Assignment : 1

When a user enters an URL in the browser, how does the browser fetch the desired result? Explain this with the below in mind and Demonstrate this by drawing a diagram for the same.

What is the main functionality of the browser?

High Level Components of a browser.

Rendering engine and its use

Parsers (HTML, CSS, etc)

Script Processors

Tree construction

Order of script processing

Layout and Painting

Ans. Computers connected to the web are called **clients** and **servers**.

Request

Response

CLIENT SERVER

* Clients are the typical web user's internet-connected devices (eg. Laptop, computers, mobiles) and web-accessing software available on those devices (usually a web browser like Chrome).
* Servers are computers that store web pages, 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.

**When you type a URL into a browser:-**

1. The browser goes to the DNS server, and finds the real address of the server that the website lives on.
2. The browser sends an HTTP request message to the server, asking it to send a copy of the website to the client. This message, and all other data sent between the client and the server, is sent across your internet connection using TCP/IP (Internet Protocol).
3. If the server approves the client's request, the server sends the client a " 200 OK " status, which means "Of course you can look at that website Here it is", and then starts sending the website's files to the browser as a series of small chunks called data packets.
4. The browser assembles the small chunks into a complete web page and displays it to you.

That’s how a browser show the result to you.

The major difference between both is that the **URL is a complete address**. URL tells about the method through which information should exchange, the path after reaching that website. Whereas the **domain name is part of a URL**

Or we can say that DNS is the address of a Website. The website is like the house and URL is the address.

BROWSER pestotech.com DNS 199.222.244.0

RDDDJA

1.FUNCTIONALITY OF A BROWSER :-

The main functions browser is to fetch or retrieve informative resources from World Wide Web to the client/ user on demand, translate those files received from web server and display those content to the user and allow the client /user to access all other relevant resources & When the user inputs any URL in the web browser, the user is navigated to that website by the browser quickly.

When user type any URL, for example <https://pestotech.in>, the prefix of the URL decide how to retrieve it. The URL prefixes that the web browser is not able to handle directly is sent to related application.

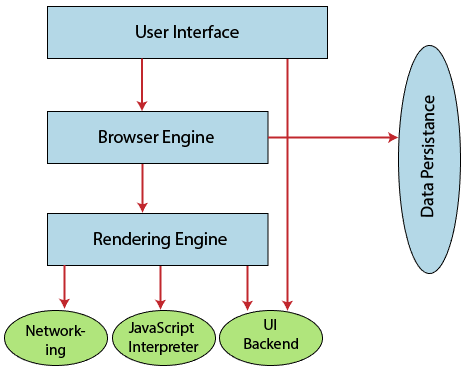
Plugins are available on web browser that supports flash content and java applets to run smoothly in any device.

Web browser allow users to interact with web pages and other dynamic contents via hyperlinks that provides navigation facility i.e. to go to different locations by clicking on links that makes internet surfing easy.

The main functions of a browser are:

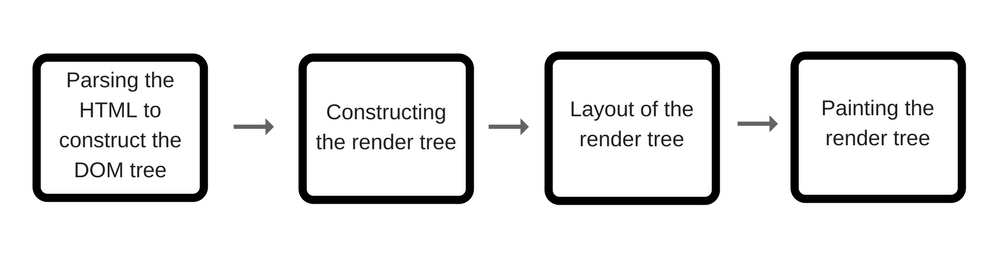
* Displaying and printing Web contents on the Internet or intranet.
* Displaying, printing, and saving a file on the Internet or intranet.
* Uploading scanned original data.

