30 Days JavaScript Series

Table of Content

30 Days JavaScript Series

Day 1: JavaScript Introduction

What JavaScript is and why every web developer needs it

1 What is JavaScript?

- JavaScript is a programming language of the web.
- It makes web pages interactive and dynamic.
- It can update content, control multimedia, animate images, and much more.
- Essential for front-end and also used in back-end (Node.js).

2 Why & Where JavaScript?

Why use JavaScript?

- Adds interactivity to websites
- Validates forms
- Updates content without relooding
- Creates animations & effects
- Improves user experience

Where JavaScript runs

- Web Browsers (Chrome, Firefox, Edge)
- Server-side with Node.js
- Mobile Apps (React Native)
- Desktop Apps (Electron)
- Games & More

JavaScript + HTML

HTML gives structure to the webpage.

```
<h1>Hello!</h1>
<button>Click me</button>
```

HTML is the skeleton.

JavaScript + CSS

CSS styles the webpage and makes it beautiful.

```
body {
  background: #f5f5f5;
  color: #333;
}
```

CSS is the skin.

3 Summary

JS

JavaScript makes websites interactive and dynamic.

It runs in browsers and many other environments.

</>

Works together with HTML (structure) and CSS (style).

A must-have skill for every modern web developer.

Follow 👉 codingwithparvez

30 Days JavaScript Series

Day 2: Adding JavaScript to a Web Page

Learn how to connect JavaScript with HTML

1 Adding JavaScript to HTML

- JavaScript adds interactivity to web pages
- Connected using the `<script>` tag
- Can be added in multiple ways
- Recommended method: **External JavaScript file**

2 Types of JavaScript Integration

Inline JavaScript

- Added directly inside HTML elements
- Quick for small actions
- Not reusable

```
<button onclick="alert('Hi')">
  Click Me
</button>
```

Quick but not recommended

Internal Script

- Written inside `<script>` tag
- Placed in same HTML file
- Good for small projects

```
<script>
    alert("Hello");
</script>
```

Single-page use

External Script

- Stored in separate .js file
- Reusable and organized
- Industry best practice

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

Recommended approach

3 Best Practice

Inline
Quick testing only

Internal
Small projects

External
Reusable & scalable

Use **External JavaScript files** for cleaner and maintainable code.

Follow 👉 codingwithparvez

30 Days JavaScript Series

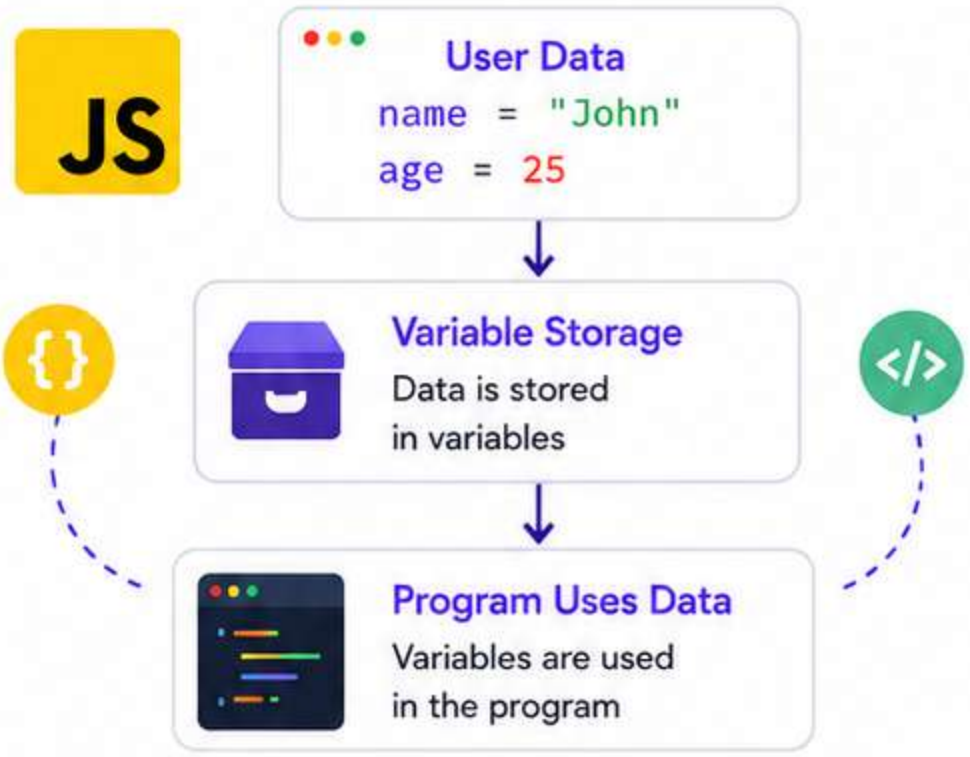
Day 3: Variables

Store and manage data in JavaScript

1 What are Variables?

- Variables store data values
- Used to save information for later use
- Makes code reusable and dynamic
- Think of variables as labeled containers

```
name = "John"
age = 25
```



2 Types of Variables

let

- Modern variable declaration
- Value can be changed
- Block scoped

```
let age = 25;
age = 26;
```

Recommended for changing values

const

- Constant value
- Cannot be reassigned
- Block scoped

```
const PI = 3.14;
```

Use for fixed values

var

- Older way to declare variables
- Function scoped
- Avoid in modern JavaScript

```
var name = "John";
```

Legacy approach

3 Naming Rules

Valid Names

userName
totalPrice
studentAge

Invalid Names

1name
user-name
var

Tips

Use meaningful names

Best Practice

Prefer `let` and `const`

Variable Naming Rules

- Start with letter, _ or $
- Case Sensitive
- No spaces allowed
- Use camelCase naming

Follow 👉 codingwithparvez

30 Days JavaScript Series

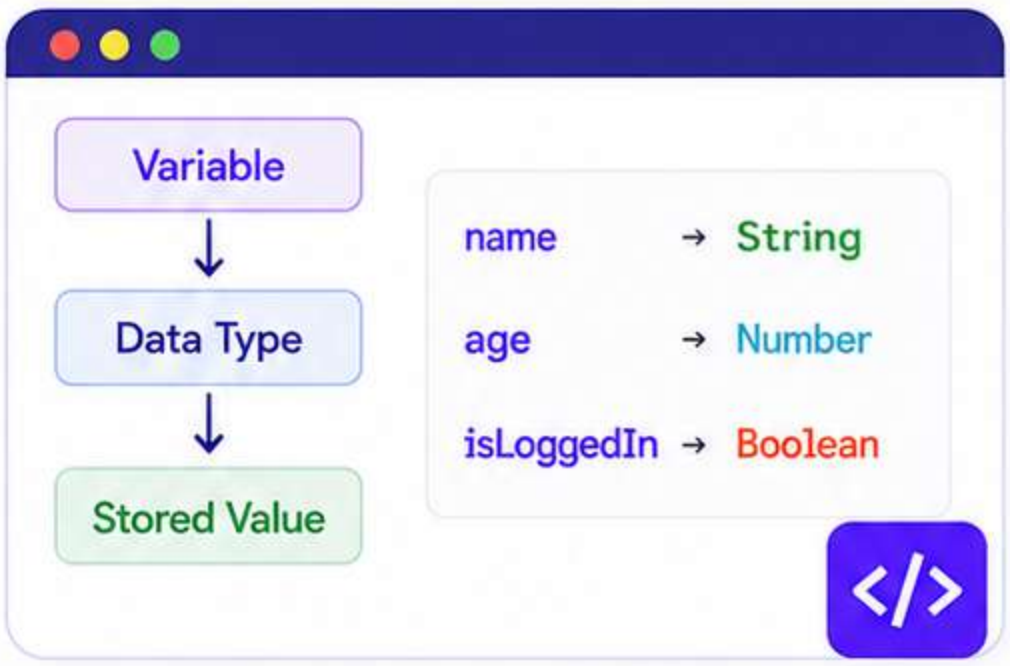
Day 4: Data Types

Understand different kinds of data

1 What are Data Types?

- Data types define the kind of value stored
- Every variable holds a specific type of data
- JavaScript uses different data types for different purposes
- Understanding data types helps prevent bugs

JS



2 Common JavaScript Data Types

String

- Stores text values
- Written inside quotes
- Used for names and messages

```
let name = "John";
```

Text Data

Number

- Stores numeric values
- Supports integers and decimals
- Used for calculations

```
let age = 25;
```

Numeric Data

Boolean

- Only true or false
- Used for conditions
- Common in decision making

```
let isActive = true;
```

True / False

Null

- Represents empty value
- Intentionally assigned
- Means "nothing here"

```
let user = null;
```

Empty Value

Undefined

- Variable declared but no value assigned
- Default JavaScript state
- Value is unknown

```
let city;
```

Not Assigned Yet

3 Quick Summary

String	Number	Boolean	Null	Undefined
Text	Numbers	True / False	Empty Value	No Value Assigned

Common Examples

`"Hello"` → String | `100` → Number | `true` → Boolean | `null` → Null | `undefined` → Undefined

Follow 👉 codingwithparvez

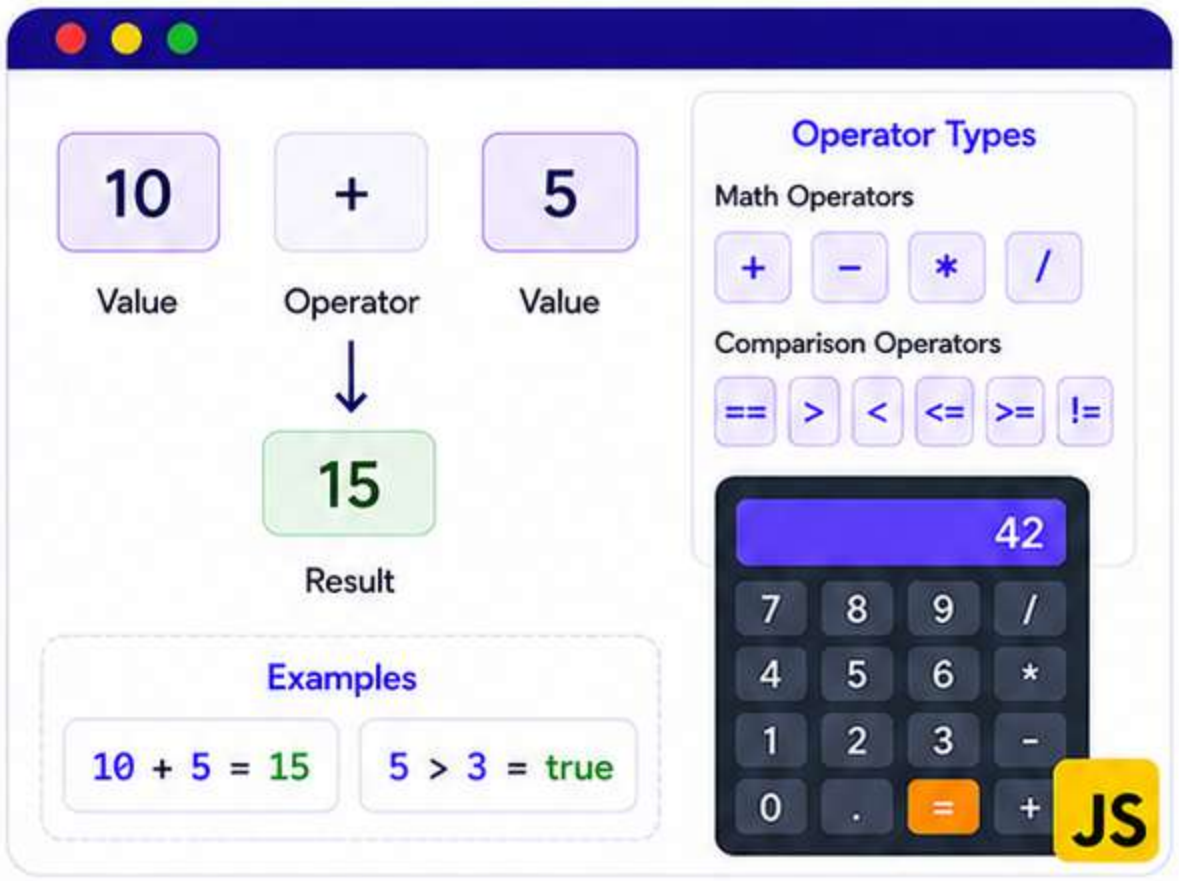
30 Days JavaScript Series

Day 5: Operators

Perform calculations and comparisons

1 What are Operators?

- Operators perform actions on values and variables
- Used for calculations and comparisons
- Essential for decision making and logic
- Help manipulate and update data



