Functional JS 101

May 2019

OpCon / GNOE

Why functional programming?

- Treat computation as individual mathematical constructs (functions)
- Reduces second/third order effects
- Forces minimal **cognitive** complexity
  - Easier to create tests for
  - Easier to debug
  - Easier to read and understand

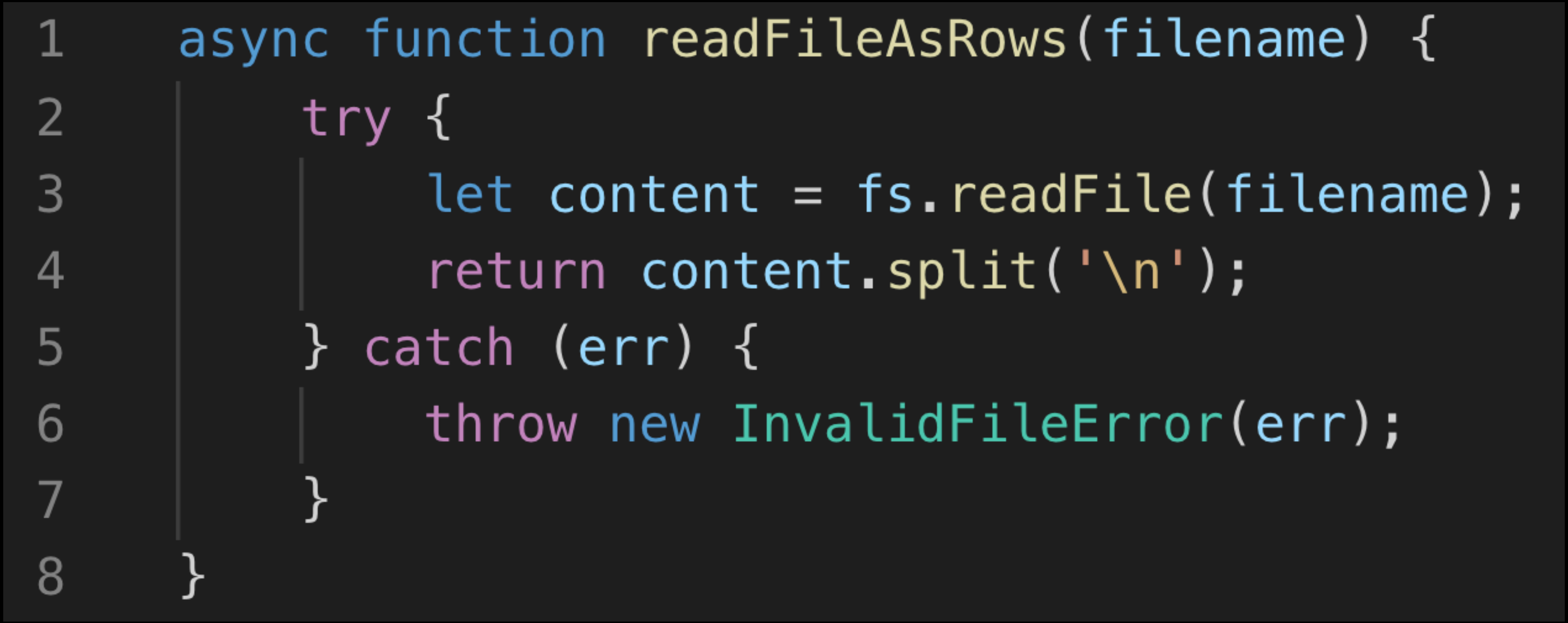
What is functional programming?

- Functions should in general:

1. Avoid changing state of any out of scope object (side effect)
2. Enforce immutability wherever possible
3. Be deterministic

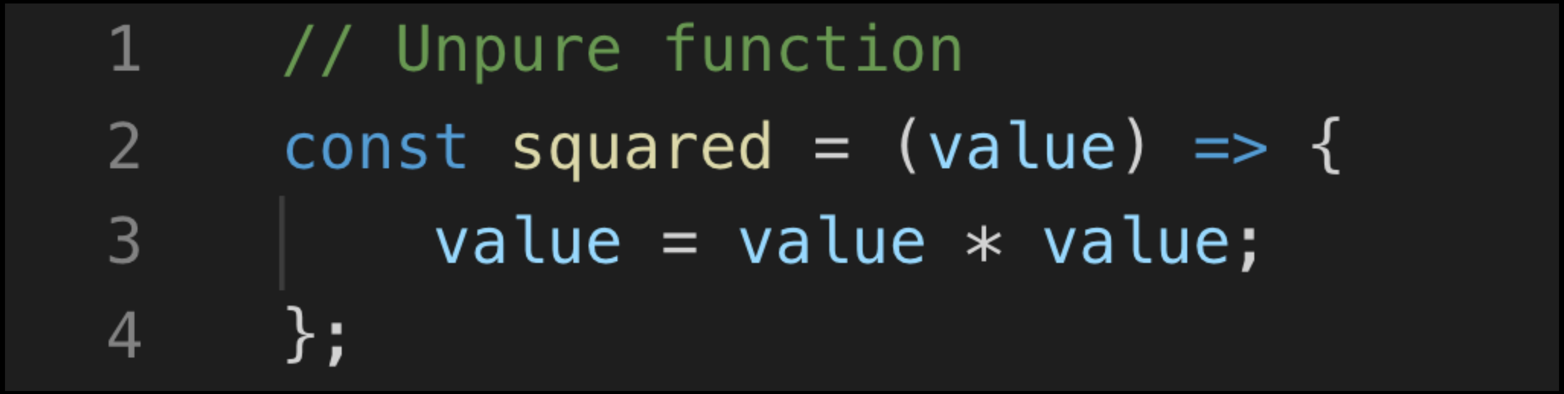
- **Pure function:** When a function is able to do these things

