__/__/____

Exercise 1.1

The main function of the browser is to present the web resource you choose, by requesting it from the server & displaying it in browser's window.

Lets take an example : https://example.com

For us, its just a domain name but for computers these are IP address. ~~we convert the domain~~ Now websites are stored in servers accessible on in internet, everywhere to everyone. To access these websites we must first get access to the IP address of the server where example.com (all the files related to it) is stored. This happens using DNS servers.

The IP address can be either found in the browser cache, OS cache, Router ~~or~~ cache, ISP cache etc. If

we are ~~se~~ searching for that URI for the first time than a DNS lookup will be done. After all this the DNS server will find the IP address for the URL. After this a lot goes into the picture that we'll ~~look~~ go through after some time.

The high level components of a browser are!-

- The User Interface : this includes the address bar, back/ forward button, bookmarking menu, etc.

- The Browser engine : communicates b/w the UI & the rendering engine.

- The Rendering engine : Responsible for displaying requested content. It parses HTML & CSS and displays the ~~pas~~ parsed content on the screen.

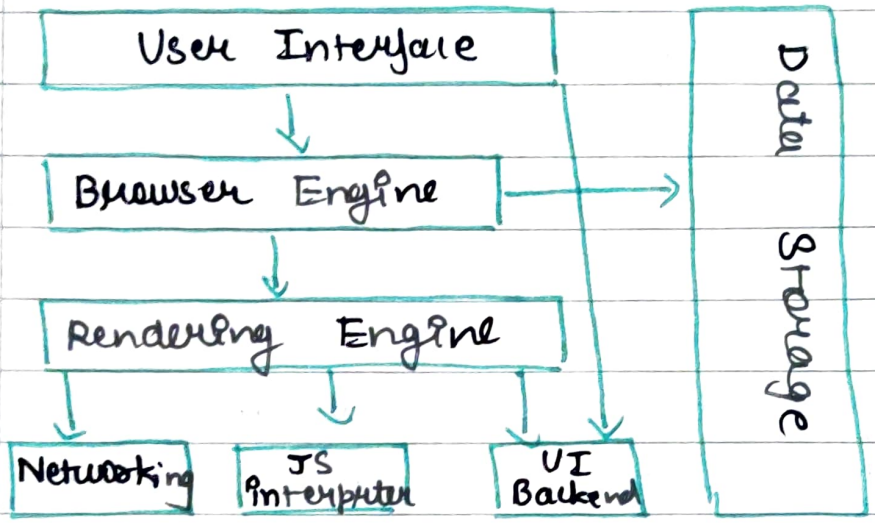
__/__/____

- Networking : For network calls such as HTTP requests, using different platform behind a platform independent surface.

- UI - backend : used for drawing basic widget like combo boxes & windows. ~~This~~

- Javascript Interpreter - Used to parse and execute JS code.

- Data Storage - This is a persistence layer. The browser may need to save all sorts of data locally, such as cookies. Browser also support storage mechanisms such as localStorage, IndexedDB, WEBSQL & FileSystem.



