The browser's main functionality

The main function of a browser is to display the web resources, by requesting it from the server and displaying it in the browser window. The resource is usually an HTML document, but may also be a PDF, image, or some other type of content. The location of the resource is specified by the user using a URI (Uniform Resource Identifier).

The way the browser interprets and displays HTML files is specified in the HTML and CSS specifications. These specifications are maintained by the W3C (World Wide Web Consortium) organization, which is the standards organization for the web.

Browser does the following things to fetch the data

- look up the location of the web server where the website is hosting.
- make the connection to the server
- send a request to get the specific page
- handle the response from the server and
- how it renders the page so that the viewer, can interact with the website

A browser takes the following steps

1. Browser checks cache for DNS entry to find the corresponding IP address of website. It looks for following cache.
    - Browser Cache
    - Operating Systems Cache
    - Router Cache
    - ISP Cache
2. If not found in cache, ISP's DNS server initiates a DNS query to find IP address of server that hosts the domain name.
   The requests are sent using small data packets that contain information content of request and IP address it is destined for.
3. Browser initiates a TCP connection with the server using synchronize (SYN) and acknowledge (ACK) messages.
4. Browser sends an HTTP request to the web server. GET or POST request.
5. Server on the host computer handles that request and sends back a response. It assembles a response in some format like JSON, XML and HTML.
6. Server sends out an HTTP response along with the status of response.
7. Browser displays HTML content
8. Finally, Done.

The browser's main components are:

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 see 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. JavaScript is an **interpreted** language. This means we do not have to compile the JavaScript source code before sending it to the browser. An interpreter can take the raw JavaScript code and run it for you.

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 localStorage, IndexedDB, WebSQL and FileSystem.

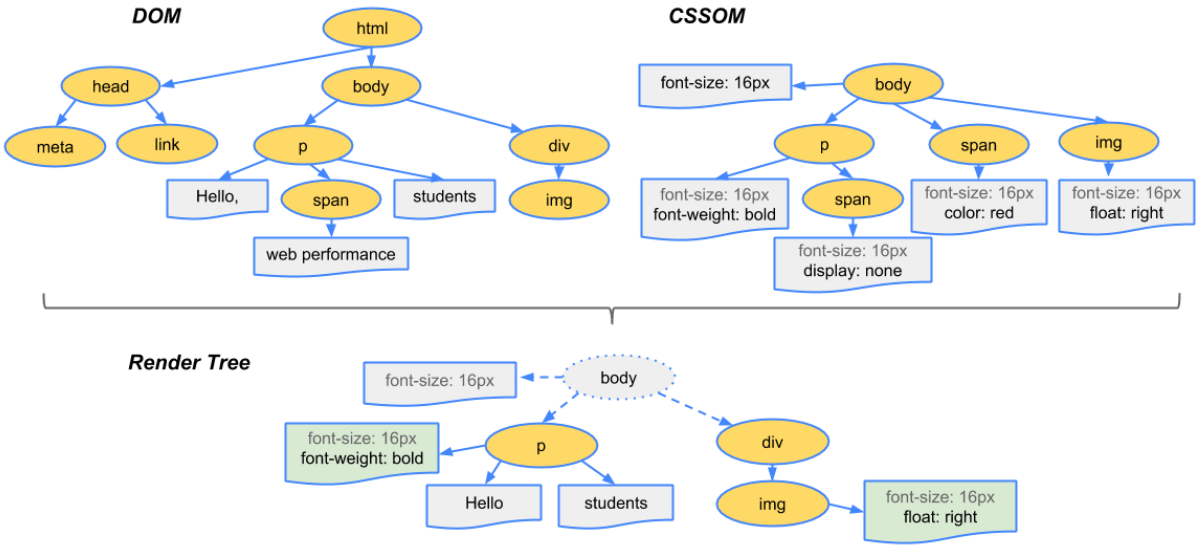
A **rendering engine** (also known as a layout engine or rendering engine) is software that draws text and images on the screen. The engine draws structured text from a document (often HTML), and formats it properly based on the given style declarations (often given in CSS). Examples of layout engines: Blink, Gecko, EdgeHTML, WebKit.

The browser parses the HTML and creates a DOM from it. Next, it parses the CSS. the browser sorts the CSS very quickly and applies that rule to the html elements, then paints the final visual representation to the screen.

Render-tree Construction, Layout, and Paint

- The DOM and CSSOM trees combine to form the render tree.
- Render tree contains only the nodes required to render the page.
- Layout computes the exact position and size of each object.
- The last step is paint, which takes in the final render tree and renders the pixels to the screen.

First, the browser combines the DOM and CSSOM into a "render tree," which captures all the visible DOM content on the page and all the CSSOM style information for each node.



To construct the render tree, the browser roughly does the following:

1. Starting at the root of the DOM tree, traverse each visible node.
    - Some nodes are not visible (for example, script tags, meta tags, and so on), and are omitted since they are not reflected in the rendered output.
    - Some nodes are hidden via CSS and are also omitted from the render tree;
2. For each visible node, find the appropriate matching CSSOM rules and apply them.
3. Emit visible nodes with content and their computed styles.

The final output is a render tree that contains both the content and style information of all the visible content on the screen. **With the render tree in place, we can proceed to the "layout" stage,** which calculates the exact position and size within the viewport of the device---that's the "layout" stage, also known as "reflow."