

JavaScript Fundamentals: A Conversational Guide

Estimated reading time: 45 minutes 🕒

Introduction

User: Hi there! I'm completely new to JavaScript and want to learn the fundamentals. Can you help me understand the basics?

Expert: Absolutely! 😊 I'd be happy to guide you through JavaScript fundamentals. JavaScript is one of the most popular programming languages in the world, powering nearly every modern website. We'll cover everything from basic syntax to more advanced concepts like loops and operators.

User: That sounds great! What topics will we be covering?

Expert: We'll be exploring these key areas:

1. Code structure
2. Variables and data types
3. Basic operators and math
4. Comparisons
5. Conditional branching with if/else statements
6. Logical operators
7. The nullish coalescing operator

Ready to get started?

User: Ready! Let's dive in!

Code Structure

User: So what's the basic structure of JavaScript code? How do I even write a simple program?

Expert: Great question! JavaScript code is made up of statements - these are commands that perform actions. Let's start with the simplest example - showing an alert:

```
alert('Hello, World!');
```

User: That seems simple enough. Do I need to add anything special at the end of statements?

Expert: JavaScript statements are usually separated with a semicolon (;). For example:

```
alert('Hello'); alert('World');
```

But for readability, we typically write each statement on its own line:

```
alert('Hello');  
alert('World');
```

User: Do I always need to include semicolons?

Expert: In many cases, JavaScript allows you to omit semicolons when you have a line break. For example:

```
alert('Hello')  
alert('World')
```

This works because JavaScript uses something called "automatic semicolon insertion." However...



Best Practice: Always use semicolons to end your statements, even when they're optional! This prevents unexpected errors and makes your code more reliable, especially as you write more complex programs.

User: Are there any special characters or syntax I should know about for commenting code?

Expert: Yes! Comments are crucial for explaining your code. JavaScript supports two types:

1. Single-line comments start with `//`:

```
// This is a single-line comment  
alert('Hello'); // This comment follows a statement
```

2. Multi-line comments start with `/*` and end with `*/`:

```
/* This is  
a multi-line comment  
that spans multiple lines */  
alert('Hello');
```

User: Can I use comments to temporarily disable code?

Expert: Absolutely! That's a common practice during development:

```
// alert('This code won't run');  
alert('This code will run');
```

Or for multiple lines:

```
/*  
alert('This code won't run');  
alert('This also won't run');  
*/
```



Fun Fact: The `/* */` comment style comes from the C programming language, created in the early 1970s. JavaScript borrowed many syntax elements from C and its descendants!

Variables

User: How do I store and use data in JavaScript?

Expert: For that, we use variables - they're like labeled boxes that store data. In modern JavaScript, we declare variables using the `let` keyword:

```
let message = 'Hello!';  
alert(message); // Shows "Hello!"
```

User: What if I just want to declare a variable without giving it a value yet?

Expert: You can do that! Then assign a value later:


```
let message;           // Declare a variable  
message = 'Hello!';    // Assign a value  
alert(message);        // Shows "Hello!"
```

User: Is there a way to declare a variable that can't be changed?

Expert: Yes, that's where the `const` keyword comes in:

```
const myBirthday = '18.04.1982';  
// myBirthday = '01.01.2000'; // This would cause an error!
```

Constants can't be reassigned after they're declared.

 **Convention Alert:** For constants that are known before execution (like colors, fixed values), use uppercase naming with underscores:



```
const COLOR_RED = "#F00";  
const COLOR_GREEN = "#0F0";  
const PI = 3.14;
```

User: What should I name my variables? Are there any rules?

Expert: Great question! Variable naming is actually really important:

1. Names can only contain letters, digits, `$` and `_`
2. The first character cannot be a digit
3. Case matters (`apple` and `APPLE` are different variables)
4. You can't use reserved words like `let`, `class`, `return`, etc.

Good variable names are descriptive and tell you what they contain:

```
let userName = 'John';    // Good   
let u = 'John';           // Bad - too short   
let data = 'John';        // Bad - too vague 
```

User: Are there any naming conventions I should follow?

Expert: Yes! JavaScript commonly uses "camelCase" for variable names - start with lowercase and capitalize each new word:

```
let firstName = 'John';
let isLoggedIn = true;
let shoppingCartTotal = 125.99;
```

Data Types

User: What kind of data can I store in JavaScript variables?