Pure functions



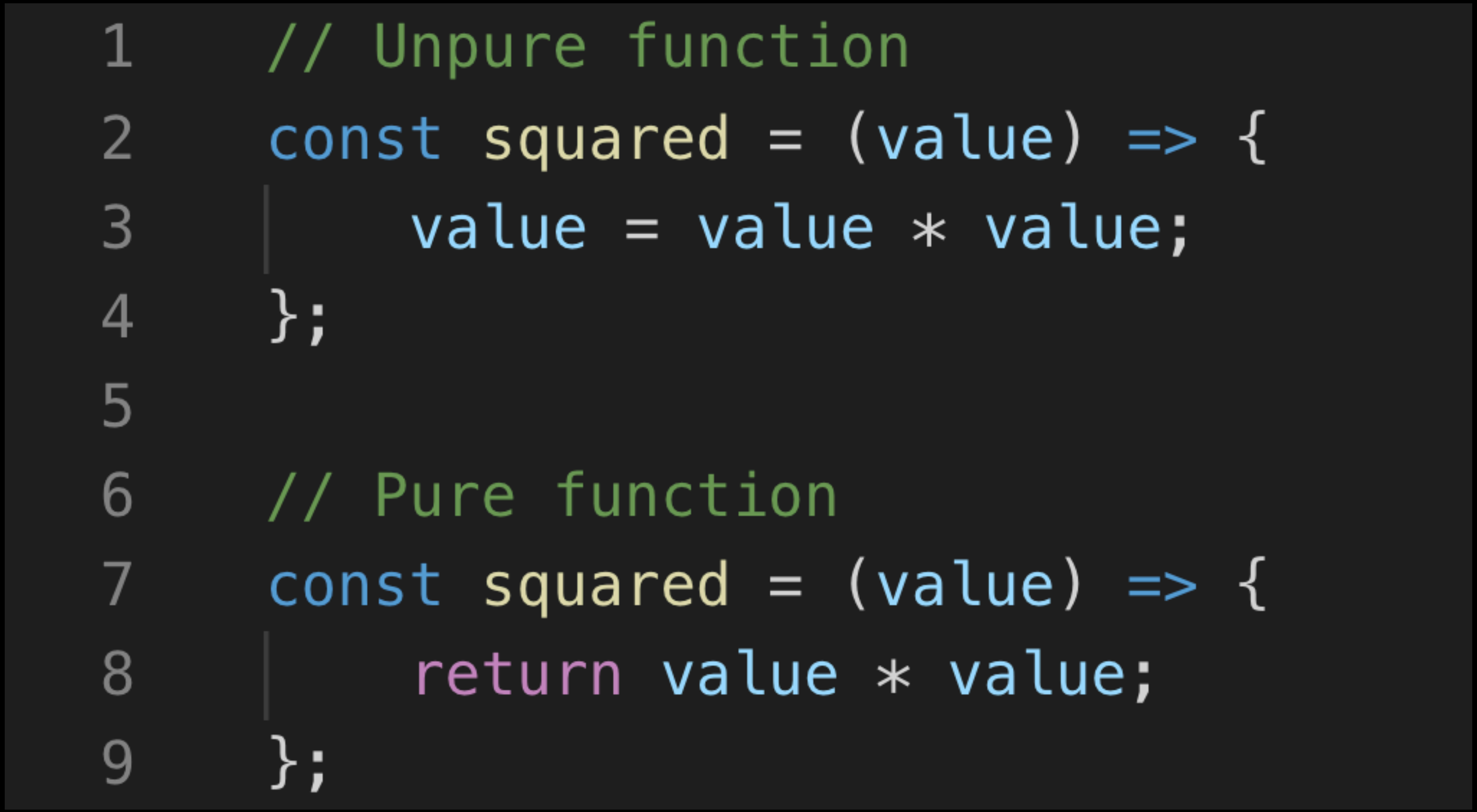
```
async function readFileAsRows(filename) {
    try {
        let content = fs.readFile(filename);
        return content.split('\n');
    } catch (err) {
        throw new InvalidFileError(err);
    }
}
```

Exercise: Suggest a pure implementation



```
// Unpure function
const squared = (value) => {
    value = value * value;
};
```

Example:



```
// Unpure function
const squared = (value) => {
    value = value * value;
};

// Pure function
const squared = (value) => {
    return value * value;
};
```

Higher order functions

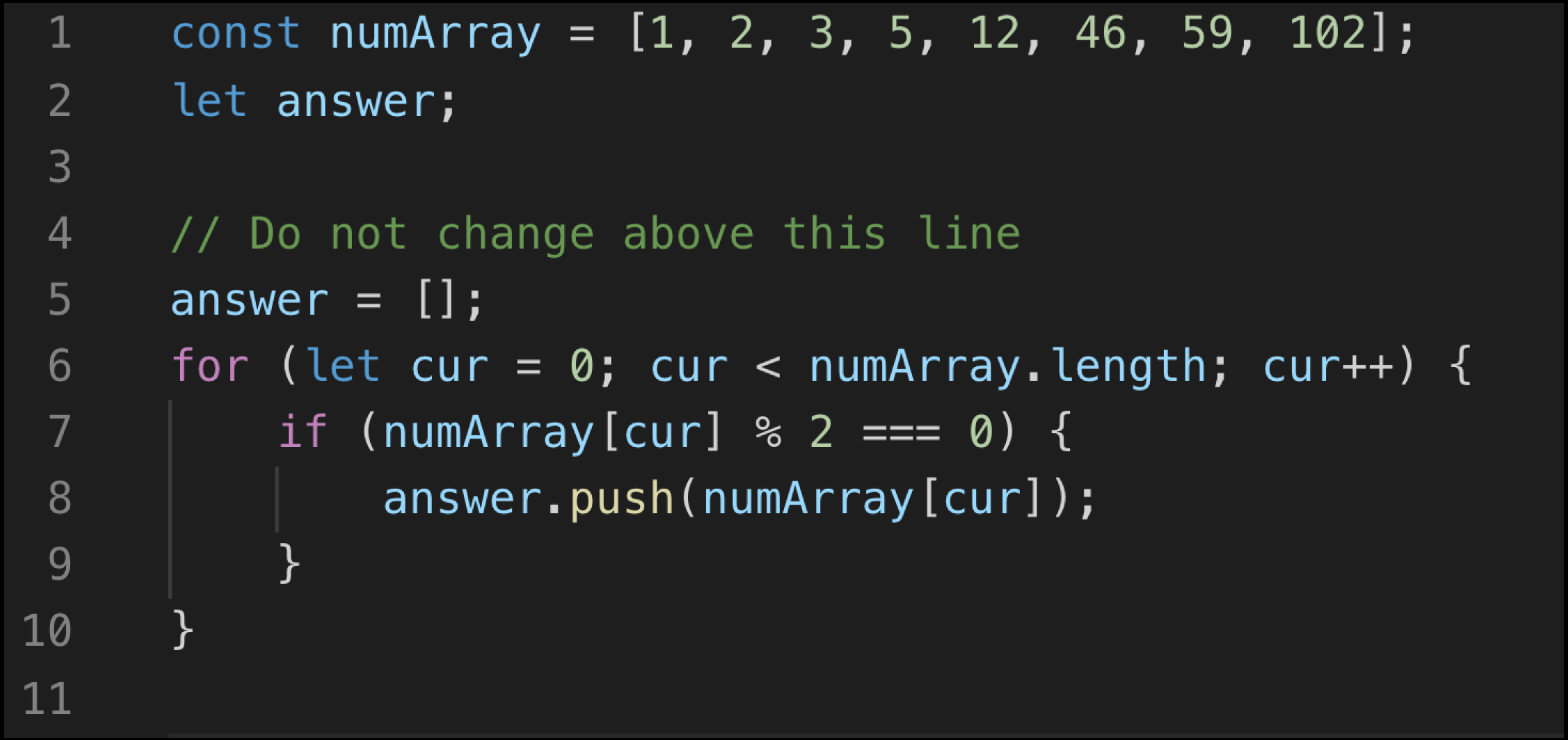
- A function that either:
  1. Returns a function, **or**
  2. Takes a function as an argument

