JavaScript Fundamentals - Part 1

> 🎓 **CS Perspective:** This section covers the foundational building blocks of JavaScript as a programming language. You'll learn about **type systems** (JavaScript uses dynamic/weak typing), **variable binding and scope** (how identifiers map to memory locations), and **primitive data types** (the atomic units of data representation). Understanding `let`, `const`, and `var` relates to concepts like mutability, block vs. function scoping, and the temporal dead zone—all important for understanding how the JS runtime manages memory and execution context.

📌 Linking JavaScript to HTML

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

- Place at the end of `<body>` or use `defer` attribute
- Use `console.log()` to output values to the browser console

📌 Values and Variables

Values

- The smallest unit of information in JS
- Can be **numbers**, **strings**, **booleans**, etc.

Variables

- Containers that store values
- Declared with `let`, `const`, or `var`

```
let firstName = "Jonas";
console.log(firstName); // Jonas
```

Variable Naming Rules

✅ Allowed	❌ Not Allowed
`firstName`	Starting with number (`2name`)
`_privateVar`	Reserved keywords (`function`, `new`)
`$function`	Containing `-` or spaces
`myFirstJob`	

Conventions

- Use **camelCase** for variables: `myFirstJob`
- Use **UPPERCASE** for constants: `const PI = 3.1415`
- Use descriptive names: `myCurrentJob` > `job1`

📌 Data Types

JavaScript has **7 primitive data types**:

Type	Description	Example
Number	Floating point numbers	`23`, `3.14`
String	Sequence of characters	`"Hello"`
Boolean	Logical true/false	`true`, `false`
Undefined	Variable declared but not assigned	`let x;`
Null	Intentionally empty value	`null`
Symbol	Unique identifier (ES6)	`Symbol()`
BigInt	Large integers (ES2020)	`9007199254740991n`

Dynamic Typing

- JavaScript automatically determines the type
- Types can change during runtime

```
let x = 23;          // Number
x = "hello";         // Now String
```

typeof Operator

```
typeof 23           // "number"
typeof "Jonas"      // "string"
typeof true         // "boolean"
typeof undefined    // "undefined"
typeof null         // "object" ⚠ (known bug)
```

📌 Variable Declaration: let, const, var

Keyword	Reassign	Redeclare	Scope	Hoisting
`let`	✅	❌	Block	TDZ
`const`	❌	❌	Block	TDZ

Keyword	Reassign	Redeclare	Scope	Hoisting
`var`	✅	✅	Function	`undefined`

```
let age = 30;
age = 31;              // ✅ OK

const birthYear = 1991;
birthYear = 1990;      // ❌ TypeError

var job = 'programmer';
var job = 'teacher';   // ✅ OK (but avoid!)
```

Best Practices

- **Default to `const`** unless you need to reassign
- Use `let` when variable needs to change
- **Avoid `var`** (legacy, function-scoped)

📌 Operators

Math Operators

```
const sum = 10 + 5;         // 15
const diff = 10 - 5;        // 5
const product = 10 * 5;     // 50
const quotient = 10 / 5;    // 2
const power = 2 ** 3;       // 8 (2³)
const remainder = 10 % 3;   // 1
```

String Concatenation

```
const firstName = 'Jonas';
const lastName = 'Smith';
console.log(firstName + ' ' + lastName); // "Jonas Smith"
```

Assignment Operators

```
let x = 10;
x += 5;   // x = x + 5  → 15
x -= 3;   // x = x - 3  → 12
x *= 2;   // x = x * 2  → 24
x /= 4;   // x = x / 4  → 6
```

```
x++;      // x = x + 1  → 7
x--;      // x = x - 1  → 6
```

Comparison Operators

```
10 > 5    // true
10 < 5    // false
10 >= 10  // true
10 <= 9   // false
10 === 10 // true (strict equality)
10 == "10"// true (loose equality)
10 !== 5  // true (strict inequality)
```

📌 Operator Precedence

Higher precedence executes first:

Priority	Operator	Example
1	`()`	`(a + b)`
2	`**`	`2 ** 3`
3	`*`, `/`, `%`	`a * b`
4	`+`, `-`	`a + b`
5	`<`, `>`, `<=`, `>=`	`a > b`
6	`===`, `!==`	`a === b`
7	`&&`	`a && b`
8	`||`	`a || b`
9	`=`	`a = b`

```
// Example
const result = 3 + 4 * 2;    // 11 (not 14)
const result2 = (3 + 4) * 2; // 14
```

📌 Strings and Template Literals

String Concatenation (Old Way)

```
const name = 'Jonas';
const age = 46;
const str = "I'm " + name + ', and I am ' + age + ' years old.';
```

Template Literals (ES6) ✨

```
const str = `I'm ${name}, and I am ${age} years old.`;
```

Multi-line Strings

```
// Old way
const multi = 'Line 1\n\
Line 2';

// Template literal (easier)
const multi = `Line 1
Line 2`;
```

