tt()

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Import / Export
    2. Scope and return values
    3. Functional patterns
    4. .map().filter().reduce()
3. All together!

1

Schedule

1. **Review assignments**
2. JavaScript deep dive
    1. Import / Export
    2. Scope and return values
    3. Functional patterns
    4. .map().filter().reduce()
3. All together!

Review

```
  residen
  work: {
    title: "Project Manag
    employer: "Clarify"

  }
}

* Filter by age, normalize capitals in names, convert ages to numbers, remov
  const data = [
    {
      name: "Robert",
      age: 29,
      residence: "Amsterdam",

    },
    {
      name: "Berend",
      age: 32,
      residence: "Rotterdam",
```

Show & tell

Review

Don’t worry, today we’ll dive deeper
in all the stuff we just discussed..

Schedule

1. Review assignments
2. **JavaScript deep dive**
    1. Import / Export
    2. Scope and return values
    3. Functional patterns
    4. .map().filter().reduce()
3. All together!

Schedule

1. Review assignments
2. JavaScript deep dive
    1. **Import/ Export**
    2. Scope and return values
    3. Functional patterns
    4. .map().filter().reduce()
3. All together!

Why?

- Using modules allow us to work in components
- Working in components allows us to:
  - Re-use snippets of code (DRY)
  - Write cleaner code (KISS)
  - Debug with more ease instead of 99999 lines of file xyz
- Prepares us to work with external modules
  (libraries, on Monday)

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Import/ Export
    2. **Scope and return values**
    3. Functional patterns
    4. .map().filter().reduce()
3. All together!

Scope

- The **scope** of a variable refers to where the variable is accessible
- const and let are **block scoped**:
  - The variable is accessible inside the block where it is defined

```
function groet(naam) {

   let wens = “Goedemorgen, ”;

   console.log(wens + naam);

}

wens=?  (A) wens == “Goedemorgen, “  (B) wens bestaat niet meer
```

Returns

The return statement ends function execution and specifies a value to be returned to the function caller.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/return

Return values

- Function calls can produce results, as in:

```
let toevalsgetal = Math.random();

function groet(naam) {
   let wens = “Goedemorgen, ”+ naam;
   return wens
}

console.log(groet(“Laura”);
```

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Import/ Export
    2. Scope and return values
    3. **Functional patterns**
    4. .map().filter().reduce()
3. All together!

Functional programming

Functional programming distinguishes between pure and impure functions.

It encourages you to write pure functions.

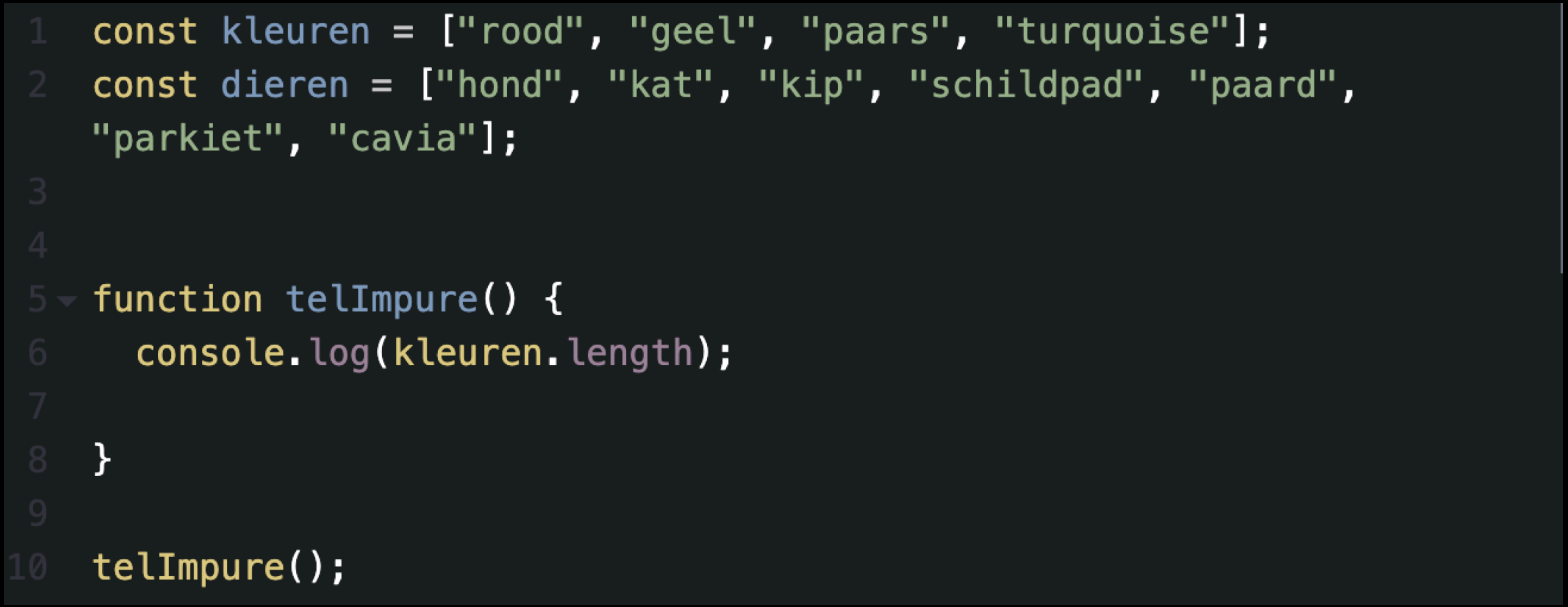
A pure function must satisfy both of the following properties:

Functional programming

Referential transparency: The function always gives the same return value for the same arguments. This means that the function cannot depend on any mutable state

Side-effect free: The function cannot cause any side effects.
Side effects may include I/O (e.g., writing to the console or a log file), modifying a mutable object, reassigning a variable, etc.

Impure functions



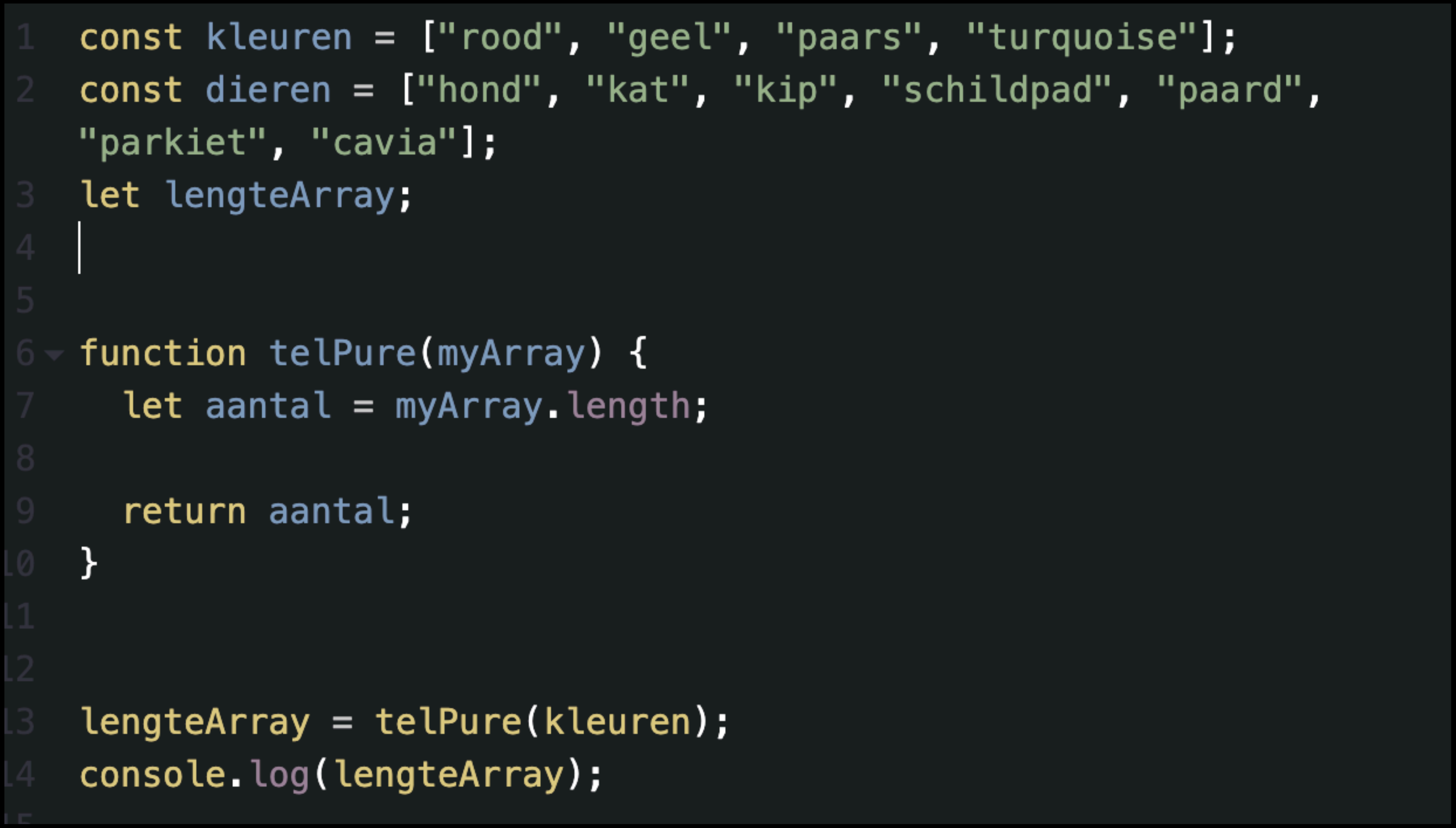
```javascript
const kleuren = ["rood", "geel", "paars", "turquoise"];
const dieren = ["hond", "kat", "kip", "schildpad", "paard", "parkiet", "cavia"];


function telImpure() {
  console.log(kleuren.length);

}

telImpure();
```

