**Exercise1.1:1.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.(2-3hours)**

**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 constructiong.**

**g.Order of script processingh.**

**h.Layout and Painting**

Answer:

When user enters an URL:

1. Browser checks cache for DNS entry to find the corresponding [IP address](https://www.geeksforgeeks.org/introduction-of-classful-ip-addressing/) of website.  
   It looks for following cache. If not found in one, then continues checking to the next until found.
   * Browser Cache
   * Operating Systems Cache
   * Router Cache
   * 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 initiates a [TCP (Transfer Control Protocol)](https://www.geeksforgeeks.org/tcp-and-udp-in-transport-layer/) connection with the server using synchronize(SYN) and acknowledge(ACK) messages.
4. Browser sends an [HTTP](https://www.geeksforgeeks.org/http-non-persistent-persistent-connection/) 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](https://www.geeksforgeeks.org/xml-basics/) and HTML.
6. Server sends out an HTTP response along with the status of response.
7. Browser displays [HTML](https://www.geeksforgeeks.org/html-tutorials/) content

**Main functionality of a web browser:**

A browser is a software application used to locate, retrieve and display content on the World Wide Web, including Web pages, images, video and other files.

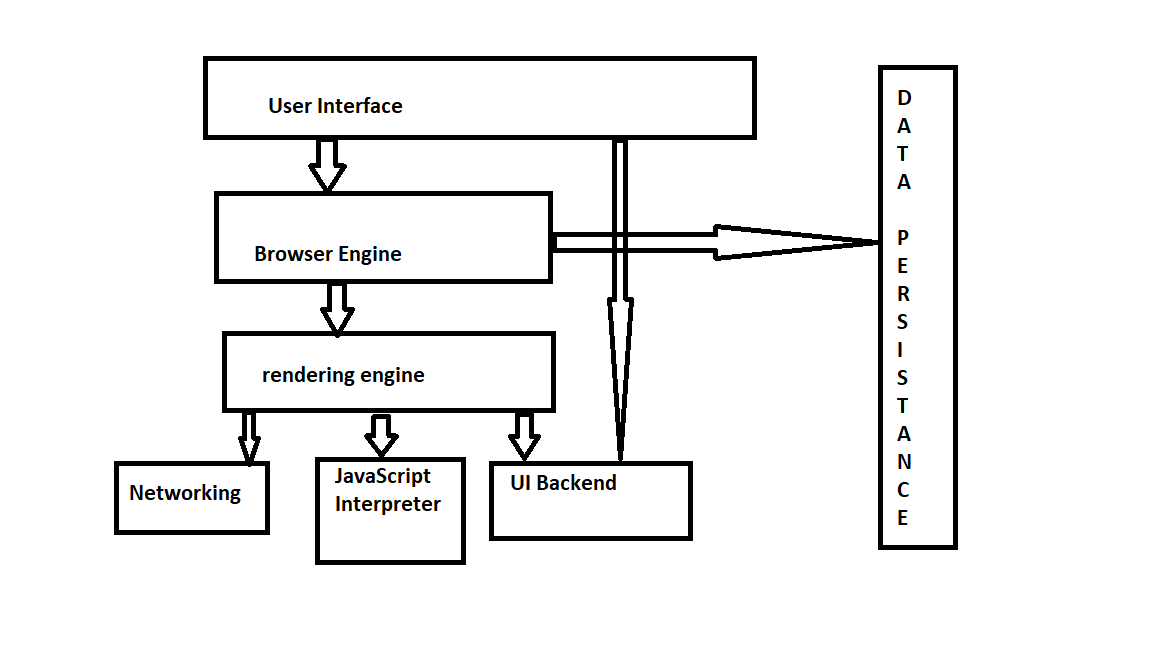
Client/server model, the browser is the client run on a computer that contacts the Web server and requests information. The Web server sends the information back to the Web browser which displays the results on the computer or other Internet-enabled device that supports a browser.

Browsers are fully-functional software suites that can interpret and display HTML Web pages, applications, JavaScript, AJAX and other content hosted on Web servers. Many browsers offer plug-ins which extend the capabilities of the software so it can display multimedia information (including sound and video), or the browser can be used to perform tasks such as videoconferencing, to design web pages or add anti-phishing filters and other security features to the browser.

A browser is a group of structured codes which together performs a series of tasks to display a web page on the screen. According to the tasks they perform, these codes are made as different components.

**High Level components of a browser:**

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:**

The responsibility of the rendering engine is display of 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.

**Parser:**

**HTML Parser:**

When the parser is created the Document object is created. During the tree construction stage the DOM tree with the Document in its root will be modified and elements will be added to it. Each node emitted by the tokenizer will be processed by the tree constructor. For each token the specification defines which DOM element is relevant to it and will be created for this token. The element is added to the DOM tree, and also the stack of open elements. This stack is used to correct nesting mismatches and unclosed tags. The algorithm is also described as a state machine. The states are called "insertion modes".

Considering the tree construction for the code below:

<html>

<body>

Hello!!

