perplexity

datatypes in js in detail

In JavaScript, **data types** specify what kind of value a variable can hold. They are divided into two broad categories: **primitive data types** and **non-primitive/reference data types**.

Primitive Data Types

Primitive types are the most basic, indivisible types—each variable directly contains its value (not a reference). JavaScript primitives are:

1. **Number**
   - Represents both integers and floating-point numbers with double-precision (IEEE 754 format).
   - Examples: `let a = 5; let b = 3.14; let c = -42;`
   - Includes special values: `Infinity`, `-Infinity`, and `NaN` (Not a Number).
   - There is no separate type for integers and floats—everything is just `number`.
2. **String**
   - Represents textual data, a sequence of UTF-16 characters wrapped in quotes (`'...'`, `"..."`, or `` `...` ``
   - Examples: `let name = "Alice"; let greeting = 'Hello!';`
   - Strings are immutable: changing a string creates a new one.
3. **Boolean**
   - Logical values: `true` or `false`.
   - Used for conditions, flags, and logic.
   - Example: `let isOpen = false;`
4. **Undefined**
   - Automatically assigned to variables that are declared but not initialized.
   - Example: `let x; // x is undefined`
5. **Null**
   - Represents the intentional "no value" state.
   - Must be explicitly assigned: `let y = null;`
6. **Symbol** (added in ES6)
   - Used to create unique identifiers for object properties.
   - Each symbol is unique, even with the same description.
   - Example: `const id = Symbol('id');`

7. **BigInt** (added in ES2020)
    - Allows safe representation of large integers beyond the safe range for `number` (beyond $\pm 2^{53}-1$).
    - Use `n` suffix: `let big = 1234567891234567891234567890n;`
    - Cannot mix with regular numbers in arithmetic.

Reference (Non-Primitive) Data Types

Non-primitives do not hold the actual value—**a variable stores a reference (pointer) to the object in memory**.

1. **Object**
    - Most complex and flexible type; used for structured data.
    - Example:

```
let person = { name: "Bob", age: 30 };
```

    - Variations:
        - **Array:** Ordered collection: `let arr = ;`
        - **Function:** Functions are objects: `function greet() {}`
        - **Date:** Special object for date/time: `let today = new Date();`
        - **RegExp:** Regular expressions: `let regex = /abc/i;`

Note: Arrays, Functions, Dates, and RegExps are technically all specialized Objects in JavaScript.

Key Properties of Reference Types

- Can store collections (objects, arrays) and more complex data.
- Assigned and copied by reference (not by value). Changing the reference changes the object everywhere it's referenced.

Differences Between Primitive and Reference Types

Property	Primitive Types	Reference Types
Assignment	By value	By reference
Mutability	Immutable	Mutable
Examples	`number`, `string`, `boolean`	`object`, `array`, `function`

Type Checking

- Use `typeof` for type inspection:

```
typeof 42        // "number"
typeof "hello"   // "string"
typeof null      // "object"     (quirk—historical bug)
typeof undefined // "undefined"
typeof Symbol()  // "symbol"
typeof BigInt(1) // "bigint"
typeof {}        // "object"
typeof []        // "object"
typeof function(){} // "function"
```

Summary Table of JavaScript Data Types

Type	Category	Description	Example
Number	Primitive	Double-precision, integer/float values	42, 3.14, NaN
String	Primitive	Text, sequence of characters	"Hello"
Boolean	Primitive	Logical: `true`, `false`	true, false
Undefined	Primitive	Declared, not initialized	let x;
Null	Primitive	Explicit 'no value'	let y = null;
Symbol	Primitive	Unique identifier (object properties)	Symbol("id")
BigInt	Primitive	Arbitrarily large integers (ES2020+)	123n
Object	Reference	Key-value store, arrays, functions, etc.	{a:1}, , fn()

In brief:

- JavaScript has **7 primitive types** (`number`, `string`, `boolean`, `undefined`, `null`, `symbol`, `bigint`) and only **one reference (object) type** that includes arrays, functions, and more.
- Primitive values are immutable and copied by value; objects are mutable and copied by reference.
- Understanding these distinctions is central to robust JavaScript programming.