JavaScript

Day 6

JavaScript Data Types

In JavaScript, data types are categorized into two broad groups: **Primitive Data Types** and **Non-Primitive (Object) Data Types**.

---

1. Primitive Data Types

Primitive data types include:

- **Number**: Used to represent both integers and floating-point numbers. Example: let x = 10;
- **String**: Represents textual data. Enclosed in quotes. Example: let name = "John";
- **Boolean**: Represents logical values — either true or false. Example: let isActive = true;
- **Null**: Represents an intentional absence of any object value. Example: let emptyValue = null;
- **Undefined**: A variable that has been declared but not assigned a value is undefined. Example: let notDefined;

---

2. Non-Primitive (Object) Data Types

Objects are used to store collections of data or more complex entities. This includes arrays, functions, and regular expressions.

For example, an object can be created like this:

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

Objects can store multiple related values in one variable, which makes them powerful and flexible.

The typeof Operator

JavaScript provides the typeof operator to check the data type of a value. For example:

- typeof 42 returns "number"
- typeof "hello" returns "string"
- typeof true returns "boolean"
- typeof null returns "object" (this is a known quirk in JavaScript)
- typeof undefined returns "undefined"
- typeof {} returns "object"
- typeof [] also returns "object" (arrays are special kinds of objects)

Type Conversions in JavaScript

JavaScript is a loosely typed language, meaning variables can hold values of any type and types can be converted automatically or manually.

---

1. String Conversion

You can convert any value to a string using the String() function.
For example:

```
let value = true;
value = String(value);
```

This changes the boolean true to the string "true".

---

2. Numeric Conversion

JavaScript often converts values to numbers automatically during mathematical operations.
For instance:

```
console.log("6" / "2"); // Outputs 3
```

You can also explicitly convert a string to a number using Number():

```
let str = "123";
let num = Number(str);
```

This converts the string "123" into the number 123.

If the string doesn’t represent a valid number (e.g., "abc"), the result will be NaN (Not-a-Number).

---

3. Boolean Conversion

In JavaScript, certain values are considered **falsy**, meaning they convert to false. These include:

- The number 0
- An empty string ""
- null
- undefined
- NaN

All other values are **truthy**, meaning they convert to true.
Examples:

```
Boolean(1);        // true
Boolean(0);        // false
Boolean("hello");  // true
Boolean("");       // false
```

JavaScript Operators – Basics & Comparisons

Operators Overview

- **Operands** are the values that operators act upon.
- A **Unary Operator** works on one operand.
- A **Binary Operator** works on two operands.

Arithmetic Operators

JavaScript supports standard arithmetic operations:

- + (Addition): Adds two numbers, e.g., 5 + 2 results in 7.
- - (Subtraction): Subtracts the second number from the first, e.g., 5 - 2 results in 3.
- * (Multiplication): Multiplies two numbers, e.g., 5 * 2 results in 10.
- / (Division): Divides the first number by the second, e.g., 5 / 2 results in 2.5.
- % (Remainder or Modulus): Gives the remainder of division, e.g., 5 % 2 results in 1.
- ** (Exponentiation): Raises the first number to the power of the second, e.g., 2 ** 3 results in 8.

String Concatenation with +

The + operator is also used to concatenate (join) strings.

- When you add a string and a number, the number is converted to a string.

Examples:

```
console.log("Hello" + " World");   // Output: "Hello World"
console.log("1" + 2);              // Output: "12"
console.log(2 + "1");              // Output: "21"
```

If you want to perform numeric addition instead, convert the string to a number first using Number() or unary plus +:

```
console.log(+"5" + 2);             // Output: 7
```

Assignment Operator =

The assignment operator assigns a value to a variable:

```
let x = 10;
```

You can chain assignments as well:

```
let a, b, c;
a = b = c = 2 + 2;
```

```
console.log(a, b, c);  // Output: 4 4 4
```

Modify-in-Place Operators

These operators modify the value of a variable based on its current value:

```
let num = 10;


num += 5;   // Equivalent to: num = num + 5  → 15
num -= 3;   // Equivalent to: num = num - 3  → 12
num *= 2;   // Equivalent to: num = num * 2  → 24
num /= 3;   // Equivalent to: num = num / 3  → 8
num %= 2;   // Equivalent to: num = num % 2  → 0
```

Increment and Decrement Operators

- ++x — Increment before returning the value.
- x++ — Return the current value, then increment.
- --x — Decrement before returning the value.
- x-- — Return the current value, then decrement.

Example:

```
let a = 1;
console.log(++a); // Output: 2 (increments first)
console.log(a++); // Output: 2 (returns first, then increments)
console.log(a);   // Output: 3 (value after increment)
```