# Nullish Coalescing Operator (??)

The nullish coalescing operator ( `??` ) returns the right-hand side operand only if the left-hand side operand is `null` or `undefined` ; otherwise, it returns the left-hand side operand.

**Example:**

```
let age;
let ageDetails = age ?? 25;
console.log(ageDetails); // 25 (since age is undefined)
```

If the provided argument is `0` , it is treated as `0` instead of returning the right-hand side operand.

# Optional Chaining (?.)

Optional chaining ( `?.` ) checks if a property or method exists before accessing it, preventing runtime errors.

**Example:**

```
let details;
console.log(details?.city); // undefined (safe access)
console.log(details.city); // TypeError: Cannot read properties of undefined
```

It can also be used with functions:

```
let kyle = { printName: () => "Kyle" };
kyle.printName?.(); // "Kyle"
```

# Plain Object vs. Map

## Differences:

- **Map**: Supports any type as keys (objects, arrays, functions, symbols, etc.).
- **Object**: Only supports string and symbol keys (other types get converted to strings).

#### Example with Map:

```
let map = new Map();
let objKey = { id: 1 };
map.set(objKey, "Object as Key");
console.log(map.get(objKey)); // "Object as Key"
```

#### Example with Object:

```
let obj = {};
obj[objKey] = "Object as Key";
console.log(obj); // { '[object Object]': 'Object as Key' } (key converted to string)
```

## Avoiding Prototype Inheritance Issues:

Objects inherit from `Object.prototype`, which can cause unintended behavior.

```
let ages = {
    Boris: 28,
    John: 29
};
console.log(ages.Boris); // 28
console.log("toString" in ages); // true (due to prototype inheritance)
```

Using a **Map** avoids this issue:

```
let ages = new Map();
ages.set("Boris", 28);
console.log(ages.get("Boris")); // 28
console.log(ages.has("toString")); // false
```