JS

JavaScript

COMPLETE NOTES

From Zero to Hero — Written for Beginners

Variables · Data Types · Operators · Conditionals · Loops

Functions · Arrays · Objects · Classes · Error Handling

Learn With Asad

learnwithasad.com • prepnex.in

Contents

01 What is JavaScript?

The language that brings websites to life!

JS

JavaScript is the brain of the internet

HTML builds the skeleton (structure), CSS adds the clothes (style), and JavaScript adds the brain — it makes things MOVE, REACT, and DO stuff!

A real-life analogy:

Imagine a traffic light. **HTML** is the metal pole and light boxes. **CSS** decides the colours (red, yellow, green). **JavaScript** is the timer and controller that actually makes the lights switch!

What can JavaScript do?

- Show a popup when you click a button
- Check if a form is filled in before submitting
- Make a countdown timer or live score board
- Load new content without refreshing the page
- Build full apps — like games, chat apps, YouTube!

Your very first JavaScript

```
// This is a comment — JavaScript ignores it!
// Comments are notes for YOU (the human)

console.log("Hello World!");   // print to screen
console.log("My name is Asad");
```

Open your browser → right-click → Inspect → Console tab → type those lines → press Enter!

Tip: console.log() is your best friend for testing. Think of it as JavaScript's way of talking back to you.

02 Variables

Labelled boxes that hold your stuff

A variable is a labelled box in memory

Imagine you have boxes at home with labels: 'Toys', 'Books', 'Clothes'. A variable is exactly that — a labelled box where you store a piece of data.

Three ways to create a variable

```
let   playerName = 'Asad';    // box you CAN repaint later
const HIGH_SCORE  = 1000;     // box that is SEALED — can't change
var   oldWay      = 'avoid';  // old style — please avoid in 2025!
```

let — The Flexible Box

Use **let** when the value will change later. Like a whiteboard — you can erase and rewrite.

```
let score = 0;
scoring log(score);    // 0

score = 50;            // changed it!
console.log(score);    // 50

score = score + 10;    // add 10
console.log(score);    // 60
```

Real Life: Like a score on a cricket board — it starts at 0 and keeps going up!

const — The Permanent Box

Use **const** when the value NEVER changes. Like your date of birth — it's always the same.

```
const birthYear = 2009;       // locked!
const PI        = 3.14159;    // locked!

// If you try to change it:
birth_year = 2010;             // ERROR! Can't do this!
```

Watch Out! You'll get an error if you try to change a const. That's actually GOOD — it protects your data!

Variable Naming Rules

```
// GOOD names:
let playerScore = 100;    // camelCase — standard JS
let isLoggedIn  = true;
let maxLives    = 3;

// BAD names:
let 1player = 'Asad';   // ERROR: can't start with number
let my name = 'Asad';   // ERROR: no spaces!
let x = 100;            // confusing — what is x?
```

Tip: Always name variables clearly. 'studentAge' is 100x better than 'sa' or 'x'.

03 Data Types

What KIND of thing is it?

Every piece of data has a TYPE. JavaScript needs to know if something is a number, text, true/false, or something else — so it can handle it correctly.

1. String — Text

Anything surrounded by quotes is a string. You can use single ' or double " or backticks `.

```
const name    = 'Asad';           // single quotes
const city    = "Patna";          // double quotes
const message = `Hello, ${name}!`; // backtick (template literal)

console.log(typeof name);  // 'string'
```

Real Life: A string is like the words on a greeting card — it's just text!

2. Number — Any number

JavaScript uses ONE type for all numbers — whole numbers and decimals.

```
const age    = 15;        // whole number
const price  = 99.99;     // decimal
const temp   = -5;        // negative
const big    = 1_000_000; // underscores for readability!

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

3. Boolean — True or False

A boolean is the simplest type — it can only be **true** or **false**. Like a light switch — on or off.