2 Common JavaScript Operators

Arithmetic Operators

- Used for calculations
- Add, subtract, multiply, divide
- Returns numeric results

```
10 + 5
10 - 5
10 * 5
10 / 5
```

Math Operations

Assignment Operators

- Assign values to variables
- Update existing values
- Frequently used in programs

```
let x = 10;
x += 5;
```

Assign Values

Comparison Operators

- Compare two values
- Return true or false
- Used in conditions

```
10 > 5
10 == 10
5 !== 3
```

Compare Values

Logical Operators

- Combine conditions
- Work with Boolean values
- Used in decision making

```
true && true
true || false
!true
```

Logic Operations

3 Quick Summary

Arithmetic — Math calculations

Assignment — Store and update values

Comparison — Returns true or false

Logical — Combine conditions

Most Common Operators

+	=	==	===	>	&&	||	!
Addition	Assignment	Equality	Strict Equality	Greater Than	Logical AND	Logical OR	Logical NOT

Follow 👉 codingwithparvez

30 Days JavaScript Series

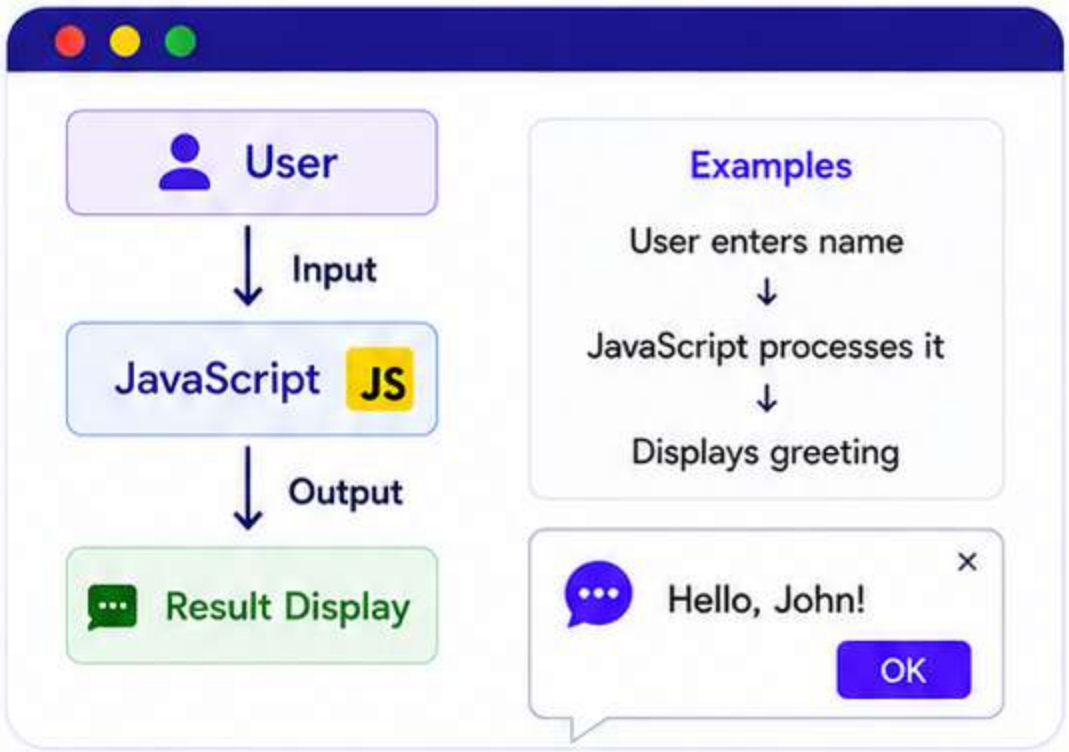
Day 6: User Input & Output

Interact with users using JavaScript

1 What is Input & Output?

- Input means receiving data from users
- Output means displaying information
- Makes web pages interactive
- Essential for user communication

JS



2 Common Input & Output Methods

alert()

- Displays a popup message
- Used for notifications
- Requires user attention

```
alert("Hello!");
```

Show Messages

prompt()

- Gets input from users
- Shows input box
- Returns entered value

```
let name =
prompt("Your Name?");
```

Receive Input

console.log()

- Displays output in console
- Useful for debugging
- Developers use it often

```
console.log("Hello");
```

Debug & Test

3 Quick Summary

alert() — Show messages

prompt() — Get user input

console.log() — Display console output

Typical Flow

User Input → JavaScript Processing → Output Result

Example:

```
let name = prompt("Name?");
alert("Hello " + name);
```

Follow 👉 codingwithparvez

30 Days JavaScript Series

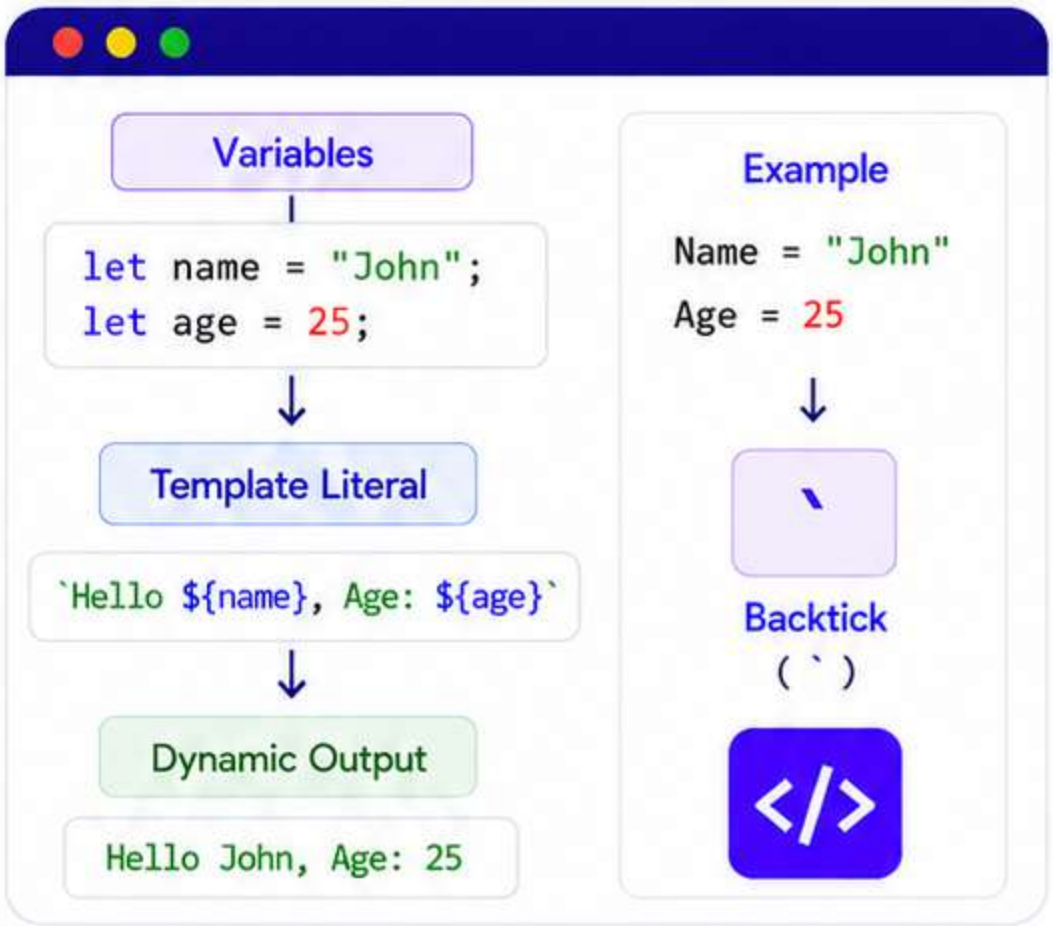
Day 7: Template Literals

Create cleaner and dynamic strings

1 What are Template Literals?

- Modern way to create strings
- Uses backticks instead of quotes
- Easily insert variables into text
- Supports multi-line strings

JS



2 Key Features

Backticks

- Uses backtick characters
- Alternative to quotes
- Required for template literals

```
`Hello World`
```

Modern String Syntax

${ } Variable Interpolation

- Insert variables directly
- Cleaner than concatenation
- Easy to read

```
let name = "John";
`Hello ${name}`
```

Dynamic Values

Multi-line Strings

- Write strings on multiple lines
- No special characters needed
- Improves readability

```
`Line 1
 Line 2
 Line 3`
```

Multiple Lines

3 Quick Summary

Backticks
Modern string syntax

${ } Insert Variables
Add dynamic values

Multi-line Text
Write across lines

Before:

```
"Hello " + name
```

Old way

After:

```
`Hello ${name}`
```

New way

- Cleaner
- Easier to Read
- Modern JavaScript

Follow 👉 codingwithparvez

30 Days JavaScript Series

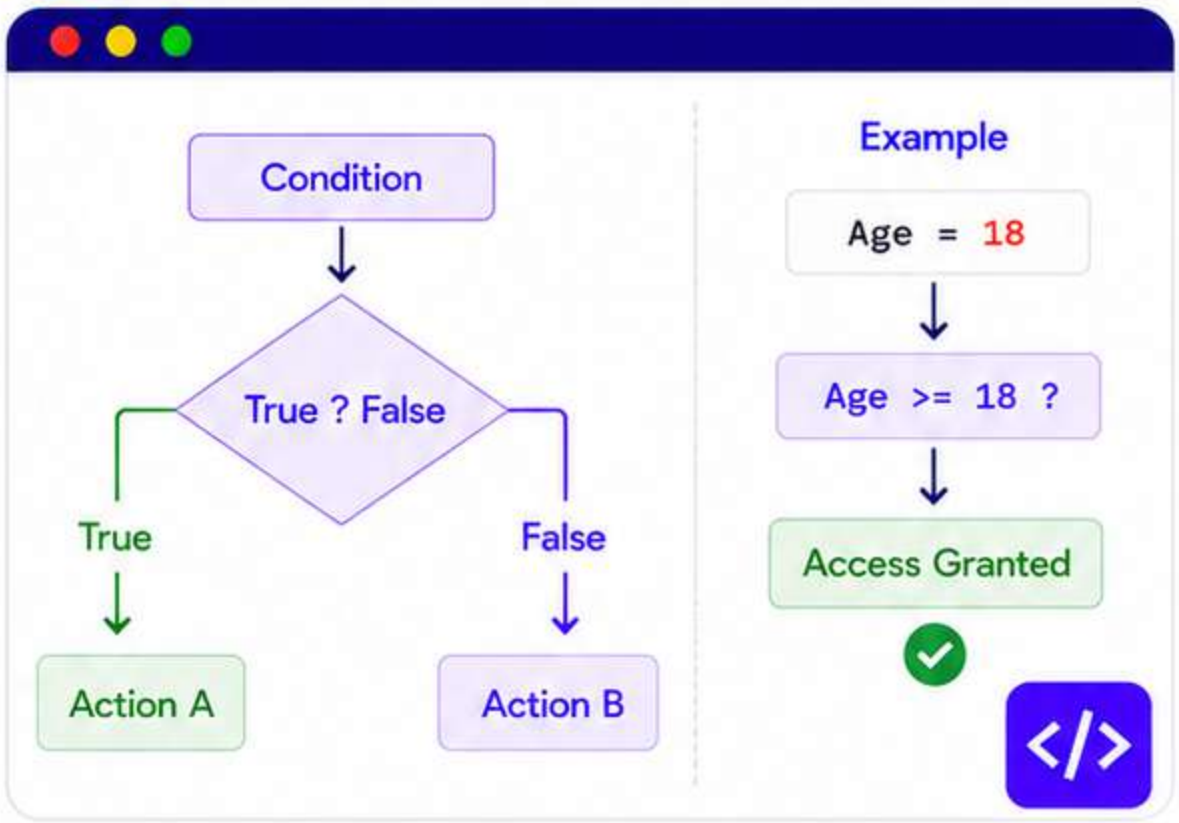
Day 8: Conditional Statements

Make decisions in your code

1 What are Conditional Statements?

- Used to make decisions in programs
- Executes code based on conditions
- Helps create dynamic applications
- Returns different outcomes for different situations

JS



2 Types of Conditional Statements

if

- Runs code when condition is true
- Simplest conditional statement
- Used for single decisions

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

Single Condition

else

- Runs when condition is false
- Used as alternative path
- Works with if statement

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

Alternative Action

else if

- Checks multiple conditions
- Executes first matching condition
- Useful for multiple outcomes

```
if (score >= 90) {
  grade = "A";
}
else if (score >= 70) {
  grade = "B";
}
```

