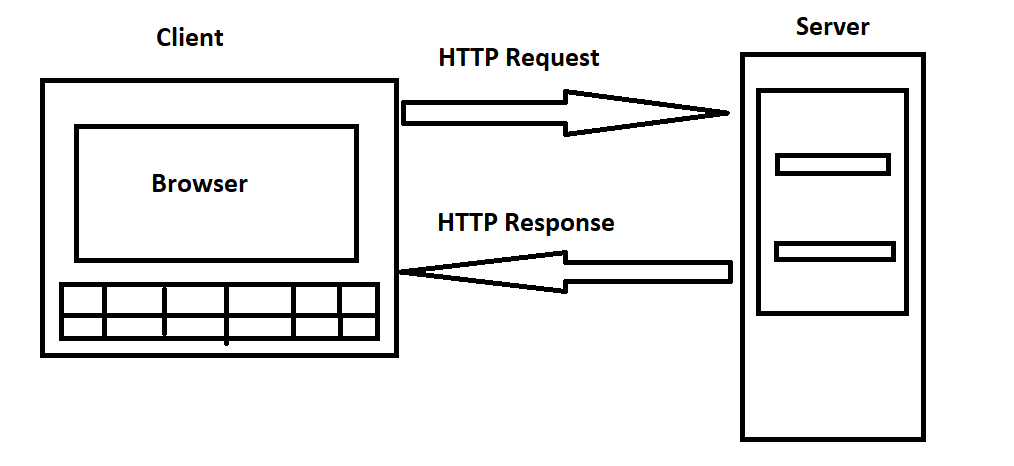
**Assignment 1**

**Exercise 1.1**

1. **What is the main functionality of the browser?**

**answer:**

****

**1.** It is used to display and print web content on internet.

**2.** It is used to saving file on internet.

**3.** the information transferred using HTTP protocol.

**4.** It is also used to provide internet services like emails.

**5.** Access and navigate web page.

**6.** Authentication & encryption function.

1. **High level components of browser?**

**answer:**

**Diagram

Description automatically generated**

**1. User Interface:** It defines the layout of elements which are available for user to interact in the browser. the elements like refresh, back & forward button. the bookmark & address bar.

2. **Browser Engine**: It is a unit that receives the input from UI and process that input to a command that Render machine will understand. It is middleware or middle unit between UI & Render machine.

3. **Rendering machine:** This unit handles rendering or displaying the requested content on window. This is most important operation. some of the famous rendering machines are blink & webkit.

4. **Networking:** This unit handles network related tasks like HTTP calls.

5. **JS Interpreter:** As the name suggests, it is used to analyse, parse and execute js code in the web page. examples: V8(C++), SpiderMonkey(C/C++).

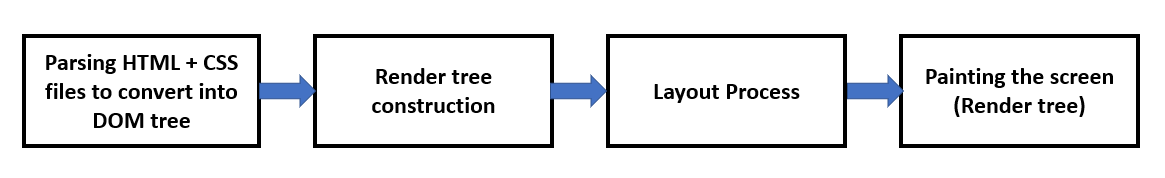
6. **UI Backend:** This unit is used for performing operation like select check box or input box. for this operation it uses UI method.

7. **Data Persistence:** Web browser needs to store various types of data like cache, cookies etc. for this browser uses data storage mechanisms WebSQL, IndexeDB etc.

1. **Rendering engines & its Uses?**

**answer:**

The rendering engine is used to fetch the content of the requested documents via the networking layer.



Uses:

* Draw text & images on the screen.
* display the web page.
* turning HTML content into web pages.

