# JavaScript Variables & Data Types - Complete Master Guide

## PART 1: VARIABLES BASICS

### What is a Variable?

A variable is a **container** that stores data values. Think of it like a labeled box where you put things.

```
// Like putting number 25 in a box labeled "age"
let age = 25;
```

### Three Ways to Declare Variables

#### 1. `var` (Old Way - Avoid Using)

```
var name = "John";
var age = 25;
```

#### 2. `let` (Modern Way - Use This)

```
let name = "John";
let age = 25;
```

#### 3. `const` (For Constants - Use When Value Won't Change)

```
const PI = 3.14159;
const country = "India";
```

### Key Differences Between var, let, const

| Feature | var | let | const |
|---|---|---|---|
| Can be redeclared | ✅ | ❌ | ❌ |
| Can be reassigned | ✅ | ✅ | ❌ |
| Block scoped | ❌ | ✅ | ✅ |

|  |  |  |  |
|---|---|---|---|
| Must be initialized | ❌ | ❌ | ✅ |

### Variable Naming Rules

#### ✅ Valid Names:

```
let firstName = "John";
let age2 = 25;
let _private = "secret";
let $price = 100;
let userName = "john123";
```

#### ❌ Invalid Names:

```
let 2age = 25;        // Can't start with number
let first-name = "John"; // Can't use hyphens
let class = "Math";    // Can't use reserved words
```

### Best Practices for Variable Names

```
// Use camelCase
let firstName = "John";
let userAge = 25;

// Be descriptive
let userEmail = "john@example.com";  // Good
let e = "john@example.com";          // Bad

// Use constants for fixed values
const MAX_RETRY_ATTEMPTS = 3;
const API_BASE_URL = "https://api.example.com";
```

---

## PART 2: DATA TYPES

JavaScript has **8 data types**: 7 primitive + 1 non-primitive

### PRIMITIVE DATA TYPES

#### 1. Number

Represents both integers and floating-point numbers.

```
let age = 25;            // Integer
let price = 19.99;        // Float
let negative = -10;        // Negative
let infinity = Infinity;   // Special number
let notANumber = NaN;      // Not a Number
```

**Number Methods & Properties:**

```
let num = 123.456;

// Common methods
num.toFixed(2);        // "123.46"
num.toString();        // "123.456"
parseInt("123.45");    // 123
parseFloat("123.45");  // 123.45

// Check if number
Number.isInteger(123);    // true
Number.isNaN(NaN);        // true
```

**2. String**

Represents text data.

```
let name = "John";           // Double quotes
let city = 'Mumbai';         // Single quotes
let message = `Hello ${name}`; // Template literals (ES6)
```

**String Methods:**

```
let str = "Hello World";

str.length;              // 11
str.toUpperCase();       // "HELLO WORLD"
str.toLowerCase();       // "hello world"
str.charAt(0);           // "H"
str.indexOf("World");    // 6
str.slice(0, 5);         // "Hello"
str.split(" ");          // ["Hello", "World"]
str.replace("World", "JS"); // "Hello JS"
```

**3. Boolean**

Represents true/false values.

```
let isActive = true;
let isComplete = false;
```

```
// Boolean conversion
Boolean(1);        // true
Boolean(0);        // false
Boolean("");       // false
Boolean("hello");  // true
```

**4. Undefined**

Variable declared but not assigned a value.

```
let name;
console.log(name); // undefined

let user = {age: 25};
console.log(user.name); // undefined
```

**5. Null**

Intentionally empty value.

```
let data = null; // Explicitly set to empty
```

**6. Symbol (ES6)**

Unique identifier.

```
let id1 = Symbol("id");
let id2 = Symbol("id");
console.log(id1 === id2); // false (always unique)
```

**7. BigInt (ES2020)**

For very large integers.

```
let bigNumber = 1234567890123456789012345678901234567890n;
let anotherBig = BigInt("1234567890123456789012345678901234567890");
```

## NON-PRIMITIVE DATA TYPE

**8. Object**

Collection of key-value pairs.

```
// Object literal
```

```
let person = {
    name: "John",
    age: 25,
    isStudent: true
};

// Arrays (special type of object)
let numbers = [1, 2, 3, 4, 5];
let mixed = [1, "hello", true, null];

// Functions (special type of object)
function greet() {
    return "Hello!";
}
```

---

## PART 3: TYPE CHECKING & CONVERSION

### Checking Data Types

```
typeof 42;            // "number"
typeof "hello";       // "string"
typeof true;          // "boolean"
typeof undefined;     // "undefined"
typeof null;          // "object" (this is a known bug!)
typeof {};            // "object"
typeof [];            // "object"
typeof function(){}; // "function"
```

