Good. We'll do this properly.
No rushing. No half-knowledge.

🟢 1️⃣ Variables in JavaScript

What is a Variable?

A **variable** is a named container used to store data.

Think of it like a labeled box 📦:

```js
let age = 20;
```

Here:

- `age` → label
- `20` → value stored inside

🔹 How to Declare Variables

JavaScript has **3 ways**:

```js
var a = 10;
let b = 20;
const c = 30;
```

But in modern web development:

Keyword	Use?	Why
`var`	❌ Avoid	Old, buggy behavior
`let`	✅ Yes	Value can change
`const`	✅ Preferred	Value cannot change

🔹 `let`

Used when value can change.

```js
let score = 50;
```

```
score = 70;  // allowed
```

🔹 `const`

Used when value should NOT change.

```
js

const pi = 3.14;
pi = 3.1415; // ❌ Error
```

Use **`const`** by default.
Use **`let`** only when needed.

🔹 Rules for Variable Names

✅ Allowed:

```
js

let myName;
let user_age;
let totalAmount;
```

❌ Not allowed:

```
js

let 1name;       // cannot start with number
let user-name;   // dash not allowed
let let = 10;    // reserved keyword
```

🔹 Important Web Dev Concept

JavaScript is **dynamically typed**.

```
js

let data = 10;
data = "hello";
data = true;
```

The type changes automatically.

This flexibility is powerful ⚡ but dangerous if you're careless.

🧠 Example (Browser Context)

```js
let username = "Meghana";
console.log(username);
```

In web dev, variables store:

- user input
- API responses
- DOM elements
- app state
- configuration

They are the foundation of everything.

📝 Your Tasks (Do Them Properly)

🔹 Task 1

Create 3 variables:

- yourName
- yourAge
- isStudent

Print them in one sentence like:

```pgsql
My name is X, I am Y years old, Student: true/false
```

🔹 Task 2

Create a variable `price = 100`.

Then:

- Increase it by 20
- Print new value
- Try the same using `const` and observe what happens

Do the tasks first.
Then I'll give you:

- ✅ Correct answers

- ⚠️ Common mistakes
- 🌐 Web dev memory points
- 🧠 Universal JS memory rules

Also, your sentence correction:

❌ for each topic give me 2 tasks, i will do.
✅ For each topic, give me 2 tasks. I will do them.

Now do the tasks. Don't just read.

Good. Straight to evaluation.

✅ TASK 1 – Expected Answer

```js
let yourName = "Meghana";
let yourAge = 21;
let isStudent = true;

console.log("My name is " + yourName +
            ", I am " + yourAge +
            " years old, Student: " + isStudent);
```

Modern cleaner version:

```js
console.log(`My name is ${yourName}, I am ${yourAge} years old, Student: ${isStudent}`);
```

✅ TASK 2 – Expected Answer

```js
let price = 100;
price = price + 20;
console.log(price);  // 120
```

OR

```js
price += 20;
```

Now using **`const`**:

```js
const price = 100;
price += 20;   // ❌ Error: Assignment to constant variable
```

Correct understanding:

- `const` cannot be reassigned.
- But objects/arrays declared with const can be modified internally (important later).

⚠️ Common Mistakes (Very Important)

1️⃣ Forgetting `let` or `const`

```js
price = 100; // ❌ creates global variable accidentally
```

2️⃣ Using `var`
Avoid it.

3️⃣ Using `const` everywhere blindly
Use it by default, but switch to `let` if value changes.

4️⃣ Confusing reassignment with mutation
(This becomes important with arrays & objects.)

