AN INTRO TO JAVASCRIPT AND THE DOM

A BRIEF HISTORY OF JAVASCRIPT

Brendan Eich, a programmer who worked for Netscape Communications Corporation, created JavaScript in 1995. It took him only 10 days to develop JavaScript, which then went under the name Mocha.

WHY JAVASCRIPT?

Eich created JavaScript for Netscape Navigator, and it quickly became known as LiveScript. Then later, the Netscape team changed the name to JavaScript to reflect Netscape's support of the Java programming language in its browser. At the time, people said it was a marketing tactic to connect a brand new language (JavaScript) to a popular language at the time (Java). This was despite the fact that Java and Javascript have nothing to do with each other. This confusion between the two languages exists to this day!

THE REASON FOR JS NAME CHANGE

According to Axel Rauschmayer in his book `Speaking JavaScript: An In-Depth Guide for Programmers`,

> **Knowing why and how JavaScript was created helps us understand why it is the way it is.**

I'd like to read a snippet from his book to give you a bit of history behind the language from a different, deeper, and more professional perspective.

WHAT IS JAVASCRIPT?

JavaScript is a scripting or programming language that allows you to create complex features on web pages. Whenever a page displays dynamic content you know that JavaScript is probably behind it. With JavaScript, you can display content updates, animated 2D/3D graphics, scrolling video jukeboxes, image slide shows, video players, audio players, etc.

In simpler and more direct terms, JavaScript is a language that the browser reads and does stuff with.

JAVASCRIPT ISN'T ONLY FOR THE BROWSER ANYMORE

JavaScript is no longer just for the browser anymore. Today, there is Node.js, an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser, and MongoDB, an Open Source database written in C++.. BUT what is great about MongoDB for JavaScript Developers, is that it supports Server-Side JavaScript execution. This allows a JavaScript developer to use a single programming language for both client and server side code.. AND `MongoDB is a document oriented database`. Node.js was released in June, 2011, and MongoDB was realased on February 11, 2009.

WHAT IS THE DOM?

The Document Object Model, or the DOM, is a platform and language neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents.

WHAT IS THE DOM?

The DOM is the DATA REPRESENTATION of the objects that comprise the structure and content of a document on the web.

WHAT IS THE DOM?

According to the WHATWG (Web Hypertext Application Technology Group),

> **The DOM Standard defines the core infrastructure used to define the web.**

and

> **The DOM defines a platform-neutral model for events, aborting activities, and node trees.**

WHO DEFINES (AND MAINTAINS) THE DOM STANDARD?

The standardization of the DOM was first handled by the World Wide Web Consortium (W3C), which last developed a recommendation in 2004.

Subsequently, WHATWG took over the development of the Standard, publishing it as a living document.

The W3C now PUBLISHES stable snapshots of the WHATWG standard.

WHATWG DOM SPECS

WHATWG breaks down the DOM Standards into the following:

- Goals/Infrastructure
- Goals/Events
- Goals/Aborting ongoing activities
- Goals/Nodes
- Goals/Ranges
- Goals/Traversal
- Goals/Sets
- Goals/XPath
- Goals/Historical

THE DOM AND JAVASCRIPT

The DOM is NOT a programming language, but without it, Javascript would not have any model or concept of web pages, HTML documents, XML documents, and their parts (elements). Every element in a document, the document as a whole, the head, the body, the footer, p tags, header tags, sections, articles, navs, ul tags, li tags, the tables within the document, table headers, text within the table cells, is part of the DOM for that document. They all can be accessed and manipulated using the DOM and Javascript.

HOW IS A WEB PAGE BUILT?

How a browser goes from a source HTML document to displaying a styled and interactive page in the viewport is called the Critical Rendering Path.

REFRESHER: THE VIEWPORT

The Viewport is the part of the web page that the viewer can see. Scrollbars move the viewport to show other parts of the page.

The viewport differs with different devices, i.e. mobile phone, ipad, tablet, laptop, or desktop, etc.

WHAT EXACTLY IS THE CRITICAL RENDERING PATH?

The Critial Rendering Path is the sequence of steps the browser goes through to convert HTML, CSS, and JavaScript into pixels on the screen. Optimizing the critical render path optimizes render performance.

