## Exercise1.1:

## When a user enters an URL in the browser, how does the browser fetch the desired result?

1. Browser checks cache for DNS entry to find the corresponding IP address of website. It looks in following cache.
   1. Browser Cache
   2. Operating Systems Cache
   3. Router Cache
   4. ISP Cache
2. If not found in cache, ISP’s (Internet Service Provider) 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 sends an HTTP request to the web server. GET or POST request.
4. Server handles that request and sends back a HTTP response along with the status of response.
5. Browser renders HTML content of the site requested by the user.

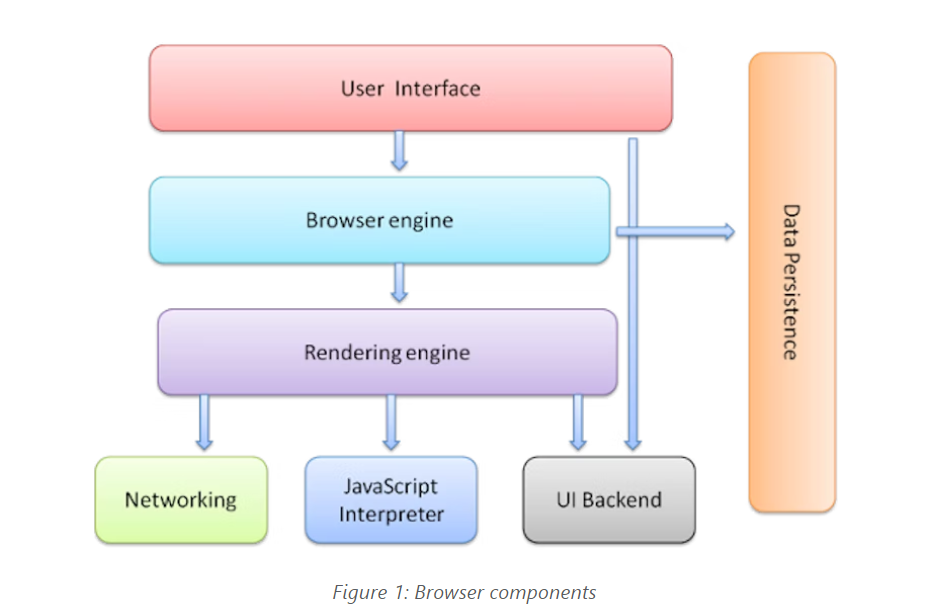
## What is the main functionality of the browser?

A web browser, also known as a “browser,” is an application software that allows users to find, access, display, and view websites. Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, and Apple Safari are all popular web browsers. Web browsers are primarily used to display and access websites on the internet, as well as other content created with programming languages such as Hypertext Markup Language (HTML) and Extensible Markup Language (XML) (XML). Browsers convert Hypertext Transfer Protocol (HTTP) web pages and websites into human-readable content.

## High Level Components of a browser.

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.
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.



## Rendering engine and its use.

As the name suggests, this component is responsible for rendering a specific web page requested by the user on their screen. It interprets HTML and XML documents along with images that are styled or formatted using CSS, and a final layout is generated, which is displayed on the user interface.

Every browser has its own unique rendering engine. Rendering engines might also differ for different browser versions. The list below mentions browser engines used by a few common browsers:

1. Google Chrome and Opera v.15+: **Blink**
2. Internet Explorer: **Trident**
3. Mozilla Firefox: **Gecko**

## Parsers (HTML, CSS, etc)

The browser parses HTML into a DOM tree. HTML parsing involves tokenization and tree construction. HTML tokens include start and end tags, as well as attribute names and values. If the document is well-formed, parsing it is straightforward and faster.

The CSS parser takes the bytes and converts them into characters, then tokens, then nodes and finally they are linked into the CSSOM. The browser does something called selector matching which means that each set of styles will be matched against all nodes (elements) on the page.

## Script Processors

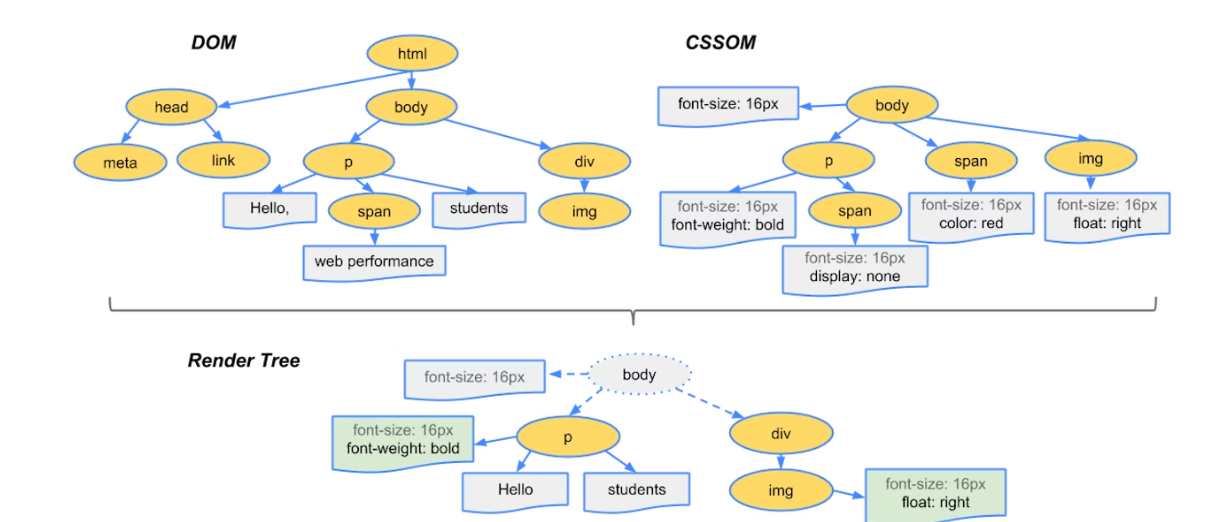
The script processor executes Javascript code to process an event. **Every time the browser encounters a script element, it will pause rendering (displaying) the document.** **It will then load the JavaScript file defined in the src attribute**.

## Order of script processing

In order to assess the consequences of any such decision, it helps to understand how browsers work: When the browser processes an HTML document, it does so from top to bottom. Upon encountering a <script> tag, it halts (“blocks”) further processing in order to download the referenced script file. Only after that download has completed and the respective JavaScript code has been processed, HTML processing continues.

## Tree Construction, Layout and Painting

To display a page on the screen, initially, 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.



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.**

The "Layout" event captures the render tree construction, position, and size calculation in the Timeline.

When layout is complete, the browser issues "Paint Setup" and "Paint" events, which convert the render tree to pixels on the screen.