ARRAY

1. ARRAY OPERATOR

In Javascript , the + operator behaves differently based on the **data type**:

**✅ When used with strings, it joins (concatenates) them:**

```
let name = "John";
let greeting = "Hello, " + name;
console.log(greeting);  // Output: Hello, John
```

**❌ When used with arrays, it doesn't add or merge them properly:**

```
let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];

console.log("Using +: " + arr1 + arr2);
// Output: Using +: 1,2,34,5,6
// This is WRONG: It merges them as strings, not numbers or arrays.
```

**✅ Correct way to display arrays:**

```
console.log("Array 1:", arr1);
console.log("Array 2:", arr2);
// Output: Array 1: [1, 2, 3]
//         Array 2: [4, 5, 6]
```

**✅ If you want to merge arrays:**

```
let merged = arr1.concat(arr2);
console.log("Merged Array:", merged);
// Output: Merged Array: [1, 2, 3, 4, 5, 6]
```

---

## 🔍 Summary:

| Type | Use of + | Output | Recommendation |
|---|---|---|---|
| String | Joins | "Hello, John" | ✅ Use + |
| Number | Adds | 3 + 2 = 5 | ✅ Use + |
| Array | Converts to string | `[1,2] + [3,4] = "1,23,4"` | ❌ Avoid + with arrays |

2. ARRAY BRACKET

In JavaScript, `[]` represents an **array**. It is used to **store multiple values** in a **single variable**, including objects, numbers, strings, or even other arrays.

---

## ✅ Example:

```
let people = [
  { firstname: "vraj", lastname: "valand" }
];
```

---

## 🔍 What is this?

- `[]` → This is an **array**.
- `{ firstname: "vraj", lastname: "valand" }` → This is an **object**.
- So you are creating an **array of objects**.

---

## 📌 Why do we write like this?

This format is **very useful** when:

### 1. You have a list of similar items (like users, products, etc.)

```
  let users = [
  { firstname: "vraj", lastname: "valand" },
  { firstname: "ravi", lastname: "patel" },
];
```

### 2. You want to loop through them:

```
users.forEach((user) => {
  console.log(user.firstname + " " + user.lastname);
});
// Output:
// vraj valand
// ravi patel
```

### 3. You want to access one object by index:

```
console.log(users[0].firstname);  // Output: vraj
```

---

### 💡 Summary:

| Symbol | Used For | Example |
|---|---|---|
| [ ] | Arrays | `[1, 2, 3]` |
| { } | Objects / Code Block | `{key: "value"}` or `{...code}` |
| ( ) | Grouping / Functions | `(x + y)` or `function() {...}` |

## 3. Array Mapping

Array mapping in JavaScript is a technique where we use the `map()` method to create a new array by applying a function to each element of an existing array, without modifying the original array."

### What is Array Mapping?

- `map()` is a method used to **loop through an array** and **transform each element**.
- It **returns a new array** with the modified values.

---

### 🔹 Syntax:

```
let newArray = oldArray.map(function(item) {
  return updatedItem;
});
```

---

### 📘 Example 1: Square each number

```
let numbers = [1, 2, 3, 4, 5];

let squares = numbers.map(function(num) {
  return num * num;
});

console.log(squares);  // Output: [1, 4, 9, 16, 25]
```

---

### 📘 Example 2: Using Arrow Function

```
let squares = numbers.map(num => num * num);
```

---

### 📘 Example 3: Array of Objects

```
let students = [
  { name: "Vraj", marks: 70 },
  { name: "Ravi", marks: 85 }
];

let namesOnly = students.map(student => student.name);

console.log(namesOnly);  // Output: ["Vraj", "Ravi"]
```

---

## ✅ 2. Function in JavaScript

A **function** is a block of code designed to perform a particular task.

---

## 📘 Example 1: Simple function

```
function greet(name) {
  return "Hello, " + name;
}

console.log(greet("Vraj"));  // Output: Hello, Vraj
```

---

## 📘 Example 2: Arrow function

```
let greet = (name) => {
  return "Hello, " + name;
};
```

---

## □ Combine map + function:

```
function double(n) {
  return n * 2;
}

let nums = [2, 4, 6];
let doubled = nums.map(double);

console.log(doubled);  // Output: [4, 8, 12]
```

# 4. Array Function

## 1. Regular Function Declaration

```
function add(a, b) {
  return a + b;
}
let result = add(5, 3);
console.log(result);  // Output: 8
```

### 🔍 Explanation:

| Line | What it does |
|---|---|
| `function add(a, b)` | Declares a **named function** called `add` that accepts two parameters `a` and `b`. |
| `{ return a + b; }` | The function returns the sum of `a` and `b`. |
| `let result = add(5, 3);` | Calls the function with 5 and 3 → `add(5, 3)` returns 8. |
| `console.log(result);` | Prints the result → 8. |

### ✅ Why use a named function?

- Helpful in debugging (error stack shows `add`).
- Easier to reuse multiple times.
- Can be **hoisted** – means you can call it even before it's written in the code.

---

## ✅ 2. Arrow Function (anonymous)

```
const add = (a, b) => {
  return a + b;
};
```

### 🔍 Explanation:

| Line | What it does |
|---|---|
| `const add =` | You are storing the function inside a variable called `add`. |
| `(a, b) => { return a + b; }` | This is an **arrow function** with two inputs. It returns their sum. |

## ⚠️ Why arrow functions don't need a name?

- Because the function is being **stored inside a variable**, like `const add`. So the name is not required in the function itself.
- It's called an **anonymous function** (function without name).
- You call it using the variable: `add(5, 3)`

---

### ✅ 3. Short Arrow Function (1 line)

```
const add = (a, b) => a + b;
```

## 🔍 Explanation:

| Line | What it does |
|---|---|
| `(a, b) => a + b` | No `{}` and no `return` needed if it's just one line. It auto-returns the result. |

---

### ⚖️ Difference Summary: Regular Function vs Arrow Function

| Feature | Regular Function | Arrow Function |
|---|---|---|
| Name | Required | Optional (usually stored in variable) |
| Hoisting | ✅ Yes | ❌ No |
| `this` binding | Traditional `this` context | Inherits `this` from outer scope (lexical) |
| Syntax | Longer | Shorter |
| Use | Useful for methods or hoisting | Useful for small, inline functions |