5️⃣ Bad naming

```js
let x = 100;   // meaningless
```

Prefer:

```js
let productPrice = 100;
```

🌐 Web Dev Memory Points

For web development:

- Use `const` for:
    - DOM elements
    - API configs
    - URLs
    - Static values

- Use `let` for:
    - Counters
    - Form input changes
    - Toggle states
    - Re-render data

Example:

```
js

const button = document.querySelector("button");
let count = 0;
```

Button won't change → const
Count changes → let

🧠 Universal JS Memory Rules

1️⃣ Default to `const`
2️⃣ Use `let` only when necessary
3️⃣ Never use `var`
4️⃣ Always declare variables
5️⃣ Use meaningful names
6️⃣ JS is dynamically typed — type can change

🔵 Next Topic: DATA TYPES

2️⃣ Data Types in JavaScript

JS has two main categories:

🔹 Primitive Types

Type	Example
Number	`10`, `3.14`
String	`"Hello"`
Boolean	`true`, `false`
Undefined	`let x;`
Null	`null`
BigInt	`123n`
Symbol	advanced

🔹 Non-Primitive (Reference Types)

Type	Example
Object	`{name: "Meghana"}`
Array	`[1,2,3]`
Function	`function(){}`

Key Concept: `typeof`

```
js

let age = 20;
console.log(typeof age); // "number"
```

🔥 Important Web Dev Concept

`null` vs `undefined`

```
js

let x;
console.log(x); // undefined

let y = null;
```

- **`undefined`** → variable declared but not assigned
- **`null`** → intentionally empty

In web dev:

- API may return **`null`**
- Missing values may be **`undefined`**

Handle both carefully.

📝 Your Tasks

🔹 Task 1

Create variables for:

- number
- string

- boolean
- undefined
- null

Print their **`typeof`**.

Observe what happens with **`null`**.

🔹 Task 2

Create:

```js
let a = "10";
let b = 10;
```

Print:

```makefile
a + b
a - b
a * b
a == b
a === b
```

Study the output carefully.

This topic is where most beginners become weak.

Do tasks. Then we move to Functions.

Good.

✅ DATA TYPES — Answers

🔹 Task 1 Expected Output

```js
let num = 25;
let str = "Hello";
let bool = true;
let undef;
let empty = null;

console.log(typeof num);    // "number"
console.log(typeof str);    // "string"
console.log(typeof bool);   // "boolean"
console.log(typeof undef);  // "undefined"
console.log(typeof empty);  // "object"  ⚠
```

Important:

`typeof null` → `"object"`

This is a historical JavaScript bug.
Remember it. Don’t argue with it.

◆ Task 2 Expected Output

```js
let a = "10";
let b = 10;

console.log(a + b);   // "1010"
console.log(a - b);   // 0
console.log(a * b);   // 100
console.log(a == b);  // true
console.log(a === b); // false
```

Why?

Operation	Reason
`"10" + 10`	String concatenation
`"10" - 10`	JS converts string to number
`==`	Loose comparison (type coercion)
`===`	Strict comparison (no coercion)

⚠️ Common Mistakes

1️⃣ Using == instead of ===
Always prefer ===.

2️⃣ Forgetting strings from input are strings.

3️⃣ Not understanding type coercion.

4️⃣ Confusing `null` and `undefined`.

🌐 Web Dev Memory Points

- Form inputs always return **string**
- API responses can contain `null`
- Use ===
- Validate data types before using

Example:

```js
if (value === null || value === undefined) {
  // handle safely
}
```

🧠 Universal JS Memory Rules

1️⃣ === over ==
2️⃣ Input is string
3️⃣ null ≠ undefined
4️⃣ **`typeof null`** is "object" (remember bug)
5️⃣ JS auto converts types — don’t rely on it

🔵 Next Topic: FUNCTIONS

3️⃣ Functions in JavaScript

A function is a reusable block of code.

🔹 Basic Function

```js
function greet() {
    console.log("Hello");
}
```

Call it:

```js
greet();
```

🔹 Function with Parameters

```js
function greet(name) {
    console.log("Hello " + name);
}
```

🔹 Return Value

```js
function add(a, b) {
    return a + b;
}

let result = add(5, 3);
console.log(result);
```

If you don't use **`return`**, function gives **`undefined`**.

🔹 Arrow Function (Modern)

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

Short version:

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

🔥 Important Web Dev Concept

Functions are used for:

- Event handlers
- API calls
- Reusable UI logic
- Data processing

Everything in React is function-based.

📝 Your Tasks

🔹 Task 1

Create a function:

- Takes two numbers
- Returns larger number

🔹 Task 2

Create an arrow function:

- Takes a name
- Returns: `"Welcome, <name>"`

Do it properly.
Then we move to Loops.