Pure functions



```javascript
const kleuren = ["rood", "geel", "paars", "turquoise"];
const dieren = ["hond", "kat", "kip", "schildpad", "paard", "parkiet", "cavia"];
let lengteArray;


function telPure(myArray) {
  let aantal = myArray.length;

  return aantal;
}


lengteArray = telPure(kleuren);
console.log(lengteArray);
```

Pure functions

- Have limited responsibility
- Their behavior is predictible
- **Can be reused** in different contexts

Functional programming

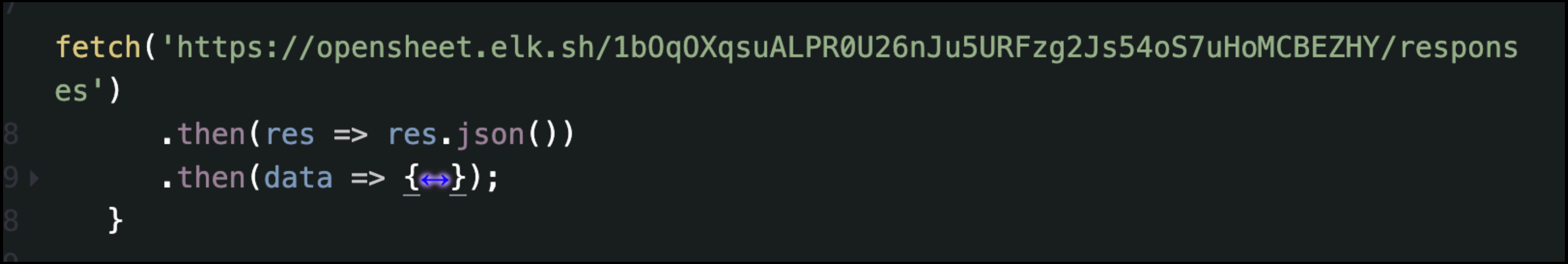
- Supports reuse through abstraction
- `console.log();` Only wites text to the console

  `newOutputFunction(console.log(), “Hello, world”)`

  `newOutputFunction(document.write(), “Hello, world”)`

  `newOutputFunction()` Also writes to the Website, maybe to paper?
- Why?
- We offer you a way of **structuring your code** which we think is clean, concise, self-explanatory and **reusable**.