Multiple Conditions

3 Quick Summary

if — Run when true

else — Run when false

else if — Check more conditions

Decision Flow

if	→	Check Condition
else if	→	Check Another Condition
else	→	Default Action

Example

```
if (marks >= 40)
  Pass
else
  Fail
```

- Smart Decisions
- Dynamic Programs
- Core JavaScript Skill

Follow 👉 codingwithparvez

30 Days JavaScript Series

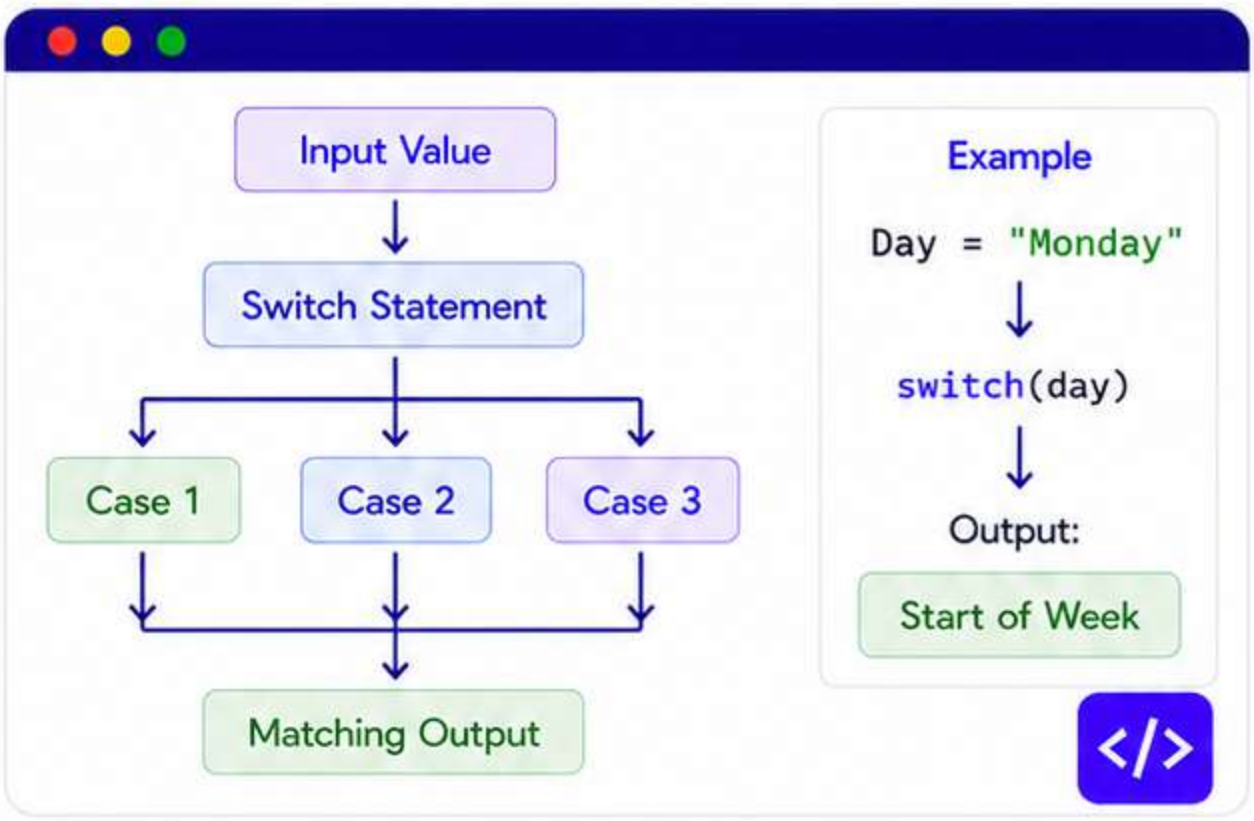
Day 9: Switch Statement

Handle multiple conditions efficiently

1 What is a Switch Statement?

- Used to handle multiple possible conditions
- Cleaner alternative to many else if statements
- Executes matching case blocks
- Improves code readability

JS



2 Switch Statement Essentials

Syntax

- Uses switch keyword
- Matches values with cases
- Executes matching block

```
switch(day) {
  case "Mon":
    break;
}
```

Basic Structure

Use Cases

- Menus
- Days of week
- User selections
- Multiple options

```
switch(role) {
  case "Admin":
    ...
}
```

Real-world Usage

Break Statement

- Stops execution
- Prevents fall-through
- Used after each case

```
case "A":
  break;
```

Important Keyword

3 Example

```
let day = "Friday";
switch(day) {
  case "Monday":
    result = "Start";
    break;
  case "Friday":
    result = "Weekend Soon";
    break;
  default:
    result = "Normal Day";
}
```

Output

Weekend Soon

Quick Summary

- **Syntax** — Match values with cases
- **Use Cases** — Handle multiple options
- **Break** — Stops execution
- **Default** — Runs when no case matches

Switch vs Else If

- Cleaner Code
- Easier to Read
- Best for Multiple Fixed Values

Follow 👉 codingwithparvez

30 Days JavaScript Series

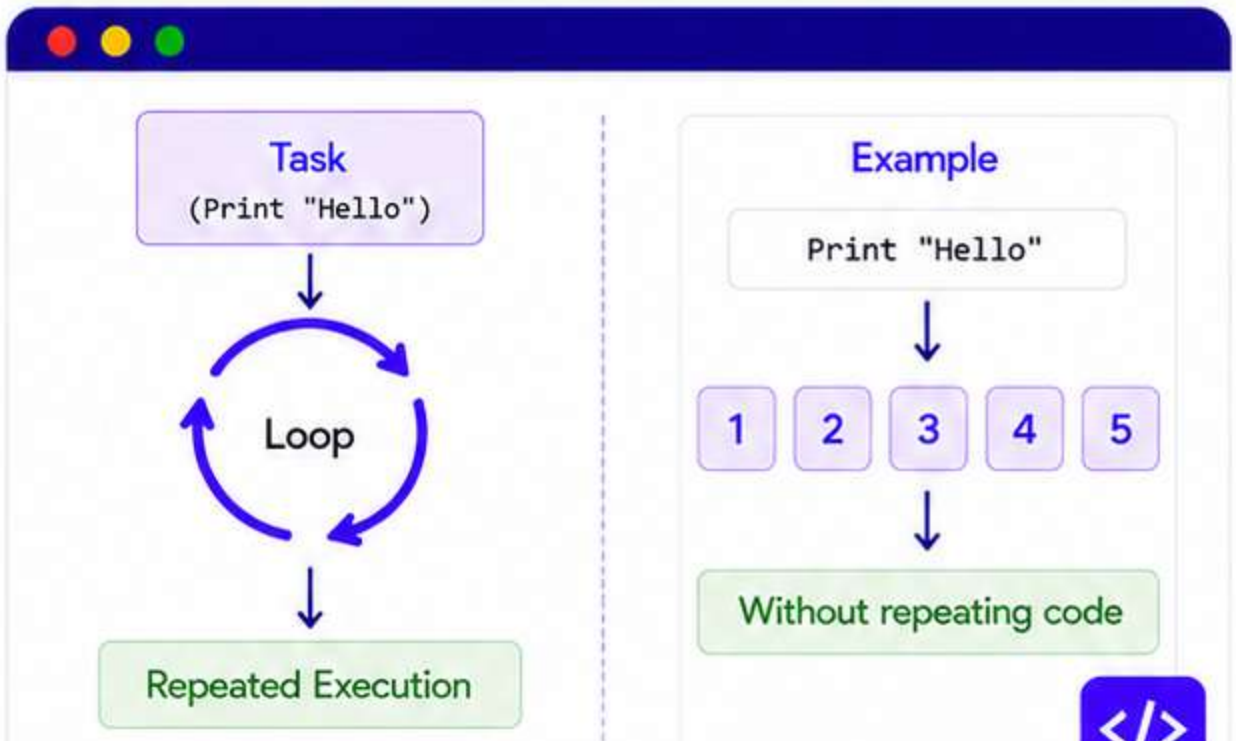
Day 10: Loops Introduction

Repeat tasks automatically

1 What are Loops?

- Loops repeat code automatically
- Avoid writing the same code multiple times
- Save time and reduce errors
- Essential for handling repetitive tasks

JS



2 Loop Fundamentals

Why Loops?

- Automate repetitive tasks
- Reduce duplicate code
- Improve efficiency

```
console.log("Hello");
console.log("Hello");
console.log("Hello");
```

Problem Without Loops

Loop Structure

- Start value
- Condition check
- Update value

```
for(let i = 1;
  i <= 5;
  i++)
```

Basic Components

Loop Result

- Runs until condition fails
- Executes repeatedly
- Produces multiple outputs

```
1
2
3
4
5
```

Repeated Execution

3 Common Use Cases

Display Lists · Count Numbers · Process Data · Game Logic

Loop Workflow

Start → Check Condition → Run Code → Update Value → Repeat

Example

Print numbers 1 to 5 automatically

Instead of writing:

```
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
```

- Cleaner Code
- Less Repetition
- Better Productivity

30 Days JavaScript Series

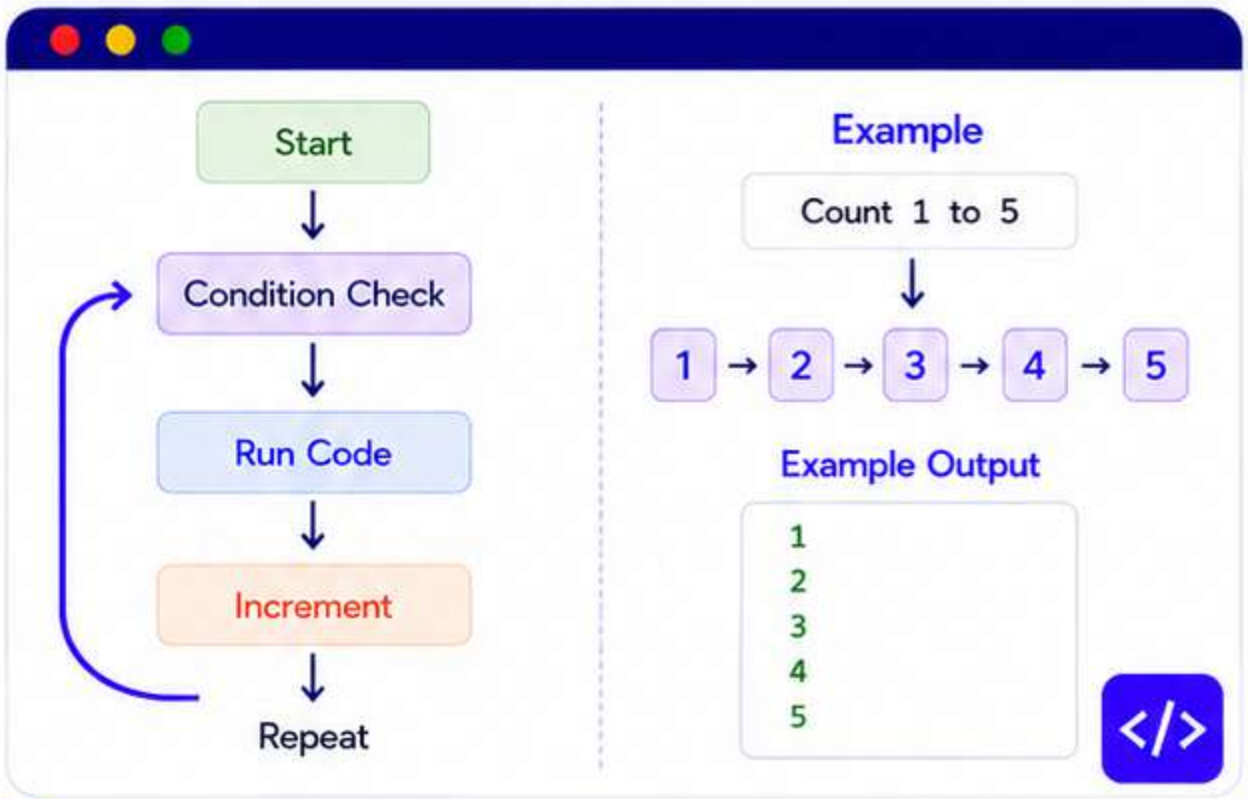
Day 11: For Loop

Repeat code with control

1 What is a For Loop?

- Most commonly used loop in JavaScript
- Repeats code a specific number of times
- Gives full control over repetition
- Perfect when iteration count is known

JS



2 For Loop Essentials

Syntax

- Has 3 parts
- Initialization
- Condition
- Update expression

```
for(let i = 1;
    i <= 5;
    i++) {
     console.log(i);
}
```

Basic Structure

Increment

- Increases value
- Uses ++ operator
- Moves loop forward

```
i++;
```

Output:

1 → 2 → 3 → 4 → 5

Count Up

Decrement

- Decreases value
- Uses -- operator
- Moves loop backward

```
i--;
```

Output:

5 → 4 → 3 → 2 → 1

Count Down

3 Practical Examples

Print Numbers

```
for(let i=1;
    i<=5;
    i++)
```

Output:

1 2 3 4 5

Reverse Count

```
for(let i=5;
    i>=1;
    i--)
```

Output:

5 4 3 2 1

Repeat Text

```
for(let i=1;
    i<=3;
    i++)
```

Output:

```
Hello
Hello
Hello
```

For Loop Formula

Initialization → Condition → Execute Code → Update → Repeat

- Cleaner Code
- Controlled Repetition
- Most Used JavaScript Loop

Follow 👉 codingwithparvez

30 Days JavaScript Series

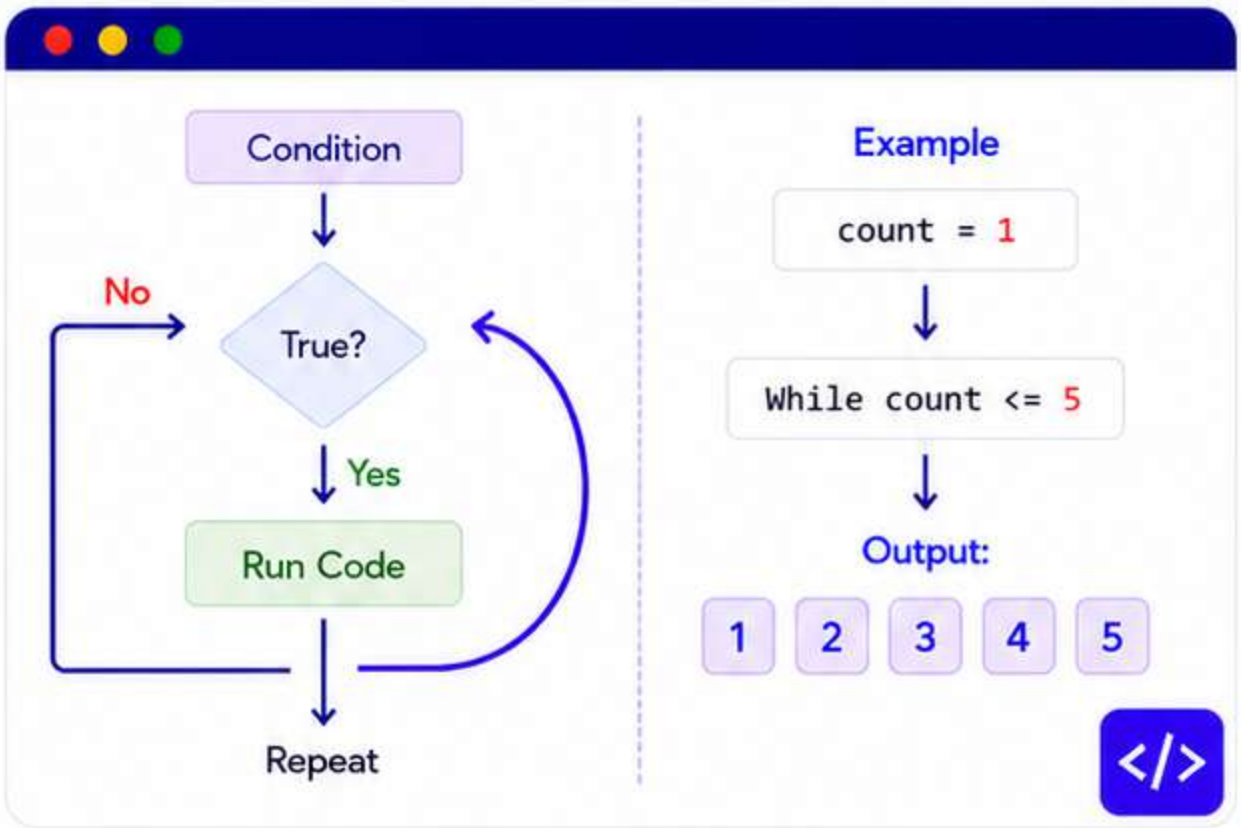
Day 12: While & Do While Loops

✦ Run code until a condition changes ✦

1 What are While Loops?

- Used when the number of repetitions is unknown
- Runs while a condition remains true
- Ideal for condition-based repetition
- Common in user input and game logic

JS



2 While vs Do While

While Loop

- Checks condition first
- May run zero times
- Executes only if condition is true

```
let i = 1;
while(i <= 5){
  console.log(i);
  i++;
}
```

Output: 1 2 3 4 5

Condition First

Do While Loop

- Executes code first
- Checks condition later
- Runs at least once

```
let i = 1;
do{
  console.log(i);
  i++;
}while(i <= 5);
```

Output: 1 2 3 4 5

Execute First

3 Key Differences

WHILE LOOP	DO WHILE LOOP	VISUAL EXAMPLE
Condition checked first	Code runs first	Condition = false
Can execute zero times	Executes at least once	While Loop → Runs 0 Times / Do While Loop → Runs 1 Time
Safer for most cases	Useful when first execution is required	

When to Use?

While Loop
When condition must be checked before execution

Do While Loop
When code should run at least once

- Flexible Repetition
- Condition-Based Execution
- Essential Loop Types

Follow 👉 codingwithparvez

30 Days JavaScript Series

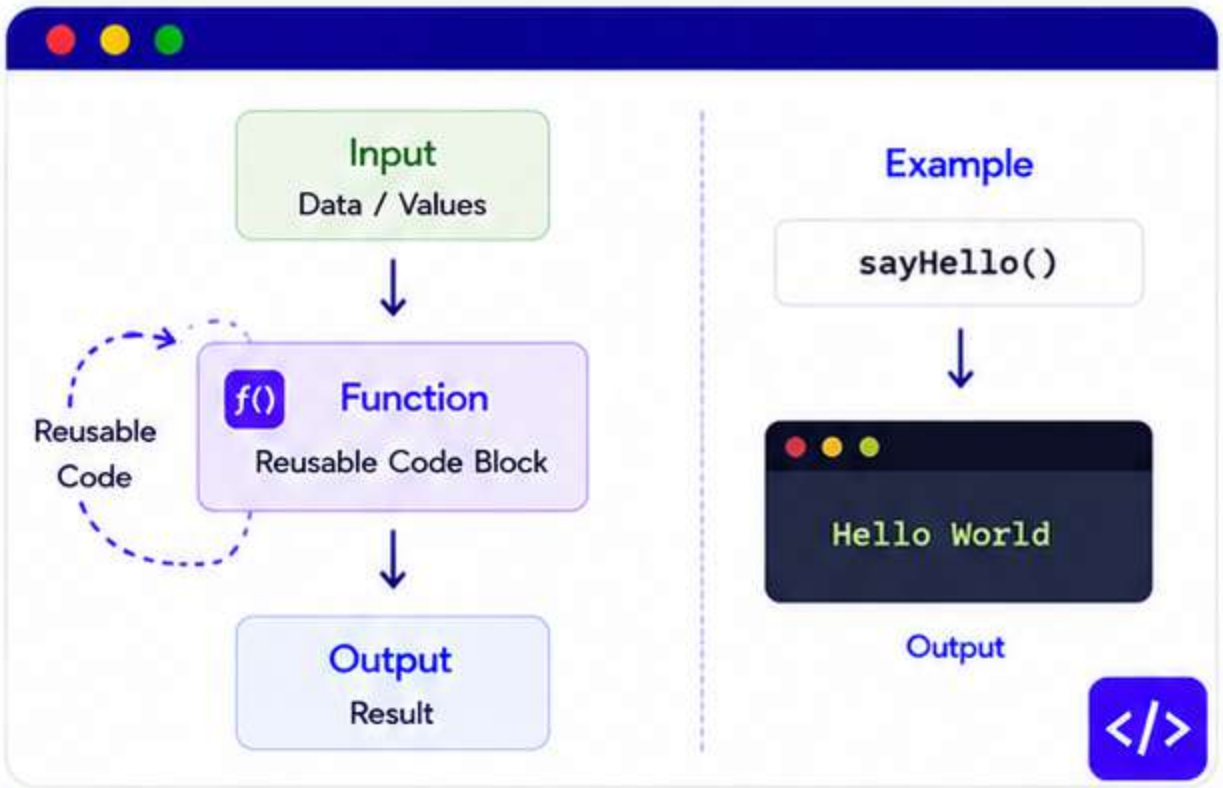
Day 13: Functions Basics

Write reusable blocks of code

1 What are Functions?

- Functions are reusable blocks of code
- Perform specific tasks when called
- Reduce code repetition
- Make programs organized and easier to maintain

JS



2 Function Essentials

Function Declaration

- Defines a function
- Contains reusable code
- Runs only when called

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

Create a Function

Calling Functions

- Executes the function
- Runs stored instructions
- Can be called multiple times

```
greet();
```

Output:

```
Hello
```

Run a Function

Parameters

- Accept input values
- Make functions flexible
- Customize output

```
function greet(name){
  console.log(name);
}
```

Function Inputs

3 Practical Example

```
1  function greet(name){
2    console.log(
3      "Hello " + name
4    );
5  }
6
7  greet("John");
```

Output:

```
Hello John
```

Declaration
Create the function

Call
Execute the function

Parameters
Pass data into function

Benefit
Write once, use many times

Function Workflow

Create Function </> → Add Parameters → Call Function → Get Output

- Reusable Code
- Cleaner Programs
- Essential JavaScript Skill

Follow 👉 codingwithparvez

30 Days JavaScript Series

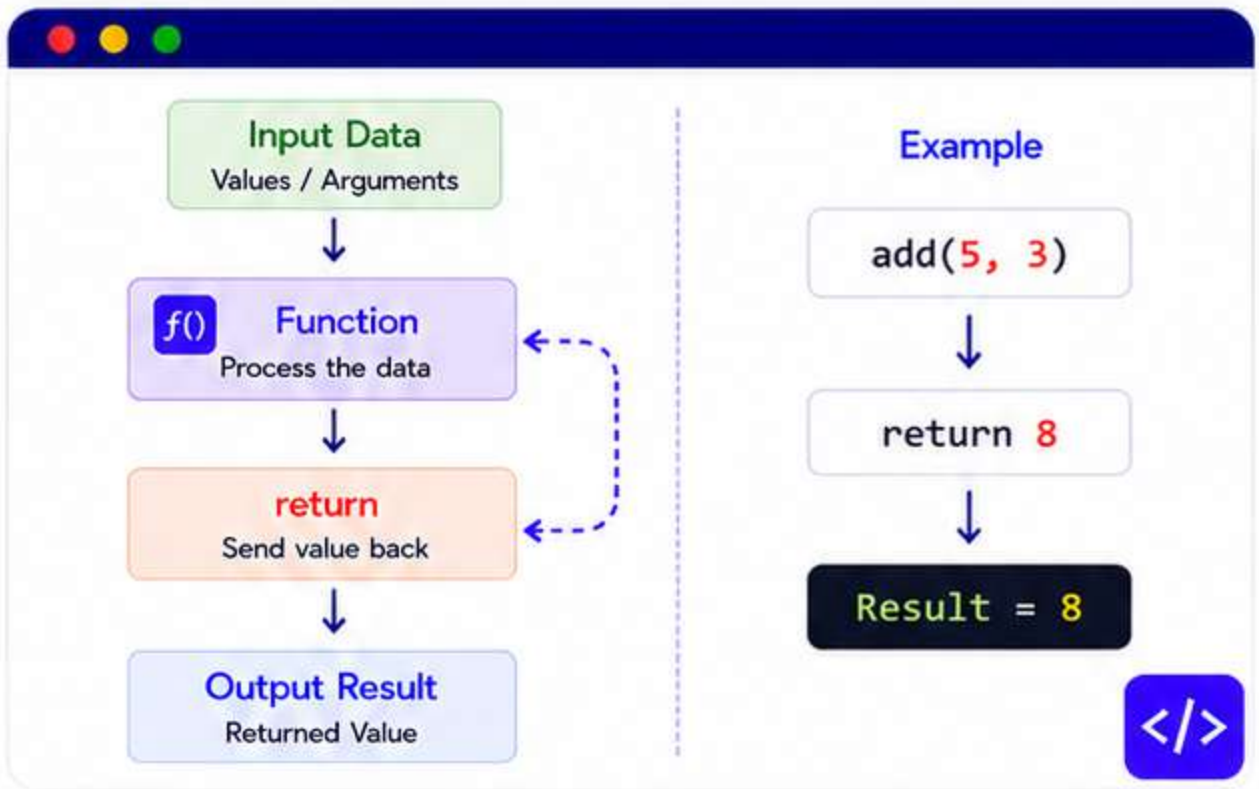
Day 14: Function Return Values

Send data back from functions

1 What is Return?

- return sends a value back from a function
- Ends function execution immediately
- Allows functions to produce results
- Makes functions reusable and powerful