Good.

✅ FUNCTIONS — Expected Answers

🔹 Task 1

```
js

function getLarger(a, b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

Cleaner:

```
js

function getLarger(a, b) {
    return a > b ? a : b;
}
```

🔹 Task 2

```
js

const welcome = (name) => {
    return "Welcome, " + name;
};
```

Short version:

```
js

const welcome = name => `Welcome, ${name}`;
```

⚠️ Common Mistakes

1️⃣ Forgetting **`return`**

```js
function add(a, b) {
    a + b;   // ❌ returns undefined
}
```

2️⃣ Confusing function declaration and function call.

3️⃣ Not storing returned value.

4️⃣ Mixing arrow function syntax incorrectly.

🌐 Web Dev Memory Points

- Event handlers are functions:

```js
button.addEventListener("click", function() {})
```

- In React:
  Everything is function-based.
- Separate logic into functions.
  Don’t write everything in global scope.

🧠 Universal JS Function Rules

1️⃣ Always return values if needed.
2️⃣ Keep functions small.
3️⃣ One function → one responsibility.
4️⃣ Use arrow functions for cleaner modern code.

🔵 Next Topic: LOOPS

4️⃣ Loops in JavaScript

Loops repeat code.

🔹 for Loop

```js
for (let i = 0; i < 5; i++) {
    console.log(i);
}
```

Structure:

```sql
initialization;
condition;
increment;
```

🔹 while Loop

```js
let i = 0;
while (i < 5) {
    console.log(i);
    i++;
}
```

🔹 for...of (Important for arrays)

```js
let arr = [10, 20, 30];

for (let value of arr) {
    console.log(value);
}
```

🔥 Important Web Dev Concept

Loops are used to:

- Render lists
- Process API data
- Validate forms
- Traverse arrays

In React, array loops render UI components.

📝 Your Tasks

🔹 Task 1

Print numbers from 1 to 10 using a loop.

🔹 Task 2

Given:

```js
let arr = [5, 10, 15, 20];
```

Use a loop to calculate total sum.

Do properly. Then we move to Conditionals.

Good.

✅ LOOPS — Expected Answers

🔹 Task 1

```js
for (let i = 1; i <= 10; i++) {
    console.log(i);
}
```

🔹 Task 2

```js
let arr = [5, 10, 15, 20];
let sum = 0;

for (let value of arr) {
    sum += value;
}

console.log(sum); // 50
```

Alternative:

```js
for (let i = 0; i < arr.length; i++) {
    sum += arr[i];
}
```

⚠️ Common Mistakes

1️⃣ Infinite loop

```
js

for (let i = 1; i <= 10; )  // ❌ no increment
```

2️⃣ Wrong condition (< vs <=)

3️⃣ Forgetting to initialize sum = 0

4️⃣ Using `for...in` instead of `for...of` for arrays
`for...in` gives index, not value.

🌐 Web Dev Memory Points

- Loop over API arrays
- Render UI lists
- Validate multiple form inputs
- Process cart items

In React:

```
js

items.map(item => <div>{item}</div>)
```

(You'll understand this deeply later.)

🧠 Universal Loop Rules

1️⃣ Always check loop condition carefully
2️⃣ Avoid modifying array while looping
3️⃣ Use `for...of` for values
4️⃣ Use `.map()` for transformation (later topic)

🔵 Next Topic: CONDITIONALS

5️⃣ Conditionals in JavaScript

Used for decision making.

🔹 if / else

```
js

let age = 18;

if (age >= 18) {
    console.log("Adult");
} else {
    console.log("Minor");
}
```

🔹 else if

```
js

let marks = 85;

if (marks >= 90) {
    console.log("A");
} else if (marks >= 75) {
    console.log("B");
} else {
    console.log("C");
}
```

🔹 Ternary Operator

Short form:

```
js

let result = age >= 18 ? "Adult" : "Minor";
```

🔹 Logical Operators

Operator	Meaning
&&	AND
`	
!	NOT

Example:

