JavaScript learning path

Perfect 👌 we'll go **one primitive at a time in depth**.
Let's start with the **first primitive: Number** — the most commonly used, but also the most **misunderstood** in JavaScript.

🔹 Primitive 1: Number

✅ Definition

In JavaScript, all numbers (integers + floats) are represented using **IEEE-754 64-bit floating-point**.
That means `42`, `3.14`, `NaN`, `Infinity`, `-0` → **all are the same type**: `number`.

⚙️ Internal Representation

A `number` is stored as **64 bits (8 bytes)**:

- **1 bit** → sign (+/-)
- **11 bits** → exponent
- **52 bits** → fraction (mantissa)

This gives:

- **53 bits of precision** for integers (~15–16 decimal digits).
- Beyond $2^{53} - 1$ (`9007199254740991`), integers are unsafe (rounded).

🌟 Special Values

- **NaN** → "Not-a-Number", but still `typeof NaN === "number"`.
- **Infinity / -Infinity** → division by zero, overflow.
- **-0** → special case; behaves like `0`, but detectable.

```js
console.log(typeof NaN); // "number"
console.log(1 / 0);     // Infinity
console.log(1 / -0);    // -Infinity
```

🔄 Type Conversion to Number

- `Number("42")` → `42`
- `Number("")` → `0`
- `Number("foo")` → `NaN`
- `Number(true)` → `1`, `Number(false)` → `0`
- `Number(null)` → `0`
- `Number(undefined)` → `NaN`

⚠️ Pitfalls & Interview Traps

1. **Floating precision**

```js
console.log(0.1 + 0.2); // 0.30000000000000004
console.log(0.1 + 0.2 === 0.3); // false
```

2. **NaN weirdness**

```js
console.log(NaN === NaN); // false
console.log(Number.isNaN(NaN)); // true
```

3. **-0 exists**

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

4. **Max safe integer**

```js
console.log(Number.MAX_SAFE_INTEGER); // 9007199254740991
console.log(9999999999999999); // 10000000000000000 (rounded!)
```

🚀 Optimization Tips

- Prefer integers under $2^{53}-1$ for exact math.
- For financial apps, avoid floats — use libraries (Decimal.js, Big.js) or **BigInt**.
- In hot loops, mixing types (`"5" + 1`) → causes de-optimizations.

🧩 Mini-Exercises (try predicting)

1. ```js
   console.log(0.1 + 0.2 == 0.3);
   ```
2. ```js
   console.log(Math.sqrt(-1));
   ```
3. ```js
   console.log(Number(null));
   console.log(Number(undefined));
   ```
4. ```js
   console.log(9999999999999999 === 10000000000000000);
   ```

👉 Do you want me to **solve these now step by step**, or first let you try predicting them yourself and then I'll reveal the answers?