Constructor functions and new operator

🛠️ Constructor Functions and the `new` Operator in JavaScript – In Detail

In JavaScript, **constructor functions** are used to create **multiple similar objects** efficiently. They are the foundation of classical object-oriented patterns before `class` syntax was introduced in ES6.

✅ What Is a Constructor Function?

A constructor function is just a **regular function** that:

- Is used with the `new` keyword.
- Capitalizes its name by convention (e.g., `Person`, `Car`).

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}
```

This function does not **return** anything — it relies on the `new` keyword to do that automatically.

✅ The `new` Operator

When you use `new Person("Abhi", 25)`, **four things** happen under the hood:

1. A **new empty object** is created.
2. `this` is set to point to that new object.
3. The function body runs, assigning properties/methods to `this`.
4. The new object is **automatically returned**.

```
const user = new Person("Abhi", 25);
console.log(user.name); // "Abhi"
console.log(user.age);  // 25
```

✅ Adding Methods to Constructor

To avoid recreating functions on every object, use the prototype:

```
Person.prototype.greet = function () {
  console.log(`Hi, I'm ${this.name}`);
};

user.greet(); // Hi, I'm Abhi
```

✅ Custom Constructor vs Object Literal

```
// Using constructor
function Animal(type) {
  this.type = type;
}

const dog = new Animal("Dog");

// Using object literal
const cat = {
  type: "Cat"
};
```

Constructor functions are **better for creating multiple instances**.

✅ Constructor Function Example

```
function Car(make, model) {
  this.make = make;
```

```
  this.model = model;
}

Car.prototype.drive = function () {
  console.log(`${this.make} ${this.model} is driving`);
};

const car1 = new Car("Toyota", "Corolla");
const car2 = new Car("Honda", "Civic");

car1.drive(); // Toyota Corolla is driving
car2.drive(); // Honda Civic is driving
```

✅ Detecting Constructor

```
console.log(car1.constructor === Car); // true
```

⚠️ What If You Forget `new`?

If you call a constructor **without** `new`, `this` refers to the global object (`window` in browser or `global` in Node.js):

```
const x = Person("Abhi", 25); // WRONG
console.log(x); // undefined
console.log(window.name); // "Abhi" (in browser) ❌
```

To avoid this, some devs use:

```
function Person(name) {
  if (!(this instanceof Person)) {
    return new Person(name);
  }
  this.name = name;
}
```

✅ Summary Table

Concept	Explanation
Constructor Function	Special function used with `new` to create objects
`new` Keyword	Creates a new object, sets `this`, and returns the object
`this`	Refers to the new object being constructed
Prototype	Shared object where you attach methods to avoid duplication

Let me know if you'd like to see how constructor functions compare to ES6 `class` syntax!