```
js

if (age >= 18 && age < 60) {
    console.log("Working age");
}
```

🔥 Important Web Dev Concept

Conditionals are used for:

- Showing/hiding UI
- Authentication checks
- Form validation
- Error handling
- Role-based rendering

📝 Your Tasks

🔹 Task 1

Check if a number is:

- Positive
- Negative
- Zero

🔹 Task 2

Given:

```js
let username = "admin";
let password = "1234";
```

Write condition:

- If username is **`"admin"`** AND password is **`"1234"`** → print "Login successful"
- Otherwise → print "Invalid credentials"

Do properly. Then we move to Arrays.

Good.

✅ CONDITIONALS — Expected Answers

🔹 Task 1

```js
js

let num = 5;

if (num > 0) {
    console.log("Positive");
} else if (num < 0) {
    console.log("Negative");
} else {
    console.log("Zero");
}
```

🔹 Task 2

```js
js

let username = "admin";
let password = "1234";

if (username === "admin" && password === "1234") {
    console.log("Login successful");
} else {
    console.log("Invalid credentials");
}
```

⚠️ Common Mistakes

1️⃣ Using = instead of ===

```js
js

if (username = "admin")  // ❌ assignment, not comparison
```

2️⃣ Using == instead of ===

3️⃣ Forgetting logical operator

4️⃣ Nested if mess instead of clean structure

🌐 Web Dev Memory Points

- Authentication checks
- Form validation
- Conditional rendering
- Role-based access
- API error handling

Example:

```js
if (!user) {
  showLogin();
}
```

🧠 Universal Conditional Rules

1️⃣ Always use ===
2️⃣ Keep conditions simple
3️⃣ Avoid deeply nested if blocks
4️⃣ Validate user input before using it

🔵 Next Topic: ARRAYS

6️⃣ Arrays in JavaScript

Arrays store multiple values.

```js
let numbers = [10, 20, 30];
```

🔹 Access Elements

```js
console.log(numbers[0]); // 10
```

Index starts from 0.

🔹 Add Elements

```js
numbers.push(40);   // add at end
numbers.unshift(5); // add at start
```

🔹 Remove Elements

```js
numbers.pop();    // remove last
numbers.shift();  // remove first
```

🔹 Length

```
js

console.log(numbers.length);
```

🔹 Important Methods

```
js

numbers.includes(20);
numbers.indexOf(30);
```

🔥 Important Web Dev Concept

Arrays are used for:

- API data lists
- Rendering products
- Users list
- Cart items
- Todo lists

Almost every dynamic UI uses arrays.

📝 Your Tasks

🔹 Task 1

Create an array of 5 numbers.

- Add one number
- Remove one number
- Print final array

🔹 Task 2

Given:

```js
let arr = [2, 4, 6, 8];
```

Multiply each element by 2 using a loop.
Print updated array.

Do it properly. Next we move to DOM basics.

Good.

✅ ARRAYS — Expected Answers

🔹 Task 1

```js
let nums = [10, 20, 30, 40, 50];

nums.push(60);    // add
nums.pop();       // remove last

console.log(nums);
```

🔹 Task 2

```js
let arr = [2, 4, 6, 8];

for (let i = 0; i < arr.length; i++) {
    arr[i] = arr[i] * 2;
}

console.log(arr); // [4, 8, 12, 16]
```

Alternative:

```js
for (let value of arr) {
    console.log(value * 2);
}
```

But this does not modify original array.

⚠️ Common Mistakes

1️⃣ Accessing wrong index

```js
arr[4] // ❌ undefined (if length is 4)
```

2️⃣ Forgetting **`.length`**

3️⃣ Using **`for...in`** incorrectly

4️⃣ Confusing mutation vs new array creation

🌐 Web Dev Memory Points

- APIs return arrays.
- UI lists are rendered from arrays.
- Shopping carts are arrays.
- Search/filter operates on arrays.
- State in React is often arrays.

Example:

```js
users.map(user => console.log(user.name));
```

Arrays = backbone of dynamic apps.

🧠 Universal Array Rules

1️⃣ Index starts from 0
2️⃣ Use **`.length`**
3️⃣ Prefer **`for...of`** or **`.map()`**
4️⃣ Understand mutation (**`push`**) vs transformation (**`map`**)
5️⃣ Arrays store references (important later)

🔵 Next Topic: DOM BASICS

Now we move from pure JS → Web Development.

7️⃣ DOM Basics

DOM = Document Object Model
It represents HTML as JavaScript objects.

HTML:

```html
<h1 id="title">Hello</h1>
<button>Click</button>
```

JS can access and modify it.

🔹 Selecting Elements

```js
const title = document.getElementById("title");
```

Modern preferred:

```js
const title = document.querySelector("#title");
const button = document.querySelector("button");
```

🔹 Changing Text

```js
title.textContent = "New Text";
```

🔹 Changing Style

```js
title.style.color = "red";
```

🔹 Adding Event Listener

```js
button.addEventListener("click", function() {
    title.textContent = "Button Clicked";
});
```

This is how websites become interactive.

🔥 Important Web Dev Concept

DOM manipulation is:

- Slow compared to normal JS
- Should be minimal
- React avoids direct DOM manipulation

📝 Your Tasks

Create simple HTML file:

```html
<h1 id="heading">Hello</h1>
<button id="btn">Change</button>
```

🔹 Task 1

When button is clicked:

- Change heading text to "Welcome"

🔹 Task 2

On button click:

- Change heading color to blue
- Change background color to yellow

Use **`addEventListener`**.

Do properly.
Next topic: Simple Interactivity (form + input handling).

Good.

✅ DOM BASICS — Expected Answers

HTML:

```html
<h1 id="heading">Hello</h1>
<button id="btn">Change</button>

<script src="script.js"></script>
```

🔹 Task 1

```js
const heading = document.querySelector("#heading");
const btn = document.querySelector("#btn");

btn.addEventListener("click", function() {
    heading.textContent = "Welcome";
});
```

🔹 Task 2

```js
btn.addEventListener("click", function() {
    heading.textContent = "Welcome";
    heading.style.color = "blue";
    document.body.style.backgroundColor = "yellow";
});
```

⚠️ Common Mistakes

1️⃣ Script placed above HTML → element not found
Fix: place **`<script>`** at bottom or use **`DOMContentLoaded`**.

2️⃣ Using wrong selector:

```js
document.querySelector("heading") // ❌ missing #
```

3️⃣ Writing **`onclick = function`** instead of **`addEventListener`**
Use **`addEventListener`** always.

4️⃣ Forgetting **`const`** for DOM elements.

🌐 Web Dev Memory Points

- Always store DOM elements in variables.
- Minimize direct DOM manipulation.
- Use event listeners, not inline HTML events.
- Every interaction on a website uses event listeners.

🧠 Universal DOM Rules

1️⃣ Select → Store → Modify
2️⃣ Always use **`querySelector`**
3️⃣ Keep JS separate from HTML
4️⃣ Avoid repetitive DOM calls
5️⃣ DOM is slower than normal JS

🔵 Final Topic: SIMPLE INTERACTIVITY

Now we combine:
Variables + Functions + Conditionals + DOM + Events

This is real web development.

8️⃣ Simple Interactivity (Input Handling)

HTML:

```html
<input type="text" id="nameInput">
<button id="submitBtn">Submit</button>
<p id="output"></p>
```

🔹 Reading Input Value

```js
const input = document.querySelector("#nameInput");
const button = document.querySelector("#submitBtn");
const output = document.querySelector("#output");

button.addEventListener("click", function() {
    const value = input.value;
    output.textContent = value;
});
```

Important:
`input.value` always returns STRING.

🔥 Add Logic

Example with validation:

```js
button.addEventListener("click", function() {
    const value = input.value;

    if (value === "") {
        output.textContent = "Please enter something";
    } else {
        output.textContent = "Hello " + value;
    }
});
```

This is real interactivity.

📝 Your Final Tasks

🔹 Task 1

Create:

- Input field
- Button
- Paragraph

When clicked:

- If input empty → show "Input required"
- Else → show "Welcome, <name>"

🔹 Task 2

Create simple counter:

HTML:

- Number display
- Increase button
- Decrease button

When clicked:

- Increase/decrease number
- Prevent it from going below 0

Use variables + conditionals + DOM.

Do this properly.

After this, I'll give:

- 🔥 Final web dev summary
- 📌 Core memory checklist
- 🚀 What you should learn next to become strong in React

Finish strong.

Good. Final evaluation.

✅ SIMPLE INTERACTIVITY — Expected Answers

🔹 Task 1 — Input Validation

```js
const input = document.querySelector("#nameInput");
const button = document.querySelector("#submitBtn");
const output = document.querySelector("#output");

button.addEventListener("click", function () {
    const value = input.value.trim();

    if (value === "") {
        output.textContent = "Input required";
    } else {
        output.textContent = `Welcome, ${value}`;
    }
});
```

Important:

- `.trim()` prevents `"     "` from passing validation.
- Input is always string.

🔹 Task 2 — Counter with Limit

```js
const display = document.querySelector("#count");
const incBtn = document.querySelector("#increase");
const decBtn = document.querySelector("#decrease");

let count = 0;

display.textContent = count;

incBtn.addEventListener("click", function () {
    count++;
    display.textContent = count;
});

decBtn.addEventListener("click", function () {
    if (count > 0) {
        count--;
        display.textContent = count;
    }
});
```

Key logic:

- State stored in variable
- UI updated after state change

- Condition prevents negative values

This is exactly how React state works conceptually.

🔥 FINAL WEB DEV SUMMARY

You now understand:

- Variables → Store state
- Data types → Understand behavior
- Functions → Reusable logic
- Loops → Process lists
- Conditionals → Control behavior
- Arrays → Dynamic data
- DOM → Connect JS to HTML
- Events → Make UI interactive

This is 70% of frontend fundamentals.

🌐 CORE WEB DEV CHECKLIST

Before touching React, you must be confident in:

☑️ Selecting elements
☑️ Handling click events
☑️ Reading input values
☑️ Updating DOM
☑️ Working with arrays
☑️ Writing small reusable functions
☑️ Using === always
☑️ Avoiding global variables

If any of these feel weak, you're not ready for React.

🧠 UNIVERSAL JAVASCRIPT MEMORY RULES

1️⃣ Default to `const`
2️⃣ Use `let` only when value changes
3️⃣ Never use `var`
4️⃣ Always use ===
5️⃣ Validate user input
6️⃣ Separate logic from UI
7️⃣ Don't repeat code → use functions
8️⃣ Understand mutation vs reassignment

9️⃣ Keep code readable
🔟 Test small pieces before building big

🚀 WHAT YOU SHOULD LEARN NEXT

To prepare for React:

1️⃣ Objects
2️⃣ Array methods:

- map
- filter
- reduce
  3️⃣ Destructuring
  4️⃣ Spread operator
  5️⃣ ES6 modules
  6️⃣ Async JS (Promises, fetch, async/await)

These are mandatory before serious React.

If you want, next we can:

- Do a **mini project using pure JS**
- Or move to **modern JS (ES6+) deep dive**
- Or directly start **React fundamentals with strong foundation**

Choose properly.

Good. Now we move from beginner → serious JavaScript.

We go one by one.
No skipping. No surface learning.

1️⃣ OBJECTS

Objects store data in key–value pairs.

```js
const user = {
    name: "Meghana",
    age: 21,
    isStudent: true
};
```

🔹 Access Values

```js
console.log(user.name);        // dot notation
console.log(user["age"]);      // bracket notation
```

🔹 Modify Values

```js
user.age = 22;
```

Even though **`user`** is **`const`**, properties can change.
You just can't reassign the whole object.

