JavaScript learning path

Perfect 👍 let’s solve those **Number mini-exercises** step by step with **reasoning + internals**.

🔹 Q1

```js
console.log(0.1 + 0.2 == 0.3);
```

Step-by-step:

- JS uses **binary floating-point (IEEE-754)**.
- `0.1` and `0.2` **cannot be represented exactly** in binary.
- Their binary approximations add up to `0.30000000000000004`.
- Comparison with `0.3` fails.

✅ **Output:** `false`

🔹 Q2

```js
console.log(Math.sqrt(-1));
```

Step-by-step:

- `Math.sqrt()` is defined only for real numbers ⩾ 0.
- For negative numbers, result is **NaN** (not-a-number).

✅ **Output:** `NaN`

(`typeof NaN === "number"` → quirky fact)

🔹 Q3

```js
```

```
console.log(Number(null));
console.log(Number(undefined));
```

Step-by-step:

- Conversion uses **ToNumber** rules from ECMAScript:
  - `null → 0`
  - `undefined → NaN`

✅ Output:

```r
0
NaN
```

🔹 Q4

```js
console.log(9999999999999999 === 10000000000000000);
```

Step-by-step:

- Both numbers are beyond `Number.MAX_SAFE_INTEGER` ($2^{53}-1$).
- JS cannot represent them precisely — they are **rounded to the nearest representable double**.
- Both become `10000000000000000`.
- Equality holds.

✅ Output: `true`

🧠 Interview Brain-Benders (with answers)

1. **Q:** Why is `typeof NaN === "number"`?
   A: Because NaN is a special numeric value in IEEE-754 — not outside the type, just "invalid number result".
2. **Q:** How can you check for `NaN` safely?
   A: Use `Number.isNaN(value)` (does not coerce), not global `isNaN`.

3. **Q:** How can you detect `-0` vs `+0`?
   A: `Object.is(-0, 0)` → `false` or check `1/x === -Infinity`.

✅ That wraps up **Numbers**.

👉 Do you want me to move to the next primitive (**BigInt**) now, or give you **extra advanced number puzzles** (like weird coercion, `parseInt` quirks, exponential limits) before we move on?