Basic Flow of engine:

1) The requested HTML page with CSS file is parsed by rendering machine. The HTML elements are then converted into DOM nodes to form "Content tree/DOM tree".

2) Then browser creates render tree which ensures that content is displayed in desired order. This render tree contains visual and styling information.

3) Further render tree goes through layout process. when render tree is created, the position & size values are not assigned. the entire process of calculating values for evaluating desired position is called layout process.

4) The final step is to paint each node of the screen using UI backend layer.

1. **Parsers (HTML, CSS etc.)**

**answer:**

Image 1:

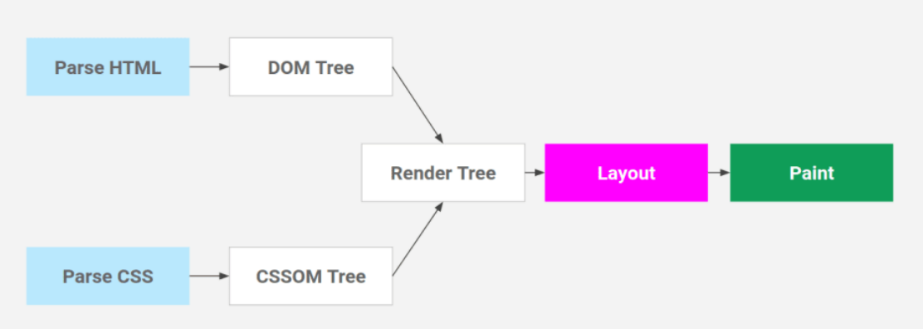
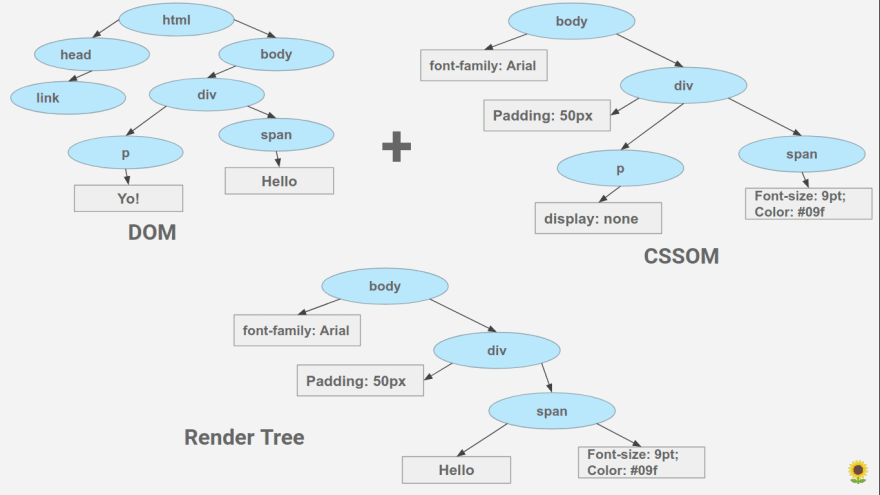


Image 2:



* HTML Parsing: HTML Text -> Tokenization -> DOM Tree
* CSS Parsing: CSS Text -> Tokenization -> CSSOM Tree

1. In the beginning HTML content is converted into tokens. this process of converting code into token is called tokenization. These tokens are easier to understand while parsing.

2. HTML parser understand what is start & end tag. which tag is it and what is inside of the tag.

3. HTML Tag sequence: [Start: HTML] [Start: HEAD]...[End: HEAD] [End: HTML] by this we get "Parse Tree" which is also called "DOM Tree".

4. DOM Tree is what we access when we do document.getElementId and document.querySelector in JavaScript.

5. Just like HTML, CSS also goes through tokenization & create something called CSSOM (CSS Object Model).

6. For rendering, this DOM & CSSOM are merged to form something called a Render Tree. Render tree has the information to paint elements on screen.