```js
user = {}  // ❌ not allowed
```

🔹 Add Property

```js
user.city = "Hyderabad";
```

🔹 Object with Function (Method)

```js
const user = {
    name: "Meghana",
    greet: function() {
        return "Hello " + this.name;
    }
};
```

`this` refers to the object.

⚠️ Common Mistakes

1️⃣ Confusing object with array
2️⃣ Forgetting **`this`** inside methods
3️⃣ Trying to reassign **`const`** object

🌐 Web Dev Importance

Objects are everywhere:

- API responses
- User data
- React props
- Configuration
- State objects

React is object-heavy.

📝 Tasks

🔹 Task 1

Create an object:

- title
- price
- inStock

Print:

```php-template
Product: <title>, Price: <price>, Available: <inStock>
```

🔹 Task 2

Add a method to object:

- **`applyDiscount(percent)`**
- It reduces price by percent
- Print updated price

Do it. Then we move to **`map`**, **`filter`**, **`reduce`**.

This is where JavaScript becomes powerful.

Good. Expected answers first.

✅ OBJECTS — Expected Answers

🔹 Task 1

```
js

const product = {
    title: "Laptop",
    price: 50000,
    inStock: true
};

console.log(`Product: ${product.title}, Price: ${product.price}, Available:
${product.inStock}`);
```

🔹 Task 2

```
js

const product = {
    title: "Laptop",
    price: 50000,
    inStock: true,
    applyDiscount: function(percent) {
        this.price = this.price - (this.price * percent / 100);
        console.log("Updated Price:", this.price);
    }
};

product.applyDiscount(10); // 45000
```

Key:

- **`this.price`**
- Not **`price`**
- Method modifies internal property

⚠️ Common Mistakes

1️⃣ Writing:

```
js

price = price - ...
```

Instead of:

```
js

this.price
```

2️⃣ Forgetting **`return`** if needed.

3️⃣ Confusing object mutation vs reassignment.

🌐 Web Dev Memory Points

- API returns arrays of objects.
- Products, users, posts = objects.
- React props = objects.
- State often = object.

You must be comfortable reading nested objects:

```js
user.address.city
```

🔥 Now Important Part: ARRAY METHODS

This separates beginners from strong JS devs.

2️⃣ map()

Transforms array → returns NEW array.

```js
let nums = [1, 2, 3];

let doubled = nums.map(num => num * 2);

console.log(doubled); // [2, 4, 6]
```

- Does not modify original array.
- Returns new array.

📝 Tasks

Task 1

Given:

```js
let prices = [100, 200, 300];
```

Return new array with 10% added to each.

Task 2

Given:

```js
let names = ["meghana", "raj", "sita"];
```

Return array with names in uppercase.

Do mentally. Then continue reading.

3️⃣ filter()

Filters based on condition → returns NEW array.

```js
let nums = [10, 15, 20, 25];

let result = nums.filter(num => num > 15);

console.log(result); // [20, 25]
```

📝 Tasks

Task 1

Given:

```js
let numbers = [1,2,3,4,5,6];
```

Return only even numbers.

Task 2

Given:

```js
let users = [
  {name: "A", active: true},
  {name: "B", active: false}
];
```

Return only active users.

4️⃣ reduce()

Reduces array → single value.

```
js

let nums = [1,2,3,4];

let sum = nums.reduce((acc, curr) => acc + curr, 0);

console.log(sum); // 10
```

- `acc` = accumulator
- `curr` = current value
- `0` = initial value

📝 Tasks

Task 1

Find sum of:

```
js

[10, 20, 30]
```

Task 2

Count total price:

```
js

let cart = [
  {price: 100},
  {price: 200},
  {price: 300}
];
```

Return total price.

⚠️ Important Difference

Method	Returns	Purpose
map	array	transform
filter	array	condition-based selection
reduce	single value	aggregate

These are heavily used in React.

Now moving forward.

5️⃣ Destructuring

Extract values easily.

```js
const user = {
    name: "Meghana",
    age: 21
};

const { name, age } = user;
```

Instead of:

```js
const name = user.name;
```

Array destructuring:

```js
let arr = [10, 20];
let [a, b] = arr;
```

6️⃣ Spread Operator (...)

Copies arrays/objects.