</body>

</html>

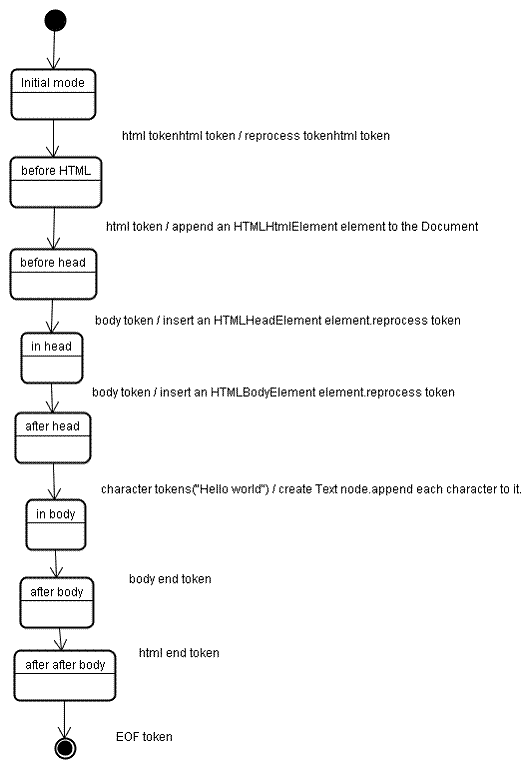
The input to the tree construction stage is a sequence of tokens from the tokenization stage. The first mode is the **"initial mode"**. Receiving the "html" token will cause a move to the **"before html"** mode and a reprocessing of the token in that mode. This will cause creation of the HTMLHtmlElement element, which will be appended to the root Document object.

The state will be changed to **"before head"**. The "body" token is then received. An HTMLHeadElement will be created implicitly although we don't have a "head" token and it will be added to the tree.

We now move to the **"in head"** mode and then to **"after head"**. The body token is reprocessed, an HTMLBodyElement is created and inserted and the mode is transferred to **"in body"**.

The character tokens of the "Hello world" string are now received. The first one will cause creation and insertion of a "Text" node and the other characters will be appended to that node.

The receiving of the body end token will cause a transfer to **"after body"** mode. We will now receive the html end tag which will move us to **"after after body"** mode. Receiving the end of file token will end the parsing.



**CSS Parser:**

CSS is a context free grammar and can be parsed using the types of parsers described in the introduction. In fact [the CSS specification defines CSS lexical and syntax grammar](http://www.w3.org/TR/CSS2/grammar.html).

Example:

div.error, a.error {

color:red;

font-weight:bold;

}

div.error and a.error are selectors. The part inside the curly braces contains the rules that are applied by this ruleset. This structure is defined formally in this definition:

ruleset

: selector [ ',' S\* selector ]\*

'{' S\* declaration [ ';' S\* declaration ]\* '}' S\*

;

This means a ruleset is a selector or optionally a number of selectors separated by a comma and spaces (S stands for white space). A ruleset contains curly braces and inside them a declaration or optionally a number of declarations separated by a semicolon. "declaration" and "selector" will be defined in the following BNF definitions.

**Order of the script processing:**

1. Download scripts, blocking other resource from downloading
2. Parse the scripts
3. Execute the scripts

**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.

A renderer knows how to layout and paint itself and its children.

WebKit's RenderObject class, the base class of the renderers, has the following definition:

class RenderObject{

virtual void layout();

virtual void paint(PaintInfo);

virtual void rect repaintRect();

Node\* node; //the DOM node

RenderStyle\* style; // the computed style

RenderLayer\* containgLayer; //the containing z-index layer

}

Each renderer represents a rectangular area usually corresponding to a node's CSS box, as described by the CSS2 spec. It includes geometric information like width, height and position.

The box type is affected by the "display" value of the style attribute that is relevant to the node (see the style computation section). Here is WebKit code for deciding what type of renderer should be created for a DOM node, according to the display attribute:

RenderObject\* RenderObject::createObject(Node\* node, RenderStyle\* style)

{

Document\* doc = node->document();

RenderArena\* arena = doc->renderArena();

...

RenderObject\* o = 0;

switch (style->display()) {

case NONE:

break;

case INLINE:

o = new (arena) RenderInline(node);

break;

case BLOCK:

o = new (arena) RenderBlock(node);

break;

case INLINE\_BLOCK:

o = new (arena) RenderBlock(node);

break;

case LIST\_ITEM:

o = new (arena) RenderListItem(node);

break;

...

}

return o;

}

The element type is also considered: for example, form controls and tables have special frames.

In WebKit if an element wants to create a special renderer, it will override the createRenderer() method. The renderers point to style objects that contains non geometric information.

**Layout**

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.

**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

Firefox display list

Firefox goes over the render tree and builds a display list for the painted rectangular. It contains the renderers relevant for the rectangular, in the right painting order (backgrounds of the renderers, then borders etc). That way the tree needs to be traversed only once for a repaint instead of several times–painting all backgrounds, then all images, then all borders etc.

Firefox optimizes the process by not adding elements that will be hidden, like elements completely beneath other opaque elements.

WebKit rectangle storage

Before repainting, WebKit saves the old rectangle as a bitmap. It then paints only the delta between the new and old rectangles.

Dynamic changes

The browsers try to do the minimal possible actions in response to a change. So changes to an element's color will cause only repaint of the element. Changes to the element position will cause layout and repaint of the element, its children and possibly siblings. Adding a DOM node will cause layout and repaint of the node. Major changes, like increasing font size of the "html" element, will cause invalidation of caches, relayout and repaint of the entire tree.

The rendering engine's threads

The rendering engine is single threaded. Almost everything, except network operations, happens in a single thread. In Firefox and Safari this is the main thread of the browser. In Chrome it's the tab process main thread.

Network operations can be performed by several parallel threads. The number of parallel connections is limited (usually 2–6 connections).

Event loop

The browser main thread is an event loop. It's an infinite loop that keeps the process alive. It waits for events (like layout and paint events) and processes them. This is Firefox code for the main event loop:

while (!mExiting)

NS\_ProcessNextEvent(thread);