions between the UI and the rendering engine.
3. **The rendering engine:** responsible for displaying requested content. For example, if the requested content is HTML, the rendering engine parses HTML and CSS, and displays the parsed content on the screen.
4. **Networking**: for network calls such as HTTP requests, using different implementations for different platform behind a platform-independent interface.
5. **UI backend**: used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. Underneath it uses operating system user interface methods.
6. **JavaScript interpreter**. Used to parse and execute JavaScript code.
7. **Data storage**. This is a persistence layer. The browser may need to save all sorts of data locally, such as cookies. Browsers also support storage mechanisms such as local Storage, Indexed DB, Web-SQL and Filesystem



**HOW PARSING WORKS**?

*Parsing is the process of breaking down code into individual chunks of code, verifying that all necessary inputs are included in the code, and acting on the instructions dictated by the code.*

In the context of the Web, parsing most commonly happens when a web browser receives the files that comprise a website. Every web browser is equipped with a rendering engine that converts the files into the web page you see in your browser. The rendering engine contains several parsers – the part of the program that parses code prior to rendering the web page. There is a different parser for every language. At a minimum, any modern browser can parse HTML, CSS, and JavaScript.

When website files are received by a browser, each file will be parsed individually, and [parsing takes places in two steps](http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/):

**Lexical analysis:** During lexical analysis the code is analyzed and broken down into individual tokens, or bits of code, that the parser can work with to create a hierarchical model of the contents of the document. Lexical analysis of HTML is also sometimes referred to as tokenization.

**Syntax analysis:**After the lexical analysis has broken the code into workable chunks, the syntax analysis determines how these chunks relate to each other and builds a model of how the rendering engine should process the code based on this analysis. Syntax analysis of HTML is also sometimes referred to as tree construction since it is the process of arranging the tokens into something called a DOM tree that will define the overall structure of the web page.

Here’s a simplified example of how a browser’s HTML parser would handle a short bit of HTML:

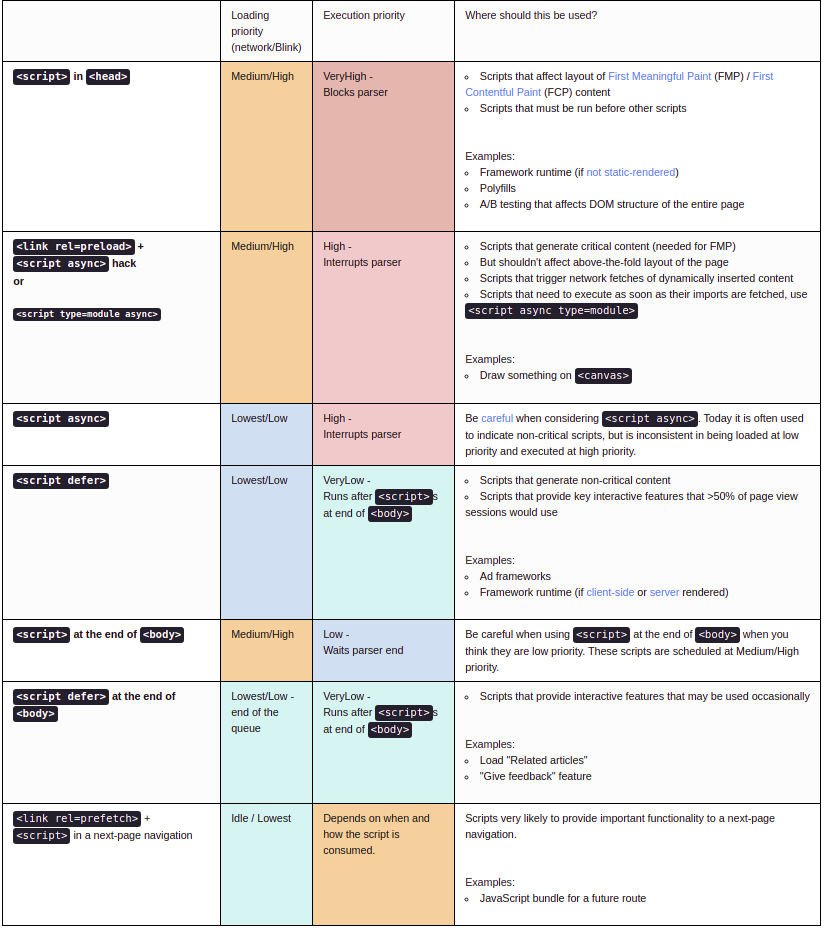
<html>  
  <body>  
    <p>Random paragraph text. </p>  
  </body>  
</html>

First, lexical analysis would tokenize this HTML into the following chunks: **HTML** element, **body** element, **paragraph** element, **text**. Second, syntax analysis would fashion the elements into a tree looking something like this:

HTML --> Body --> Paragraph --> Text

In this simplified example, the tree only has a single branch. In virtually every real-life example there would be many branches.

**THE ORDER OF EXECUTION OF SCRIPTS**

****