1. **Script Processors?**

**answer:**

Timeline

Description automatically generated with medium confidence

Two decades ago, the WWW was a static landscape, with very little interaction between user & information provided on screen. In such dark time, user would use the web for reading badly formatted article & research purpose. but after some years, lots of web advancement had happen which is also called as "Web 2.0". Now user can experience dynamic content. in order to get such content, developers often utilize 2 forms of interactive scripting - client side scripting server side scripting.

1. When User presses any keyboard key or any mouse response which modifies webpage is due to client side scripting.

2. For example, web developer can use hover function to make button change colour or image become transparent when user brought mouse over it.

3. Some of the client-side scripting languages are JavaScript, ActionScript.

4. Client-side scripting is used grab attention of user by adding sound, animation, text changes and other effects.

5. Unlike client-side scripting which is executed on user's browser, server side scripting is executed on website's server before it sent over to user's computer.

6. Server-side scripting creates web content on various pages, manage user sessions & control website workflow.

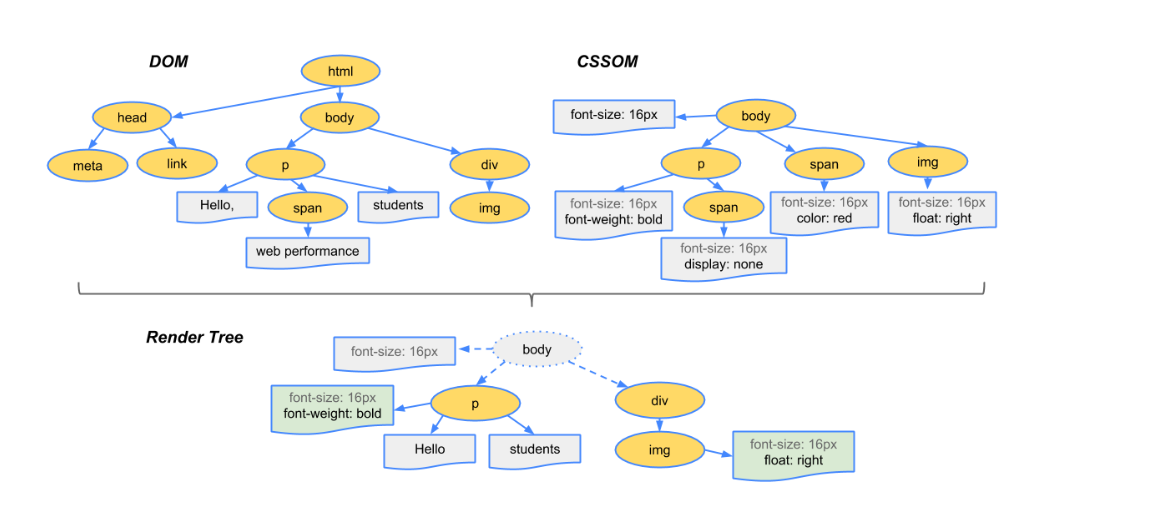
7. Server-side scripting is also used for security purpose like hiding source code. it is also used to respond to queries, access database and return result to a browser.

8. Example of server-side scripting is "Facebook". like we can create our own profile & also search for friends in their database.

9. This how by using client-side & server-side scripting, we can create more user friendly, attractive, intuitive, intelligent and engaging websites ever before.

1. **Tree Construction?**

**answer:**



1. Process HTML content and build the DOM tree similarly process CSS markup and build the CSSOM tree.

2. Merging of DOM and CSSOM trees form the render tree.

3. Render tree contains only the nodes required to render the tree.

4. Layout computes the exact size and position of each object.

5. Paint is the last step that takes in the final render tree and renders the pixels to the screen.

Browser follows below steps to construct tree:

1. Starting at the root of the DOM tree, transverse each visible node.

2. Some nodes are not visible in rendered output e.g. script tags and meta tags.

3. Some nodes are hidden via CSS e.g. span node because it has 'Display: None' property.

4. For each visible node find CSSOM rules & apply them.

5. print or display nodes with content and its styling.