1. High Level Components of a browser:-
2. **User Interface:** The user interface is an area where the user can use several options like address bar, back and forward button, menu, bookmarking, and many other options to interact with the browser.
3. **Browser Engine:** It connects the UI (User Interface) and the rendering engine as a bridge. It queries and manipulates the rendering engine based on inputs from several user interfaces.
4. **Rendering Engine:** It is responsible for displaying the requested content on the browser screen. It translates the HTML, XML files, and images, which are formatted by using the CSS. It generates the layout of the content and displays it on the browser screen. Although it can also display the other types of content by using different types of plugins or extensions. such as:
   * Internet Explorer uses **Trident**
   * Chrome & Opera 15+ use **Blink**
   * Chrome (iPhone) & Safari use **Webkit**
   * Firefox & other Mozilla browsers use **Gecko**
5. **Networking:** It retrieves the URLs by using internet protocols like HTTP or FTP. It is responsible for maintaining all aspects of Internet communication and security. Furthermore, it may be used to cache a retrieved document to reduce network traffic.
6. **JavaScript Interpreter:** As the name suggests, JavaScript Interpreter translates and executes the JavaScript code, which is included in a website. The translated results are sent to the rendering engine to display results on the device screen.
7. **UI Backend:** It is used to draw basic combo boxes and Windows (widgets). It specifies a generic interface, which is not platform-specific.
8. **Data Storage:** The data storage is a persistence layer that is used by the browser to store all sorts of information locally, like cookies. A browser also supports different storage mechanisms such as IndexedDB, WebSQL, localStorage, and FileSystem. It is a database stored on the local drive of your computer where the browser is installed. It handles user data like cache, bookmarks, cookies, and preferences.



3.Rendering engine and its use:-

Once a user requests a particular document, the rendering engine starts fetching the content of the requested document. This is done via the networking layer. The rendering engine starts receiving the content of that specific document in chunks of 8 KBs from the networking layer. After this, the basic flow of the rendering engine begins.

The four basic steps include: 

1. The requested HTML page is parsed in chunks, including the external CSS files and in style elements, by the rendering engine. The HTML elements are then converted into DOM nodes to form a **“content tree” or “DOM tree.”**
2. Simultaneously, the browser also creates a **render tree.**This tree includes both the styling information as well as the visual instructions that define the order in which the elements will be displayed. The render tree ensures that the content is displayed in the desired order.
3. Further, the render tree goes through the **layout process.** When a render tree is created, the position or size values are not assigned. The entire process of calculating values for evaluating the desired position is called a layout process. In this process, every node is assigned the exact coordinates. This ensures that every node appears at an accurate position on the screen.
4. The final step is to paint the screen, wherein the render tree is traversed, and the renderer’s **paint()** method is invoked, which paints each node on the screen using the UI backend layer.

As discussed earlier, **every browser has its own unique rendering engine.**So naturally, every browser has its own way of interpreting web pages on a user’s screen. Here’s where a challenge arises for web developers regarding the cross-browser compatibility of their website.

This is where cross-browser testing comes into the picture.

[Cross browser testing](https://www.browserstack.com/live) is a quality assurance method used to verify the consistency of web applications in terms of functionality and design across multiple browsers. These tests enable QA teams to explore any issues by conducting a [responsive test](https://www.browserstack.com/responsive), that may occur when their website is accessed via different browsers or browser versions.

Organizations and teams can leverage online platforms like Browser Stack for performing [cross browser compatibility testing](https://www.browserstack.com/guide/cross-browser-compatibility-testing-beyond-chrome) across a wide range of real browsers and devices. For example, one can perform [Edge Testing](https://www.browserstack.com/test-on-microsoft-edge-browser) on Windows 10 to ensure that their web-app performs flawlessly on Microsoft Edge.

By understanding how a rendering engine works, web developers can gain greater insight into how websites operate. Consequently, they can develop, design and deploy content more efficiently. If one comprehends the nuance of how web content is displayed on a user’s screen by various browsers, one is simply more equipped to create content that is compatible with multiple browsers.

4.Parsers:-

HTMLCSS is a lightweight HTML/CSS parser written in C that allows applications to prepare a HTML document for rendering or conversion. HTMLCSS is extremely portable and only requires a C99 compiler like GCC, Clang, Visual C, etc. HTMLCSS is also extremely memory efficient, utilizing a shared string pool and smart CSS cache to minimize the size of a HTML document in memory.

Features include:

* HTML 5 markup parser
* CSS 3 stylesheet parser
* OFC/OFF/TTC/TTF font file parser (metadata only)
* GIF/JPG/PNG image file parser (metadata only)
* Functions to calculate CSS properties for a given node in a HTML document
* Functions to extract HTML “runs” consisting of CSS properties, content strings, and image references that can be rendered directly, including the :before and :after content from a stylesheet

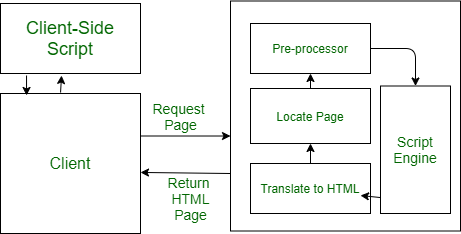
HTMLCSS does *not* support dynamic HTML content created using Javascript in a HTML document, as such content is typically used for interactive web pages while HTMLCSS is intended for use with static content.