Functional pattern: chaining



```
fetch('https://opensheet.elk.sh/1bOqOXqsuALPR0U26nJu5URFzg2Js54oS7uHoMCBEZHY/responses')
      .then(res => res.json())
      .then(data => {↔});
   }
```

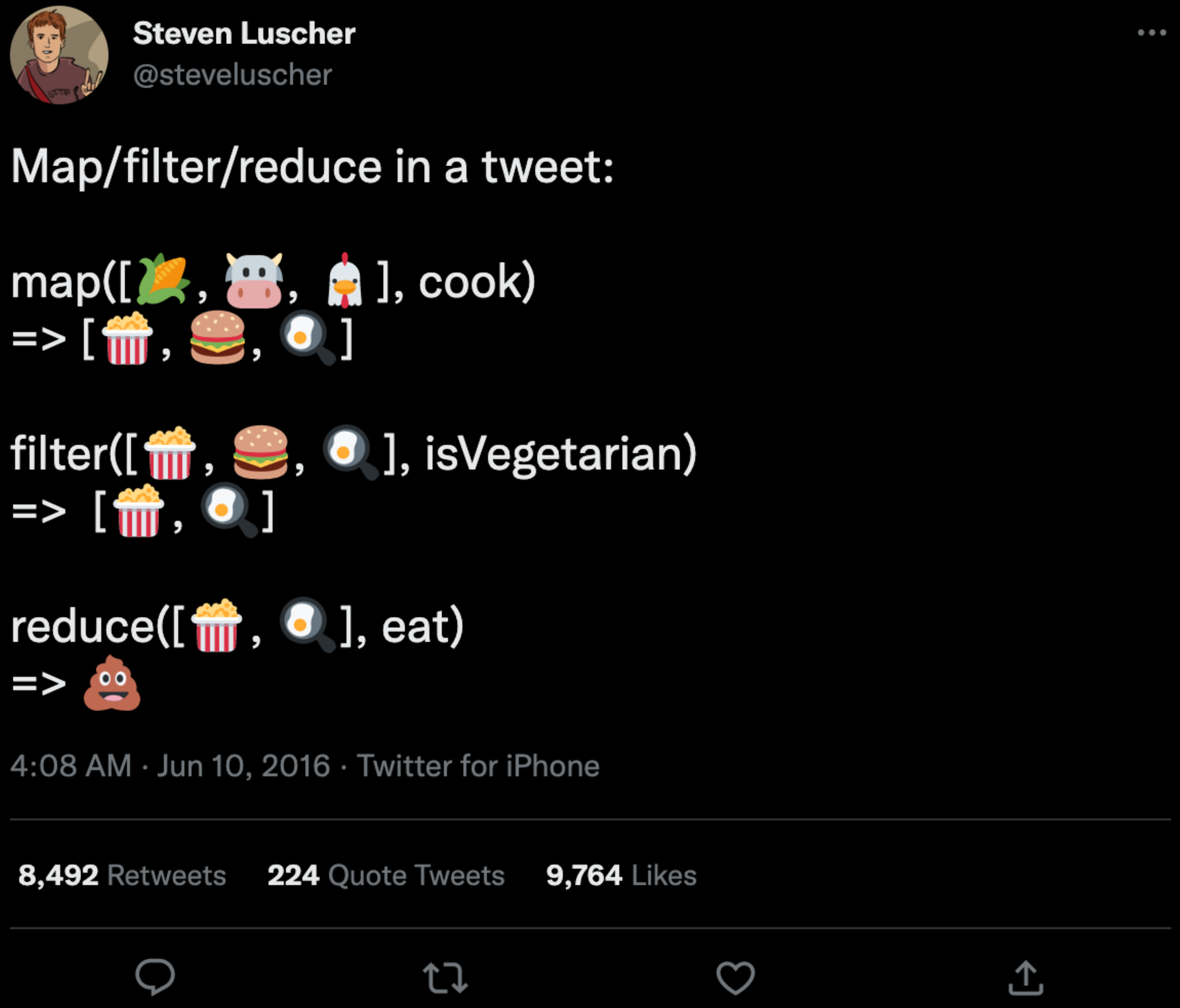
Schedule

1. Review assignments
2. JavaScript deep dive
   1. Import/ Export
   2. Scope and return values
   3. Functional patterns
   4. **.map().filter().reduce()**
3. All together!

