JavaScript learning path

Great 🚀 let's move on to the **second primitive: BigInt** — which was introduced in **ES2020** to solve some of the exact problems we saw with `Number`.

🔹 Primitive 2: BigInt

✅ Definition

- `BigInt` is a primitive type for representing **integers of arbitrary size**, beyond the safe limit of `Number` ($\pm 2^{53} - 1$).
- Created by appending `n` at the end of an integer or using the `BigInt()` function.

```
js

const big1 = 1234567890123456789012345678890n;
const big2 = BigInt("9007199254740993");
```

⚙️ Why BigInt?

- With `Number`:

  ```
  js

  console.log(9999999999999999); // 10000000000000000 (rounded!)
  ```

- With `BigInt`:

  ```
  js

  console.log(9999999999999999n); // 9999999999999999n (exact!)
  ```

✅ BigInt preserves exact integer precision, no rounding.

🌟 Operations with BigInt

```
js
```

```
let a = 10n;
let b = 3n;

console.log(a + b); // 13n
console.log(a - b); // 7n
console.log(a * b); // 30n
console.log(a / b); // 3n (decimal part truncated)
console.log(a % b); // 1n
```

⚠ Division **always truncates** toward zero — no decimals allowed.

🔄 Mixing BigInt with Number

- **Not allowed directly** (to avoid silent precision loss).

```
js

console.log(1n + 2); // ❌ TypeError
```

- Must explicitly convert:

```
js

console.log(1n + BigInt(2)); // ✅ 3n
console.log(Number(1n) + 2); // ✅ 3
```

⚠ Special Cases

1. **Cannot use Math functions**

   ```
   js

   console.log(Math.sqrt(16n)); // ❌ TypeError
   ```

   (BigInt not supported by `Math`, use libraries instead).

2. **Boolean context works**

   ```
   js

   console.log(0n ? "yes" : "no"); // no
   ```

```
console.log(Boolean(123n)); // true
```

3. **Comparisons with Number**

```
js

console.log(10n == 10);  // true (loose equality)
console.log(10n === 10); // false (different types)
```

🚀 Use Cases

- Cryptography (RSA, hashing).
- Arbitrary precision finance/calculations.
- Scientific computing.
- Blockchain / IDs beyond safe integer range.

🧩 Mini-Exercises

1.
```
js

console.log(5n / 2n);
```

2.
```
js

console.log(5n == 5);
console.log(5n === 5);
```

3.
```
js

console.log(BigInt(Number.MAX_SAFE_INTEGER) + 2n);
```

👉 Should I solve these **BigInt exercises now step by step** (like I did with Number), or move to the **next primitive (String)** after just listing them?

Perfect 👌 let's **solve the BigInt exercises step by step**, and then I'll dive into **edge cases + brain-benders** that trip people up in interviews.

✅ BigInt Exercises with Solutions

🔹 Q1

```js
console.log(5n / 2n);
```

Step-by-step:

- BigInt **only works with integers**.
- Division result is **truncated toward zero** (decimal discarded).

➡ `5n / 2n = 2n`

✅ **Output:** `2n`

🔹 Q2

```js
console.log(5n == 5);
console.log(5n === 5);
```

Step-by-step:

1. `==` does **type coercion**:
   - `5n` (BigInt) → `5` (Number) for comparison.
   - `5 == 5` → `true`.
2. `===` checks **type + value**:
   - Left operand: `BigInt`
   - Right operand: `Number`
   - Different types → `false`.

✅ **Output:**

```arduino
true
```

```
false
```

🔹 Q3

```
js

console.log(BigInt(Number.MAX_SAFE_INTEGER) + 2n);
```

Step-by-step:

- `Number.MAX_SAFE_INTEGER = 9007199254740991`
- Converting to BigInt → `9007199254740991n`
- Adding `2n` → `9007199254740993n`

✅ **Output:** `9007199254740993n`
(This would have been **unsafe** with `Number`, but BigInt handles it perfectly.)

⚠ BigInt Edge Cases & Brain-Benders

---

1. Mixing BigInt with Number

```
js

console.log(10n + 2); // ❌ TypeError
```

- You **must convert explicitly**:

```
js

console.log(10n + BigInt(2)); // ✅ 12n
console.log(Number(10n) + 2); // ✅ 12
```

---

2. Division always truncates

```
js

console.log(7n / 2n); // 3n (not 3.5n)
```

⚠ There is **no BigDecimal in JS**, so BigInt can't hold fractions.

---

3. Boolean behavior

```js
console.log(Boolean(0n));    // false
console.log(Boolean(123n));  // true
```

👉 Works like `0` vs non-zero Numbers.

---

4. Comparisons

```js
console.log(20n > 15);   // true
console.log(20n < 30);   // true
console.log(20n == 20);  // true
console.log(20n === 20); // false
```

---

5. No `Math` support

```js
console.log(Math.max(10n, 20n)); // ❌ TypeError
```

➡ You need libraries like `bigint-math` for sqrt, pow, etc.

---

6. JSON doesn't support BigInt

```js
console.log(JSON.stringify({ big: 10n }));
// ❌ TypeError: Do not know how to serialize a BigInt
```

👉 You must convert manually:

```js
JSON.stringify({ big: 10n.toString() });
```

---

7. BigInt with bitwise ops

- Works fine, but only with BigInts on both sides:

```js
console.log(10n & 7n); // 2n
```

---

8. Negative zero doesn't exist

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

(BigInt removes this IEEE-754 quirk.)

🔑 Interview Tip

When to use BigInt vs Number?

- Use `Number` for:
  - General-purpose math
  - Floating-point / decimals
  - Performance (faster in most cases)
- Use `BigInt` for:
  - Exact integers beyond $2^{53}-1$
  - Cryptography, finance, unique IDs
  - When rounding errors are unacceptable

👉 Should I now move on to the **next primitive: String** (with the same deep dive, edge cases, and puzzles), or do you want a **BigInt optimization & performance internals** section first?