1. **Order of script processing?**

**answer:**

In a web page, the order of script processing is important, as it can affect the behavior and functionality of the page. Generally, the order of script processing is as follows:

HTML parsing: The browser first parses the HTML code of the web page, creating a Document Object Model (DOM) tree that represents the structure of the page.

CSS parsing: The browser then parses the CSS code of the web page, creating a Cascading Style Sheets (CSS) object model that represents the styles to be applied to the page.

Script loading: The browser then loads any external JavaScript files referenced in the HTML code, as well as any inline scripts.

Script execution: Once all the scripts are loaded, the browser executes them in the order they appear in the HTML code, unless the scripts contain asynchronous behaviour. In the case of asynchronous scripts, the order of execution may not necessarily be the same as the order in the HTML code.

DOM manipulation: During script execution, the scripts can manipulate the DOM tree, adding, modifying, or removing nodes and elements.

Rendering: Finally, the browser uses the updated DOM tree and CSS object model to render the web page with the appropriate styles and behaviour.

It is important to note that the order of script processing can affect the behaviour of the web page. For example, if a script that manipulates the DOM tree is executed before the elements it is manipulating have been loaded, the script may not work as intended. It is also important to consider the performance implications of script processing, as loading and executing large or complex scripts can slow down the rendering of the web page.

The order of execution of scripts in a web page can affect how the page behaves and is rendered. Here are some key points to consider:

Scripts in the head section: Scripts included in the head section of an HTML document are typically loaded and executed before the content of the document is parsed and rendered. This means that if a script in the head section modifies the content of the page, the modified content may not be immediately visible to the user.

Scripts in the body section: Scripts included in the body section of an HTML document are typically loaded and executed after the content of the document is parsed and rendered. This means that if a script in the body section modifies the content of the page, the modified content will be immediately visible to the user.

External scripts: Scripts that are loaded from external sources, such as a separate JavaScript file or a script loaded from a CDN, may take longer to load, and execute than scripts included in the document itself. This can affect the perceived performance of the web page, especially if the external scripts are large or take a long time to load.

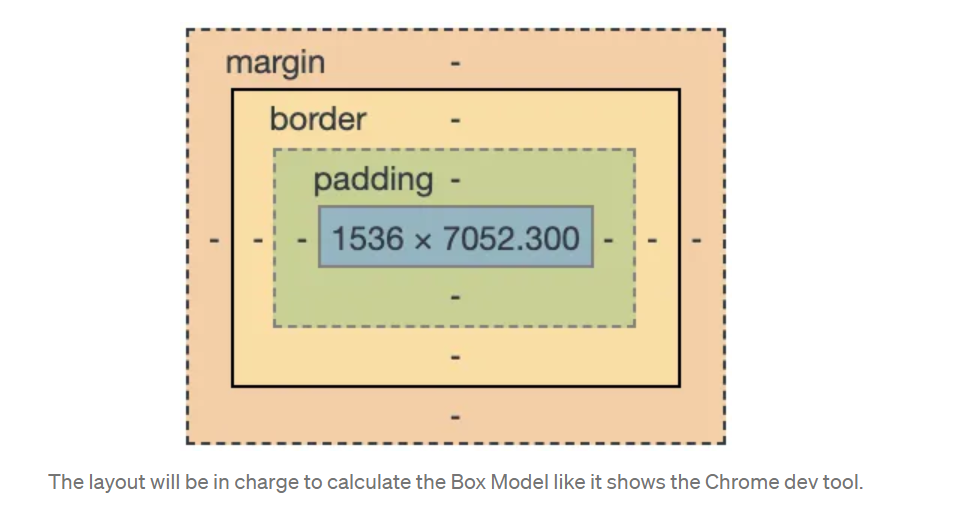
Asynchronous loading: Scripts can be loaded asynchronously using the async attribute on the script tag. This allows the browser to continue parsing and rendering the page while the script is being loaded and executed. However, asynchronous loading can also cause scripts to execute out of order, which can lead to unexpected behaviour.

