



JONAS.IO  
SCHMEDTMANN

# THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!



@JONASSCHMEDTMAN

SECTION

MODERN JAVASCRIPT DEVELOPMENT:  
MODULES AND TOOLING

LECTURE

DECLARATIVE AND FUNCTIONAL  
JAVASCRIPT PRINCIPLES

JS

# IMPERATIVE VS. DECLARATIVE CODE

Two fundamentally different ways  
of writing code (paradigms)

IMPERATIVE

DECLARATIVE

- 👉 Programmer explains “**HOW** to do things”
- 👉 We explain the computer *every single step* it has to follow to achieve a result
- 👉 **Example:** Step-by-step recipe of a cake
- 👉 Programmer tells “**WHAT** do do”
- 👉 We simply *describe* the way the computer should achieve the result
- 👉 The **HOW** (step-by-step instructions) gets abstracted away
- 👉 **Example:** Description of a cake

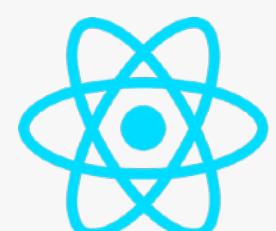
```
const arr = [2, 4, 6, 8];
const doubled = [];
for (let i = 0; i < arr.length; i++)
  doubled[i] = arr[i] * 2;
```

```
const arr = [2, 4, 6, 8];
const doubled = arr.map(n => n * 2);
```

# FUNCTIONAL PROGRAMMING PRINCIPLES

## FUNCTIONAL PROGRAMMING

- 👉 **Declarative** programming paradigm
- 👉 Based on the idea of writing software by combining many **pure functions**, avoiding **side effects** and **mutating** data
- 👉 **Side effect:** Modification (mutation) of any data **outside** of the function (mutating external variables, logging to console, writing to DOM, etc.)
- 👉 **Pure function:** Function without side effects. Does not depend on external variables. **Given the same inputs, always returns the same outputs.**
- 👉 **Immutability:** State (data) is **never** modified! Instead, state is **copied** and the copy is mutated and returned.
- 👉 Examples:



React



Redux

## FUNCTIONAL PROGRAMMING TECHNIQUES

- 👉 Try to avoid data mutations
- 👉 Use built-in methods that don't produce side effects
- 👉 Do data transformations with methods such as `.map()`, `.filter()` and `.reduce()`
- 👉 Try to avoid side effects in functions: this is of course not always possible!

## DECLARATIVE SYNTAX

- 👉 Use array and object destructuring
- 👉 Use the spread operator (...)
- 👉 Use the ternary (conditional) operator
- 👉 Use template literals