```
const isRaining  = false;
const hasTicket  = true;
const isAdult    = age >= 18;  // this becomes true or false

console.log(typeof isRaining);  // "boolean"
```

Real Life: Your phone's WiFi button — it's either ON (true) or OFF (false). Nothing in between!

4. null — Intentionally Empty

null means 'I know this box exists, but right now there is NOTHING in it on purpose.'

```
let winner = null;   // game not over yet, no winner

// Later when game ends:
winner = 'Asad';     // now we have a winner!
```

5. undefined — Not Set Yet

undefined means the box was created but you never put anything in it.

```
let score;            // created but no value given
console.log(score);   // undefined

let player = {};
console.log(player.name);  // undefined (no 'name' key)
```

Watch Out! null means 'empty on purpose'. undefined means 'forgotten to set'. They are different!

04 Operators

The tools of math and logic

Arithmetic Operators — Math

```
const a = 10;
const b = 3;

a + b   // 13   (add)
a - b   // 7    (subtract)
a * b   // 30   (multiply)
a / b   // 3.33 (divide)
a % b   // 1    (remainder — called MODULO)
a ** b  // 1000 (power: 10 to the power of 3)
```

The modulo % operator is super useful — it gives you the leftover after dividing.

```
// Is a number even or odd?
console.log(10 % 2);  // 0 → even (no leftover)
console.log(7  % 2);  // 1 → odd  (1 leftover)

// How many chocolates left after packing boxes of 5?
console.log(23 % 5);  // 3 chocolates left over
```

Comparison Operators

```
5 === 5    // true  (strictly equal — same value AND type)
5 !== 3    // true  (strictly not equal)
10 > 7     // true  (greater than)
3  < 8     // true  (less than)
5  >= 5    // true  (greater than OR equal to)
4  <= 6    // true  (less than or equal to)
```

Watch Out! Always use === (triple equals), not == (double equals). Triple equals checks both value AND type, which is safer.

Logical Operators — AND, OR, NOT

```
const hasTicket = true;
const isAdult   = true;
const isBanned  = false;

// && means AND — BOTH must be true:
const canEnter = hasTicket && isAdult;       // true

// || means OR — at least ONE must be true:
const canPass  = hasTicket || isAdult;       // true

// ! means NOT — flips true to false:
const isAllowed = !isBanned;                 // true
```

Real Life: To enter a theme park you need a ticket AND you must be tall enough. That is the && operator in real life!

Increment & Shortcut Operators

```
let lives = 3;

lives++;        // lives = lives + 1  → 4
lives--;        // lives = lives - 1  → 3
lives += 5;     // lives = lives + 5  → 8
lives *= 2;     // lives = lives * 2  → 16
lives -= 1;     // lives = lives - 1  → 15
```

05 Conditionals

Making decisions in your code

Conditionals let your code make decisions

Just like YOU make decisions every day — 'if it is raining, take an umbrella, otherwise wear sunglasses' — JavaScript does the same thing!

The if / else if / else Chain

```
const marks = 78;

if (marks >= 90) {
  console.log('Grade A – Excellent!');
} else if (marks >= 75) {
  console.log('Grade B – Great job!');
} else if (marks >= 60) {
  console.log('Grade C – Keep it up!');
} else {
  console.log('Needs improvement.');
}
```

JavaScript reads from top to bottom. The FIRST true condition runs and all the others are skipped.

Tip: Always use curly braces {} even for a single line. It makes your code clearer and safer.

Ternary Operator — One-line if / else

When you have a simple true/false choice, the ternary operator is a shorter way to write it.

```
// Full if-else:
let greeting;
if (hour < 12) {
  greeting = 'Good morning';
} else {
  greeting = 'Good afternoon';
}

// Same thing with ternary (one line!):
const greeting = hour < 12 ? 'Good morning' : 'Good afternoon';
//               condition    if true            if false
```

Real Life: Like asking someone: 'Are you hungry? Yes → eat pizza. No → keep working.'

Switch Statement — The Menu

When you have many specific cases, switch is cleaner than lots of else-ifs.

