JavaScript learning path

Excellent catch 👌 — you're absolutely right. Those **extra details** are part of the *depth* you want, so I'll always include **internal reasoning + hidden quirks + memory/engine-level behavior** when explaining.

Let's refine our **Number deep dive** by adding exactly what you pointed out (with clarification where needed, since some explanations online can be misleading).

🔹 More Depth on `Number`

1. Whole numbers vs decimals

```js
let price = 200.00;
console.log(price); // 200
```

- Even though you wrote `200.00`, JS **stores it as** `200` because **IEEE-754 doesn't distinguish between "int" and "float"**.
- JavaScript has **only one numeric type** → `number`.
- ✅ Engine optimization: If the value is *mathematically an integer*, JS engines (like V8 in Chrome) may internally store it in a smaller memory representation (called **SMI – Small Integer Optimization**) to save memory and speed up arithmetic.

⚠ Correction to your note: It's not about "floats take twice the memory" (JS doesn't have distinct int/float types). Instead, it's that engines use **different internal representations** to optimize whole numbers.

---

2. Range of numbers

```js
console.log(Number.MAX_VALUE); // 1.7976931348623157e+308
console.log(Number.MIN_VALUE); // 5e-324
```

- `Number.MAX_VALUE`: Largest positive number representable (~1.79 × 10^{308}).

- `Number.MIN_VALUE`: Smallest positive number greater than 0 (~5 × 10^{-324}), **not the most negative number** (that's `-Number.MAX_VALUE`).

---

3. Infinity & -Infinity

```js
console.log(Number.MAX_VALUE + Number.MAX_VALUE); // Infinity
console.log(-Number.MAX_VALUE - Number.MAX_VALUE); // -Infinity
```

- Arithmetic overflow (beyond max value) → `Infinity` or `-Infinity`.
- Division by 0 also produces infinities:

  ```js
  console.log(1 / 0);   // Infinity
  console.log(-1 / 0);  // -Infinity
  ```

---

4. Safe integer range

```js
console.log(Number.MAX_SAFE_INTEGER); // 9007199254740991
console.log(Number.MIN_SAFE_INTEGER); // -9007199254740991
```

- Only integers within this range are **reliably represented without rounding errors.**
- Anything beyond may collapse into the same float (as we saw earlier with `9999999999999999`).

---

5. Zero quirk

```js
console.log(0 === -0);        // true
console.log(Object.is(0, -0)); // false
```

- `-0` exists because of IEEE-754 encoding (sign bit can flip even when value is 0).

- Useful in some math/graph contexts (like slope direction).

---

6. Special numeric constants

- `Number.POSITIVE_INFINITY`
- `Number.NEGATIVE_INFINITY`
- `Number.NaN`
- `Number.EPSILON` → difference between 1 and the next representable float.

  ```js
  console.log(Number.EPSILON); // 2.220446049250313e-16
  ```

✅ So, a **true in-depth explanation of Number** should cover:

- One numeric type (IEEE-754 double precision).
- Engine optimizations (SMI, hidden classes).
- Safe integer limits.
- `Infinity`, `-Infinity`, `NaN`, `-0`.
- Constants like `MAX_VALUE`, `MIN_VALUE`, `EPSILON`.
- Precision pitfalls (floating-point math).

👉 Do you want me to **consolidate all this into a single "final cheat sheet" for** `Number` (like a one-page reference), or move to the **next primitive (**`BigInt`**)** with the same in-depth style?