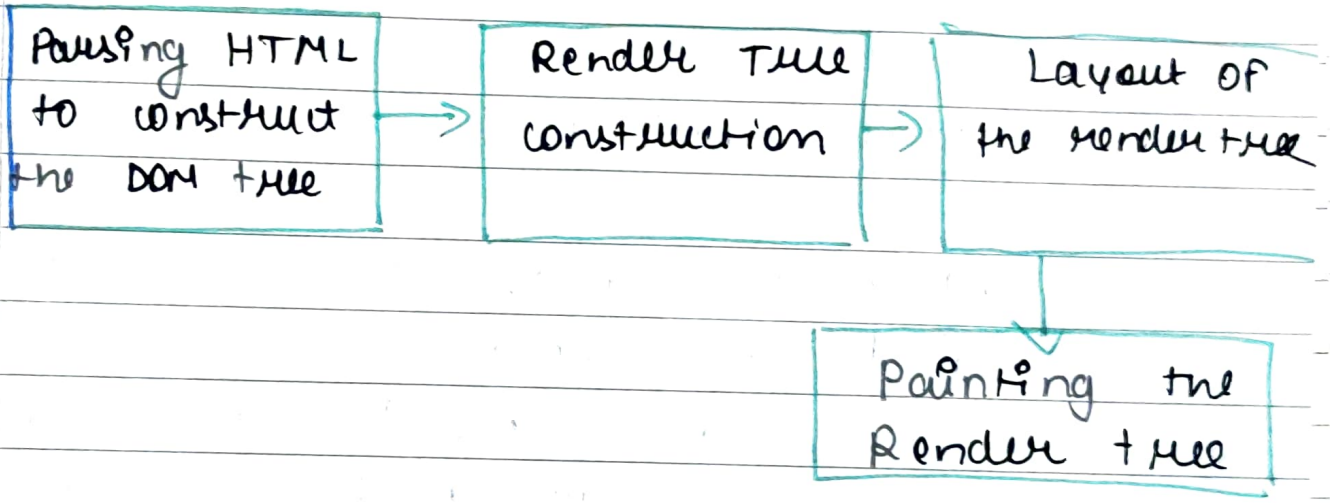
Rendering Engine

The responsibility of the rendering engine is displaying the requested contents on the browser screen.
By default it can display HTML & XML Documents.
Different Browsers uses different rendering engines: Safari uses WebKit, Chrome & Opera use Blink, a fork of WebKit.

The rendering engine will start getting the contents of the requested

documents from the networking layer.

Basic flow of Rendering engine:



The rendering engine will start parsing the HTML Document elements and create a tree called the DOM tree. Similary CSS is also parsed and CSSOM tree is constructed. Both of them combined together results in the Render Tree.

After the construction of the render tree it goes through a layout process. This means giving each node the exact co-ordinates where it

should appear on the screen. Now the render tree will be traversed and each node will be painted using UI backend layer.

HTML Parsing

After the request has been processed and a response (HTML Document) has been received by the browser the HTML Parsing is done by the Rendering Engine (Blink in case of Chrome)

After the HTML document is received, Parsing takes place.

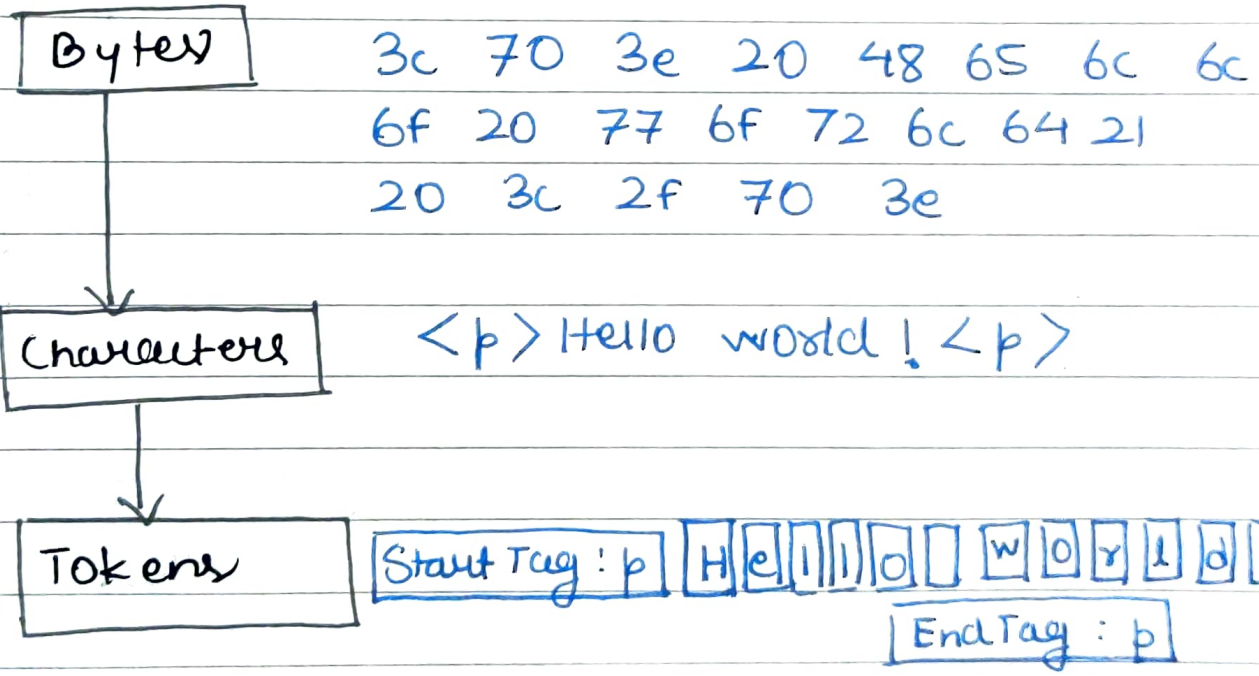
~~For~~

Parsing consists of Tokenization & Building the DOM Tree.