- When handling Javascript collections (Arrays):
  - Array.prototype.filter()
  - Array.prototype.map()
  - Array.prototype.reduce()
  - Array.prototype.sort()
  - others…

Array.prototype.filter()

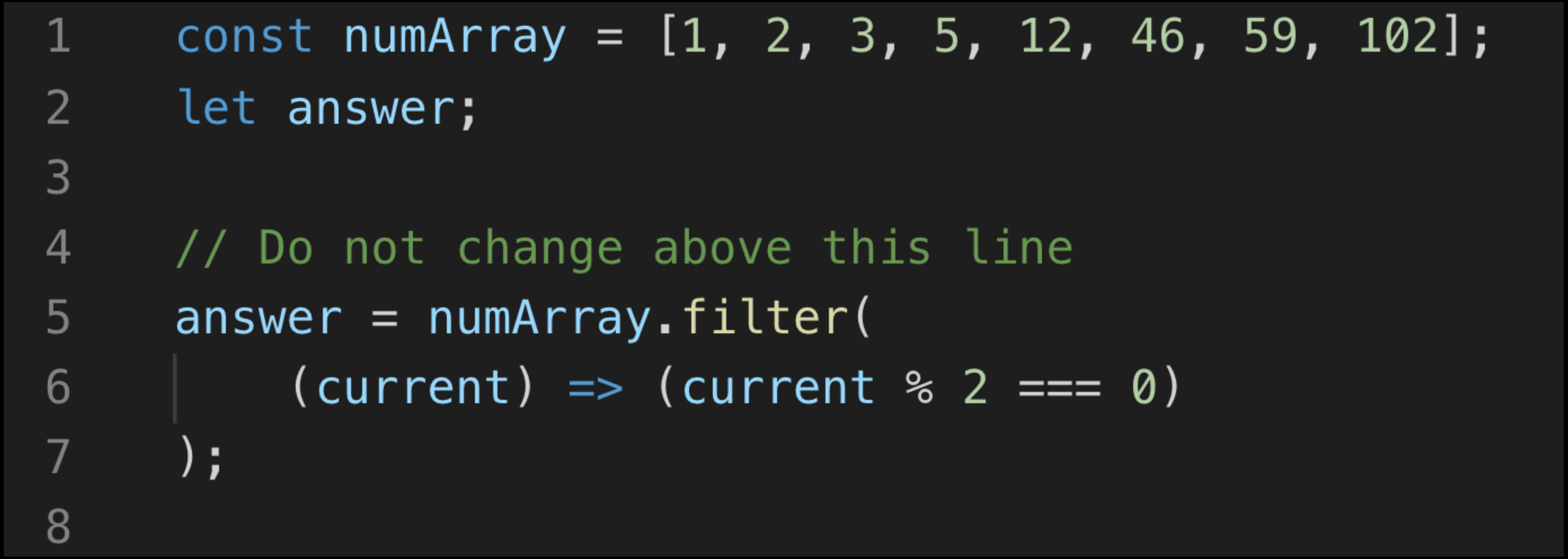
- Replaces **if/elseif/else** and **switch** statements
- Takes function as argument which must return **true** or **false**
- Returns a subset of the original array
- Minimizes size of array prior to computationally intensive tasks
- Not the same as Array.prototype.reduce()!

Exercise: Replace with filter()



```javascript
const numArray = [1, 2, 3, 5, 12, 46, 59, 102];
let answer;

// Do not change above this line
answer = [];
for (let cur = 0; cur < numArray.length; cur++) {
    if (numArray[cur] % 2 === 0) {
        answer.push(numArray[cur]);
    }
}
```

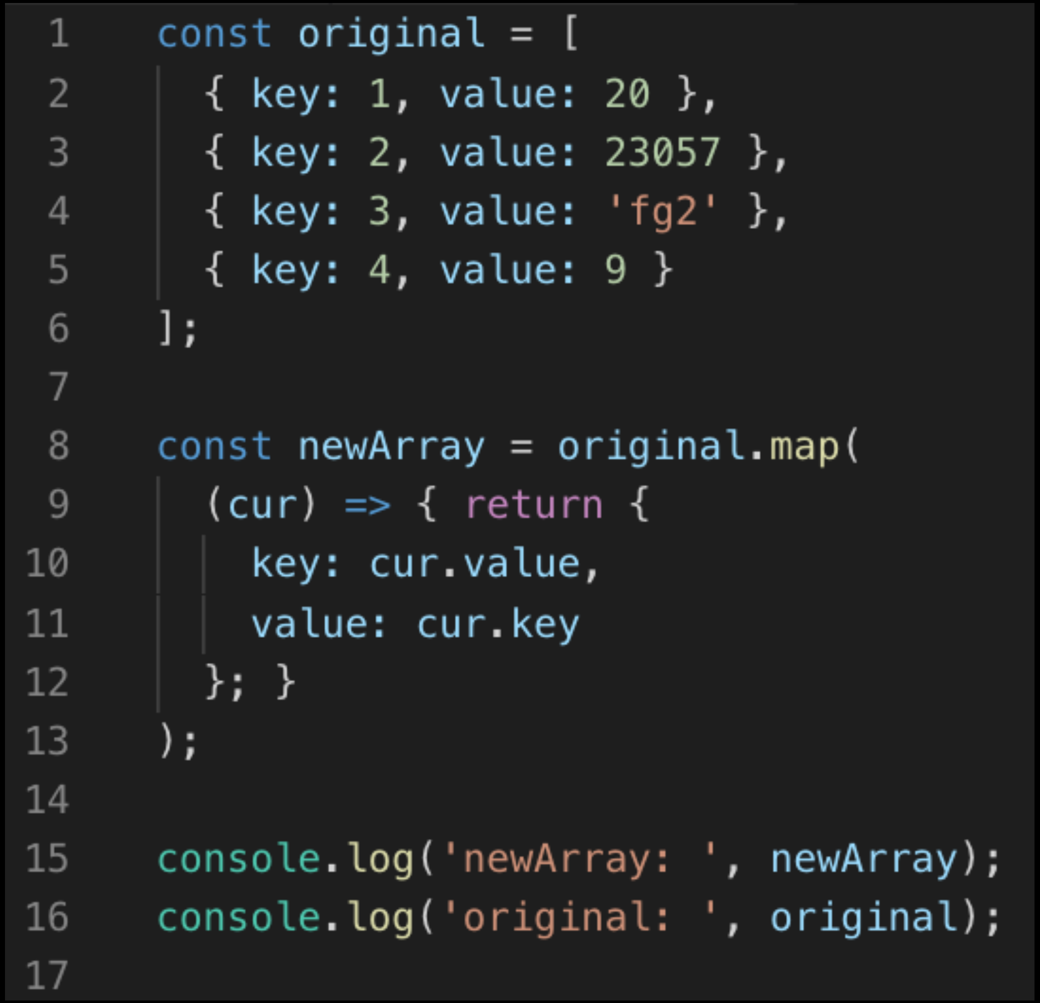
Exercise: Replace with filter()



