Below is a **clean, corrected, modern, and deeply explained version of "Cookie Attributes in JavaScript"**, rewritten to be:

- ✅ **Conceptually accurate (modern browser behavior)**
- ✅ **Interview-ready**
- ✅ **Security-aware**
- ✅ **NoteLM / study-notes friendly**
- ✅ **Internals + practical usage combined**

I have **fixed syntax mistakes**, **removed deprecated/incorrect attributes**, and **explained what JavaScript can vs cannot control**.

🍪 Cookie Attributes in JavaScript — Deep & Practical Guide

1️⃣ What Are Cookie Attributes?

Cookie attributes are **metadata** attached to a cookie that control:

- Where the cookie is sent
- How long it lives
- Whether JavaScript can access it
- Whether it is sent over HTTPS
- Whether it is sent in cross-site requests

📌 **Cookies are not just `key=value`** — attributes define their **behavior and security**.

2️⃣ Important Rule (Very Important)

> JavaScript can **SET** some attributes,
> JavaScript can **READ ONLY name & value**,
> JavaScript **CANNOT read security attributes** like `HttpOnly`, `Secure`, `SameSite`.

Those attributes are enforced by the **browser**, not JS.

3️⃣ Cookie Attribute Summary (Modern & Correct)

Attribute	Purpose	Default
`name=value`	Actual data	Required
`path`	URL scope	Current path
`domain`	Domain scope	Current domain
`expires`	Expiry date (UTC)	Session
`max-age`	Lifetime in seconds	Session

Attribute	Purpose	Default
`secure`	HTTPS only	false
`HttpOnly`	Block JS access	false
`SameSite`	Cross-site control	Lax
`Priority`	Eviction priority	Medium

❌ Attributes like **Size, SourcePort, StoragePartition** are **browser-internal**

❌ JavaScript **cannot set or modify them**

4️⃣ Name / Value Attribute (Core Data)

Purpose

Stores actual information in `key=value` form.

Rules

- Must be **URL-encoded**
- No spaces, semicolons, or special characters

Correct Way

```js
document.cookie = "subscribed=" + encodeURIComponent("false");
```

Reading

```js
function getCookie(name) {
  const cookies = document.cookie.split("; ");
  for (const c of cookies) {
    const [k, v] = c.split("=");
    if (k === name) return decodeURIComponent(v);
  }
  return null;
}
```

5️⃣ Path Attribute (Scope Control)

What it does

Defines **where the cookie is accessible** on the site.

Examples

Path	Accessible Where
`/`	Entire site
`/admin`	`/admin/*` only

Syntax

```js
document.cookie = "signIn=true; path=/";
```

📌 **Most cookies should use** `path=/`

6️⃣ Expires Attribute (Absolute Time)

Purpose

Sets **exact expiry date** (UTC).

Behavior

- Past date → cookie deleted
- No expires → session cookie

Syntax

```js
document.cookie = "product=mobile; expires=Tue, 12 Jan 2050 12:00:00 UTC; path=/";
```

📌 `expires` is **older**, but still supported

7️⃣ Max-Age Attribute (Recommended)

Purpose

Defines **lifetime in seconds**

Advantages over `expires`

- Relative time

- More reliable across time zones

Syntax

```js
document.cookie = "token=abc123; max-age=86400; path=/";
```

⏱️ 86400 seconds = 1 day

Deleting via max-age

```js
document.cookie = "token=; max-age=0; path=/";
```

8️⃣ Domain Attribute (Cross-Subdomain Access)

Default

Cookie is **only accessible to current domain**

Use Case

Share cookies across subdomains

```js
document.cookie = "user=anoop; domain=example.com; path=/";
```

✔️ Available to:

- example.com
- api.example.com
- app.example.com

❌ Cannot set cookies for **other domains**

9️⃣ Secure Attribute (HTTPS Only)

Purpose

Prevents cookie from being sent over HTTP

Syntax

```js
```

```
document.cookie = "sessionId=abc; Secure; path=/";
```

📌 Browser sends cookie **only over HTTPS**

🚨 Mandatory for:

- Authentication cookies
- `SameSite=None` cookies

🔟 HttpOnly Attribute (Security Critical)

Purpose

Prevents **JavaScript access**

Result

```js
document.cookie  // ❌ HttpOnly cookies not visible
```

Why Important?

- Prevents XSS attacks stealing cookies
- Used for auth/session cookies

📌 **HttpOnly can only be set by the server**, not JavaScript

1️⃣1️⃣ SameSite Attribute (CSRF Protection)

Controls cross-site cookie sending

Value	Behavior
`Strict`	Same-site only
`Lax`	Default, safe
`None`	Cross-site allowed (needs Secure)

Example

```js
document.cookie = "token=abc; SameSite=Lax; path=/";
```

📌 Modern browsers default to `Lax`

1️⃣2️⃣ Priority Attribute (Advanced)

Purpose

Helps browser decide **which cookies to evict** first

```js
document.cookie = "id=123; Priority=High; path=/";
```

Values:

- `Low`
- `Medium` (default)
- `High`

📌 Mostly used by large applications

1️⃣3️⃣ Attributes You CANNOT Control via JavaScript

❌ These are **read-only / browser-managed**:

- Size
- SourcePort
- StoragePartition
- Site/Service metadata

You can **view them in DevTools**, but not modify.

1️⃣4️⃣ Viewing Cookie Attributes in Browser

Chrome Steps

1. Right-click → Inspect
2. Application tab
3. Storage → Cookies
4. Select domain
5. View attributes (Secure, HttpOnly, SameSite)

📌 JavaScript **cannot read these values**

1️⃣5️⃣ Updating Cookie Attributes

Cookies are updated by **re-setting the same name**:

```js
```

```
document.cookie = "token=xyz; max-age=86400; path=/";
```

✔️ Same name

✔️ New attributes

✔️ Old cookie replaced

1️⃣6️⃣ Common Mistakes (Interview Traps)

❌ Forgetting `path` when deleting

❌ Storing sensitive data without HttpOnly

❌ Using cookies instead of localStorage blindly

❌ Using `expires` without UTC

❌ Not encoding values

1️⃣7️⃣ Best Practice Summary

✅ Use `max-age` over `expires`

✅ Always set `path=/`

✅ Use `Secure + HttpOnly` for auth

✅ Prefer `SameSite=Lax`

✅ Encode values

❌ Never store secrets in JS-accessible cookies

1️⃣8️⃣ One-Line Rule (Very Important)

> **Cookie attributes define security, scope, and lifetime — not just storage.**

⏭️ What Should We Cover Next?

Recommended next deep topics:

1️⃣ **Cookies vs LocalStorage vs SessionStorage (Internals)**

2️⃣ **Authentication Cookies (JWT + HttpOnly)**

3️⃣ **CSRF & SameSite Attacks (Real Examples)**

4️⃣ **XSS Attacks & Cookie Theft**

Reply with **1 / 2 / 3 / 4**.