.map().filter.reduce()

Map, filter & reduce are **array methods.** We can call them on any array to loop over them and perform actions on their items

.map().filter.reduce()



Steven Luscher
@steveluscher

Map/filter/reduce in a tweet:

map([🌽, 🐮, 🐔], cook)
=> [🍿, 🍔, 🍳]

filter([🍿, 🍔, 🍳], isVegetarian)
=> [🍿, 🍳]

reduce([🍿, 🍳], eat)
=> 💩

4:08 AM · Jun 10, 2016 · Twitter for iPhone

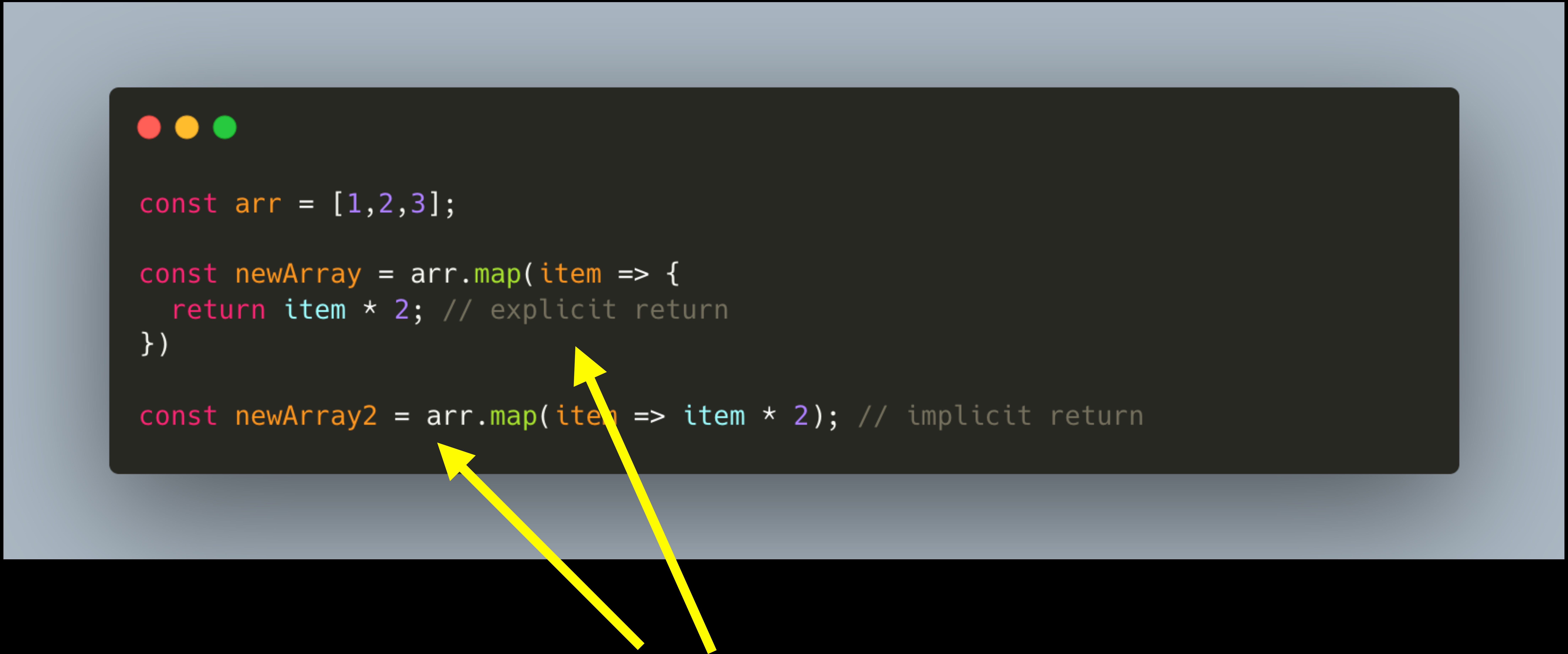
8,492 Retweets **224** Quote Tweets **9,764** Likes