Expert: JavaScript has eight basic data types. Let me walk you through them:

1. **Number** - for both integer and floating-point numbers:

```
let n = 123;
n = 12.345;
```

2. **BigInt** - for very large integers (append `n`):

```
const bigInt = 1234567890123456789012345678901234567890n;
```

3. **String** - for text:

```
let str = "Hello";
let str2 = 'Single quotes work too';
let phrase = `Can embed ${str}`; // Backticks allow embedding variables
```

4. **Boolean** - true/false:

```
let isChecked = true;
let isDisabled = false;
```

5. **null** - represents "nothing" or "empty":

```
let age = null; // age is unknown or empty
```

6. **undefined** - represents a value that hasn't been assigned:

```
let x; // value is undefined
```

7. **Symbol** - for unique identifiers (advanced)

8. **Object** - for complex data structures (we'll cover this later)

User: How can I check what type a value is?

Expert: Use the `typeof` operator:

```
typeof 0           // "number"
typeof "Hello"     // "string"
typeof true        // "boolean"
typeof undefined   // "undefined"
typeof alert       // "function"
```



Quirky Detail: `typeof null` returns `"object"`, which is actually a known bug in JavaScript! Null is not an object; it's a primitive value, but this behavior has been kept for backward compatibility.

User: Can I convert between different types?

Expert: Yes, JavaScript allows type conversion. Here are some common conversions:

String conversion:

```
let value = true;
value = String(value); // now value is "true"
```

Numeric conversion:

```
let str = "123";
let num = Number(str); // becomes number 123
```

Boolean conversion:

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

Basic Operators and Math

User: What mathematical operations can I perform in JavaScript?

Expert: JavaScript supports all the standard math operations:

```
let a = 5;
let b = 2;

alert(a + b); // 7  (addition)
alert(a - b); // 3  (subtraction)
alert(a * b); // 10 (multiplication)
alert(a / b); // 2.5 (division)
alert(a % b); // 1  (remainder)
alert(a ** b); // 25 (exponentiation - a to the power of b)
```

User: What about the plus sign? I've seen it used with strings too.

Expert: Great observation! The `+` operator is special. It does two things:

1. Adds numbers: `5 + 2 = 7`
2. Concatenates strings: `"Hello" + "World" = "HelloWorld"`

If any operand is a string, the others are converted to strings too:

```
alert('1' + 2);      // "12" (string concatenation)
alert(2 + 2 + '1');  // "41" (not "221")
alert('1' + 2 + 2);  // "122" (not "14")
```

User: That's a bit confusing. How can I make sure something is treated as a number?

Expert: You can use the unary `+` operator to convert values to numbers:

```
let apples = "2";
let oranges = "3";

// Without conversion
alert(apples + oranges); // "23" (string concatenation)


// With conversion
alert(+apples + +oranges); // 5 (numeric addition)
```

User: What about when we have multiple operations in one expression?

Expert: JavaScript follows an order of operations (operator precedence):

1. Parentheses `()`
2. Unary operators (like unary `+` or `-`)
3. Multiplication/division
4. Addition/subtraction
5. Assignment `=`

```
let result = 2 + 2 * 2; // 6, not 8
let result2 = (2 + 2) * 2; // 8
```

 **Clarity Tip:** When in doubt, use parentheses to make your intentions clear! Even when not strictly necessary, they can make your code more readable and prevent mistakes.

User: What about shortcuts for modifying variables?

Expert: JavaScript has handy "modify-and-assign" operators:

```
let n = 2;
n += 5; // Same as n = n + 5
n *= 2; // Same as n = n * 2

// Works with all arithmetic operators
n -= 1; // Same as n = n - 1
n /= 2; // Same as n = n / 2
```

And for adding or subtracting 1, we have increment/decrement operators:

```
let counter = 1;
counter++; // Same as counter = counter + 1
++counter; // Same as counter = counter + 1

counter--; // Same as counter = counter - 1
--counter; // Same as counter = counter - 1
```

User: Is there a difference between `counter++` and `++counter`?