Tokenization is a lexical analysis and it converts some inputs into tokens. What results at the end of tokenization is a series of zero or more of the following tokens:

DOCTYPE, Start tag (<tag>), end tag (</tag>), self closing tag (<tag/>), attribute names, values, comments, characters, end-of-file or plain text content within a element.

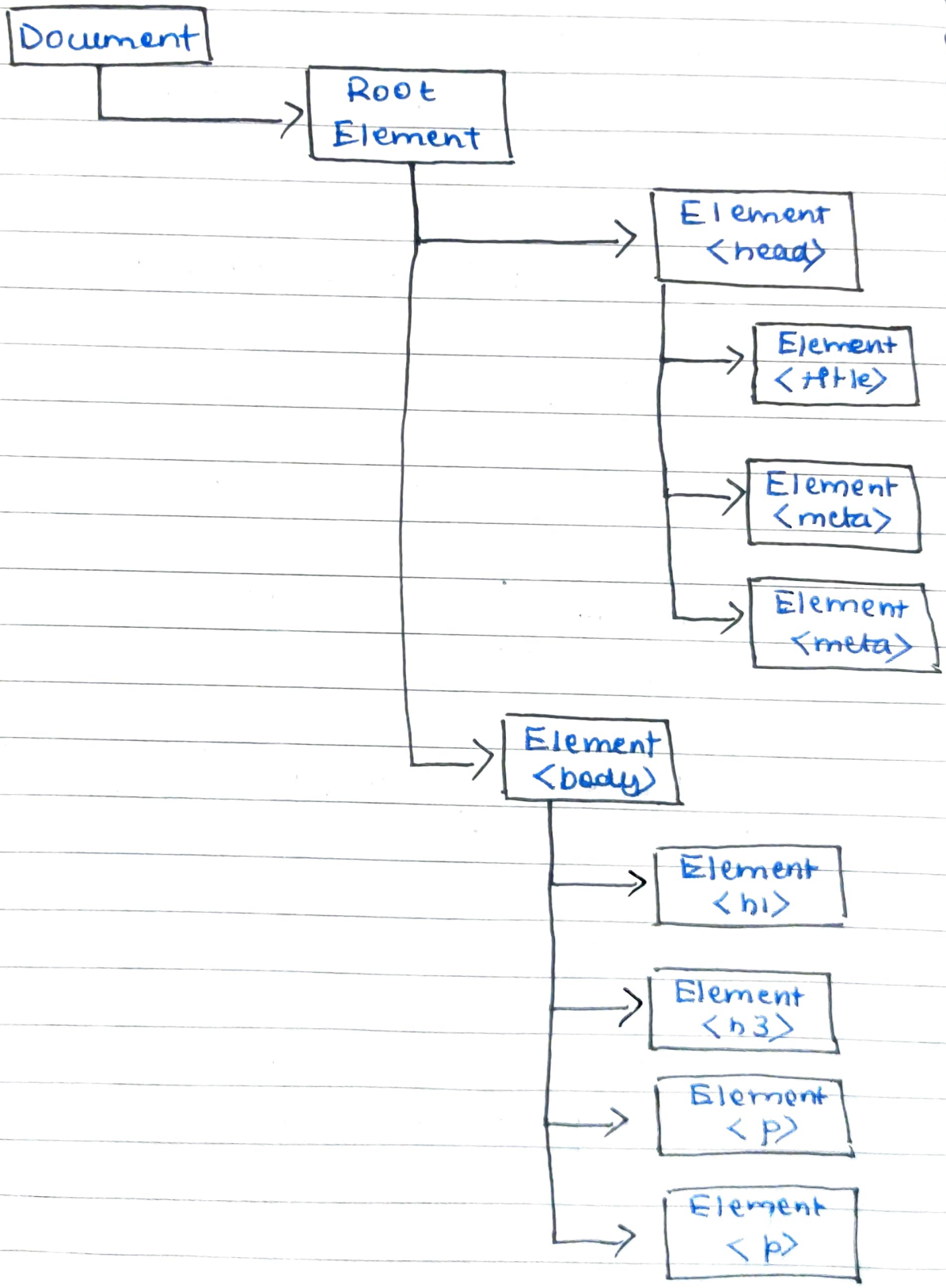
HTML 5 Parser Tokenization



Building the DOM

After the first token gets created, tree building starts. This is called the DOM tree.

