**1st Week 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.

a. What is the main functionality of the browser?

b. High Level Components of a browser.

c. Rendering engine and its use.

d. Parsers (HTML, CSS, etc)

e .Script Processors

f .Tree construction

g .Order of script processing

h .Layout and Painting

Answer- When a user enters an URL in the browser URL (**Uniform Resource Locator)**

as the same suggests, it has the location of the resources which we want

to access. It is an **address of the place** where we want to go to interact with or

find information.

The browser checks the cache for a DNS record to find the corresponding IP

address of URL.

**DNS(Domain Name System)** is a database that maintains the name of the website

(URL) and the particular IP address it links to. Every single URL on the internet has

a unique IP address assigned to it. The IP address belongs to the computer which

hosts the server of the website we are requesting to access.

If the requested URL is not in the cache, ISP’s DNS server initiates a DNS query

to find the IP address of the server that hosts that URL.

The browser initiates a TCP connection with the server. Once the browser receives

the correct IP address, it will build a connection with the server that matches the IP

address to transfer information. Browsers use internet protocols to build such

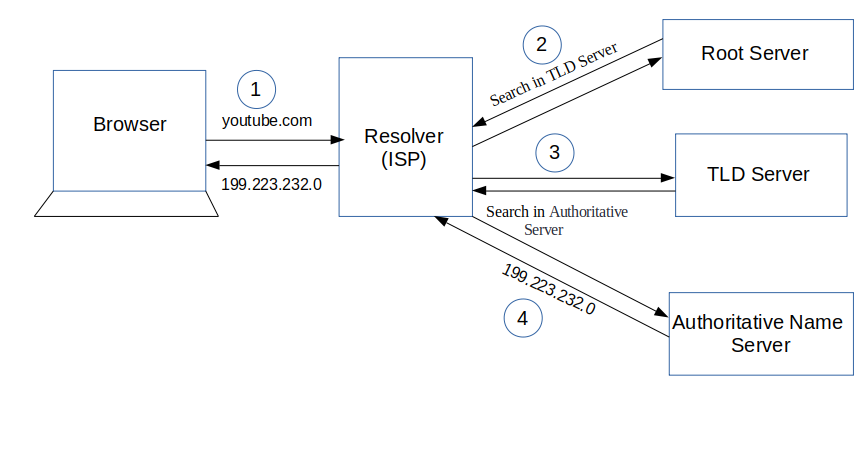
connections. There are several different internet protocols that can be used, but

TCP is the most common protocol used for many types of HTTP requests.

The browser sends an HTTP request to the web server. Once the TCP connection is established, it is time to start transferring data. The server handles the request and sends backan HTTP response. The browser then displays the HTML content.

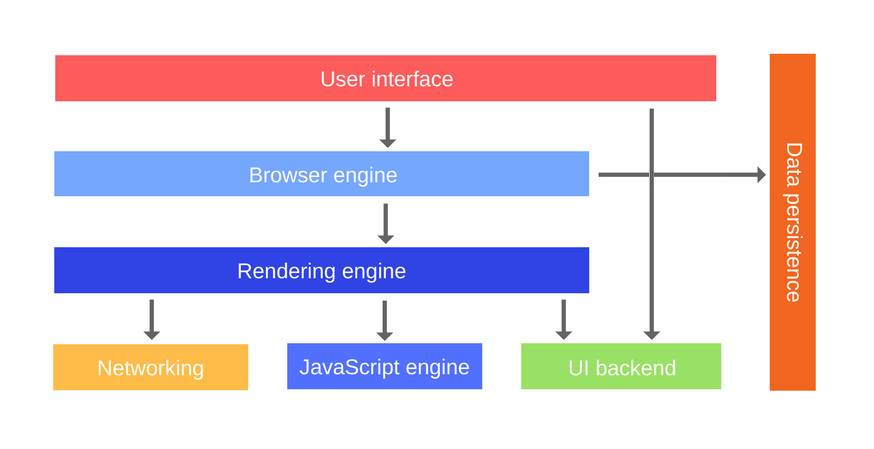
1. the main functionalities of the browser are -