### Better Type Checking

```
// For arrays
Array.isArray([1, 2, 3]);     // true
Array.isArray("hello");       // false

// For null
let value = null;
value === null;               // true

// For objects (excluding arrays and null)
function isObject(val) {
    return typeof val === 'object' && val !== null && !Array.isArray(val);
}
```

### Type Conversion

**Implicit Conversion (Coercion)**

```
// String + Number = String
"5" + 3;        // "53"
"5" + true;     // "5true"

// Number operations
"5" - 3;        // 2
"5" * 2;        // 10
"5" / 1;        // 5

// Boolean context
if ("hello") {} // true (non-empty string)
if (0) {}       // false (zero is falsy)
if ([]) {}      // true (empty array is truthy)
```

**Explicit Conversion**

```
// To String
String(123);           // "123"
(123).toString();      // "123"
123 + "";              // "123"

// To Number
Number("123");         // 123
parseInt("123px");     // 123
parseFloat("12.34px"); // 12.34
+"123";                // 123

// To Boolean
Boolean(1);            // true
Boolean(0);            // false
!!1;                   // true (double negation)
```

---

# PART 4: ADVANCED CONCEPTS

## Truthy and Falsy Values

**Falsy Values (only 8):**

```
false
0
-0
0n (BigInt zero)
"" (empty string)
null
```

undefined
NaN

**Everything else is truthy:**

true
1
-1
"hello"
"0" (string zero)
[] (empty array)
{} (empty object)
function() {}

## Comparison Operators

### Loose Equality (==) vs Strict Equality (===)

```
// Loose equality (converts types)
5 == "5";        // true
0 == false;      // true
null == undefined; // true

// Strict equality (no conversion)
5 === "5";       // false
0 === false;     // false
null === undefined; // false
```

**Always use === (strict equality) unless you specifically need type conversion!**

## Variable Scope

### Global Scope

```
var globalVar = "I'm global";
let globalLet = "I'm also global";

function test() {
   console.log(globalVar); // Can access global variables
}
```

### Function Scope

```
function myFunction() {
   var functionScoped = "Only inside function";
   let alsoFunctionScoped = "Also only inside function";
}
```

```
// console.log(functionScoped); // Error: not defined
```

**Block Scope (let & const only)**

```
if (true) {
   var varVariable = "var is function-scoped";
   let letVariable = "let is block-scoped";
   const constVariable = "const is also block-scoped";
}

console.log(varVariable);    // Works
// console.log(letVariable); // Error: not defined
// console.log(constVariable); // Error: not defined
```

### Hoisting

```
// Variable hoisting
console.log(hoistedVar); // undefined (not error)
var hoistedVar = "Hello";

// Let and const are not hoisted the same way
// console.log(letVar); // Error: Cannot access before initialization
let letVar = "Hello";

// Function hoisting
sayHello(); // Works! (function is hoisted)

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

---

# PART 5: PRACTICAL PROBLEMS

## Beginner Problems

**Problem 1: Variable Declaration and Assignment**

```
// Create variables for a user profile
let userName = "john_doe";
let userAge = 25;
let isLoggedIn = true;
const userID = 12345;

console.log("User: " + userName);
console.log("Age: " + userAge);
console.log("Logged in: " + isLoggedIn);
console.log("ID: " + userID);
```

**Problem 2: Type Checking Function**

```
function checkType(value) {
   console.log(`Value: ${value}, Type: ${typeof value}`);
}

checkType(42);
checkType("hello");
checkType(true);
checkType(null);
checkType(undefined);
checkType([1, 2, 3]);
```

**Problem 3: String Manipulation**

```
let firstName = "john";
let lastName = "DOE";

// Fix the names (capitalize first letter, lowercase rest)
function fixName(name) {
   return name.charAt(0).toUpperCase() + name.slice(1).toLowerCase();
}

let fullName = fixName(firstName) + " " + fixName(lastName);
console.log(fullName); // "John Doe"
```

## Intermediate Problems

**Problem 4: Type Conversion Challenge**

```
function convertAndCalculate(a, b, operation) {
   // Convert inputs to numbers
   let num1 = Number(a);
   let num2 = Number(b);

   // Check if conversion was successful
   if (isNaN(num1) || isNaN(num2)) {
      return "Invalid input: cannot convert to number";
   }

   switch(operation) {
      case "add":
         return num1 + num2;
      case "subtract":
            return num1 - num2;
        case "multiply":
            return num1 * num2;
        case "divide":
            return num2 !== 0 ? num1 / num2 : "Cannot divide by zero";
        default:
            return "Invalid operation";
    }
}

// Test cases
console.log(convertAndCalculate("5", "3", "add"));      // 8
console.log(convertAndCalculate("10", "hello", "add")); // Invalid input
console.log(convertAndCalculate("10", "0", "divide"));  // Cannot divide by zero
```

**Problem 5: Truthy/Falsy Detector**

