Here’s your **JavaScript Class 1 Notes** fully corrected, structured, and formatted in Markdown, ready to be pasted into a `notes.md` file:

JavaScript Class 1 Notes

1. History of JavaScript

- The company **Netscape Communications Corporation** wanted to develop its own web browser.
- Initially, they tried to integrate **Java** into the browser. But Java was not ideal for frontend development as browsers require a more dynamic and lightweight language.
- So, **Brendan Eich**, an employee at Netscape, created a new language in 1993 called **Mocha** (considered the world’s first browser scripting language).
- Later, the name *Mocha* was changed to **Mosaic**, and then renamed again to **LiveScript** in 1994.
- In 1995, they officially released the **LiveScript** language along with the browser **Netscape Navigator** (the world’s second web browser).
- However, LiveScript didn’t gain popularity. So, Netscape renamed it to **JavaScript** for marketing purposes, leveraging the popularity of Java.
- The **Netscape Navigator** browser became very popular, and JavaScript started gaining attention.
- Meanwhile, **Microsoft** copied around 70% of JavaScript’s features and added 30% of their own. They named it **JScript** and launched it with their own browser called **Internet Explorer** in **1995** (the world’s third browser).

🧠 Summary Diagram

```
Netscape Communications Corporation
           |
     Brendan Eich (Developer)
           |
        Mocha → Mosaic → LiveScript → JavaScript
           |
   Netscape Navigator (Browser)

Microsoft (Competitor)
           |
        Copied 70% JS + 30% new features → JScript
           |
        Internet Explorer (Browser)
```

☑ 2. Emergence of JavaScript

- Initially created to enhance interactivity within the browser (client-side scripting).
- Became the **standard scripting language for browsers**.
- Later standardized by **ECMA International** in 1997 (as ECMAScript).
- Evolved to be used both on **frontend** and **backend** (via Node.js).

☑ 3. How JavaScript Became Popular

- Lightweight and easy to integrate into HTML.
- Became the **only scripting language supported by all major browsers**.
- Community support, frequent updates via **ECMAScript versions (ES5, ES6, etc.)**.
- Rise of frameworks (React, Angular, Vue) and backend (Node.js) boosted its usage.
- Runs directly in the browser – **no compilation needed**.

✅ 4. Flow of JavaScript Code in VS Code

- JavaScript files are usually saved with `.js` extension.
- JavaScript code is written in files or embedded directly into HTML using the `<script>` tag.
- When run in the browser:
  1. The HTML is parsed first (top to bottom).
  2. `<script>` tags are encountered and JS is executed immediately unless `defer` or `async` is used.
- In **VS Code**, we usually:
  - Write code in `.js` files.
  - Use **Live Server extension** to see the output dynamically in a browser.

---

5. `defer` vs `async` in `<script>` Tag

Attribute	Load Order	Execution Timing	Best Use Case
`defer`	Loads in parallel	Executes **after HTML parsing is complete**	Multiple scripts that depend on DOM elements
`async`	Loads in parallel	Executes **as soon as it's loaded**	Independent scripts (e.g., analytics)

Example:

```html
<script src="main.js" defer></script>
<script src="analytics.js" async></script>
```

---

6. Order of Execution When Using Multiple `<script>` Tags

- **Without `async` or `defer`**:
  - Scripts are executed **in the order they appear** in the HTML (top to bottom).
  - HTML parsing **pauses** for script execution.
- **With `defer`**:
  - All deferred scripts are downloaded in parallel.
  - Execution happens **in order**, after HTML parsing is complete.
- **With `async`**:
  - Scripts are downloaded in parallel.
  - Executed **as soon as ready**, **not guaranteed in order**.

Example:

```html
<!-- Executed in order, blocks HTML parsing -->
<script src="a.js"></script>
<script src="b.js"></script>

<!-- Executed after HTML parsing, in order -->
<script src="c.js" defer></script>
<script src="d.js" defer></script>

<!-- Executed as soon as loaded, order not guaranteed -->
<script src="e.js" async></script>
<script src="f.js" async></script>
```

☑ 7. What is a Module?

- A **module** is a JavaScript file that can **export** values (functions, variables, classes) to be used in another file.
- Helps in code organization and reusability.

Example:

math.js

```js
export function add(a, b) {
    return a + b;
}
```

main.js

```js
import { add } from './math.js';
console.log(add(5, 3)); // Output: 8
```

> Note: To use modules, you must use `type="module"` in your HTML:

```html
<script type="module" src="main.js"></script>
```

Let me know if you want a visual infographic or diagram version for print or presentation!