JS



2 Understanding Return

return Keyword

- Sends value back
- Stops function execution
- Returns a result

```
function greet(){
  return "Hello";
}
```

Return a Value

Return Example

- Calculate inside function
- Return final answer
- Use result later

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

Return Calculation

Store Returned Value

- Save function result
- Reuse returned data
- Common practice

```
let total =
  add(5,3);
```

Store Results

3 Why Returns Matter

Return Values
Functions produce results

Reusable Data
Store and use later

Calculations
Return answers easily

Cleaner Code
Avoid unnecessary variables

```
1  function multiply(a,b){
2    return a*b;
3  }
4  let result =
5    multiply(4,5);
```

Output:

20

Without Return

Function performs action only

No value returned

VS

With Return

Function produces reusable value

Value returned & used

- Better Logic
- Reusable Functions
- Essential JavaScript Concept

Follow 👉 codingwithparvez

30 Days JavaScript Series

Day 15: Arrow Functions

Modern way to write functions

1 What are Arrow Functions?

- Introduced in ES6 (ECMAScript 2015)
- Shorter syntax for writing functions
- Cleaner and easier to read
- Widely used in modern JavaScript

JS

Traditional Function

```
function greet() {
  return "Hello";
}
```

↓

Arrow Function

```
const greet = () => "Hello";
```

↓

Cleaner Code ✓

Example

```
greet()
```

↓

```
return "Hello"
```

↓

```
Result = "Hello"
```

</>

2 Arrow Function Essentials

Syntax

- Uses => operator
- Shorter than regular functions
- Easy to write and read

```
const greet = () => {
  console.log("Hi");
};
```

Modern Syntax

Examples

- Works with parameters
- Can return values
- Ideal for small functions

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

Simple Functions

Benefits

- Less code
- Better readability
- Popular in React and modern JS

```
const square =
  n => n * n;
```

Cleaner Code

3 Traditional vs Arrow Function

Traditional Function

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

Arrow Function

```
const greet =
  (name) =>
  "Hello " + name;
```

Output

```
Hello John
```

Quick Summary

- **Arrow Syntax** — Uses =>
- **Modern JavaScript** — ES6 Feature
- **Supports Parameters** — Accepts Input
- **Cleaner Code** — Less Boilerplate

Why Use Arrow Functions?

=>

- ✓ Shorter Syntax
- ✓ Easier to Read
- ✓ Common in React
- ✓ Modern JavaScript Standard

30 Days JavaScript Series

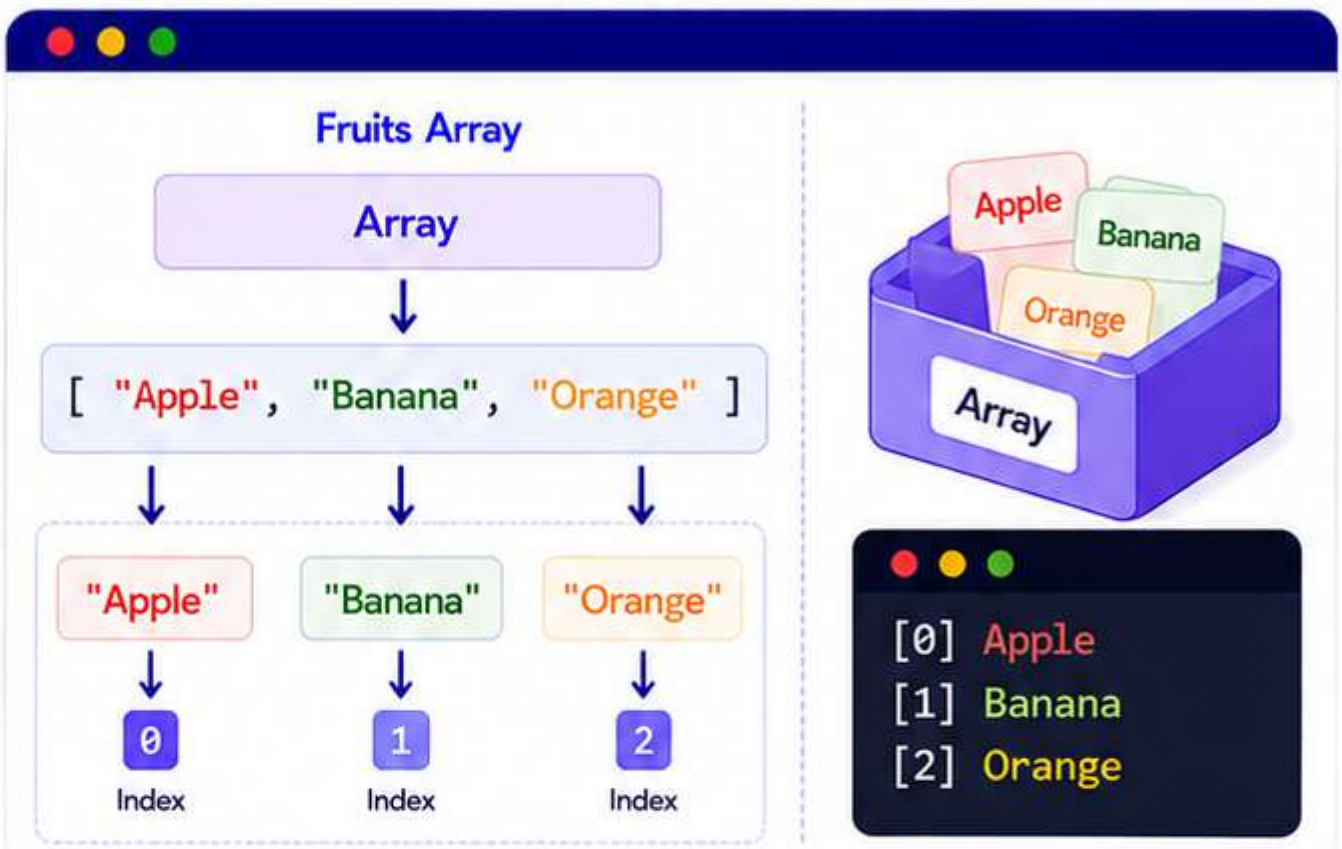
Day 16: Arrays Introduction

Store multiple values together

1 What are Arrays?

- Arrays store multiple values in a single variable
- Organize related data efficiently
- Access values using indexes
- One of the most important JavaScript data structures

JS



2 Array Fundamentals

Creating Arrays

- Use square brackets []
- Store multiple values
- Separate items with commas

```
const fruits = [
  "Apple",
  "Banana",
  "Orange"
];
```

Create an Array

Accessing Elements

- Use index numbers
- Index starts from 0
- Retrieve specific values

```
fruits[0]
```

Output:

```
Apple
```

Get Values

Array Length

- Counts total elements
- Uses .length property
- Useful in loops

```
fruits.length
```

Output:

```
3
```

Count Items

3 Quick Examples

Create Array

```
const colors =
["Red", "Blue"]
```

First Element

```
colors[0]
```

Output:

```
Red
```

Length

```
colors.length
```

Output:

```
2
```

Index Rule

First item = 0

Second item = 1

Array Structure

```
const fruits = [
  "Apple",
  "Banana",
  "Orange"
];
```

Indexes:

0 → Apple

1 → Banana

2 → Orange

- Store Multiple Values
- Easy Data Access
- Essential JavaScript Concept

Follow 👉 codingwithparvez

30 Days JavaScript Series

Day 17: Common Array Methods

Work with arrays efficiently

1 What are Array Methods?

- Array methods help modify array data
- Add or remove elements easily
- Built into JavaScript arrays
- Save time and reduce code complexity

JS

Array (Before)

```
["Apple", "Banana"]
```

↓

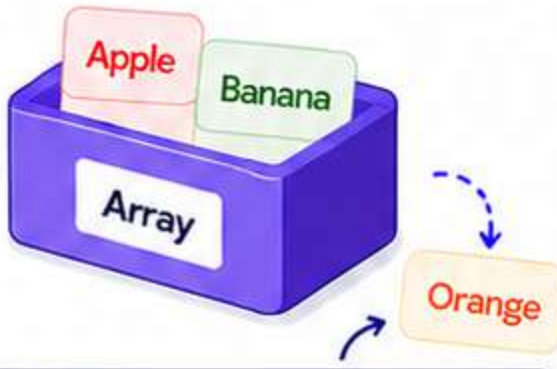
Method Applied

```
push("Orange")
```

↓

Updated Array (After)

```
["Apple", "Banana", "Orange"]
```




```
// Before
["Apple","Banana"]
push("Orange")
// After
["Apple","Banana","Orange"]
```

2 Essential Array Methods

push()

- Adds element to the end
- Modifies original array
- Very commonly used

```
let fruits =
["Apple","Banana"];

fruits.push("Orange");
```

Output:

```
["Apple","Banana","Orange"]
```

Add to End

pop()

- Removes last element
- Returns removed item
- Updates original array

```
fruits.pop();
```

Output:

```
["Apple","Banana"]
```

Remove from End

shift()

- Removes first element
- Shifts remaining items
- Changes array indexes

```
fruits.shift();
```

Output:

```
["Banana","Orange"]
```

Remove from Start

unshift()

- Adds item to beginning
- Moves existing elements
- Updates indexes automatically

```
fruits.unshift("Mango");
```

Output:

```
["Mango","Apple","Banana"]
```

Add to Start

3 Quick Summary

push()	pop()	shift()	unshift()
Add item at end	Remove item from end	Remove item from start	Add item at start

Array Before

```
["Apple", "Banana"]
 0        1
```

After push("Orange")

```
["Apple", "Banana", "Orange"]
 0        1         2
```

After pop()

```
["Apple", "Banana"]
 0        1
```

- Easy Array Management
- Built-in JavaScript Methods
- Essential for Everyday Coding

30 Days JavaScript Series

Day 18: Objects Introduction

✦ Organize related data ✦

1 What are Objects?

- Objects store related information together
- Data is stored as key-value pairs
- Helps organize complex data
- One of the most important JavaScript concepts

JS

Person Object

```
{
  name: "John",
  age: 25,
  city: "Mumbai"
}
```

Person

Name	→	John
Age	→	25
City	→	Mumbai

```
// Person Object
name: "John"
age: 25
city: "Mumbai"
```

2 Object Fundamentals

Object Syntax

- Uses curly braces {}
- Stores multiple properties
- Organized structure

```
const person = {
  name: "John",
  age: 25
};
```

Create an Object

Properties

- Keys inside an object
- Describe the data
- Unique identifiers

```
name
age
city
```

Object Keys

Values

- Actual stored information
- Linked to properties
- Can be any data type

```
"John"
25
"Mumbai"
```

Stored Data

3 Quick Example

```
const student = {
  name: "Alex",
  age: 20,
  course: "JavaScript"
};
```

Object Structure

name	→	Alex
age	→	20
course	→	JavaScript

Quick Summary

- **Object** — Stores related data
- **Property** — Key name
- **Value** — Actual data
- **Key-Value Pair** — Foundation of Objects

Real World Example

Car Object

```
{
  brand: "Toyota",
  model: "Camry",
  year: 2025
}
```

brand	→	Toyota
model	→	Camry
year	→	2025

- Organized Data
- Easy Management
- Essential JavaScript Structure

30 Days JavaScript Series

Day 19: Accessing Object Data

Read and update object properties

1 Accessing Object Data

- Object properties can be read and modified
- JavaScript provides multiple ways to access data
- Access values using property names
- Objects become useful when data can be retrieved easily

JS

Object

```
person = {
    name: "John",
    age: 25
}
```

Property Access

```
person.name
```

Value Returned

```
"John"
```


```
// Object
person = {
  name: "John",
  age: 25
}
// Access Property
person.name; // "John"
```

2 Ways to Access Data

Dot Notation

- Most common method
- Easy to read
- Uses dot operator

```
const person = {
  name: "John"
};
person.name;
```

Output:

```
"John"
```

Simple & Popular

Bracket Notation

- Uses square brackets
- Property name as string
- Useful for dynamic keys

```
const person = {
  name: "John"
};
person["name"];
```

Output:

```
"John"
```

Flexible Access

Updating Values

- Modify existing properties
- Assign new values
- Objects are mutable

```
const person = {
  age: 25
};
person.age = 26;
```

Output:

```
age → 26
```

Update Data

3 Practical Example

```
const student = {
  name: "Alex",
  age: 20
};

student.age = 21;
```

Before	After
name → Alex	name → Alex
age → 20	age → 21

Quick Summary

- Dot Notation: `person.name`
- Bracket Notation: `person["name"]`
- Update Value: `person.age = 26`
- Result: Read & Modify Data

Choose the Right Method

Dot Notation
Best for known property names

Bracket Notation
Best for dynamic property names

Updating Values
Keeps object data current

Follow 👉 codingwithparvez

30 Days JavaScript Series

Day 20: DOM Introduction

JavaScript meets the webpage

1 What is the DOM?

- DOM stands for Document Object Model
- Represents a webpage as a tree structure
- Allows JavaScript to access HTML elements
- Makes web pages interactive and dynamic

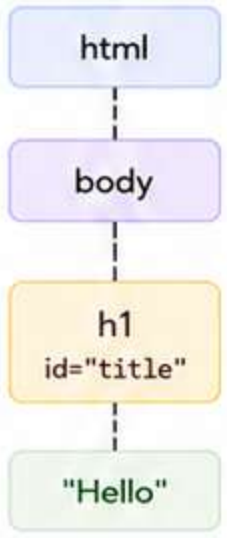
JS

HTML Document

```
<!DOCTYPE html>
<html>
  <body>
    <h1 id="title">
      Hello
    </h1>
  </body>
</html>
```

HTML → DOM Tree → JavaScript Manipulation → Hello

DOM Tree (Structure)



JavaScript can access and update elements on the page!