* The main function is to retrieve information from the World Wide Web and making it available for users
* Visiting any website can be done using a web browser. When a URL is entered in a browser, the web server takes us to that website
* To run Java applets and flash content, plugins are available on the web browser
* It makes Internet surfing easy as once we reach a website we can easily check the hyperlinks and get more and more useful data online
* Browsers user internal cache which gets stored and the user can open the same webpage time and again without losing extra data
* Multiple web pages can be opened at the same time on a web browser
* Options like back, forward, reload, stop reload, home, etc. are available on these web browsers, which make using them easy and convenient



1. High Level Components of a browser are –

* **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.
* **The browser engine**: marshals actions between the UI and the rendering engine.
* **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.
* **Networking**: for network calls such as HTTP requests, using different implementations for different platform behind a platform-independent interface.
* **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.
* **JavaScript interpreter**. Used to parse and execute JavaScript code.
* **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, IndexedDB, WebSQL and File System.

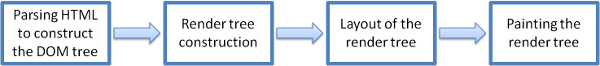


1. Rendering engine and its use –

The responsibility of the rendering engine, that is to display the requested contents on the browser screen.

By default the rendering engine can display HTML and XML documents and images. It can display other types of data via plug-ins or extension; for example, displaying PDF documents using a PDF viewer plug-in.

The main flow of Rendering engine



The rendering engine will start getting the contents of the requested document from the networking layer. This will usually be done in 8kB chunks.

The rendering engine will start parsing the HTML document and convert elements to DOM nodes in a tree called the "content tree". The engine will parse the style data, both in external CSS files and in style elements. Styling information together with visual instructions in the HTML will be used to create another tree: the render tree.

The render tree contains rectangles with visual attributes like color and dimensions. The rectangles are in the right order to be displayed on the screen.

After the construction of the render tree it goes through a "layout" process. This means giving each node the exact coordinates where it should appear on the screen. The next stage is painting–the render tree will be traversed and each node will be painted using the UI backend layer.

**d)** Parsers (HTML, CSS, etc)

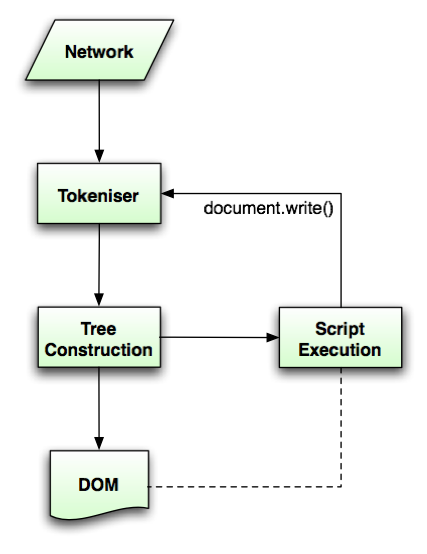
Parsing means analyzing and converting a program into an internal format that a runtime environment can actually run.

The browser parses HTML into a DOM tree. HTML parsing involves tokenization and tree construction. HTML token include start and end tags, as well as attribute names and values.

If document is well formed, parsing it straightforward and faster. The parser parses tokenized input into the document, building up the document tree.

**HTML Parser**

HTML Parser is a Java library used to parse HTML in either a linear or nested fashion. Primarily used for transformation or extraction, it features filters, visitors, custom tags and easy to use JavaBeans. It is a fast, robust and well tested package.



**CSS Parser**

The *CSS Parser* is implemented as a package of Java classes, that inputs Cascading Style Sheets source text and outputs a Document Object Model Level 2 Style.

**e)** Script Processors

The model of the web is synchronous. We expect scripts to be parsed and executed immediately when the parser reaches a <script> tag. The parsing of the document halts until the script has been executed. If the script is external then the resource must first be fetched from the network–this is also done synchronously, and parsing halts until the resource is fetched. This was the model for many years and is also specified in HTML4 and 5 specifications. We can add the "defer" attribute to a script, in which case it will not halt document parsing and will execute after the document is parsed. HTML5 adds an option to mark the script as asynchronous so it will be parsed and executed by a different thread.

**f)** **Render tree construction**

While the DOM tree is being constructed, the browser constructs another tree, the render tree. This tree is of visual elements in the order in which they will be displayed. It is the visual representation of the document. The purpose of this tree is to enable painting the contents in their correct order.