WHAT DOES THE CRITICAL RENDERING PATH INCLUDE?

It includes the DOM, the CSS Object Model (CSSOM), render tree, layout, and paint.

SO HOW DOES BROWSER RENDERING WORK?

In order for the browser to render content on the page, it has to go through the five steps mentioned in the previous slide:

`The DOM, the CSS Object Model (CSSOM), the render tree, layout, and paint.`

WHEN AND HOW IS THE DOM CREATED IN THE CRITICAL RENDERING PATH PROCESS?

To process an HTML file, and get to the DOM, the browser has to go through four steps:

```
Convert bytes to characters, identify tokens, convert tokens to nodes, build the DOM tree
```

CSS OBJECT MODEL (CSSOM)

Just as the DOM contains all page content, the CSS Object Model contains all the info on how to style the DOM. CSSOM is similar to the DOM but different. The building of the DOM is bit by bit, and CSSOM is render blocking. What does this mean? The browser blocks the page from rendering until it receives and processes all the CSS. CSS is render blocking because rules can be overwritten in the CSS, so content can't be rendered until the CSSOM is complete.

CSS OBJECT MODEL (CSSOM) (CONTD)

CSS rules "cascade" down (Cascading Style Sheet). In the stylesheet itself, you could have one rule state one thing higher up the page, and as you go down the page, the same rule could state something else. The rule lower down the page is the rule that is then recognized, and it overwrites the previous rule. In the html page, things cascade down as well. Everything in the head and then then between the body tags has to render first, and then the script tags render afterwards. That's why it is important, to usually have your JavaScript tags at the bottom of the page. You want everything else to render BEFORE the JavaScript is rendered. In essence, the JavaScript DEPENDS on the CSS. Order is important.