Expert: Yes! Great question. They both increment `counter`, but:

- `counter++` (postfix) returns the old value, then increments
- `++counter` (prefix) increments first, then returns the new value

```
let a = 1;
let b = ++a; // a = 2, b = 2

let c = 1;
let d = c++; // c = 2, d = 1
```

Comparisons

User: How do I compare values in JavaScript?

Expert: We use comparison operators that return boolean values (true or false):

```
alert(2 > 1); // true (greater than)
alert(2 < 1); // false (less than)
alert(2 >= 1); // true (greater than or equal)
alert(2 <= 1); // false (less than or equal)
alert(2 == 1); // false (equality)
alert(2 !== 1); // true (not equal)
```

User: How does JavaScript compare strings?

Expert: Strings are compared letter-by-letter, using the Unicode value of each character:

```
alert('Z' > 'A');      // true
alert('Glow' > 'Glee'); // true
alert('Bee' > 'Be');   // true
```


The comparison works alphabetically, but remember it's case-sensitive - capital letters are "less than" lowercase:

```
alert('a' > 'Z'); // true
```

User: What about comparing different types?

Expert: When comparing different types, JavaScript converts the values to numbers:

```
alert('2' > 1);      // true, string '2' becomes number 2
alert('01' == 1);    // true, string '01' becomes number 1
alert(true == 1);    // true, true becomes number 1
alert(false == 0);   // true, false becomes number 0
```


 **Equality Quirk:** `null == undefined` is true in JavaScript, but `null` or `undefined` are not equal to any other value. This is a special case in the language!

User: Is there a way to compare without this automatic conversion?

Expert: Yes! Use the strict equality operator `===` and strict non-equality operator `!==`:

```
alert(0 == false);    // true (0 converts to false)
alert(0 === false);   // false (different types)

alert('0' == 0);      // true (string converts to number)
alert('0' === 0);     // false (different types)
```

 **Best Practice:** Use strict equality (`===`) and strict inequality (`!==`) whenever possible to avoid unexpected type conversions. This makes your code more predictable and reduces bugs!

Conditional Branching

User: How can I make my code do different things based on conditions?

Expert: For that, we use conditional statements, primarily the `if` statement:

```
let year = prompt('In which year was JavaScript created?', '');

if (year == 1995) {
  alert('You are right!');
}
```

User: What if I want to do something else when the condition is false?

Expert: You can use the `else` clause:


```
let year = prompt('In which year was JavaScript created?', '');

if (year == 1995) {
  alert('You are right!');
} else {
  alert('Wrong answer!');
}
```

User: What about multiple conditions?

Expert: You can chain conditions with `else if`:

```
let year = prompt('In which year was JavaScript created?', '');

if (year < 1995) {
  alert('Too early!');
} else if (year > 1995) {
  alert('Too late!');
} else {
  alert('Exactly!');
}
```

User: Is there a shorter way to write a simple condition?

Expert: Yes! For simple conditional assignments, you can use the ternary operator `?:`:

```
// Instead of:
let accessAllowed;
if (age > 18) {
  accessAllowed = true;
} else {
  accessAllowed = false;
}


// You can write:
let accessAllowed = (age > 18) ? true : false;

// Or even simpler:
let accessAllowed = age > 18;
```

User: Can I use multiple conditions with the ternary operator?

Expert: Yes, you can chain them, but it gets harder to read:

```
let message = (age < 3) ? 'Hi, baby!' :
  (age < 18) ? 'Hello!' :
  (age < 100) ? 'Greetings!' :
  'What an unusual age!';
```

 **Readability Tip:** While the ternary operator is convenient for simple conditions, use regular `if/else` statements for complex logic. Your future self (and other developers) will thank you!

Logical Operators

User: How do I combine multiple conditions?

Expert: JavaScript has three main logical operators:

1. **OR** (`||`): Returns `true` if ANY operand is `true`
2. **AND** (`&&`): Returns `true` if ALL operands are `true`
3. **NOT** (`!`): Returns the opposite boolean value

Let's see them in action:

```
// OR examples
alert(true || false); // true
alert(false || false); // false

// AND examples
alert(true && true); // true
alert(true && false); // false

// NOT examples
alert(!true); // false
alert(!false); // true
```

User: How would I use these in real code?