Returns

Map, filter (and reduce) **return** something: an **Array or object.**

This is what differentiates them from **forEach()**

Implicit vs explicit



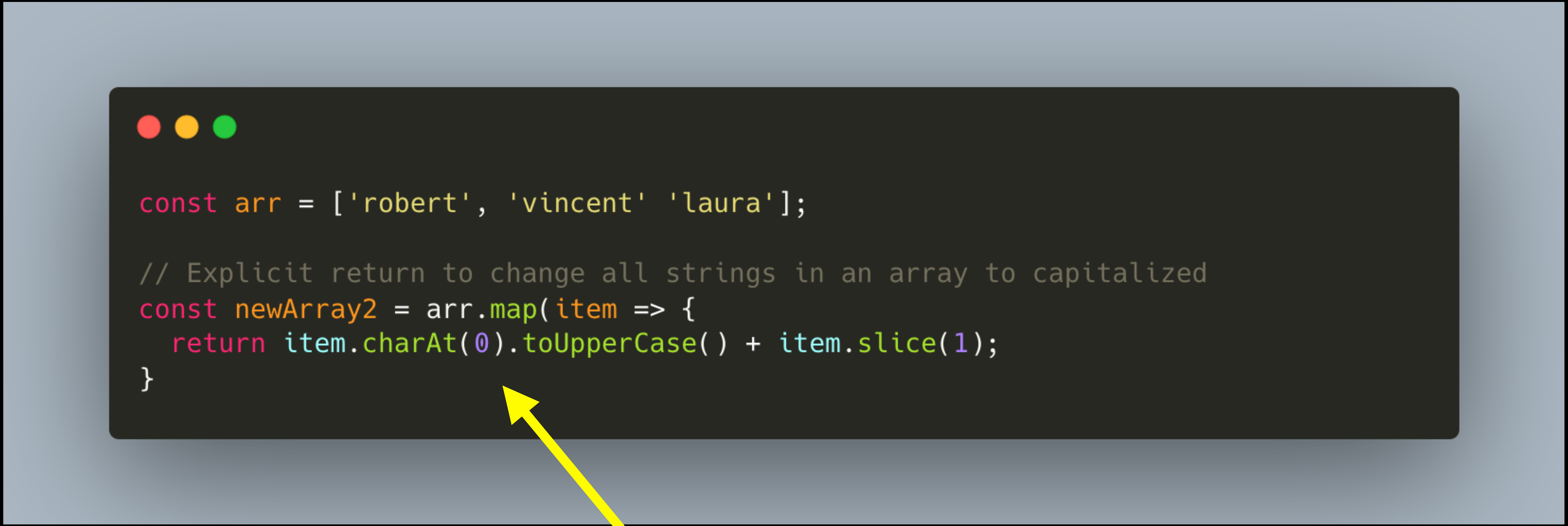
```javascript
const arr = [1,2,3];

const newArray = arr.map(item => {
  return item * 2; // explicit return
})

const newArray2 = arr.map(item => item * 2); // implicit return
```

This code does the same

Implicit vs explicit



```
const arr = ['robert', 'vincent' 'laura'];

// Explicit return to change all strings in an array to capitalized
const newArray2 = arr.map(item => {
  return item.charAt(0).toUpperCase() + item.slice(1);
}
```

Remember the homework assignment?

.map().filter().reduce()

Livecode!

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Import/ Export
    2. Scope and return values
    3. Functional patterns
    4. .map().filter().reduce()
3. **All together!**

All together!

Live example from the editor combining everything…

Uncaught SyntaxError
Unexpected end of input