JavaScript Data Types and OOP Concepts

JavaScript Data Types

JavaScript has two main categories of data types:

1. Primitive Data Types (Immutable)

These are basic, single-value data types.

- Number: let age = 21;
- String: let name = "Aryan";
- Boolean: let isOnline = true;
- Undefined: let x; // no value assigned
- Null: let temp = null;
- Symbol: let id = Symbol("id");
- BigInt: let big = 12345678912345678 9n;

2. Non-Primitive (Reference) Data Types

These hold collections or complex values.

- Object: { name: "Aryan", age: 21 }
- Array: [1, 2, 3, 4]
- Function: function sayHi() { console.log("Hi"); }

How to check data type?

Use typeof operator:

```
let a = "Aryan";
console.log(typeof a); // string
```

Special Notes:

- typeof null => "object" (bug in JS)
- Arrays/functions are technically objects
- BigInt used for very large integers
- Symbol creates unique keys

Summary Table:

Primitive : Number, String, Boolean, Null, Undefined, Symbol, BigInt
Non-Primitive : Object, Array, Function

OOP (Object-Oriented Programming) in JavaScript

OOP Concepts:
1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

1. Encapsulation
Group data and functions inside a class.

```
class Student {
constructor(name, age) {
this.name = name;
this.age = age;
}
getDetails() {
return `${this.name} is ${this.age} years old.`;
}
}
```

2. Abstraction
Hiding internal logic using private fields (#)

```
class BankAccount {
#balance = 0;
```

```
deposit(amount) { this.#balance += amount; }
getBalance() { return this.#balance; }
}
```

3. Inheritance

Child class reuses parent class methods/properties.

```
class Animal {
speak() { console.log("Animal speaks"); }
}
class Dog extends Animal {
bark() { console.log("Dog barks"); }
}
```

4. Polymorphism

Same method behaves differently in different classes.

```
class Shape { area() { return "Undefined"; } }
class Circle extends Shape { area() { return 3.14 * 5 * 5; } }
class Square extends Shape { area() { return 4 * 4; } }
```

Summary Table:

```
Encapsulation : class with properties/methods
Abstraction   : hiding with #private
Inheritance   : extends keyword
Polymorphism  : method overriding
```