```
function checkTruthiness(value) {
    return {
        value: value,
        type: typeof value,
        isTruthy: !!value,
        description: !!value ? "This value is truthy" : "This value is falsy"
    };
}

// Test with different values
let testValues = [0, 1, "", "hello", null, undefined, [], {}, false, true, NaN];

testValues.forEach(val => {
    console.log(checkTruthiness(val));
});
```

## Advanced Problems

**Problem 6: Deep Type Checker**

```
function getDetailedType(value) {
    if (value === null) return "null";
    if (Array.isArray(value)) return "array";
    if (value instanceof Date) return "date";
    if (value instanceof RegExp) return "regexp";

    let type = typeof value;

    if (type === "object") {
        return "object";
    }
    if (type === "number") {
        if (isNaN(value)) return "NaN";
        if (!isFinite(value)) return "infinity";
        if (Number.isInteger(value)) return "integer";
        return "float";
    }

    return type;
}

// Test cases
console.log(getDetailedType(42));           // "integer"
console.log(getDetailedType(3.14));         // "float"
console.log(getDetailedType(NaN));          // "NaN"
console.log(getDetailedType([]));           // "array"
console.log(getDetailedType({}));           // "object"
console.log(getDetailedType(null));         // "null"
console.log(getDetailedType(new Date()));   // "date"
```

**Problem 7: Variable Scope Challenge**

```
var globalVar = "I'm global";
let globalLet = "I'm also global";

function scopeTest() {
    var functionVar = "I'm function scoped";
    let functionLet = "I'm also function scoped";

    if (true) {
        var blockVar = "I'm still function scoped";
        let blockLet = "I'm block scoped";
        const blockConst = "I'm also block scoped";

        console.log("Inside block:");
        console.log(globalVar, globalLet, functionVar, functionLet, blockVar, blockLet,
blockConst);
    }

    console.log("Outside block:");
    console.log(globalVar, globalLet, functionVar, functionLet, blockVar);
    // console.log(blockLet); // This would cause an error
    // console.log(blockConst); // This would cause an error
}

scopeTest();
```

# PART 6: COMMON MISTAKES & BEST PRACTICES

## Common Mistakes

### 1. Using var instead of let/const

```
// Bad
for (var i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 100); // Prints 3, 3, 3
}

// Good
for (let i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 100); // Prints 0, 1, 2
}
```

### 2. Not using strict equality

```
// Bad
if (userInput == 0) {
  // This will match 0, "0", false, "", null, undefined
}

// Good
if (userInput === 0) {
  // This only matches the number 0
}
```

### 3. Not checking for null/undefined

```
// Bad
function processUser(user) {
  return user.name.toUpperCase(); // Error if user is null
}

// Good
function processUser(user) {
  if (user && user.name) {
    return user.name.toUpperCase();
  }
  return "No user name";
}
```

## Best Practices

1. **Use `const` by default, `let` when you need to reassign, avoid `var`**
2. **Always use strict equality (`===`) unless you specifically need type coercion**

3. **Give variables descriptive names**
4. **Initialize variables when you declare them**
5. **Check for null/undefined before using object properties**
6. **Use template literals for string interpolation**

---

# PART 7: PRACTICE EXERCISES

## Exercise Set 1: Basics (Solve these first)

### Variable Declaration

```
 // Create variables for a student record
// Name: "Alice Johnson"
// Age: 20
// Grade: "A"
// Is Enrolled: true
// Student ID: 2024001 (this shouldn't change)
```

1.

### Type Identification

```
 // Write a function that takes any value and returns its type
// Handle special cases like null, arrays, etc.
function identifyType(value) {
   // Your code here
}
```

2.

### String Operations

```
 // Given firstName and lastName, create a full name
// Make sure first letter of each name is capitalized
let firstName = "jOhN";
let lastName = "dOE";
// Expected output: "John Doe"
```

3.

## Exercise Set 2: Intermediate

### Safe Type Conversion

```
 // Write a function that safely converts string to number
// Return the number if valid, otherwise return 0
function safeStringToNumber(str) {
    // Your code here
}
```

4.

**Comparison Function**

```
 // Write a function that compares two values and returns:
// "equal" if they're strictly equal
// "loose equal" if they're loosely equal but not strictly
// "not equal" if they're not equal at all
function compareValues(a, b) {
    // Your code here
}
```

5.

## Exercise Set 3: Advanced

**Variable Validator**

```
 // Write a function that checks if a string is a valid JavaScript variable name
function isValidVariableName(name) {
    // Your code here
}
```

6.

**Deep Clone Detector**

```
 // Write a function that checks if two objects have the same structure and values
function deepEqual(obj1, obj2) {
    // Your code here
}
```

7.

This comprehensive guide covers everything you need to master JavaScript variables and data types. Practice the exercises, and you'll have a solid foundation!