**ScopeTreasure Courses**

JAVASCRIPT COURSES · LESSON 3.7 · CODE SCOPE SERIES

# Functions in JavaScript

Reusable machines for your code: declaring and calling, parameters vs arguments, return values, expressions, scope's wall, and a real form validator.

## Introduction

You have code that works — but you are copying it again and again: calculate a bill here, calculate a bill there. A **function** fixes this. A function is a **machine you build once**: give it inputs, it does its job, and it can hand a result back. Then you ring its bell whenever you need it, from anywhere in your code.

In this lesson you will declare and call a function, learn the difference between **parameters** and **arguments**, get values back with `return`, meet function expressions, hit the wall called **scope** — honestly, with its real error — and finish with a validator function that guards a real form.

*(Roman Urdu: Function aik machine hai — aik dafa banao, phir jab chaho ghanti baja kar chala lo.)*

## Key points

- **Declaring is not running:** `function makeChai() { ... }` only **builds** the machine. Nothing happens until you **call** it: `makeChai();` — the call runs the code inside.
- **Parameters are the input slots:** `function bill(price, cups)` has two parameters — named slots waiting for values. **Arguments** are the actual values you pass in the call: in `bill(250, 3)`, the arguments are `250` and `3`.

  *(Roman Urdu: Parameter machine ke khaali khaanay hain; argument woh asal value hai jo tum daalte ho.)*
- **`return` hands a value back:** the result of `bill(250, 3)` can be stored: `let total = bill(250, 3);`. Without `return`, a function gives back `undefined`.
- **Code after `return` never runs:** `return` exits the function at once. Anything written below it inside the function is **dead code** — JavaScript will never reach it.
- **Function expression:** a function can be stored in a variable — `const add = function (a, b) { return a + b; };` — and called by the variable's name, `add(2, 3)`. The newer arrow style gets its deep lesson in 3.16.

  *(Roman Urdu: Function ko variable mein bhi rakha ja sakta hai — phir variable ka naam hi machine ka naam ban jata hai.)*
- **Scope is a wall:** a variable born **inside** a function lives inside that function's wall. Code outside cannot see it — asking for it throws a real `ReferenceError`.
- **Functions keep code honest and short:** one machine, tested once, used everywhere — a validator function can guard every field of a form.

## Code examples

### Example 1 — Build the machine, ring the bell

```
function makeChai() {
  console.log("Boiling water...");
  console.log("Adding tea and milk...");
  console.log("Chai is ready!");
}

console.log("Morning starts");
makeChai();
console.log("---");
makeChai();
```

What it does: The declaration only builds the machine — note that "Morning starts" prints **before** any chai lines, even though the function is written first in the file. Each call `makeChai();` runs the whole machine once, so the three chai lines appear twice.

**What you will see:** nine lines in the Console: `Morning starts`, then `Boiling water...`, `Adding tea and milk...`, `Chai is ready!`, then `---`, then the same three chai lines again.

### Example 2 — Parameters in, return out

```
function bill(price, cups) {
  let total = price * cups;
  return total;
  console.log("This line never runs");
}

let myBill = bill(250, 3);
console.log(myBill);
console.log(bill(200, 2));
```

What it does: The parameters `price` and `cups` receive the arguments `250` and `3`. The function multiplies them and `return`s `750`, which is stored in `myBill`. The `console.log` after `return` is dead code — it never prints. The second call shows a return value can be printed directly too.

**What you will see:** two lines in the Console: `750` and then `400`. No third line — proof that code after `return` never runs.

**Example 3 — Scope's wall, and a real form validator**

```
function checkName(name) {
  let message = "Checking: " + name;

  if (name === "") {
    return "Name is empty!";
  }
  return message + " — OK";
}

console.log(checkName("Ali"));
console.log(checkName(""));
console.log(message);
```

What it does: `checkName` is a small validator: an empty name gets a warning, a real name gets an OK. The variable `message` is born inside the function, behind scope's wall. The last line tries to read it from outside — and JavaScript answers honestly with an error.

**What you will see:** two lines print first — `Checking: Ali — OK` and `Name is empty!` — then the script stops with `ReferenceError: message is not defined`. That error is scope protecting the function's private world.

## Quick recap

- Declare to build, **call to run**: `makeChai` does nothing until `makeChai()` rings the bell.
- Parameters are slots, arguments are the real values; `return` hands a value back — and ends the function.
- Variables born inside a function stay inside its wall (**scope**); outside code gets a `ReferenceError`.

## Common mistakes

1. **Declaring but never calling** — the function exists but its code never runs, and beginners stare at a silent Console. Fix: after every declaration, write the call with brackets, like `makeChai();`.
2. **Forgetting `return`** — you calculate the right answer inside the function, but the caller receives `undefined`. Fix: if a value must come out, `return` it — printing inside is not returning.
3. **Using a function's variable outside it** — scope's wall blocks you with `ReferenceError: ... is not defined`. Fix: pass values in as arguments and get results out with `return`; do not reach through the wall.

## Practice

1. Write a function `welcome(name)` that prints `Welcome to ChaiPoint,` plus the name. Call it with three different names. Then change it to `return` the message instead of printing, store the result in a variable,

and print that.

2. Write a validator function `checkAge(age)` that returns `"Allowed"` if the age is `18` or more, and `"Not allowed"` otherwise. Test it with `20`, `18`, and `15`, and print each result.

## Practice quiz

**1. function sayHi() { console.log('Hi'); — the page breaks. What is missing?**

- The closing brace }
- A semicolon
- The word return
- Parentheses

**Answer:** The closing brace }

---

**Watch the video lesson:** https://youtu.be/0_r-uStgA6I

**Read online:** https://scopetreasure.com/notes/js-functions · More free tools and notes at https://scopetreasure.com