ScopeTreasure Courses

JAVASCRIPT COURSES · LESSON 3.16 · CODE SCOPE SERIES

ES6: Arrow Functions

The shorter function: arrow syntax, one-line returns, the object-literal trap, arrows inside map and filter, and a brief honest note on this.

Introduction

You already know how to write functions. An **arrow function** is not a new kind of function — it is the same function in a shorter costume, the ES6 way of writing it. Wherever the old forms work, the arrow works too, and in callbacks (like the ones in `map` and `filter`) it makes the code much easier to read. *(Roman Urdu: Arrow function wohi purana function hai, bas likhne ka andaaz chhota aur saaf hai.)*

Key points

- **Same function, three costumes:** declaration `function add(a, b) { return a + b; }`, expression `const add = function (a, b) { return a + b; };`, and arrow `const add = (a, b) => a + b;` — all three do exactly the same job.
- **One expression = automatic return:** when the body after `=>` is a single expression, the arrow **returns it for you** — no braces, no word `return`.
- **Block body brings `return` back:** if you write the body in braces `{ }` (for several steps), you must write `return` yourself again, exactly like a normal function.

  (Roman Urdu: Aik line wala arrow khud return karta hai, braces lagao to return lafz wapas likhna parta hai.)
- **Parameter brackets are flexible:** with **one** parameter, the brackets are optional — `price => price * 2` is fine. With **zero** parameters, keep the empty brackets: `() => "Hello"`. With two or more, brackets are required.
- **The object-literal trap:** in a one-line arrow, `{ }` is read as a **block**, not an object. Returning an object needs the object wrapped in round brackets: `() => ({ name: "Chai" })`.
- **Arrows shine in callbacks:** `prices.map(price => price * 2)` is the lesson 3.11 code with all the ceremony removed.
- **An honest note on `this`:** arrows do **not** get their own `this` — they borrow the `this` of the code around them. That is why arrows are perfect for callbacks, and the wrong choice for object methods that need their own `this`. The full `this` story belongs to the advanced lessons.

  (Roman Urdu: Arrow ka apna this nahi hota, woh bahar wala this udhaar leta hai.)

Code examples

Example 1 — Three costumes, and when return comes back

```
function addNormal(a, b) {
  return a + b;
}

const addExpression = function (a, b) {
  return a + b;
};

const addArrow = (a, b) => a + b;

console.log(addNormal(2, 3));
console.log(addExpression(2, 3));
console.log(addArrow(2, 3));

const multiplySteps = (a, b) => {
  const result = a * b;
  return result;
};
console.log(multiplySteps(4, 5));
```

What it does: The same add job is written three ways — declaration, expression and arrow — and all three give the same answer. The arrow needs no `return` because its body is one expression. The last function shows the other form: a block body with two steps, where the word `return` is required again to send the value out.

What you will see: the console prints `5`, then `5`, then `5`, then `20`.

Example 2 — Flexible brackets and the object-literal trap

```
const double = price => price * 2;
console.log(double(250));

const greet = () => "Hello, Code Scope";
console.log(greet());

const getChaiWrong = () => { name: "Masala Chai" };
console.log(getChaiWrong());

const getChai = () => ({ name: "Masala Chai", price: 250 });
console.log(getChai().name);
console.log(getChai().price);
```

What it does: `double` has one parameter, so no brackets are needed; `greet` has zero parameters, so the empty brackets stay. Then the trap: in `getChaiWrong`, the braces after the arrow are read as a **block** — the line inside is not an object being returned, so the function gives back `undefined`. The fix in `getChai` wraps the object in round brackets, and now it really is an object that comes out.

What you will see: the console prints `500`, then `Hello, Code Scope`, then `undefined` (the trap), then `Masala Chai`, then `250` (the fix working).

Example 3 — Arrows inside map and filter

```
const prices = [250, 200, 300];

const doubled = prices.map(price => price * 2);
console.log(doubled.join(", "));

const expensive = prices.filter(price => price > 220);
console.log(expensive.join(", "));

const labels = prices.map(price => "Rs " + price);
console.log(labels.join(" | "));
```

What it does: This is the lesson 3.11 factory line, rewritten with arrows. Each callback is now a single short expression: `map` doubles every price, `filter` keeps only the prices above `220`, and a second `map` turns each price into a label. No `function` word, no braces, no `return` — the machine on the belt is now one small line.

What you will see: the console prints `500, 400, 600`, then `250, 300`, then `Rs 250 | Rs 200 | Rs 300`.

Quick recap

- An arrow function is the same function, shorter: `(a, b) => a + b` returns its one expression automatically.
- One parameter needs no brackets, zero parameters keep `()`, and a block body `{ }` needs the word `return` again.
- Wrap a returned object in round brackets — `() => ({ ... })` — and remember: arrows borrow `this`, they never own it.

Common mistakes

1. **The object-literal trap** — `() => { name: "Chai" }` returns `undefined`, because JavaScript reads those braces as a block, not as an object. The fix: wrap the object in round brackets — `() => ({ name: "Chai" })`.
2. **Forgetting return in a block body** — the moment you add braces for extra steps, the automatic return is gone, and a missing `return` silently gives `undefined`. The fix: either keep the body to one expression, or write `return` inside the block.

3. **Dropping the brackets with two parameters** — `a, b => a + b` is a syntax error; the no-brackets shortcut works for **one** parameter only. The fix: zero parameters get `()`, two or more get full brackets `(a, b)`.

Practice

1. Rewrite these three jobs as arrow functions: a function that squares a number, a function that joins a first name and a last name with a space, and a function with no parameters that returns the text `Code Scope`. Print the result of each one.
2. Take `const marks = [45, 80, 62, 90, 30]`. Using only arrow callbacks, use `filter` to keep the marks of 50 and above, then use `map` to turn each kept mark into a label like `Pass: 80`. Print both results.

Practice quiz

1. What does (a, b) => a + b return?

- Nothing, you must write return
- a + b, automatically, because a one-expression body returns itself
- undefined, always
- A function, not a value

Answer: a + b, automatically, because a one-expression body returns itself

2. What happens when an arrow body uses braces { } for several steps?

- It still returns automatically
- The automatic return is gone, you must write return yourself
- It becomes a normal function declaration
- It is a syntax error

Answer: The automatic return is gone, you must write return yourself

3. How do you return an object from a one-line arrow?

- () => { name: "Chai" }
- () => ({ name: "Chai" }), wrapped in round brackets, or the braces read as a block
- () => name: "Chai"
- Arrows cannot return objects

Answer: () => ({ name: "Chai" }), wrapped in round brackets, or the braces read as a block

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

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