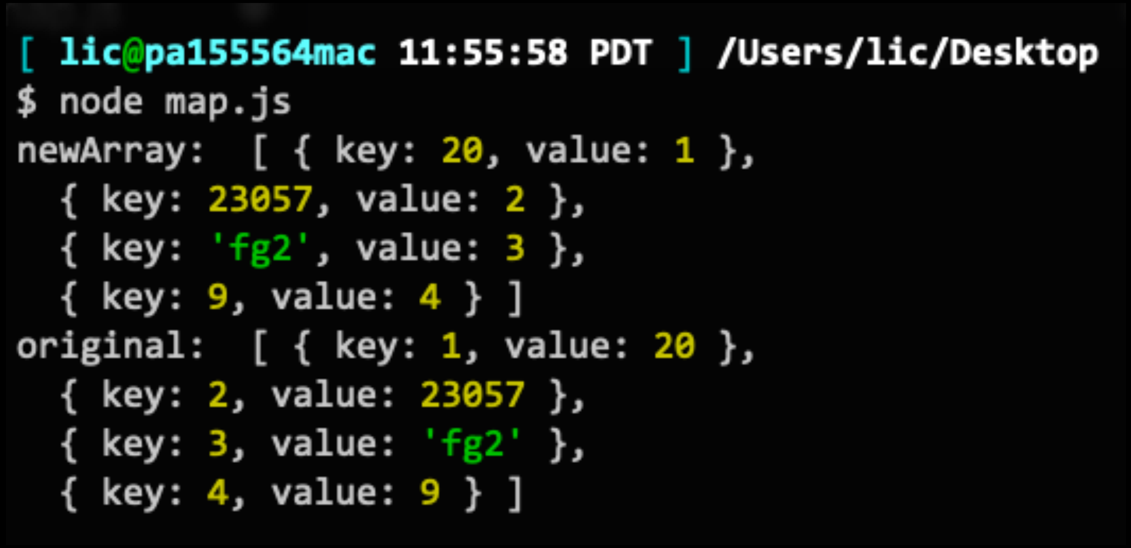
```javascript
const numArray = [1, 2, 3, 5, 12, 46, 59, 102];
let answer;

// Do not change above this line
answer = numArray.filter(
    (current) => (current % 2 === 0)
);

```

Array.prototype.map()

- Creates a **new array** from the original array with the results of a callback function run on each item in the original array.



```
const original = [
  { key: 1, value: 20 },
  { key: 2, value: 23057 },
  { key: 3, value: 'fg2' },
  { key: 4, value: 9 }
];

const newArray = original.map(
  (cur) => { return {
    key: cur.value,
    value: cur.key
  }; }
);

console.log('newArray: ', newArray);
console.log('original: ', original);
```

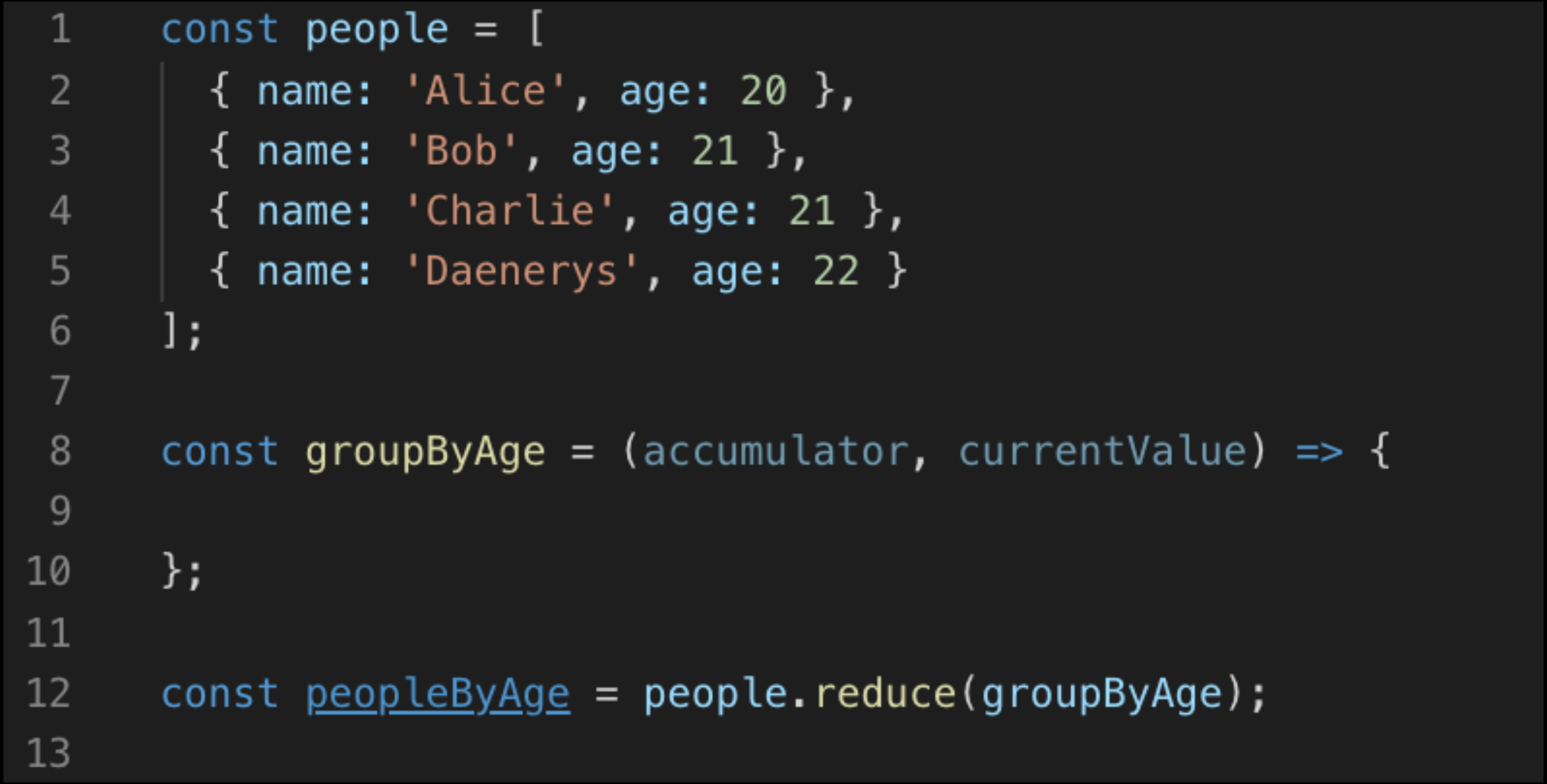


```
[ lic@pa155564mac 11:55:58 PDT ] /Users/lic/Desktop
$ node map.js
newArray:  [ { key: 20, value: 1 },
  { key: 23057, value: 2 },
  { key: 'fg2', value: 3 },
  { key: 9, value: 4 } ]
original:  [ { key: 1, value: 20 },
  { key: 2, value: 23057 },
  { key: 3, value: 'fg2' },
  { key: 4, value: 9 } ]
```