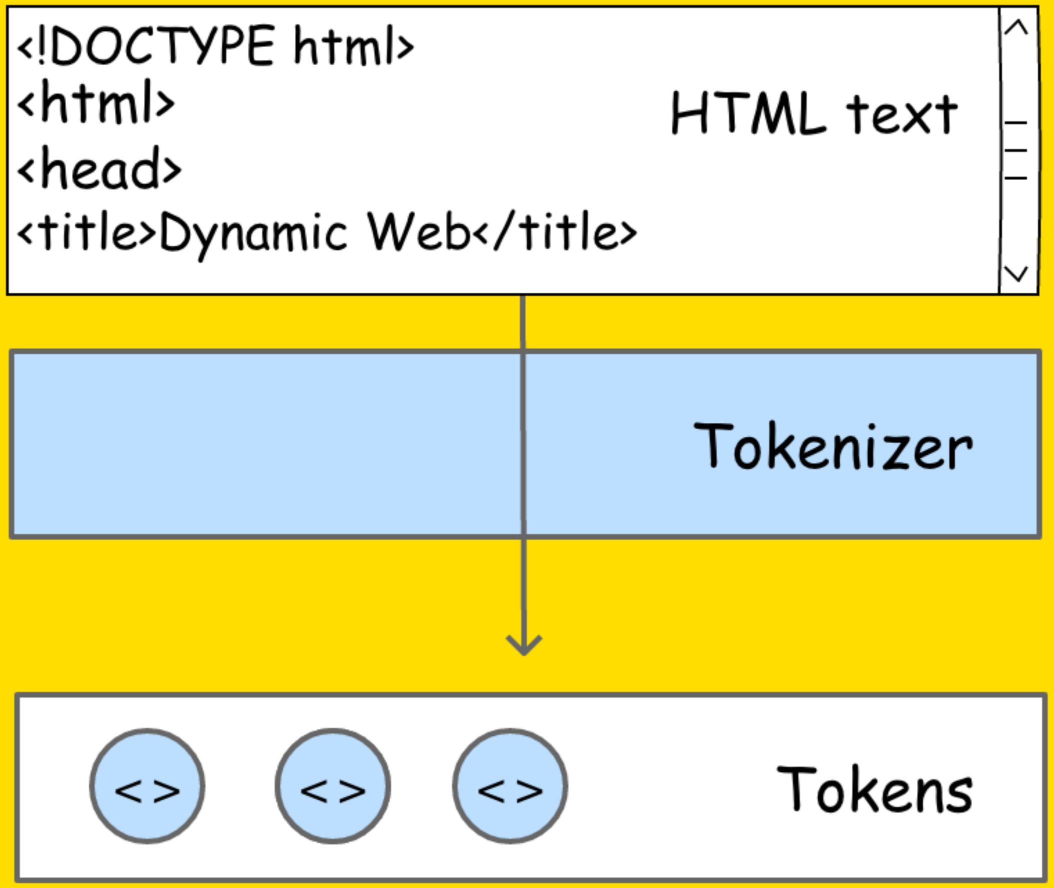
HTML PARSER

Basically, it is the HTML parser, i.e. implemented by the browser, that composes encoding, pre-parsing, tokenization, and tree formation.

TOKENS

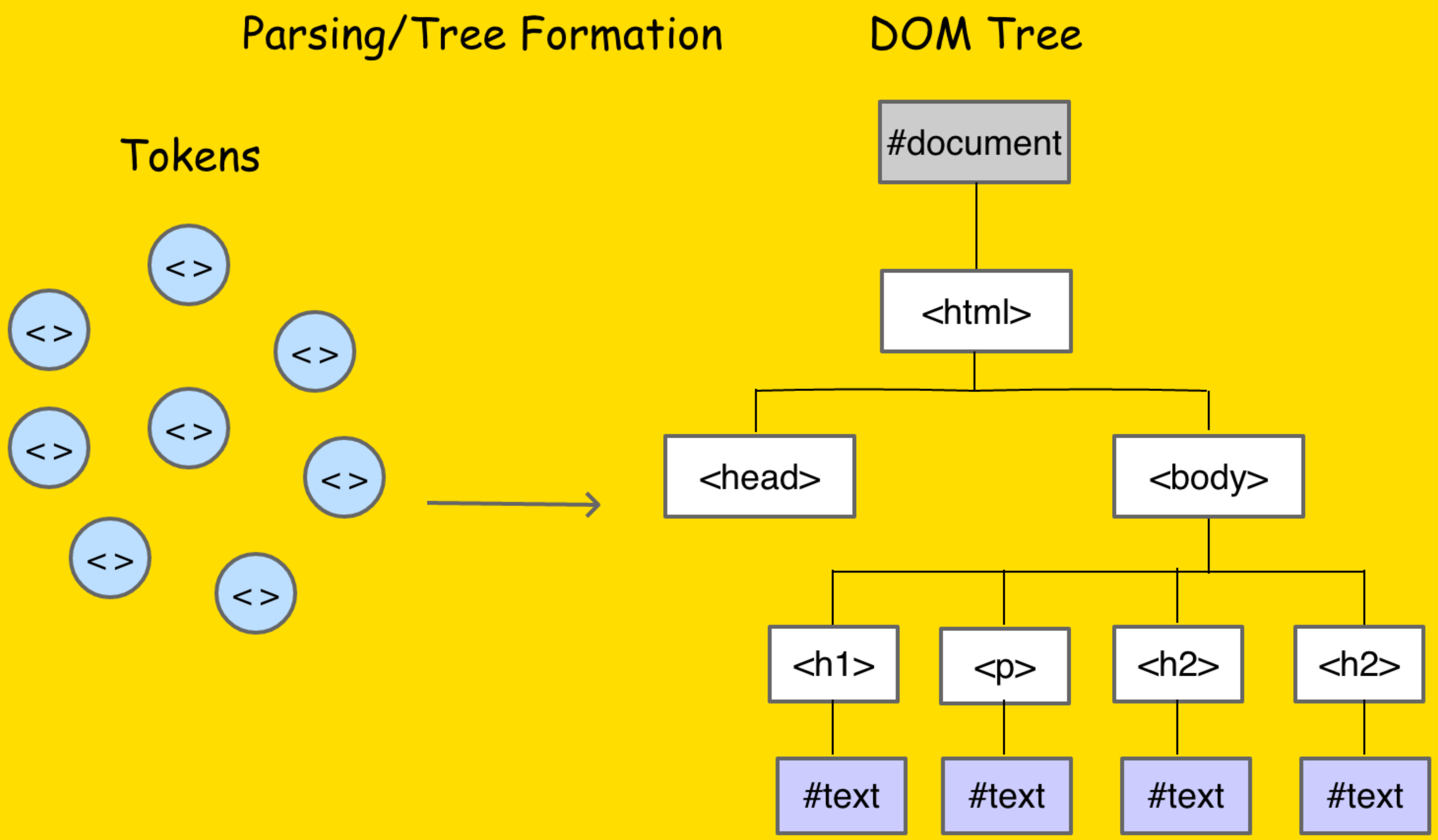
Tokenization is the first half of *parsing* `HTML`. It means turning the `HTML` *markup* into individual tokens such as "begin tag" or "opening tag", "end tag", etc.