2 Why DOM Matters

Access Elements

- Find HTML elements
- Read page content
- Work with webpage data

```
document
  .getElementById(
    "title"
);
```

Find Elements

Change Content

- Update text dynamically
- Modify HTML content
- Create interactive pages

```
element.textContent =
  "Welcome";
```

Update Content

Change Styles

- Modify CSS with JavaScript
- Create dynamic designs
- Respond to user actions

```
element.style.color =
  "blue";
```

Update Styles

3 Basic Example

HTML

```
<h1 id="title">
  Hello
</h1>
```

JavaScript

```
document
  .getElementById(
    "title"
  )
  .textContent =
    "Welcome";
```

Result

Hello → Welcome

Quick Summary

- **DOM** — Webpage Structure
- **Access Elements** — Find HTML Tags
- **Change Content** — Update Text
- **Change Styles** — Modify Appearance

DOM Workflow

HTML Page → DOM Created → JavaScript Access → Webpage Updated

- Interactive Websites
- Dynamic Content
- Foundation of Frontend Development

Follow codingwithparvez

30 Days JavaScript Series

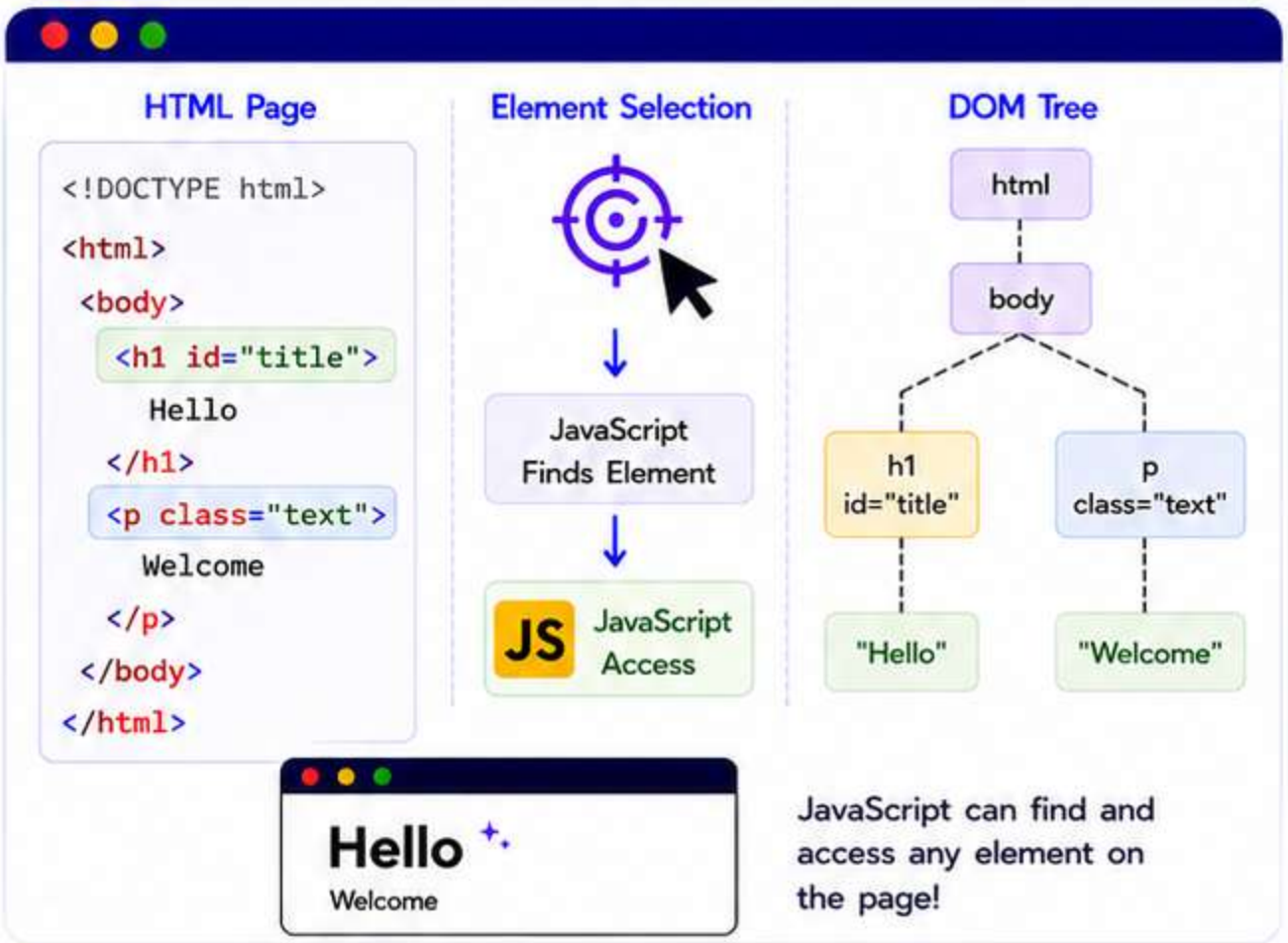
Day 21: Selecting Elements

Find HTML elements with JavaScript

1 What is Element Selection?

- JavaScript must find elements before modifying them
- Element selection is the first step in DOM manipulation
- Allows access to text, styles, and events
- Essential for interactive web pages

JS



2 Common Selection Methods

getElementById()

- Selects element by ID
- Returns one element
- Fast and commonly used

```
document
  .getElementById(
    "title"
);
```

Output:

```
<h1 id="title">
```

Select by ID

querySelector()

- Selects first matching element
- Uses CSS selectors
- Flexible and modern

```
document
  .querySelector(
    ".box"
);
```

Output:

```
First .box Element
```

First Match

querySelectorAll()

- Selects all matching elements
- Returns a NodeList
- Great for multiple elements

```
document
  .querySelectorAll(
    ".box"
);
```

Output:

```
All .box Elements
```

Multiple Matches

3 Quick Comparison

Method	What It Uses	What It Returns	Best For
getElementById()	Uses ID	Returns One Element	Unique Elements
querySelector()	Uses CSS Selectors	Returns First Match	Single Element
querySelectorAll()	Uses CSS Selectors	Returns All Matches	Multiple Elements

Highlighted Example

HTML

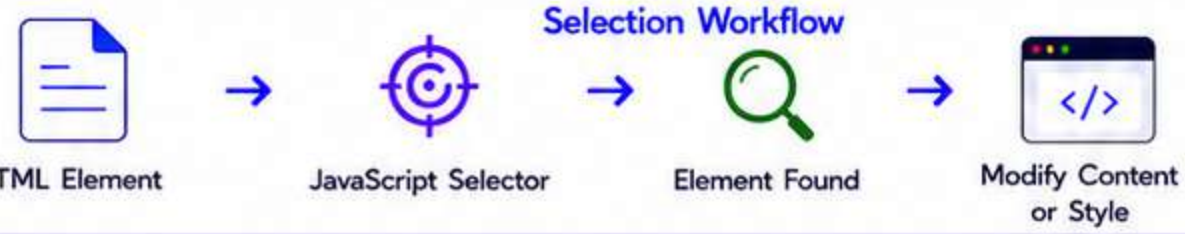
```
<p class="text">
  Hello
</p>
```

JavaScript

```
document
  .querySelector(".text");
```

Result: Selected Paragraph Element

Selection Workflow



- Foundation of DOM Manipulation
- Required for Events
- Essential Frontend Skill

Follow codingwithparvez

30 Days JavaScript Series

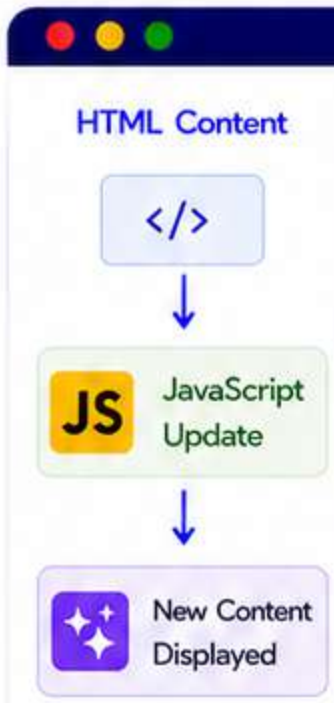
Day 22: Changing Content

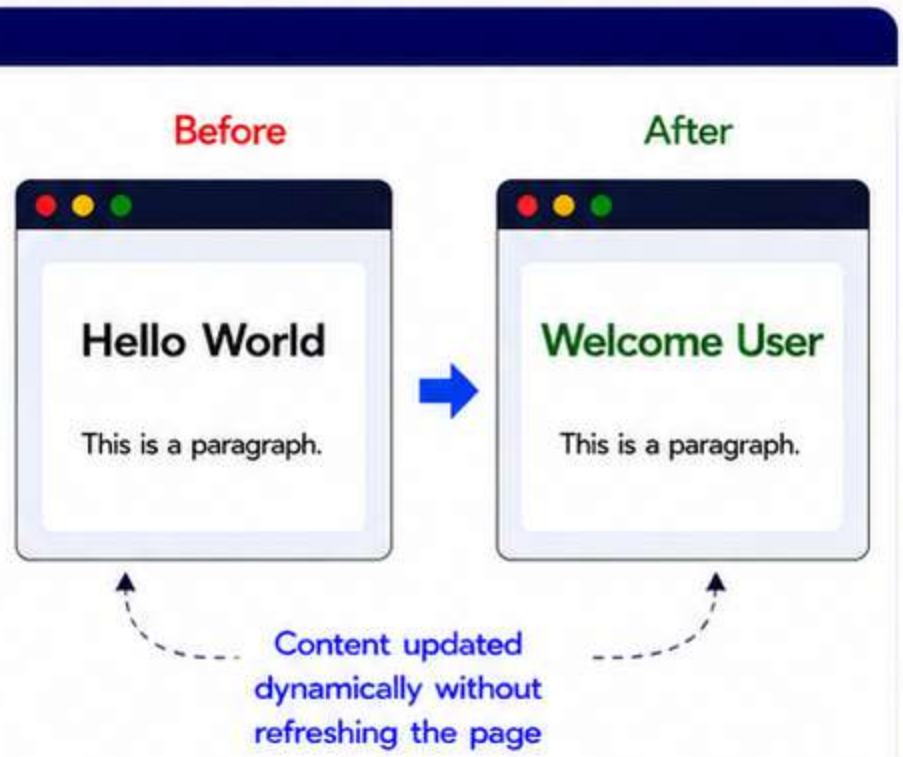
Update webpage content dynamically

1 What is Dynamic Content?

- JavaScript can change webpage content instantly
- No page refresh required
- Makes websites interactive
- Updates text and HTML elements dynamically

JS





2 Ways to Change Content

textContent

- Changes plain text
- Safe and simple
- Ignores HTML tags

```
element.textContent =
  "Hello User";
```

Output:

Hello User

Text Only

innerHTML

- Changes HTML content
- Can add HTML tags
- More flexible

```
element.innerHTML =
  "<b>Hello</b>";
```

Output:

Hello

HTML Content

Examples

- Update headings
- Change paragraphs
- Display dynamic data

```
title.textContent =
  "Welcome";
```

Real-world Usage

3 Practical Example

HTML

```
<h1 id="title">
  Hello
</h1>
```

JavaScript

```
document
  .getElementById(
   "title")
  .textContent =
   "Welcome";
```