Array.prototype.reduce()

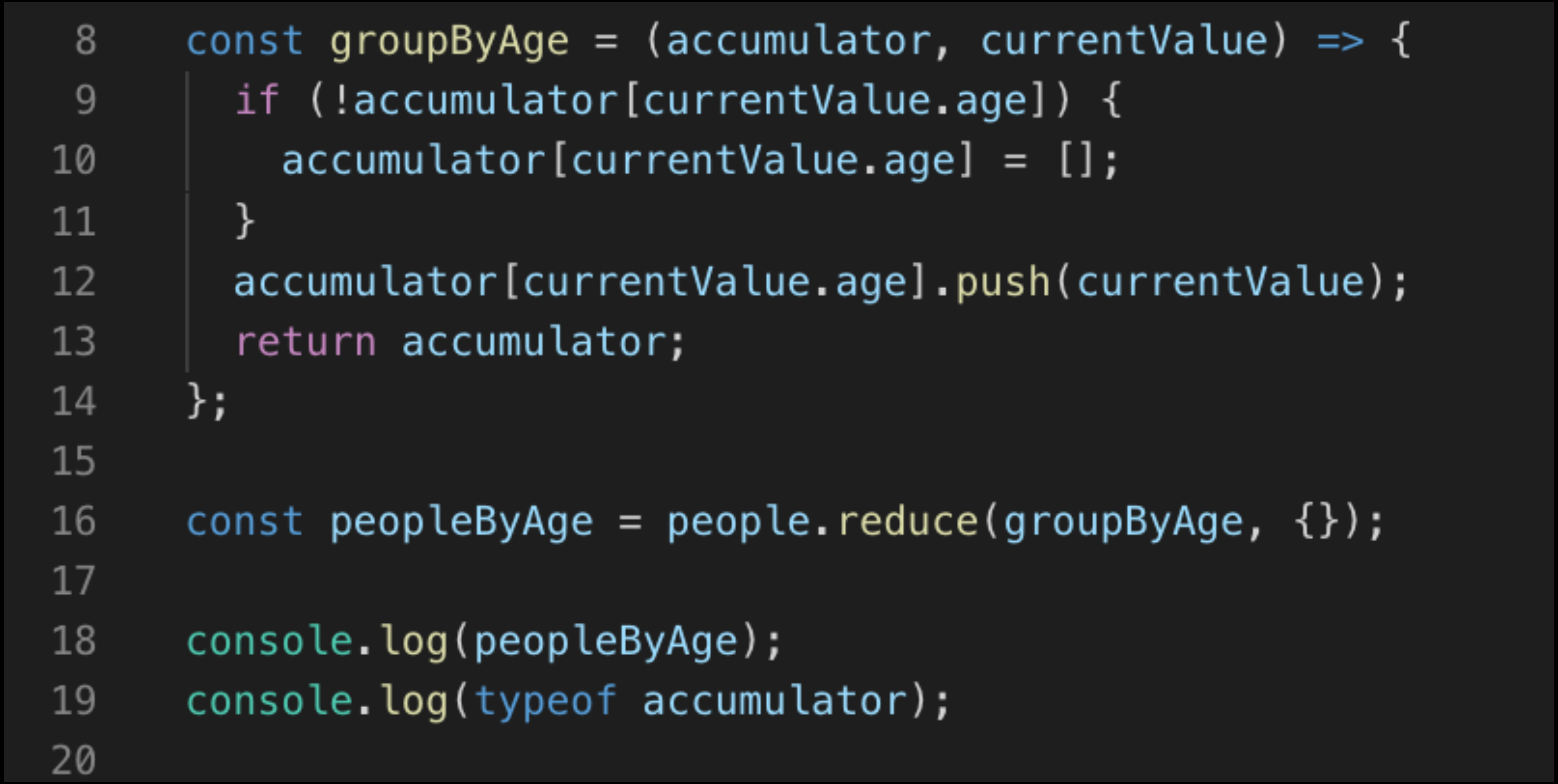
- Simplifies an array into a single value
- Replace a **forEach()** when re-using a tracking or “accumulator” variable
- Keeps accumulator from polluting scopes

Exercise: groupBy() using reduce()