Defer loading: Scripts can also be loaded using the defer attribute on the script tag. This delays the execution of the script until after the page has been parsed and rendered, but before the DOM Content Loaded event is fired. This can improve the perceived performance of the web page, especially for scripts that are not critical for the initial rendering of the page.

In general, it's a good practice to include scripts at the end of the body section, just before the closing tag. This can improve the perceived performance of the web page by allowing the content to be rendered first and by ensuring that the scripts don't block the initial rendering of the page. However, the specific order of script execution may depend on the requirements of the web page and the behaviour of the scripts themselves.

1. **Layout & painting?**

**answer:**

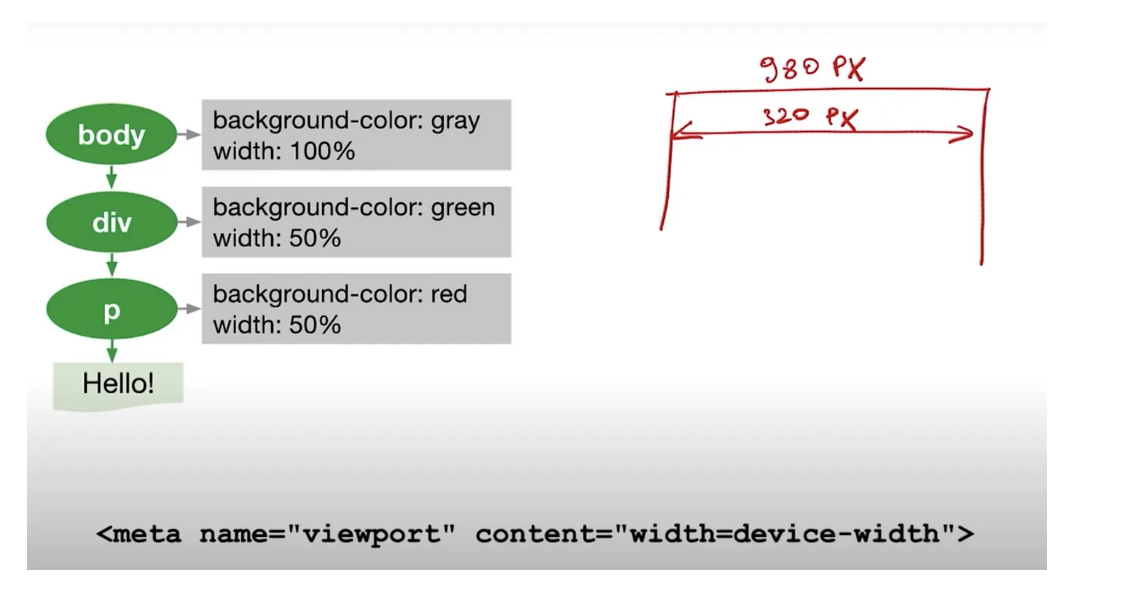
****

**Layout:**

1. The layout which is also called as reflow will be in charge to calculate the positions and dimensions of each node on the screen.

2. The layout peace execute again if we resize the browser or rotate the mobile.

let's take an example:



Here we have render tree at left and <meta> tag indicates pixels of the device, in this ex. 320px. if we don't have meta tag, by default is will be 980px that will cause the unwanted zoom in the browser.

**Paint:**

1. Paint is the final stage in this process, as we know all visible nodes, their styles and geometry. we pass this information to last stage.

2. Paint converts each node (of render tree) into pixels on the screen.

3. This stage is also called as "Painting", "Rasterizing" or "Repainting".

4. Painting breaks elements into layers. there are some specific properties that instantiate layers.