```
const day = 'Wednesday';

switch (day) {
  case 'Monday':
  case 'Tuesday':
  case 'Wednesday':
    console.log('School day! Study hard.');
    break;                          // MUST add break!
  case 'Saturday':
  case 'Sunday':
    console.log('Weekend! Time to relax.');
    break;
  default:
```

```
    console.log('Just another day.');
}
```

Watch Out! Always add break at the end of each case! Without it, JavaScript runs ALL cases below it.

06 Loops

Doing things over and over — automatically!

Loops save you from writing the same thing 1000 times

Imagine you need to print 'Good morning!' 100 times. You could type it 100 times... or write one loop that does it for you in 3 lines!

for loop — When you know HOW MANY times

```
// Print numbers 1 to 10:
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
//   start  condition  step (i++ means +1 each time)

// Print only even numbers:
for (let i = 2; i <= 20; i += 2) {
  console.log(i);  // 2, 4, 6, 8... 20
}
```

Real Life: Like stairs — you start at step 1, go up one at a time, and stop when you reach the top.

while loop — When you wait for a CONDITION

A while loop keeps going as long as a condition is true. Good when you don't know how many times upfront.

```
// Simulate rolling a dice until you get 6:
let dice = 0;
let rolls = 0;

while (dice !== 6) {
  dice  = Math.floor(Math.random() * 6) + 1;
  rolls++;
  console.log(`Rolled: ${dice}`);
}

console.log(`Got a 6 after ${rolls} rolls!`);
```

Watch Out! ALWAYS change the loop variable inside while! If you forget, it runs forever (infinite loop) and crashes!

for...of — Loop through an array

```
const subjects = ['Maths', 'Science', 'History'];

for (const subject of subjects) {
  console.log(`Studying: ${subject}`);
}
// Output:
// Studying: Maths
// Studying: Science
// Studying: History
```

forEach — Array method loop

```
const scores = [85, 92, 78, 61];

scores.forEach((score, index) => {
  console.log(`Student ${index + 1}: ${score} marks`);
});
```

```
// With condition inside:
scores.forEach(score => {
  if (score >= 80) {
    console.log(`${score} — Pass with distinction!`);
  }
});
```

Tip: Use for...of for simple loops. Use forEach when you also need the index number.

07 Functions

Reusable magic spells!

A function is a reusable block of code

Think of a function like a recipe. You write the recipe once (the function). Whenever you want to cook that dish, you just call the recipe by name! No need to rewrite all the steps every time.

Defining and Calling Functions

```
// Define the function (write the recipe)
function makeGreeting(name) {
  const message = `Hello, ${name}! Welcome to JavaScript!`;
  return message;        // send result back
}

// Call the function (cook the dish)
const result = makeGreeting('Asad');
console.log(result);  // Hello, Asad! Welcome to JavaScript!

// Call it many times with different inputs:
console.log(makeGreeting('Riya'));
console.log(makeGreeting('Zara'));
```

Functions with Multiple Parameters

```
function addMarks(maths, science, english) {
  const total   = maths + science + english;
  const average = total / 3;
  return `Total: ${total}, Average: ${average.toFixed(1)}`;
}

console.log(addMarks(85, 92, 78));
// Total: 255, Average: 85.0
```

Default Parameters

```
// If no argument given, use the default
function greet(name = 'Friend', lang = 'English') {
  if (lang === 'Hindi') {
    return `Namaste, ${name}!`;
  }
  return `Hello, ${name}!`;
}

greet()              // Hello, Friend!
greet('Asad')        // Hello, Asad!
greet('Riya','Hindi')// Namaste, Riya!
```

Arrow Functions — The Shorter Way

Arrow functions are a shorter way to write functions. Common in modern JavaScript.