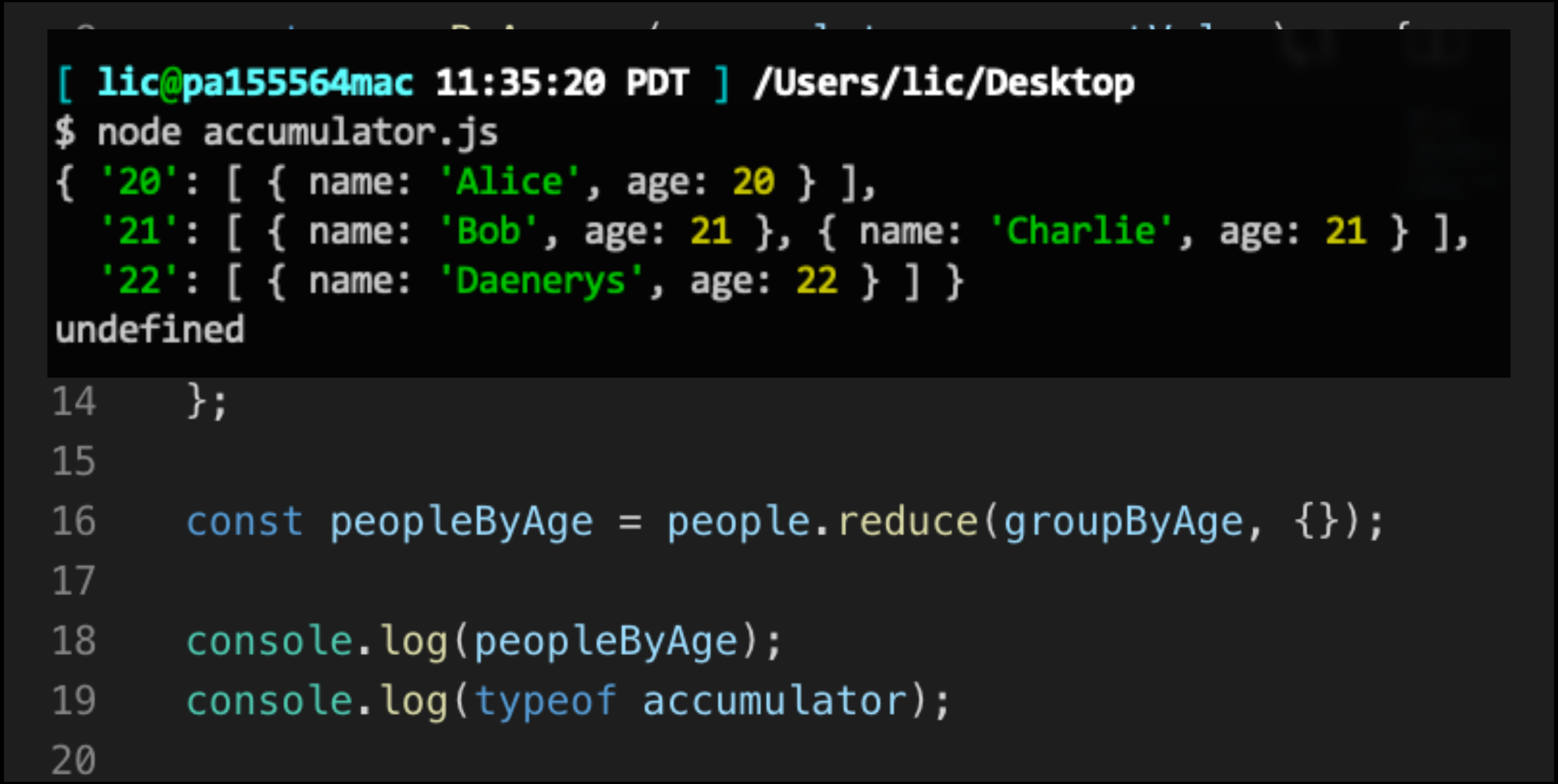
```javascript
const people = [
  { name: 'Alice', age: 20 },
  { name: 'Bob', age: 21 },
  { name: 'Charlie', age: 21 },
  { name: 'Daenerys', age: 22 }
];

const groupByAge = (accumulator, currentValue) => {

};

const peopleByAge = people.reduce(groupByAge);

```

Exercise: groupBy() using reduce()



```
const groupByAge = (accumulator, currentValue) => {
  if (!accumulator[currentValue.age]) {
    accumulator[currentValue.age] = [];
  }
  accumulator[currentValue.age].push(currentValue);
  return accumulator;
};

const peopleByAge = people.reduce(groupByAge, {});

console.log(peopleByAge);
console.log(typeof accumulator);

```

Exercise: groupBy() using reduce()

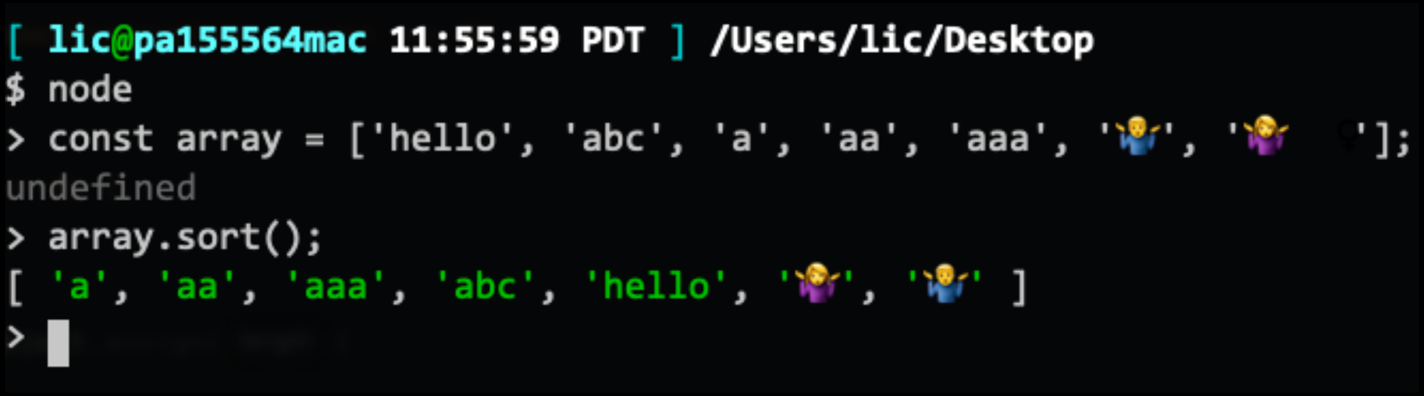


```
[ lic@pa155564mac 11:35:20 PDT ] /Users/lic/Desktop
$ node accumulator.js
{ '20': [ { name: 'Alice', age: 20 } ],
  '21': [ { name: 'Bob', age: 21 }, { name: 'Charlie', age: 21 } ],
  '22': [ { name: 'Daenerys', age: 22 } ] }
undefined
```

```
};

const peopleByAge = people.reduce(groupByAge, {});

console.log(peopleByAge);
console.log(typeof accumulator);

```

Array.prototype.sort()