Tokenization



```
<!DOCTYPE html>
<html>
<head>
<title>Dynamic Web</title>
```

Parsing/Tree Formation



LAYOUT

Once the `Render Tree` is built, *establishment* of the `Layout` becomes possible.

- `Layout` is dependent on the size of the 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, such as `imgs`, have a default width of 100% of the width of their parent.

- An element with a width of 75%, will be three quarters of the width of its parent.
- Unless otherwise specified, the `body` will have a *default* `width` of 100%. This means that it will fill 100% of the width of the viewport.
- The width of the device impacts Layout.

THE VIEWPORT REVISITED

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 `960px`.

On by-default browsers, i.e., your iphone or tablet, if you set the viewport meta tag to `name="viewport" content="width=device-width"`, then the width will be the width of the device instead of the default viewport width.

COMPLETE VIEWPORT META TAG CONTENTS

`name="viewport" content="width=device-width, initial-scale=1.0"`

`initial-scale=1.0"` sets the initial zoom level when the page is first loaded by the browser.

PAINT

The *last* `step` in the *critical rendering path* is PAINT, as in painting pixels to the screen.

- *Onload*, the entire screen is painted.
- After that, only impacted areas, areas on the screen which have been modified, will be repainted. Browsers are optimized to repaint the minimum required. True to a certain extent. Depends on how much is going on with your JavaScript AND CSS at any given time.

PAINT TIME

Paint time *depends on* what updates are being *applied* to the `render tree`.

Painting in of itself is fast, and not the most impactful in improving performance of your app, but one does have to take into consideration LAYOUT and REPAINT when measuring how long an animation frame may take to render/re-render, for example.

STYLES AND PAINT

Styles applied to each node increase paint time, but one should not just remove styles to improve performance. One has to consider whether the potentially incremental (very small) performance improvement outweighs the experience it provides to the visitor.

UNDERSTANDING THE DOM MEANS A BETTER UNDERSTANDING OF JAVASCRIPT AND HOW IT WORKS

The DOM is essential to making websites interactive. JavaScript is the client-side scripting language that connects to the DOM and directly manipulates it in the browser.

Understanding the `DOM` means a *better understanding* of `JavaScript` and how it works. For those of you who want to *further* your **understanding** of `JavaScript`, or want to go on to *using* `libraries` such as `React` or `frameworks` such as `Vue.js`, `Angular.js`, `Nuxt.js`, or `Next.js`, for example, *understanding* the `DOM` is *crucial*.

RELATED RESOURCES

- Places to Learn JavaScript Online: *Laurence Bradford*, for **the balance careers**
- The History of JavaScript [INFOGRAPHIC]: **checkmarx**
- WHATWG - Standards: **WHATWG**
- WHATWG - DOM Standard: **WHATWG**
- Document Object Model (DOM): **W3C**
- What, exactly, is the DOM?: *Ire Aderinokun*, for **bitsofcode**

RELATED RESOURCES (CONTD)

- Introduction To The DOM: **MDN**
- Critical Rendering Path: **MDN**
- The Document Object Model (DOM): **W3C**
- Speaking JavaScript: An In-Depth Guide for Programmers: *free* **O'Reilly** book online
- Introduction to MongoDB: **w3resource**
- Document Object Model: **wikipedia**

RELATED RESOURCES (CONTD)

- What is viewport in HTML?: **stackoverflow**
- Critical Rendering Path: **Google**
- Understanding the critical rendering path, rendering pages in 1 second: *Luis Vieira*, on **Medium**
- Performance Analysis Reference: **Google**
- Tags to DOM: *Travis Leithead*, for **A List Apart**