Below is an example DOM tree.



The parser works line by line, from top to bottom. When the parser will encounter non-blocking resources, the browser will request those images from the server & continue parsing. On the other hand if it encounters blocking - resources (CSS stylesheets, JS files added in the <head> section of HTML or fonts added from a CDN, the parser will stop execution and download the blocking resources.

CSS Parsing

After the html has been parsed, its time to parse the CSS & and build the CSSOM tree. (CSS Object Model)

When a browser encounters a CSS stylesheet, be it external or embedded it needs to parse the text into something it can use for styling the layouts. Similar to HTML Tokenization

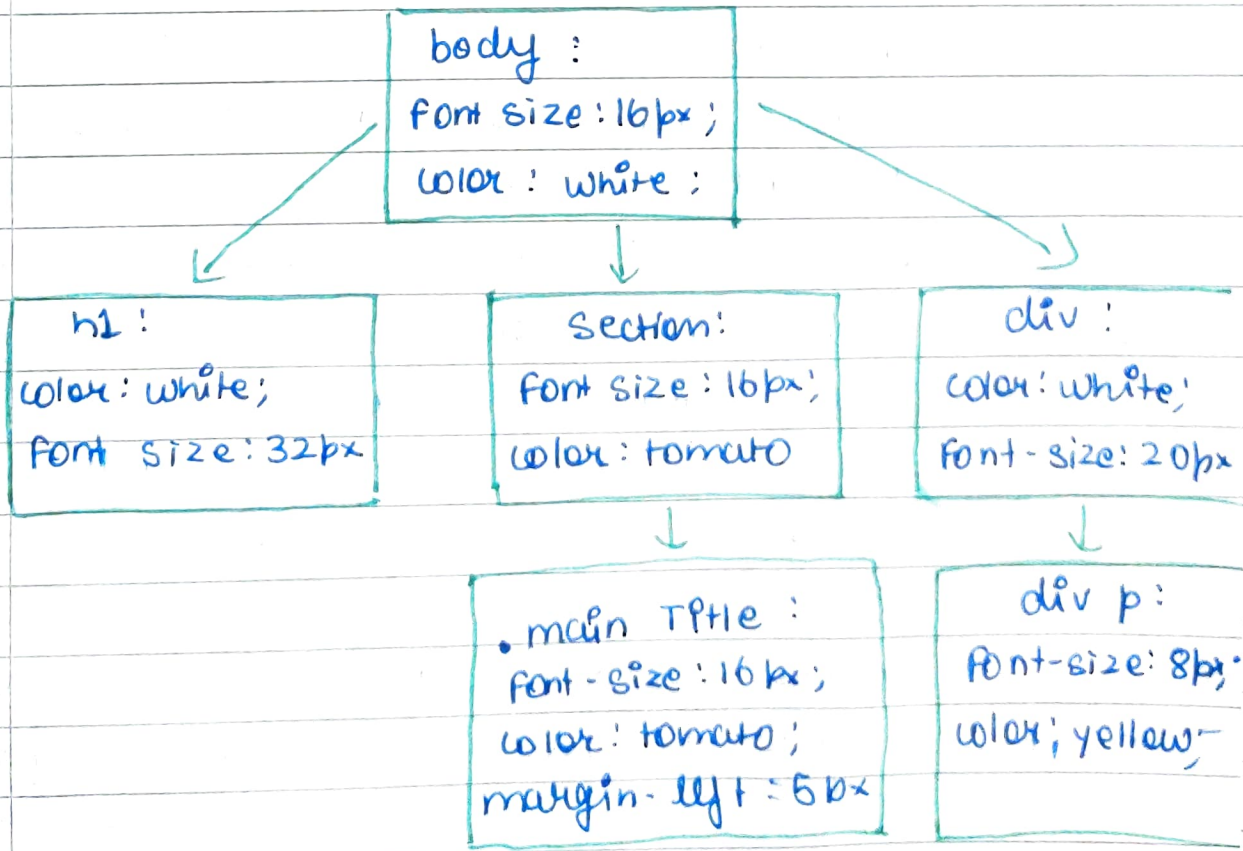
takes place in CSS parsing.