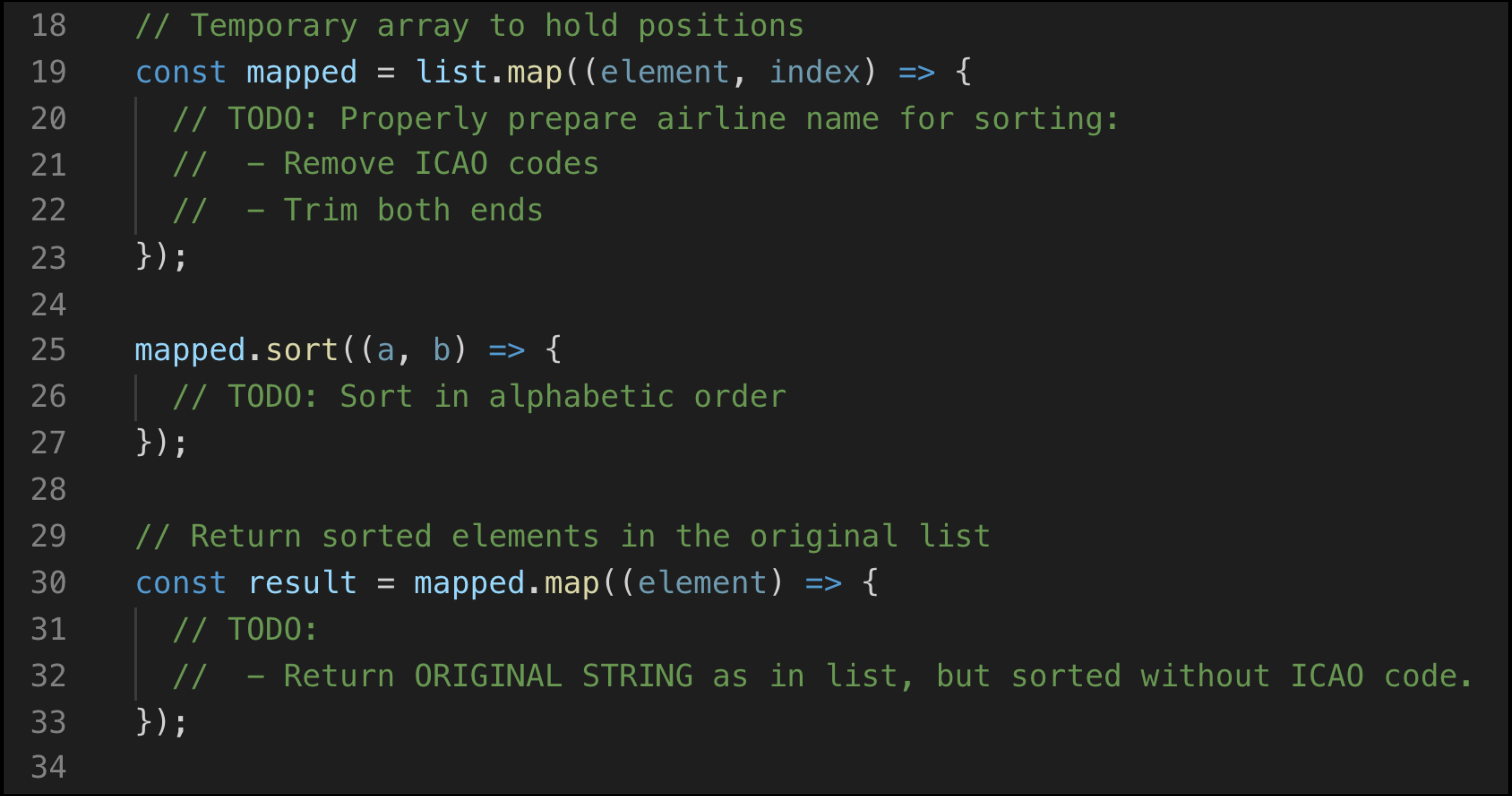
- Sorts an array **in-place** based on a defined algorithm
- Default is to sort based on string type conversion, then in ascending UTF-16 byte order values



```
[ lic@pa155564mac 11:55:59 PDT ] /Users/lic/Desktop
$ node
> const array = ['hello', 'abc', 'a', 'aa', 'aaa', '🤷‍♂️', '🤷‍♀️'];
undefined
> array.sort();
[ 'a', 'aa', 'aaa', 'abc', 'hello', '🤷‍♀️', '🤷‍♂️' ]
> 
```

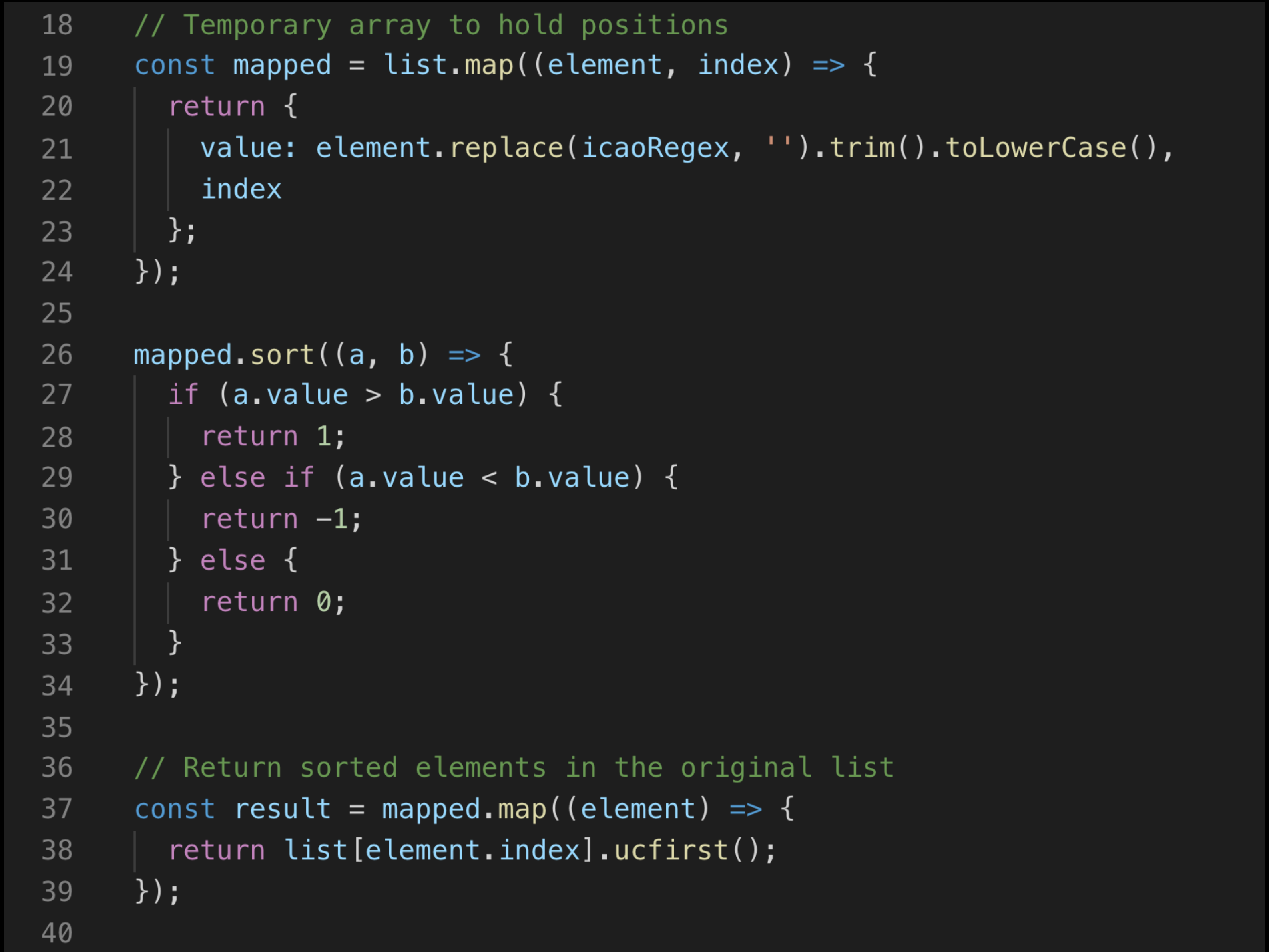
- Returns integers when comparing elementA and elementB:
  - -1 or less: elementB before elementA
  - 0: no change
  - 1 or more: elementA before elementB

Exercise: sort() with a daisy-chained map()



```javascript
// Temporary array to hold positions
const mapped = list.map((element, index) => {
  // TODO: Properly prepare airline name for sorting:
  //  - Remove ICAO codes
  //  - Trim both ends
});

mapped.sort((a, b) => {
  // TODO: Sort in alphabetic order
});

// Return sorted elements in the original list
const result = mapped.map((element) => {
  // TODO:
  //  - Return ORIGINAL STRING as in list, but sorted without ICAO code.
});

```

Exercise: sort() with a daisy-chained map()



```javascript
// Temporary array to hold positions
const mapped = list.map((element, index) => {
  return {
    value: element.replace(icaoRegex, '').trim().toLowerCase(),
    index
  };
});

mapped.sort((a, b) => {
  if (a.value > b.value) {
    return 1;
  } else if (a.value < b.value) {
    return -1;
  } else {
    return 0;
  }
});

// Return sorted elements in the original list
const result = mapped.map((element) => {
  return list[element.index].ucfirst();
});

```

Project Time!