Expert: Here are some practical examples:

```
// OR is great for checking if ANY condition is true
let hour = 12;
let isWeekend = true;

if (hour < 10 || hour > 18 || isWeekend) {
  alert('The office is closed.');// it is weekend
}

// AND is great for checking if ALL conditions are true
let hour = 12;
let minute = 30;

if (hour == 12 && minute == 30) {
  alert('The time is 12:30');
}
```

User: I've heard that OR can also return values, not just true or false?

Expert: Very observant! Both `||` and `&&` have an interesting behavior in JavaScript - they actually return one of their operand values, not just `true` or `false`.

OR (`||`) returns the first truthy value it finds, or the last value if all are falsy:

```
alert(null || 0 || "" || undefined); // all falsy, returns the last value: undefined
alert(null || 0 || "Hello" || undefined); // returns "Hello" (first truthy value)
```

This leads to a common pattern - providing default values:

```
let firstName = "";
let lastName = "";
let nickName = "SuperCoder";

// Shows the first non-empty value
alert(firstName || lastName || nickName || "Anonymous"); // SuperCoder
```

User: What about AND? Does it have a similar behavior?

Expert: Yes! AND (`&&`) returns the first falsy value it finds, or the last value if all are truthy:

```
alert(1 && 2 && null && 3); // returns null (first falsy value)
alert(1 && 2 && 3); // all truthy, returns the last value: 3
```

🚀 **Shortcut Evaluation:** These behaviors of `&&` and `||` are called "short-circuit evaluation" - the operators stop evaluating as soon as they can determine the result. This can be used to execute code conditionally!

```
true || alert("not printed"); // alert is never run
false || alert("printed"); // alert is run
```

User: What is the nullish coalescing operator? I've seen it in some code.

Expert: The nullish coalescing operator (`??`) is a newer addition to JavaScript. It's similar to `||` but with an important difference:

- `||` returns the first truthy value
- `??` returns the first defined value (not null or undefined)

```
let height = 0;

alert(height || 100); // 100 (because 0 is falsy)
alert(height ?? 100); // 0 (because 0 is defined)
```

This is incredibly useful when you want to provide default values only for null/undefined, but treat 0, empty strings, and false as valid values!

Loops

User: How can I repeat code multiple times?

Expert: Loops are perfect for that! JavaScript has several types of loops, but let's start with the most common ones:

The `while` loop:

```
let i = 0;
while (i < 3) {
  alert(i);
  i++;
}
// Shows 0, then 1, then 2
```

The `for` loop (most commonly used):


```
for (let i = 0; i < 3; i++) {
  alert(i);
}
// Shows 0, then 1, then 2
```

User: What are the parts of a for loop?

Expert: A `for` loop has three parts:

```
for (initialization; condition; step) {
  // loop body
}
```

1. **Initialization:** Runs once when the loop starts (`let i = 0`)
2. **Condition:** Checked before every iteration - if false, the loop stops (`i < 3`)
3. **Step:** Runs after each iteration (`i++`)

 **Loop Tip:** You can omit any part of the `for` loop, but don't forget the semicolons! For example, an infinite loop: `for(;;) { ... }`

User: What if I need to exit a loop early?

Expert: You can use the `break` statement to exit a loop immediately:

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // exit the loop when i is 5
  }
  alert(i);
}
// Shows 0, 1, 2, 3, 4
```

And you can use `continue` to skip the current iteration:

```
for (let i = 0; i < 5; i++) {  
  if (i === 2) {  
    continue; // skip iteration when i is 2  
  }  
  alert(i);  
}  
// Shows 0, 1, 3, 4
```

User: What about the switch statement? Is it related to loops?

Expert: The `switch` statement isn't a loop, but it's a way to replace multiple `if` checks. It's especially useful when you need to compare a value against multiple variants:

```
let day = 2;  
let dayName;  
  
switch (day) {  
  case 0:  
    dayName = 'Sunday';  
    break;  
  case 1:  
    dayName = 'Monday';  
    break;  
  case 2:  
    dayName = 'Tuesday';  
    break;  
  // ... and so on  
  default:  
    dayName = 'Unknown';  
}  
  
alert(dayName); // 'Tuesday'
```



Loop History: The `for` loop syntax in JavaScript was inherited from the C programming language, which was developed in the early 1970s. Many modern programming languages use this same loop structure!

Practice Exercises

User: This is a lot to take in! Do you have some exercises I could try to practice these concepts?

Expert: Absolutely! Let's try a few:

Exercise 1: Create a simple alert

