Coding & Development / JavaScript

Generate functional JavaScript code (map, filter, reduce) to transform data structures.
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Data Transformation, Array Methods, Functional Programming
Updated: May 2026
Why This Prompt Exists
Most data transformation is done with loops — which are verbose and error-prone.

You get:

- for loops for simple transformations (too much code)
- mutating original data (side effects)
- complex nested loops for simple operations
- no method chaining (hard to read)
- bugs from off-by-one errors

But functional methods are not just shorter.

They are declarative, testable, and chainable.

- map: transform each element (one-to-one)
- filter: select elements that match a condition
- reduce: accumulate into a single value
- forEach: side effects (use sparingly)
- Chaining: combine methods for complex transforms

Without functional methods, data code is verbose and buggy.

This framework forces AI to write declarative data transformations.

The Prompt

```
Assume the role of a functional programmer who transforms data with
map, filter, reduce.

Your task is to generate JavaScript code to transform an array or
object.

Generate:

1. INPUT DATA STRUCTURE (example)
   - Show what the input looks like

2. OUTPUT DATA STRUCTURE (example)
   - Show what the output should look like

3. TRANSFORMATION CODE
   - Use map, filter, reduce, Object.entries, etc.
   - Chain methods for readability

4. EXPLANATION
   - What each step does

5. EDGE CASE HANDLING
   - Empty arrays, missing properties, null values

6. TEST EXAMPLES
   - Input and expected output
```

```
INPUTS:

Input Data Structure (describe):
[E.G., "Array of objects with name, age, active"]

Desired Output Structure (describe):
[E.G., "Array of names for active users only"]

Transformation Type:
[MAP / FILTER / REDUCE / GROUP / SORT / COMBINATION]

Handle Empty/Null Values:
[YES / NO]

Performance Concern (array size):
[SMALL (<1000) / LARGE (1000+)]

RULES:
- Use map for one-to-one transformations
- Use filter for subset selection
- Use reduce for aggregation or grouping
- Chain methods for readability (not one giant function)
- Don't mutate original data (return new array/object)
- Use optional chaining (?.) for missing properties
- Use nullish coalescing (??) for default values
- Prefer arrow functions for callbacks
```

How To Use It

- Describe the input data structure clearly — include examples.

- Describe the desired output — show what you want to produce.
- If performance is a concern (large arrays), mention it.
- Test the generated code with edge cases (empty array, missing fields).
- Chain methods for complex transformations (don't nest).

Example Input

**Input Data Structure:** Array of objects: { id: number, name: string, price: number, category: string, inStock: boolean }

**Desired Output Structure:** Object where keys are categories and values are arrays of product names for in-stock products only, sorted by price ascending

**Transformation Type:** COMBINATION (filter + group + sort)

**Handle Empty/Null Values:** YES (skip products with missing name or price)

**Performance Concern:** SMALL (<1000)

Why It Works
Most data transformation uses verbose loops.

This framework improves outcomes by forcing:

- map/filter/reduce (declarative)
- method chaining (readability)
- no mutation (safety)
- edge case handling (robustness)
- test examples (verification)

Great data transformations don't loop — they map, filter, and reduce.

## Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, JavaScript frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

### Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The JavaScript Function Generator