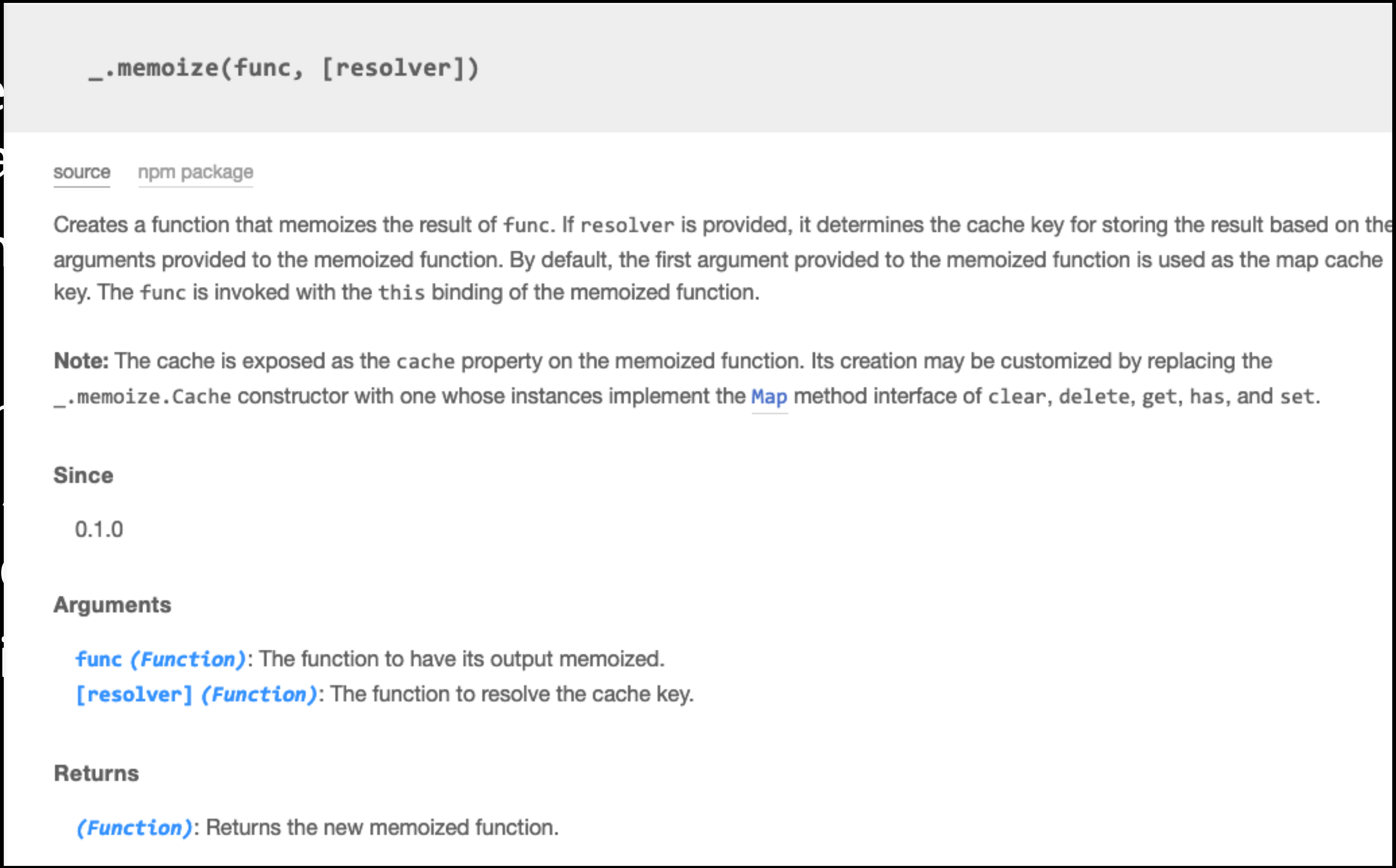
1. How much has the Marvel Cinematic Universe grossed in the US?
2. What is the average gross per film for each studio? Sort it by descending average gross.
3. Which movie was the first Disney-produced (Buena Vista) film?
4. Has every Avengers movie grossed more than the last?
5. Up to today, what is the average gross **per day** of each MCU film in descending order?
6. Which actor(s) have appeared in more than one film? Order them in descending order of number of appearances.

Bonus: Memoization

- Practice of making expensive, recursive/iterative functions run much faster (aka “cache”)
- Is a higher order function: memoize(function)
- Keeps a record of function call with a set of arguments
  - If same function + arguments are called again, just returns that
  - If not, calculates the result and stores in memory
- The function being memorized **MUST** be pure!
- Try writing your own implementation!

Bonus: Memoization

- Practice … much faster (a
- Is a high
- Keeps a
  - If sam
  - If not
- The fun
- Try writi



```
_.memoize(func, [resolver])
```

source npm package

Creates a function that memoizes the result of `func`. If `resolver` is provided, it determines the cache key for storing the result based on the arguments provided to the memoized function. By default, the first argument provided to the memoized function is used as the map cache key. The `func` is invoked with the `this` binding of the memoized function.

Note: The cache is exposed as the `cache` property on the memoized function. Its creation may be customized by replacing the `_.memoize.Cache` constructor with one whose instances implement the `Map` method interface of `clear`, `delete`, `get`, `has`, and `set`.

Since

0.1.0

Arguments

`func` *(Function)*: The function to have its output memoized.
`[resolver]` *(Function)*: The function to resolve the cache key.

Returns

(Function): Returns the new memoized function.

Further Exercises

- Challenge: Create pure higher-order implementation of .filter(), .map() and .reduce()
  - Note: You can overload Array.prototype.<function> with your own implementation, test to see if result is the same as the standard library
- This Gist: https://gist.github.com/oskarkv/3168ea3f8d7530ccd94c97c19aafe266
- https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d
- https://codeburst.io/understanding-memoization-in-3-minutes-2e58daf33a19