Result

Before: Hello

After: Welcome

Quick Summary

- **textContent** — Updates plain text
- **innerHTML** — Updates HTML content
- **Dynamic Updates** — Change content instantly
- **DOM Manipulation** — Interactive webpages

textContent vs innerHTML

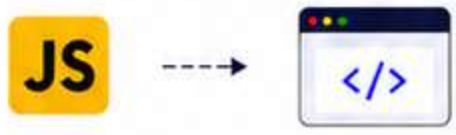
textContent
Safer for plain text

```
element.textContent =
  "Hello";
```

innerHTML
Useful for HTML elements

```
element.innerHTML =
  "<b>Hello</b>";
```

Both are essential for dynamic web pages



Follow codingwithparvez

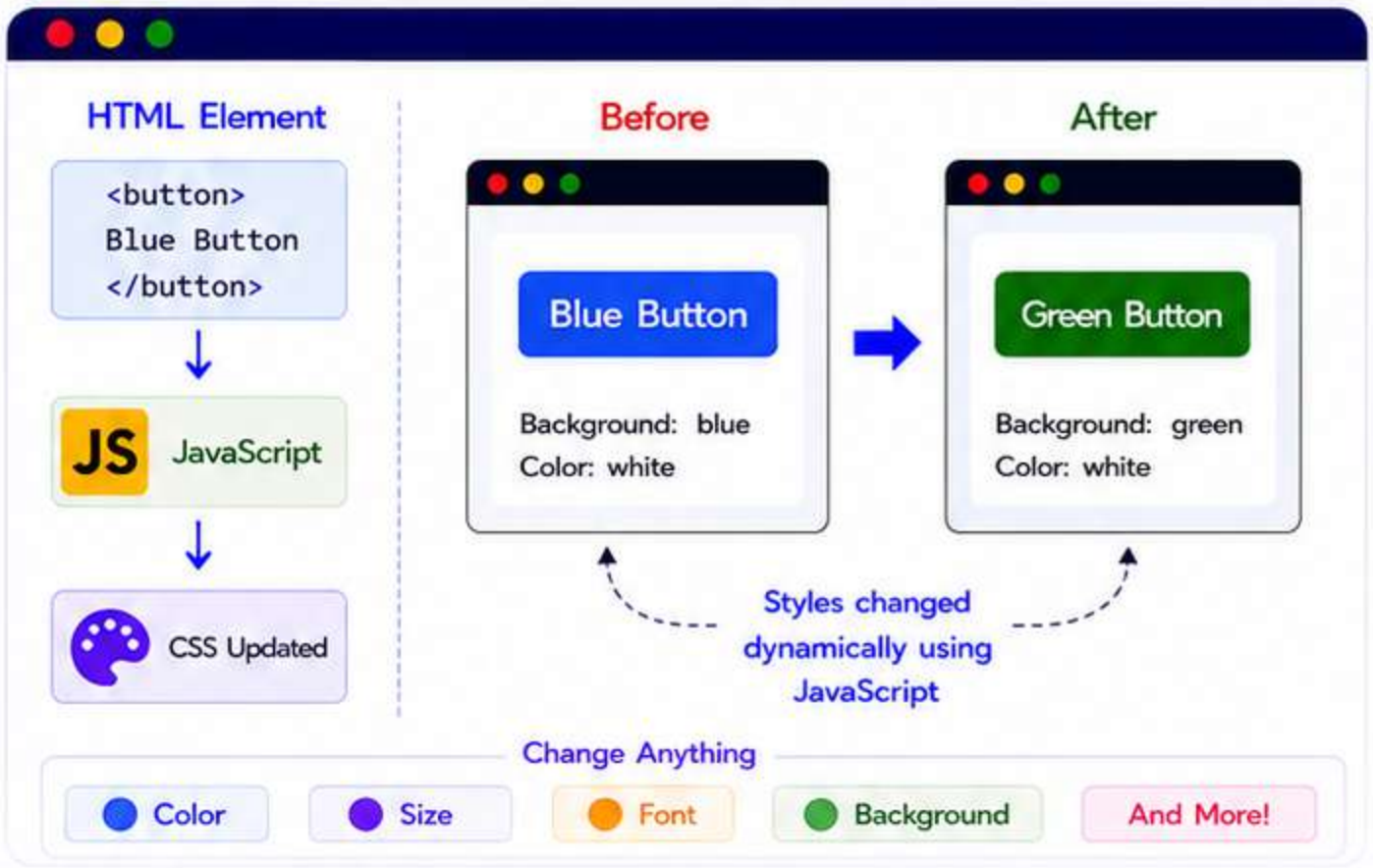
30 Days JavaScript Series

Day 23: Changing Styles

Modify CSS using JavaScript

1 Why Change Styles with JavaScript?

- JavaScript can modify CSS dynamically
- Create interactive user experiences
- Respond to user actions instantly
- Update appearance without page refresh



2 Common Styling Techniques

.style

- Change individual CSS properties
- Apply styles directly
- Quick and simple updates

```
element.style.color =
  "blue";
```

Output:

Text becomes blue

Direct Styling

Class Manipulation

- Add or remove CSS classes
- Reuse existing styles
- Best practice approach

```
element.classList
  .add("active");
```

Output:

Class "active" added

Manage Classes

Examples

- Change colors
- Update fonts
- Show visual feedback

```
button.classList
  toggle("dark");
```

Output:

Toggle dark mode

Interactive UI

3 Practical Example

HTML

```
<button id="btn">
  Click Me
</button>
```

JavaScript

```
document
  .getElementById(
    "btn"
  )
  .style.background =
    "green";
```

Result

Before:

Gray Button

After:

Green Button

Quick Summary

- **.style** — Change CSS directly
- **classList** — Add or remove classes
- **Dynamic UI** — Update styles instantly
- **Better UX** — Interactive webpages

Most Common Methods

element.style.color

```
element.style.color =
  "red";
```

classList.add()

```
element.classList
  .add("active");
```

classList.remove()

```
element.classList
  .remove("active");
```

classList.toggle()

```
element.classList
  .toggle("dark");
```

- Dynamic Styling
- Interactive Interfaces
- Essential DOM Skill

Follow codingwithparvez

30 Days JavaScript Series

Day 24: Events Introduction

Respond to user actions

1 What are Events?

- Events are actions that happen on a webpage
- JavaScript can detect and respond to events
- Makes websites interactive
- Trigger code when users interact with elements

JS

2 Event Fundamentals

What are Events?

- Actions performed by users
- Trigger JavaScript code
- Enable interaction

```
Click Button
     ↓
Run Function
```

User Actions

Event Types

- click
- mousoever
- keydown
- submit

```
click
keydown
submit
```

Common Events

Event Listeners

- Listen for events
- Execute functions automatically
- Core DOM feature

```
button
  .addEventListener(
   "click",
   sayHello
);
```

Listen & Respond

3 Practical Example

HTML

```
<button id="btn">
   Click Me
</button>
```

JavaScript

```
btn
 .addEventListener(
   "click",
   function(){
     alert("Hello!");
   }
);
```

Result

User Clicks Button

Alert Appears

Quick Summary

- **Event** — User Action
- **Event Types** — click, submit, keydown
- **Event Listener** — Waits for Events
- **Response** — Runs JavaScript Code

Event Workflow

- Interactive Websites
- Dynamic User Experience
- Essential DOM Concept

Follow codingwithparvez

30 Days JavaScript Series

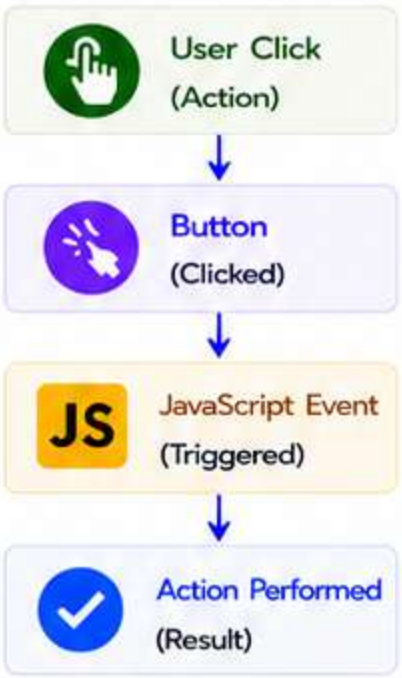
Day 25: Click Events

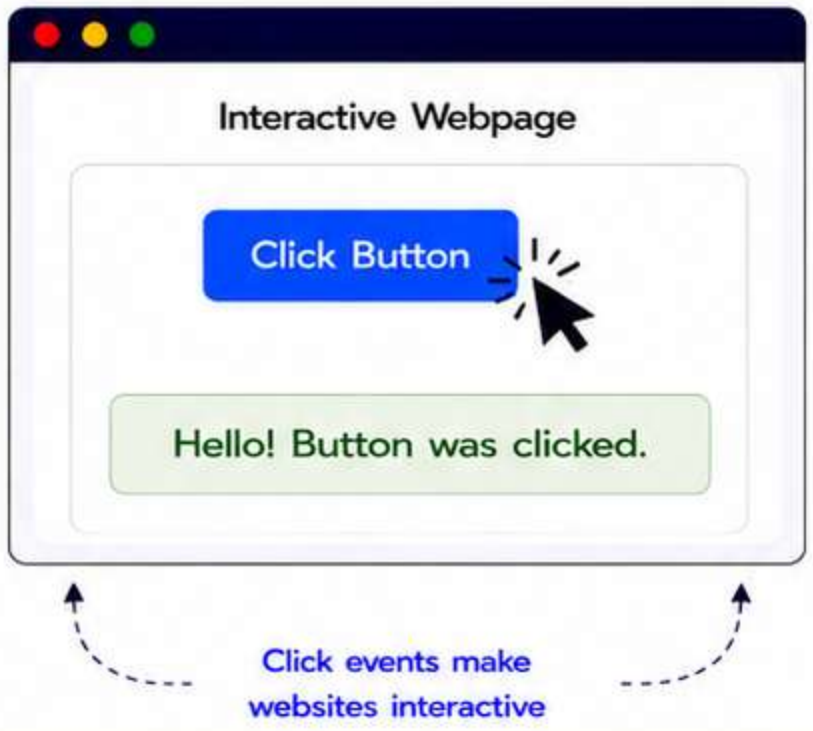
Make buttons interactive

1 What are Click Events?

- Click events occur when users click an element
- One of the most common JavaScript events
- Used to trigger actions and interactions
- Essential for interactive web applications

JS





2 Click Event Essentials

click Event

- Triggered when user clicks
- Works on buttons and elements
- Most commonly used event

```
button.onclick =
 function(){
   alert("Clicked");
};
```

User Click Action

Event Listener

- Modern event handling
- Attach multiple events
- Preferred approach

```
button
  .addEventListener(
    "click",
    showMessage
  );
```

Best Practice

Practical Uses

- Open menus
- Submit forms
- Show messages
- Toggle content

```
button.addEventListener(
  "click",
  toggleMenu
);
```

Real-world Usage

3 Practical Example

HTML

```
<button id="btn">
  Click Me
</button>
```

JavaScript

```
const btn =
  document
    .getElementById(
      "btn"
);

btn.addEventListener(
  "click",
  function(){
    alert(
      "Button Clicked!"
    }
);
```

Result

Quick Summary

- **click** — Detect user clicks
- **Event Listener** — Listen for events
- **Action** — Execute JavaScript
- **Result** — Interactive Webpage

Click Event Workflow

- Interactive Buttons
- Better User Experience
- Essential Frontend Skill

Follow codingwithparvez

30 Days JavaScript Series

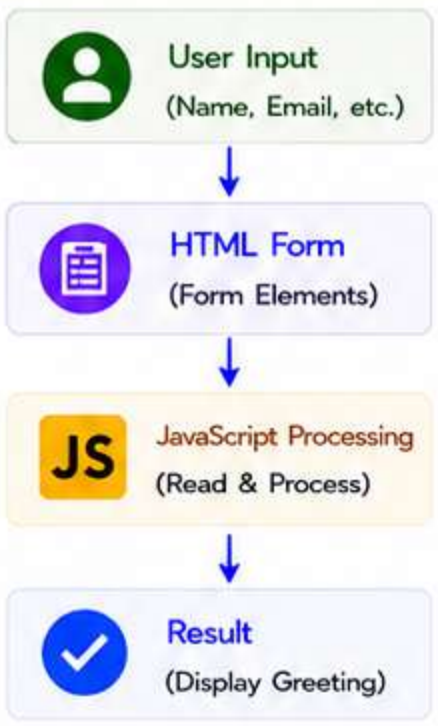
Day 26: Form Handling

Capture user input from forms

1 What is Form Handling?

- Forms collect data from users
- JavaScript reads and processes form inputs
- Used in login, signup, search, and contact forms
- Essential for interactive web applications

JS



2 Form Handling Essentials

Input Values

- Read user-entered data
- Access input field values
- Most common form operation

```
const name =
  input.value;
```