The browser starts with the most general rule applicable to a node (eg: if a node ~~tis~~ its the child of the body element, then all body ~~sy~~ styles are inherited by that node) and the recursively refines the computed styles by applying more specific rules.
For example

```
body {
  font-size : 16px;
  color : white;
}
h1 {
    font size : 32 px;
}
section {
   color : tomato;
}
```

```
section.mainTitle {
    margin-left: 5px
}
div {
    font-size: 20px;
}
div p {
    font-size: 8px;
    color: yellow;
}
```

CSSOM For this would be:



Since we can have multiple sources for our CSS and they can contain rules that apply to the same node, the browser must decide which rule will apply in the end.

SCRIPT PROCESSING

So, after we get the Javascript file from the server, the code is interpreted, compiled, parsed & executed. The computer can't understand Javascript code, only the browser can. The JS code needs to be translated into something the computer can work with and THIS IS the job of the Javascript browser engine.

Google uses V8 as its Javascript Engine.
When JS code enters the Javascript engine it gets parsed as a first

__/__/____

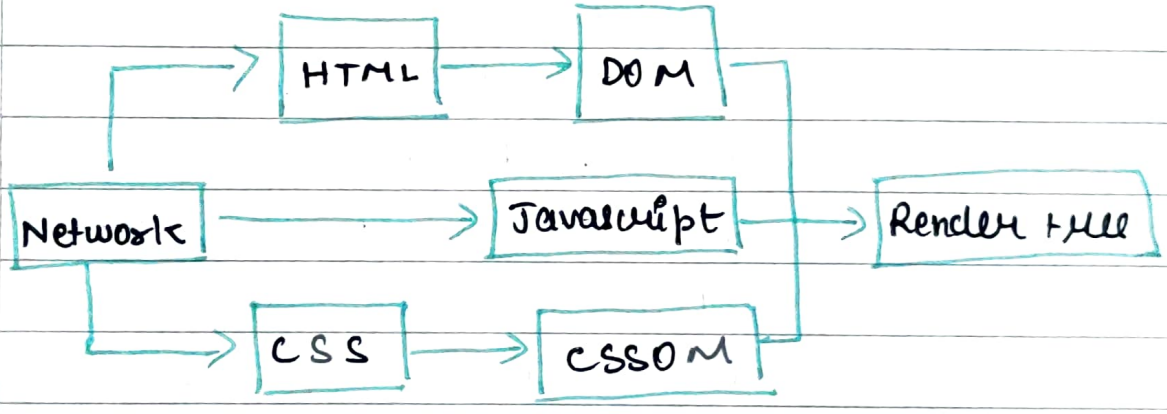
step. This means the code is read, and while this is happening, the code is transformed into a data structure called the Abstract Syntax Tree.

After the AST has been built, it gets translated into machine code & executed right away, since modern Javascript ~~used~~ uses JIT (Just in Time) compilation. ~~Thi~~ The execution of this code will be done by the Javascript engine, making use of something called the "call stack".

The trees built in the parsing phase (DOM, CSSOM) are combined into something called the render tree. This is used to compute the layout of all visible elements that will be painted to the screen in the end.

Combining The DOM with the CSSOM

After combining the DOM tree & the CSSOM tree, render tree is generated that contains all visible nodes with content & styles.



▲ The Render Tree.

The LayOut Stage

The render tree holds information on which nodes are displayed along with their computed styles, but not the dimensions or location of each node.

What ~~next~~ needs The browser starts the process at the root of the render tree & traverses it.

The layout step doesn't happen only once, but every time we change something in the DOM that affects the layout of the page even partially.

The Painting Stage

After the browser decides which nodes need to be visible and calculates their position in the viewport, it's time to paint them on the screen. Here the browser converts each box calculated in the layout phase to actual pixels on the screen.

Painting means the browser needs to draw every visual part of an element

__/__/____

to the screen, including text, colors, borders, shadows and replaced elements like buttons & images and its needs to be done super quickly.