5.Script Processors:-

The process of creating and embedding scripts in a web page is known as **web-scripting**. A script or a computer-script is a list of commands that are embedded in a web-page normally and are interpreted and executed by a certain program or scripting engine.

* Scripts may be written for a variety of purposes such as for automating processes on a local-computer or to generate web pages.
* The programming languages in which scripts are written are called scripting language, there are many scripting languages available today.
* Common scripting languages are [VBScript](https://www.geeksforgeeks.org/vbscript-introduction/), [JavaScript](https://www.geeksforgeeks.org/javascript-tutorial/), [ASP](https://www.geeksforgeeks.org/asp-full-form/), [PHP](https://www.geeksforgeeks.org/php/), [PERL](https://www.geeksforgeeks.org/perl-programming-language/), [JSP](https://www.geeksforgeeks.org/introduction-to-jsp/) etc.

**Types of Script :**  
Scripts are broadly of following two type :



**Client-Side Scripts :**

1. Client-side scripting is responsible for interaction within a web page. The client-side scripts are firstly downloaded at the client-end and then interpreted and executed by the browser (default browser of the system).
2. The client-side scripting is browser-dependent. i.e., the client-side browser must be scripting enables in order to run scripts
3. Client-side scripting is used when the client-side interaction is used. Some example uses of client-side scripting may be :
   * To get the data from user’s screen or browser.
   * For playing online games.
   * Customizing the display of page in browser without reloading or reopening the page.
4. Here are some popular client-side scripting languages VBScript, JavaScript, Hypertext Processor(PHP).

**Server-Side Scripts :**

1. Server-side scripting is responsible for the completion or carrying out a task at the server-end and then sending the result to the client-end.
2. In server-side script, it doesn’t matter which browser is being used at client-end, because the server does all the work.
3. Server-side scripting is mainly used when the information is sent to a server and to be processed at the server-end. Some sample uses of server-scripting can be :
   * Password Protection.
   * Browser Customization (sending information as per the requirements of client-end browser)
   * Form Processing
   * Building/Creating and displaying pages created from a database.
   * Dynamically editing changing or adding content to a web-page.
4. Here are some popular server-side scripting languages PHP, Perl, ASP (Active Server Pages), JSP ( Java Server Pages).

6.Tree construction:-

The CSSOM and DOM trees are combined into a render tree, which is then used to compute the layout of each visible element and serves as an input to the paint process that renders the pixels to screen. Optimizing each of these steps is critical to achieving optimal rendering performance.

In the previous section on constructing the object model, we built the DOM and the CSSOM trees based on the HTML and CSS input. However, both of these are independent objects that capture different aspects of the document: one describes the content, and the other describes the style rules that need to be applied to the document.

* The DOM and CSSOM trees are combined 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; for example, the span node---in the example above---is missing from the render tree because we have an explicit rule that sets the "display: none" property on it.
2. For each visible node, find the appropriate matching CSSOM rules and apply them.
3. Emit visible nodes with content and their computed styles.

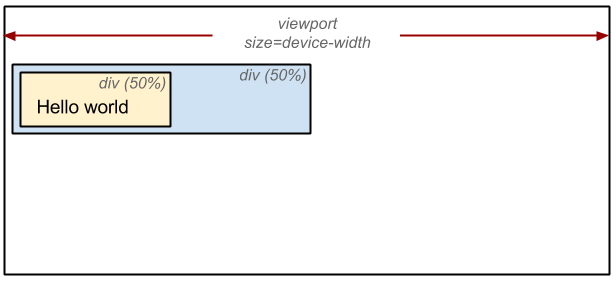
The final output is a render that contains both the content and style information of all the visible content on the screen**. With the render tree in place, we can proceed to the "layout" stage.**

Up to this point we've calculated which nodes should be visible and their computed styles, but we have not calculated their exact position and size within the viewport of the device---that's the "layout" stage, also known as "reflow."

To figure out the exact size and position of each object on the page, the browser begins at the root of the render tree and traverses it.

!DOCTYPE html>  
<html>  
 <head>  
 <meta name="viewport" content="width=device-width,initial-scale=1" />  
 <title>Critial Path: Hello world!</title>  
 </head>  
 <body>  
 <div style="width: 50%">  
 <div style="width: 50%">Hello world!</div>  
 </div>  
 </body>  
</html>

The body of the above page contains two nested div's: the first (parent) div sets the display size of the node to 50% of the viewport width, and the second div---contained by the parent---sets its width to be 50% of its parent; that is, 25% of the viewport width.