```
// Create a page that shows a message "I'm JavaScript!"  
alert("I'm JavaScript!");
```

Exercise 2: Working with variables

```
// Declare two variables: admin and name
// Assign "John" to name
// Copy the value from name to admin
// Show the value of admin
let name = "John";
let admin = name;
alert(admin); // Should show "John"
```

Exercise 3: Simple sum function

```
// Fix this code to properly add two numbers from prompts
let a = prompt("First number?", 1);
let b = prompt("Second number?", 2);

// Convert strings to numbers before adding
alert(+a + +b); // Should show the sum
```

Exercise 4: Login check

```
// Simple login system
let username = prompt("Who's there?", "");

if (username === "Admin") {
  let password = prompt("Password?", "");

  if (password === "TheMaster") {
    alert("Welcome!");
  } else if (password === "" || password === null) {
    alert("Canceled");
  } else {
    alert("Wrong password");
  }
} else if (username === "" || username === null) {
  alert("Canceled");
} else {
  alert("I don't know you");
}
```

User: These are great! I think I'm starting to understand JavaScript better now. Any final advice before I start coding?

Expert: Absolutely! Here are my top tips for JavaScript beginners:

Success Tips:

1. **Practice regularly** - Even 15 minutes a day makes a big difference!
2. **Read others' code** - Looking at well-written JavaScript helps you learn patterns and best practices.
3. **Use the console** - The browser console (F12) is your best friend for testing and debugging.

4. **Be patient with yourself** - Programming is a skill that takes time to develop.
5. **Build real projects** - Apply what you learn by building small projects that interest you.

Remember that every expert was once a beginner. The JavaScript journey is rewarding, and I'm excited to see what you'll build! Happy coding! 🚀

JavaScript in Action: Building a Shopping Cart Calculator

User: Hey! I'm trying to build a simple shopping cart for an online bookstore. I need to calculate the total price with discounts and shipping. Can you help me work through this using the JavaScript concepts we just learned?

Expert: Absolutely! This is a perfect project to practice variables, operators, and conditional statements. Let's break it down step by step.

User: Great! So I need to track the book prices, quantities, maybe apply discounts if the total is over a certain amount, and add shipping costs unless they order enough for free shipping.

Expert: That's a perfect scenario! Let's start by defining our variables. What variables do you think we'll need?

User: Hmm, I guess we need variables for the prices, quantities, and then some for the calculations?

Expert: Exactly! Let's define those:

```
// Book information
let bookPrice1 = 12.99;
let bookPrice2 = 9.99;
let bookPrice3 = 15.50;

// Quantities
let quantity1 = 2;
let quantity2 = 1;
let quantity3 = 3;

// For calculations
let subtotal = 0;
let discount = 0;
let shipping = 4.99;
let finalTotal = 0;
```

Now let's calculate the subtotal. How would you use operators to do this?

User: I think I would multiply each book price by its quantity and then add them all together?

Expert: Perfect! Let's implement that:

```
// Calculate subtotal
subtotal = (bookPrice1 * quantity1) + (bookPrice2 * quantity2) + (bookPrice3 * quantity3);

// Let's see what we have so far
console.log("Subtotal: $" + subtotal);
```

User: What value would this give us? Let me try to calculate... that's $\$12.99 \times 2 + \$9.99 \times 1 + \$15.50 \times 3 = \$25.98 + \$9.99 + \$46.50 = \$82.47$, right?

Expert: That's correct! 🎉 Now let's add some conditional logic for discounts:

1. If the subtotal is over \$75, apply a 10% discount
2. If the subtotal is over \$100, apply a 15% discount

How would you write this using if statements?

User: I think I'd use if and else if statements to check the subtotal amount:

```
if (subtotal > 100) {  
    discount = subtotal * 0.15;  
} else if (subtotal > 75) {  
    discount = subtotal * 0.10;  
} else {  
    discount = 0;  
}
```

Expert: Excellent! That's exactly right. Next, let's handle the shipping cost. If the subtotal (before discount) is over \$50, shipping should be free. How would you write this condition?

User: Would this work?