Output:

John

Get User Data

Form Submit

- Triggered when form is submitted
- Sends user data
- Common in web applications

```
form.addEventListener(
  "submit",
  handleSubmit
);
```

Submit Data

preventDefault()

- Stops page refresh
- Gives JavaScript control
- Commonly used in forms

```
event
  .preventDefault();
```

Prevent Refresh

3 Practical Example

HTML

```
<form id="myForm">
  <input type="text"
         id="name" />
  <button>Submit</button>
</form>
```

JavaScript

```
myForm.addEventListener(
  "submit",
  function(event){
   event.preventDefault();

  const name =
    document.getElementById(
     "name").value;
  }
);
```

Result

User Enters Name → Form Submitted → JavaScript Gets Value

Quick Summary

- **Input Value** — Read user input
- **Submit Event** — Handle form submission
- **preventDefault()** — Stop page refresh
- **Result** — Process form data

Form Workflow

- User Interaction
- Dynamic Forms
- Essential Web Development Skill

Follow codingwithparvez

30 Days JavaScript Series

Day 27: ES6 Features

Modern JavaScript essentials

1 What is ES6?

- ES6 (ECMAScript 2015) introduced modern JavaScript features
- Makes code cleaner and easier to write
- Reduces repetitive code
- Widely used in modern web development

JS

Old JavaScript (Verbose Code)

↓

ES6 Features (Modern Syntax)

↓

Cleaner Code (Less & Better)

↓

Better Development (Faster & Efficient)

JS

Old JavaScript

```
var user = {
  name: "John",
  age: 25
};

var name = user.name;
var age = user.age;
```

More Code
More Lines

ES6 JavaScript

```
const user = {
  name: "John",
  age: 25
};

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

Less Code
Less Lines

ES6 makes code cleaner, shorter and easier to maintain.

2 Popular ES6 Features

Destructuring

- Extract values easily
- Works with objects and arrays
- Cleaner variable assignment

```
const user = {
  name: "John",
  age: 25
};
const { name } = user;
```

Extract Values

Spread Operator

- Expands arrays and objects
- Copies data easily
- Simplifies merging

```
const nums = [1,2,3];

const copy = [...nums];
```

Expand Data

Default Parameters

- Set fallback values
- Avoid undefined arguments
- Cleaner functions

```
function greet(
   name = "Guest"
){
  return name;
}
```

Default Values

3 Practical Examples

Destructuring

```
const user = {
  name: "John"
};
const {name} = user;
```

Output:

John

Spread Operator

```
const arr1 = [1,2];

const arr2 = [...arr1,3];
```

Output:

[1,2,3]

Default Parameter

```
function greet(
    name = "Guest"
){
  return name;
}
greet();
```

Output:

Guest

Quick Summary

- **Destructuring** — Extract values easily
- **Spread Operator** — Copy and merge data
- **Default Parameters** — Provide fallback values
- **ES6** — Cleaner modern JavaScript

Why ES6 Matters

- **Less Code** — Write more with less
- **Better Readability** — Clean and understandable
- **Faster Development** — Build features quickly
- **Industry Standard JavaScript** — Used by modern frameworks

30 Days JavaScript Series

Day 28: Local Storage

Save data in the browser

1 What is Local Storage?

- Local Storage stores data in the browser
- Data remains available after page refresh
- Easy way to save user preferences
- Useful for small amounts of data

JS

2 Local Storage Methods

Store Data

- Save data in browser
- Uses key-value pairs
- Data persists after refresh

```
localStorage
 .setItem(
   "name",
   "John"
);
```

Save Data

Retrieve Data

- Read stored values
- Access using key name
- Returns saved data

```
localStorage
 .getItem(
   "name"
);
```

Output:

John

Get Data

Remove Data

- Delete stored values
- Remove unwanted data
- Free browser storage

```
localStorage
 .removeItem(
   "name"
);
```

Delete Data

3 Practical Example

Save Data

```
localStorage
 .setItem(
   "user",
   "Alex"
);
```

Retrieve Data

```
localStorage
 .getItem(
   "user"
);
```

Output:

Alex

Output

Data Retrieved Successfully

Alex

Remove Data

```
localStorage
 .removeItem(
   "user"
);
```

Quick Summary

- setItem() — Store data
- getItem() — Retrieve data
- removeItem() — Delete data
- Local Storage — Data survives refresh

Common Use Cases

- Dark Mode Preference
- Remember Username
- Save Settings
- Small Browser Storage

Storage Workflow

Save (Store Data) → Store (In Browser) → Retrieve (Get Data) → Remove (Delete Data)

Follow codingwithparvez

30 Days JavaScript Series

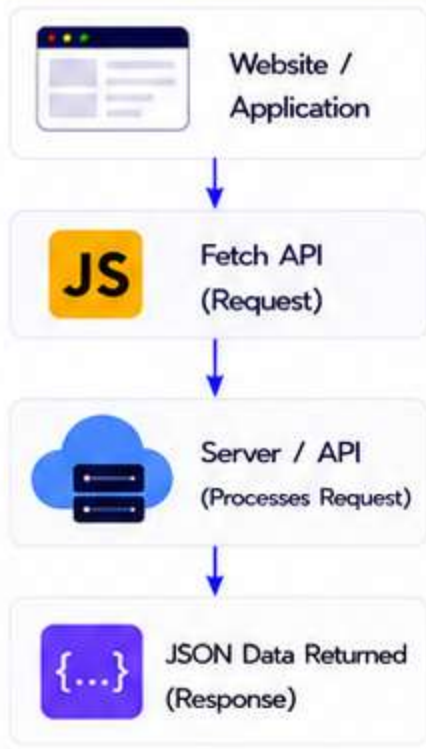
Day 29: Fetch API Basics

Get data from APIs

1 What is an API?

- API stands for Application Programming Interface
- Allows applications to exchange data
- Fetch data from servers and external services
- Foundation of modern web applications

JS



Fetch API connects your app to servers and returns JSON data

2 Fetch API Essentials

What is fetch()?

- Built-in JavaScript function
- Sends HTTP requests
- Retrieves data from APIs

```
fetch(
  "https://api.com"
);
```

Request Data

JSON Response

- Most APIs return JSON
- Easy to read and use
- JavaScript converts JSON to objects

```
response.json()
```

Output:

```
{
  "name": "John"
}
```

API Data Format

Using Data

- Display API data
- Build dynamic applications
- Update webpage content

```
data.name
```

Output:

John

Use API Results

3 Practical Example

JavaScript

```
fetch("api/users")
  .then(response =>
    response.json()
  )
  .then(data =>
    console.log(data)
  );
```

Workflow

Send Request → Receive Response → Convert JSON → Use Data

Result

API Data Available Inside JavaScript

Quick Summary

- fetch() — Request data
- JSON — Common API format
- Data Usage — Display API results
- Modern Web Apps — Powered by APIs

Fetch API Flow

Request → Response → JSON → JavaScript Object → Display Data

- Connect to APIs
- Retrieve Live Data
- Essential Modern JavaScript Skill

Follow codingwithparvez

Day 30 Final Project

Day 30: Mini Project

Build something real with JavaScript

1 Apply What You've Learned

- Combine multiple JavaScript concepts
- Build interactive applications
- Practice DOM manipulation
- Strengthen problem-solving skills

JS

JavaScript Skills

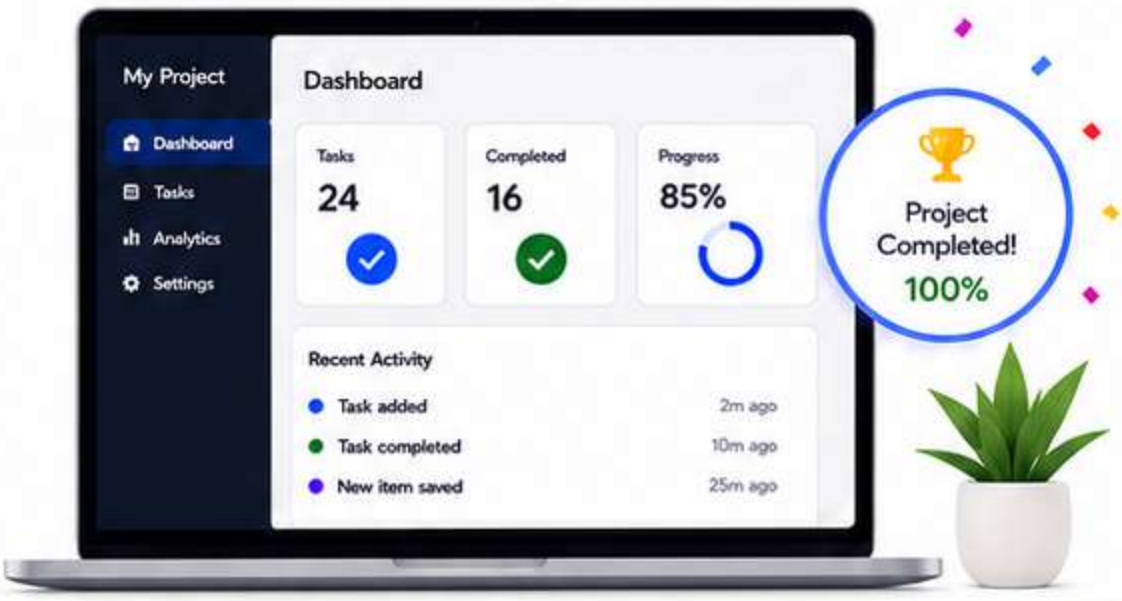
↓

Mini Project

↓

Real Application

Variables + Functions + DOM + Events

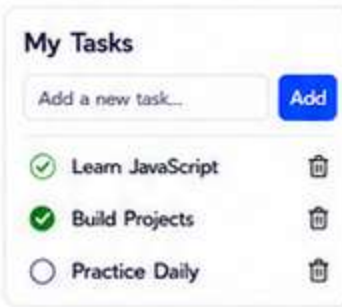


2 Beginner Project Ideas

To-Do List App

Features:

- Add tasks
- Delete tasks
- Mark completed
- Store tasks



Skills Used: DOM, Events, Arrays

Most Popular Beginner Project

Counter App

Features:

- Increase count
- Decrease count
- Reset value
- Live updates

Skills Used: Variables, Functions, Events

Perfect for Practice

Random Quote Generator

Features:

- Show random quotes
- Button interaction
- Dynamic content
- Fun UI

Skills Used: Arrays, Functions, DOM

Interactive Project

3 What You've Learned

JavaScript Fundamentals

{ }

- Variables
- Data Types
- Operators

Control Flow

- Conditions
- Loops

Functions

f(x)

- Parameters
- Return Values
- Arrow Functions

DOM & Events

- Element Selection
- Content Updates
- Click Events

Modern JavaScript

ES6+

- ES6 Features
- Local Storage
- Fetch API

30 Days Completed

30 Lessons Learned

3 Project Ideas Ready

Ready for Real Development

You can now build:

- Interactive Web Pages
- DOM-Based Applications
- Event-Driven Features
- Small JavaScript Projects

Next Step:

- Learn React.js
- Build Portfolio Projects
- Continue Your Developer Journey

Follow codingwithparvez

Series Completed

30 Days JavaScript Roadmap Completed

You've completed the fundamentals of modern JavaScript development

1 What You've Learned

Day 1	Day 10	Day 20	Day 30
JavaScript Basics	Loops & Logic	DOM Manipulation	Ready for Projects
Completed	Completed	Completed	Completed

2 Skills Unlocked

JavaScript Fundamentals

Topics Covered:

- Variables
- Data Types
- Operators
- Conditions
- Loops
- Functions

DOM Manipulation

Topics Covered:

- DOM Basics
- Selecting Elements
- Changing Content
- Styling Elements

Events & Forms

Topics Covered:

- Event Listeners
- Click Events
- Form Handling
- User Interaction

ES6 Concepts

Topics Covered:

- Template Literals
- Arrow Functions
- Destructuring
- Spread Operator

APIs & Storage

Topics Covered:

- Local Storage
- Fetch API
- JSON Data
- Browser Storage

3 What's Next?

NEXT STEPS

- Learn React.js
- Learn Async JavaScript
- Build Real Projects
- Create Responsive Apps
- Learn Backend Development
- Explore Full Stack Development

ACHIEVEMENT SUMMARY

- 30 Days Completed
- 29+ Core Concepts Learned
- DOM & Events Mastered
- Ready for Real Projects

Congratulations!

You now understand:

- JavaScript Fundamentals
- DOM Manipulation
- Events & Forms
- ES6 Concepts
- APIs & Browser Storage

Your JavaScript journey starts here.

Keep building. Keep learning. Keep coding.

Follow codingwithparvez