The output of the layout process is a "box model," which precisely captures the exact position and size of each element within the viewport: all of the relative measurements are converted to absolute pixels on the screen.

Finally, now that we know which nodes are visible, and their computed styles and geometry, we can pass this information to the final stage, which converts each node in the render tree to actual pixels on the screen. This step is often referred to as "painting" or "rasterizing."

This can take some time because the browser has to do quite a bit of work. However, Chrome DevTools can provide some insight into all three of the stages described above. 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.

The time required to perform render tree construction, layout and paint varies based on the size of the document, the applied styles, and the device it is running on: the larger the document, the more work the browser has; the more complicated the styles, the more time taken for painting also (for example, a solid color is "cheap" to paint, while a drop shadow is "expensive" to compute and render).

Here's a quick recap of the browser's steps:

1. Process HTML markup and build the DOM tree.
2. Process CSS markup and build the CSSOM tree.
3. Combine the DOM and CSSOM into a render tree.
4. Run layout on the render tree to compute geometry of each node.
5. Paint the individual nodes to the screen.

7.Order of script processing:-

The order in which commands to download external Cascading Style Sheets (CSS) files and JavaScript files are placed in an HTML page can affect how quickly the page is rendered and even whether the page is rendered at all.

When a CSS file is loaded before a JavaScript file, the page can begin rendering and the file downloads can happen in parallel which speeds up the rendering time.

If the order is reversed, the files are downloaded sequentially and the JavaScript file must finish loading completely before any other file can be downloaded. This means that the page cannot begin rendering until the JavaScript download is complete and the CSS file can begin loading.

This Best Practice Deep Dive looks at the issues involved with the order in which files are downloaded, tells you how AT&T Video Optimizer can help identify when certain files are loaded in an inefficient order, and provides recommendations for downloading style sheets and scripts.

## Background:

As a developer building HTML 5 style apps or webpages, you can control the order that files are requested by your mobile app. It is important to consider the order you request the files in, because some ways are faster than others.

## The Issue:

Script files (such as JavaScript) are loaded sequentially unless asynchronous downloading is specified. Sequential (synchronous) download means that the entire script file must be downloaded before any other files can be downloaded and before any other instructions on the page, such as inline code or HTML, can be executed.

Another issue that can occur when a JavaScript file is loaded before a CSS file, is that any JavaScript code which relies on properties set in that CSS file can't be executed until both files have finished loading.

## Best Practice Recommendation:

The Best Practice Recommendation is to download CSS before script files, and to move SCRIPT tags to the bottom of the page whenever possible.

8.Layout and Painting:-

### Layout:-

Once the render tree is built, layout becomes possible. Layout is dependent on the size of screen. The layout step determines where and how the elements are positioned on the page, determining the width and height of each element, and where they are in relation to each other.

What is the width of an element? Block level elements, by definition, have a default width of 100% of the width of their parent. An element with a width of 50%, will be half of the width of its parent. Unless otherwise defined, the body has a width of 100%, meaning it will be 100% of the width of the viewport. This width of the device impacts layout.

The viewport meta tag defines the width of the layout viewport, impacting the layout. Without it, the browser uses the default viewport width, which on by-default full screen browsers is generally 960px. On by-default full screen browsers, like your phone's browser, by setting

<meta name="viewport" content="width=device-width">

the width will be the width of the device instead of the default viewport width. The device-width changes when a user rotates their phone between landscape and portrait mode. Layout happens every time a device is rotated or browser is otherwise resized.

Layout performance is impacted by the DOM -- the greater the number of nodes, the longer layout takes. Layout can become a bottleneck, leading to jank if required during scrolling or other animations. While a 20ms delay on load or orientation change may be fine, it will lead to jank on animation or scroll. Any time the render tree is modified, such as by added nodes, altered content, or updated box model styles on a node, layout occurs.

To reduce the frequency and duration of layout events, batch updates and avoid animating box model properties.

### Paint:-

The last step is painting the pixels to the screen. Once the render tree is created and layout occurs, the pixels can be painted to the screen. On load, the entire screen is painted. After that, only impacted areas of the screen will be repainted, as browsers are optimized to repaint the minimum area required. Paint time depends on what kind of updates are being applied to the render tree. While painting is a very fast process, and therefore likely not the most impactful place to focus on in improving performance, it is important to remember to allow for both layout and re-paint times when measuring how long an animation frame may take. The styles applied to each node increase the paint time, but removing style that increases the paint by 0.001ms may not give you the biggest bang for your optimization buck. Remember to measure first. Then you can determine whether it should be an optimization priority.