```js
let arr1 = [1,2];
let arr2 = [...arr1, 3, 4];
```

Object copy:

```js
let user = {name: "A"};
let updated = {...user, age: 25};
```

Very important in React state updates.

7️⃣ ES6 Modules

Export:

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

Import:

```js
import { add } from "./file.js";
```

Default export:

```js
export default function greet() {}
```

Import:

```js
import greet from "./file.js";
```

Used everywhere in React.

8️⃣ Async JS

JavaScript is asynchronous.

🔹 Promise

```js
let promise = new Promise((resolve, reject) => {
    resolve("Success");
});
```

🔹 fetch()

```js
fetch("https://api.example.com")
  .then(res => res.json())
  .then(data => console.log(data));
```

🔹 async / await (cleaner)

```js
async function getData() {
    const res = await fetch("https://api.example.com");
    const data = await res.json();
    console.log(data);
}
```

Modern standard.

🔥 What You Must Be Comfortable With Before React

☑ map
☑ filter
☑ reduce
☑ destructuring
☑ spread
☑ async/await

If any of these feel weak → React will feel confusing.

Now answer honestly:

Which one feels weakest right now?

Good. Now we go deeper — not surface explanation.
This is **intermediate JavaScript mastery layer**.

🔥 3️⃣ Destructuring (Deep Understanding)

🔹 Object Destructuring

Instead of:

```js
const user = { name: "Meghana", age: 21 };

const name = user.name;
const age = user.age;
```

We write:

```js
const { name, age } = user;
```

🔹 Rename While Destructuring

```js
const { name: userName } = user;
console.log(userName);
```

Very useful in React.

🔹 Default Values

```js
const { city = "Unknown" } = user;
```

Prevents undefined errors.

🔹 Nested Destructuring

```js
const user = {
  name: "Meghana",
  address: {
    city: "Hyderabad"
  }
};

const { address: { city } } = user;
```

Used frequently with API responses.

🧠 Why Important?

In React:

```js
function Card({ title, price }) {
   return <h1>{title}</h1>;
}
```

That is destructuring props.

🔥 4️⃣ Spread Operator (...)

🔹 Copy Array

```js
let arr1 = [1,2,3];
let arr2 = [...arr1];
```

Without spread:

```js
let arr2 = arr1;  // reference copy ❌
```

Now modifying arr2 modifies arr1. Dangerous.

🔹 Add Elements

```js
let newArr = [...arr1, 4, 5];
```

🔹 Copy Object

```js
let user = { name: "A" };
let updated = { ...user, age: 21 };
```

React state updates rely on this.

🔥 Important Concept: Reference vs Copy

Objects & arrays are reference types.

```js
let a = [1,2];
let b = a;
b.push(3);

console.log(a); // [1,2,3]
```

Spread prevents this issue.

🔥 5️⃣ ES6 Modules

Modern JS is modular.

Named Export

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

Import:

```js
import { add } from "./math.js";
```

Default Export

```js
export default function greet() {}
```

Import:

```js
import greet from "./file.js";
```

React uses default export for components.

🔥 6️⃣ Async JavaScript (Very Important)

JavaScript is single-threaded but non-blocking.

🔹 Promise

Represents future value.

```js
const promise = new Promise((resolve, reject) => {
   setTimeout(() => {
      resolve("Done");
   }, 2000);
});
```

🔹 fetch()

```js
fetch("https://jsonplaceholder.typicode.com/posts")
  .then(res => res.json())
  .then(data => console.log(data));
```

🔹 async / await (Modern Way)

```
js

async function getData() {
   try {
      const res = await fetch("https://jsonplaceholder.typicode.com/posts");
      const data = await res.json();
      console.log(data);
   } catch (error) {
      console.log("Error:", error);
   }
}
```

Always use try/catch.

🔥 Important Async Concepts

1️⃣ Code does not wait unless **`await`** is used
2️⃣ Always handle errors
3️⃣ fetch returns promise
4️⃣ async function always returns promise

🧠 Final Readiness Checklist for React

If you can confidently:

- Transform arrays using map
- Filter data
- Calculate totals using reduce
- Update objects using spread
- Destructure props
- Fetch API data with async/await

Then React will feel natural.

If not → React will feel confusing.

Now serious question:

Do you want:

A) Deep practice problems on these
B) Mini real-world project using all of them
C) Direct jump into React fundamentals
D) Brutal interview-style JS drills

Choose one.