```
if (subtotal > 50) {  
    shipping = 0;  
}
```

Expert: Yes, that's perfect! Now let's calculate the final total by putting it all together:

```
// Calculate final total  
finalTotal = subtotal - discount + shipping;  
  
// Show the customer their order summary  
console.log("Subtotal: $" + subtotal.toFixed(2));  
console.log("Discount: -$" + discount.toFixed(2));  
console.log("Shipping: $" + shipping.toFixed(2));  
console.log("Total: $" + finalTotal.toFixed(2));
```

User: Great! Can we put everything together and test some different scenarios?

Expert: Absolutely! Here's the complete code, and we'll test it with our current values:

```
// Book information  
let bookPrice1 = 12.99;  
let bookPrice2 = 9.99;  
let bookPrice3 = 15.50;  
  
// Quantities  
let quantity1 = 2;  
let quantity2 = 1;  
let quantity3 = 3;
```

```

// For calculations
let subtotal = 0;
let discount = 0;
let shipping = 4.99;
let finalTotal = 0;

// Calculate subtotal
subtotal = (bookPrice1 * quantity1) + (bookPrice2 * quantity2) + (bookPrice3 * quantity3);

// Apply discount based on subtotal
if (subtotal > 100) {
    discount = subtotal * 0.15;
} else if (subtotal > 75) {
    discount = subtotal * 0.10;
} else {
    discount = 0;
}

// Determine shipping cost
if (subtotal > 50) {
    shipping = 0;
}

// Calculate final total
finalTotal = subtotal - discount + shipping;

// Show the customer their order summary
console.log("Subtotal: $" + subtotal.toFixed(2));
console.log("Discount: -$" + discount.toFixed(2));
console.log("Shipping: $" + shipping.toFixed(2));
console.log("Total: $" + finalTotal.toFixed(2));

```

Let's trace through the execution:

1. Subtotal = \$82.47
2. Since \$82.47 > \$75, we apply a 10% discount: \$8.25
3. Since \$82.47 > \$50, shipping is free: \$0
4. Final total = \$82.47 - \$8.25 + \$0 = \$74.22

User: That makes sense! What if I wanted to add a coupon code discount that would apply after the regular discount?

Expert: Great idea! Let's add that. We'll need a new variable for the coupon code and its discount:

```

let couponCode = "BOOKS10";
let couponDiscount = 0;

// After calculating the regular discount, add:
if (couponCode === "BOOKS10") {
    couponDiscount = (subtotal - discount) * 0.10;
    console.log("Coupon discount: -$" + couponDiscount.toFixed(2));
    finalTotal = subtotal - discount - couponDiscount + shipping;
} else {
    finalTotal = subtotal - discount + shipping;
}

```

User: What if I want to check if the customer is a member, and if they are, give an additional 5% off?

Expert: Let's add that with another conditional check:

```

let isMember = true; // This could come from a login system
let memberDiscount = 0;

// After coupon code logic
if (isMember) {
    memberDiscount = (subtotal - discount - couponDiscount) * 0.05;
    console.log("Member discount: -$" + memberDiscount.toFixed(2));
    finalTotal = finalTotal - memberDiscount;
}

```

User: This is getting complex! Could we use the ternary operator anywhere to simplify our code?

Expert: Yes! The ternary operator would be perfect for the shipping calculation:

```

// Instead of:
if (subtotal > 50) {
    shipping = 0;
}

// We could write:
shipping = (subtotal > 50) ? 0 : 4.99;

```

And we could use it for checking membership:

```

memberDiscount = isMember ? (subtotal - discount - couponDiscount) * 0.05 : 0;

```



Tip: When building e-commerce calculations, always round monetary values appropriately and be careful with floating-point arithmetic. JavaScript's math with decimals can sometimes produce tiny rounding errors. For critical financial applications, consider using libraries designed for precise money calculations!

User: Thanks! I feel much more confident now about using variables, operators, and conditional statements in a practical scenario. Can you suggest how I might extend this for a full shopping cart with any number of items?

Expert: For a full shopping cart with variable items, you'd want to use arrays and loops, which we'll cover next! But your foundation with variables, operators, and conditionals is exactly what you needed to understand first. Great job working through this problem! 🎉