```
// Regular function:
function square(n) { return n * n; }

// Arrow function — same thing!:
const square = (n) => n * n;
```

```
// Even shorter (1 param, no brackets needed):
const cube = n => n * n * n;

// With multiple lines:
const getGrade = score => {
  if (score >= 90) return 'A';
  if (score >= 75) return 'B';
  return 'C';
};

console.log(square(5));  // 25
console.log(getGrade(88)); // B
```

Tip: Use regular functions for longer, named functions. Use arrow functions for short, one-job helpers.

08 Arrays

Ordered lists of anything!

An array is like a numbered shelf

Imagine a shelf with slots numbered 0, 1, 2, 3... Each slot holds one item. The shelf is the array. The slot number is the INDEX. Counting always starts at 0!

Creating and Accessing Arrays

```
const fruits = ['apple', 'mango', 'grape', 'kiwi'];

console.log(fruits[0]);     // 'apple' (first!)
console.log(fruits[1]);     // 'mango'
console.log(fruits[3]);     // 'kiwi' (last)
console.log(fruits[-1]);    // undefined! (JS doesn't support this)

// Better way to get last item:
const last = fruits[fruits.length - 1]; // 'kiwi'
console.log(fruits.length); // 4 (total items)
```

Adding and Removing Items

```
const cart = ['pen', 'book'];

cart.push('eraser');    // add to END → ['pen','book','eraser']
cart.unshift('ruler');  // add to START → ['ruler','pen','book','eraser']

cart.pop();             // remove from END → returns 'eraser'
cart.shift();           // remove from START → returns 'ruler'

// Remove 1 item at index 1:
cart.splice(1, 1);      // removes 'book'
console.log(cart);      // ['pen']
```

Powerful Array Methods

```
const scores = [45, 92, 78, 61, 88];

// map — transform every item:
const doubled = scores.map(n => n * 2);
// [90, 184, 156, 122, 176]

// filter — keep only items that pass:
const passed = scores.filter(n => n >= 60);
// [92, 78, 61, 88]

// find — get first match:
const first90 = scores.find(n => n > 90);
// 92

// reduce — combine to one value:
const total = scores.reduce((sum, n) => sum + n, 0);
// 364 (sum of all)

// includes — does it exist?
scores.includes(78);   // true
scores.includes(100);  // false
```

```
// sort:
scores.sort((a, b) => a - b);  // ascending
scores.sort((a, b) => b - a);  // descending
```

Tip: map, filter, and reduce are the three most powerful array tools. Master them and you'll write beautiful code!

09 Objects

Things with named properties

An object is like a labelled file folder

An array stores items by NUMBER (0, 1, 2). An object stores items by NAME (key). Think of it like a student's ID card — it has a name field, age field, grade field — each piece of info has its own label!

Creating an Object

```
const student = {
  name:   'Asad',        // key: value
  age:    15,
  grade:  'A',
  hobbies: ['coding', 'gaming'],  // array inside!
  address: {              // object inside object!
    city:  'Patna',
    state: 'Bihar'
  }
};

// Access by dot notation:
console.log(student.name);          // 'Asad'
console.log(student.address.city);  // 'Patna'

// Access by bracket notation:
console.log(student['grade']);      // 'A'
```

Modifying Objects

```
// Add a new property:
student.school = 'LWA School';

// Update existing property:
student.age = 16;

// Delete a property:
delete student.hobbies;

// Check if a property exists:
console.log('name' in student);  // true
console.log('salary' in student); // false
```

Methods — Functions inside Objects

```
const dog = {
  name:  'Bruno',
  breed: 'Labrador',
  age:   3,
  bark() {
    console.log(`${this.name} says: WOOF!`);
  },
  info() {
    return `${this.name} is a ${this.age} year old ${this.breed}`;
  }
};

dog.bark();          // Bruno says: WOOF!
```

```
console.log(dog.info()); // Bruno is a 3 year old Labrador
```

this refers to the object itself. Inside dog's bark(), 'this.name' means dog.name.

Destructuring — Unpack Objects Quickly