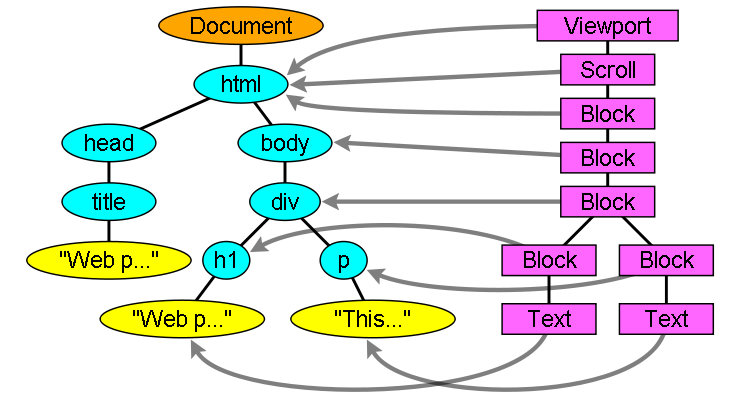
Firefox calls the elements in the render tree "frames". WebKit uses the term renderer or render object.  
A renderer knows how to lay out and paint itself and its children.

The renderers correspond to DOM elements, but the relation is not one to one. Non-visual DOM elements will not be inserted in the render tree. An example is the "head" element. Also elements whose display value was assigned to "none" will not appear in the tree (whereas elements with "hidden" visibility will appear in the tree).

There are DOM elements which correspond to several visual objects. These are usually elements with complex structure that cannot be described by a single rectangle. For example, the "select" element has three renderers: one for the display area, one for the drop down list box and one for the button. Also when text is broken into multiple lines because the width is not sufficient for one line, the new lines will be added as extra renderers.

Another example of multiple renderers is broken HTML. According to the CSS spec an inline element must contain either only block elements or only inline elements. In the case of mixed content, anonymous block renderers will be created to wrap the inline elements.

Some render objects correspond to a DOM node but not in the same place in the tree. Floats and absolutely positioned elements are out of flow, placed in a different part of the tree, and mapped to the real frame. A placeholder frame is where they should have been.



**g) Order of script processing**

There are basically three locations into which we can attach JavaScript:

* Directly into the head of the page
* Directly into the body of the page
* From an event handler/listener

It doesn't make any difference whether the JavaScript is within the web page itself or in external files linked to the page. It also doesn't matter whether the event handlers are hard-coded into the page or added by the JavaScript itself (except that they can't be triggered before they are added).

**h) Layout and Painting**

When the renderer is created and added to the tree, it does not have a position and size. Calculating these values is called layout or reflow.

HTML uses a flow based layout model, meaning that most of the time it is possible to compute the geometry in a single pass. Elements later ``in the flow'' typically do not affect the geometry of elements that are earlier ``in the flow'', so layout can proceed left-to-right, top-to-bottom through the document. There are exceptions: for example, HTML tables may require more than one pass.

The coordinate system is relative to the root frame. Top and left coordinates are used.

Layout is a recursive process. It begins at the root renderer, which corresponds to the <html> element of the HTML document. Layout continues recursively through some or all of the frame hierarchy, computing geometric information for each renderer that requires it.

The position of the root renderer is 0,0 and its dimensions are the viewport–the visible part of the browser window.

All renderers have a "layout" or "reflow" method, each renderer invokes the layout method of its children that need layout.

**The layout process**

The layout usually has the following pattern:

* Parent renderer determines its own width.
* Parent goes over children and:
  + Place the child renderer (sets its x and y).
  + Calls child layout if needed–they are dirty or we are in a global layout, or for some other reason–which calculates the child's height.
* Parent uses children's accumulative heights and the heights of margins and padding to set its own height–this will be used by the parent renderer's parent.
* Sets its dirty bit to false.

## Painting

In the painting stage, the render tree is traversed and the renderer's "paint()" method is called to display content on the screen. Painting uses the UI infrastructure component.

**The painting order**

CSS2 defines the order of the painting process. This is actually the order in which the elements are stacked in the stacking contexts. This order affects painting since the stacks are painted from back to front. The stacking order of a block renderer is:

* background color
* background image
* border
* children
* outline