HTML learning roadmap

Got it 👍
Let's now **deep dive into** `async` **vs** `defer` **vs normal** `<script>` **loading**, because this is one of the **biggest real-world performance killers** that most devs misunderstand.

🔹 How Browser Loads Scripts

When the browser sees a `<script>` tag:

1. **Pause HTML parsing**
2. **Fetch the script** (if external)
3. **Execute it immediately**
4. Then continue parsing HTML

👉 This causes **render-blocking**, meaning nothing else (like showing content) happens until the script is done.

That's why Google PageSpeed often screams about "Eliminate render-blocking resources".

1️⃣ Normal `<script>` (Default Behavior)

```html
<script src="app.js"></script>
```

⚡ Timeline

- HTML parsing **stops**
- Script downloads
- Script executes
- HTML parsing resumes

📉 **Slow page load** because user waits for script before content continues.

2️⃣ `async` (Independent Mode)

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

⚡ Timeline

- Script is **downloaded in parallel** with HTML parsing.
- As soon as it's ready → **execution happens immediately** (even if HTML parsing isn't done).

✅ Good for

- Scripts that don't depend on DOM or other scripts.
- Example: **Analytics, Ads, Trackers**.

⚠ Pitfall

- Execution order is **NOT guaranteed**.
- If you load multiple `async` scripts, they may run in random order.

3️⃣ `defer` (Team Player Mode)

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

⚡ Timeline

- Script downloads **in parallel** with HTML parsing.
- **Execution happens only after HTML parsing is complete**, in the same order as in HTML.

✅ Good for

- Page logic (JS that manipulates DOM).
- Multiple scripts that depend on each other.

⚠ Pitfall

- Doesn't work with inline scripts.
- Old IE versions had buggy support (not a problem today).

🏆 Comparison Table

Feature	Normal `<script>`	`async`	`defer`
Blocks HTML parsing?	✅ Yes	❌ No	❌ No
Execution timing	Immediately when downloaded	As soon as ready (random order)	After HTML parsing finished
Execution order	Sequential	Unpredictable	Guaranteed (HTML order)
Best use case	Rarely (legacy)	Analytics, Ads, 3rd-party	Main app logic, DOM-related scripts

🔥 Visual Timeline Example

Assume page has 3 scripts:

```html
<script src="a.js"></script>
<script src="b.js" async></script>
<script src="c.js" defer></script>
```

How it plays out:

- `a.js` → blocks parsing, runs immediately.
- `b.js` → downloads in background, runs whenever ready (may interrupt parsing).
- `c.js` → downloads in background, waits until DOM is fully parsed, then executes.

💡 Pro Developer Insights

✅ Rule of Thumb:

- Use `defer` for **your app scripts** (safe, predictable).

- Use `async` for **3rd-party scripts** (independent).
- Avoid plain `<script>` unless absolutely necessary.

✅ Modern Pro Hack:

- Inline **critical JS** in `<head>` (tiny logic only).
- Put **main scripts with** `defer`.
- Load **3rd-party scripts with** `async`.

Example:

```html
<head>
  <script>/* small inline config */</script>
  <script src="/main.js" defer></script>
  <script src="https://analytics.com/track.js" async></script>
</head>
```

👉 This way, **user sees content instantly** + scripts load smartly.

⚡ Next Step:

Do you want me to **deep dive into render-blocking CSS (critical CSS, media queries, font-display)** OR continue with **meta tags that influence performance (cache-control, CSP, theme-color, manifest)**?