```
const {name, age, grade} = student;
console.log(name);   // 'Asad'
console.log(age);    // 15

// With rename:
const {name: studentName, grade: studentGrade} = student;

// In function parameter:
function printInfo({name, age}) {
  console.log(`${name} is ${age} years old`);
}
printInfo(student);
```

10 Classes

Blueprints for creating objects!

A class is a blueprint, an object is the real thing

A class is like a cookie cutter. The cutter (class) is the blueprint. Every cookie you make (object) is a real instance from that same cutter. One blueprint → unlimited objects!

Creating a Class

```
class Animal {
  constructor(name, age) {  // runs when object is created
    this.name = name;
    this.age  = age;
  }

  introduce() {
    return `I am ${this.name}, aged ${this.age}`;
  }

  birthday() {
    this.age++;
    console.log(`Happy Birthday ${this.name}! Now ${this.age}.`);
  }
}

// Create objects from the blueprint:
const cat = new Animal('Kitty', 2);
const dog = new Animal('Bruno', 3);

console.log(cat.introduce());  // I am Kitty, aged 2
dog.birthday();                // Happy Birthday Bruno! Now 4.
```

Inheritance — Child Classes

```
class Dog extends Animal {   // Dog IS AN Animal
  constructor(name, age, breed) {
    super(name, age);        // call parent constructor first!
    this.breed = breed;
  }

  bark() {
    return `${this.name} says WOOF!`;
  }

  // Override parent method:
  introduce() {
    return `I am ${this.name}, a ${this.breed}`;
  }
}

const rex = new Dog('Rex', 4, 'Husky');
console.log(rex.bark());        // Rex says WOOF!
console.log(rex.introduce());   // I am Rex, a Husky
rex.birthday();                 // inherited from Animal!
```

Tip: Always call super() first in the child constructor. It sets up everything from the parent class.

11 Error Handling

Catching mistakes before they crash everything!

Errors happen — handle them gracefully!

Imagine you are a chef. Sometimes an ingredient is missing. A good chef has a backup plan — 'if we have no tomatoes, use ketchup'. try/catch is JavaScript's backup plan system!

try / catch / finally

```
try {
  // code that might fail:
  const result = riskyOperation();
  console.log(result);

} catch (error) {
  // runs only if something went wrong:
  console.log('Oops! Error:', error.message);

} finally {
  // ALWAYS runs (cleanup code):
  console.log('Done — this always runs!');
}
```

A Real Example

```
function divideNumbers(a, b) {
  if (b === 0) {
    throw new Error('Cannot divide by zero!');
  }
  return a / b;
}

try {
  console.log(divideNumbers(10, 2));  // 5
  console.log(divideNumbers(10, 0));  // throws error!

} catch (err) {
  console.log(`Caught: ${err.message}`);
  // Caught: Cannot divide by zero!
}
```

Common Error Types

Knowing the error type helps you fix it quickly:

`ReferenceError`	Used a variable that doesn't exist
`TypeError`	Called something that's not a function
`SyntaxError`	Broken code — missing bracket, etc.
`RangeError`	Number out of allowed range
`Custom Error`	Thrown by you: throw new Error('msg')

```
// Defensive programming — check before using:
function getFirstItem(arr) {
  if (!Array.isArray(arr)) {
    throw new TypeError('Expected an array!');
  }
  if (arr.length === 0) {
```

```
    return null;
  }
  return arr[0];
}

console.log(getFirstItem([5, 3, 1]));  // 5
console.log(getFirstItem([]));         // null
console.log(getFirstItem('oops'));     // TypeError!
```

Tip: Good code expects things to go wrong and handles it politely. Bad code crashes and shows ugly red errors!