📌 if/else Statements

```
const age = 15;

if (age >= 18) {
  console.log('You can drive! 🚗 ');
} else {
  const yearsLeft = 18 - age;
  console.log(`Wait ${yearsLeft} more years`);
}
```

else if

```
if (score >= 90) {
  grade = 'A';
} else if (score >= 80) {
  grade = 'B';
} else if (score >= 70) {
  grade = 'C';
} else {
  grade = 'F';
}
```

📌 Type Conversion vs Coercion

Type Conversion (Manual)

```
// String to Number
const input = '1991';
Number(input);        // 1991
Number('Jonas');      // NaN (Not a Number)

// Number to String
String(23);           // "23"
```

Type Coercion (Automatic)

```
// + triggers string coercion
'I am ' + 23 + ' years old'  // "I am 23 years old"
'23' + '10' + 3              // "23103"

// -, *, / trigger number coercion
'23' - '10' - 3              // 10
'23' * '2'                   // 46
'23' / '2'                   // 11.5
```

Tricky Example

```
let n = '1' + 1;  // "11" (string)
n = n - 1;        // 10 (number)
```

📌 Truthy and Falsy Values

5 Falsy Values

Value	Boolean
`0`	`false`
`''` (empty string)	`false`
`undefined`	`false`
`null`	`false`
`NaN`	`false`

Everything else is **truthy** (including `{}`, `[]`, `'0'`)

```
// Used in conditionals
const money = 0;
if (money) {
  console.log("Don't spend it all");
} else {
  console.log('Get a job!'); // ✅ This runs
}
```

📌 Equality Operators: == vs ===

Operator	Name	Coercion
`===`	Strict equality	❌ No
`==`	Loose equality	✅ Yes
`!==`	Strict inequality	❌ No
`!=`	Loose inequality	✅ Yes

```
18 === 18     // true
'18' === 18   // false
'18' == 18    // true (coercion)

18 !== '18'   // true
18 != '18'    // false (coercion)
```

⚠ Best Practice

Always use `===` and `!==` to avoid unexpected bugs!

📌 Logical Operators

Operator	Name	Description
`&&`	AND	Both must be true
`||`	OR	At least one true
`!`	NOT	Inverts boolean

```
const a = true;
const b = false;

a && b    // false
a || b    // true
!a        // false
```

Practical Example

```
const hasLicense = true;
const hasGoodVision = true;
const isTired = false;

if (hasLicense && hasGoodVision && !isTired) {
  console.log('You can drive!');
}
```

📌 The switch Statement

Alternative to multiple `if/else if`:

```
const day = 'monday';

switch (day) {
  case 'monday':
    console.log('Start of week');
    break;
  case 'tuesday':
  case 'wednesday':
    console.log('Midweek');
    break;
  case 'friday':
    console.log('Almost weekend!');
    break;
  default:
    console.log('Invalid day');
}
```

Key Points

- Uses **strict equality** (`===`)
- `break` prevents fall-through
- `default` handles unmatched cases
- Multiple cases can share same code

📌 Statements vs Expressions

Expression

- Produces a value

```
3 + 4           // 7
true && false   // false
1991            // 1991
```

Statement

- Performs an action, doesn't produce a value

```
if (23 > 10) {
  const str = '23 is bigger';
}
```

Why It Matters

- Template literals only accept **expressions**

```
console.log(`I am ${2037 - 1991} years old`); // ✅
console.log(`${if (true) {}}`);               // ❌ Error
```

📌 The Ternary (Conditional) Operator

Syntax: `condition ? valueIfTrue : valueIfFalse`

```
const age = 23;

// Instead of if/else
const drink = age >= 18 ? 'wine 🍷' : 'water 💧';

// Can be used in template literals
console.log(`I like to drink ${age >= 18 ? 'wine' : 'water'}`);
```

When to Use

- ✅ Simple conditional assignments
- ❌ Complex logic (use if/else instead)

🧪 Quick Reference Cheatsheet

```
// Variables
let x = 10;       // Reassignable
const y = 20;     // Constant
```

```
var z = 30;       // Avoid (legacy)

// Types
typeof 42         // "number"
typeof "hi"       // "string"
typeof true       // "boolean"
typeof undefined  // "undefined"

// Conversions
Number("23")      // 23
String(23)        // "23"
Boolean(1)        // true

// Template literal
`Hello ${name}`

// Strict equality (always use!)
===  !==

// Logical
&&   ||   !

// Ternary
condition ? ifTrue : ifFalse
```

✅ Exam Tips

1. **Always use** `===` instead of `==`
2. **Default to** `const`, use `let` only when needed
3. **Avoid** `var` - it has function scope, not block scope
4. Remember the **5 falsy values**: `0`, `''`, `undefined`, `null`, `NaN`
5. **Template literals** use backticks `` ` `` not quotes
6. `typeof null` returns `"object"` (JS bug!)
7. Use **parentheses** to control operator precedence
8. `NaN` means "Not a Number" but `typeof NaN` is `"number"`