```
tt()
```

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Imports & Exports
    2. Functional patterns
    3. .map().filter().reduce()
3. All together!

Schedule

1. **Review assignments**
2. JavaScript deep dive
    1. Imports & Exports
    2. Functional patterns
    3. .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. Imports & Exports
    2. Functional patterns
    3. .map().filter().reduce()
3. All together!

Deep dive

```
async function request(url) {
    let res = await fetch(url);
    return await res.json();
}

export default request
```

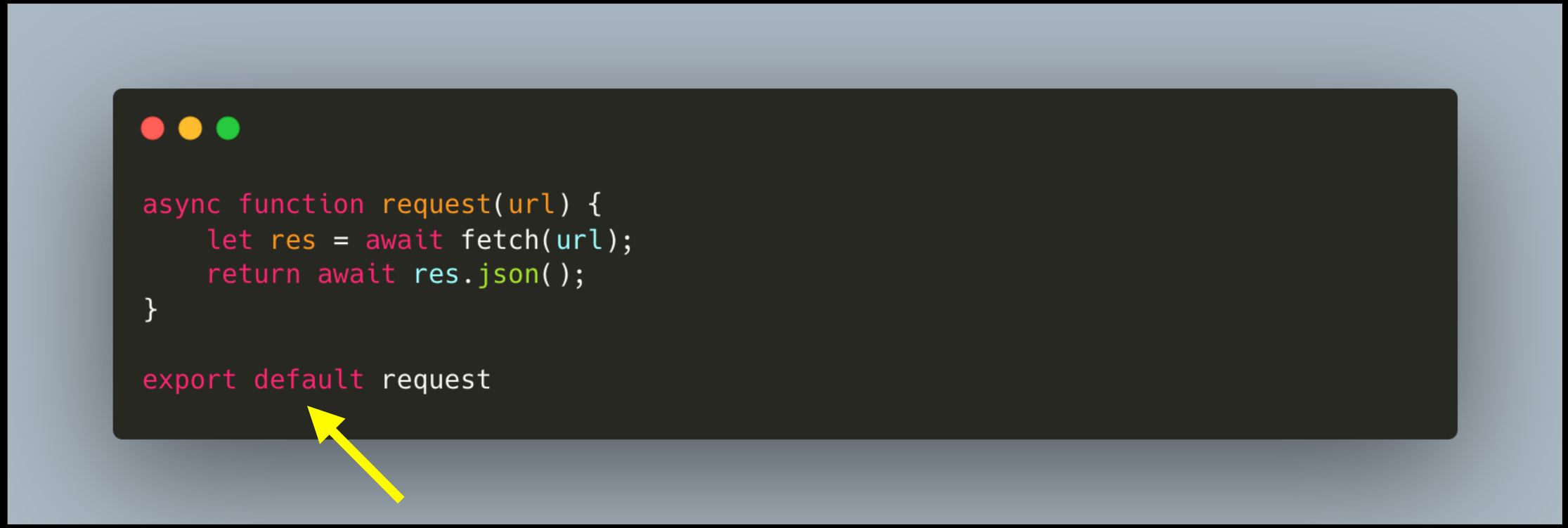
Deep dive

- You know how to work in modules / components
- You know something about shorthands, implicit and explicit returns, as well as promises
- You’re able to write functional programming patterns for your application
- You are able to chain multiple functions / promises together to parse a raw dataset and return something ready for production

Schedule

1. Review assignments
2. JavaScript deep dive
    1. **Imports & Exports**
    2. Functional patterns
    3. .map().filter().reduce()
3. All together!

Deep dive



export default whaaaat?

Deep dive

```
import CONFIG from './config.js';
import request from './request.js';
import makeHtml from './make.js';

const data = await request(CONFIG.url);
```

Import default whaaaat?

Deep dive

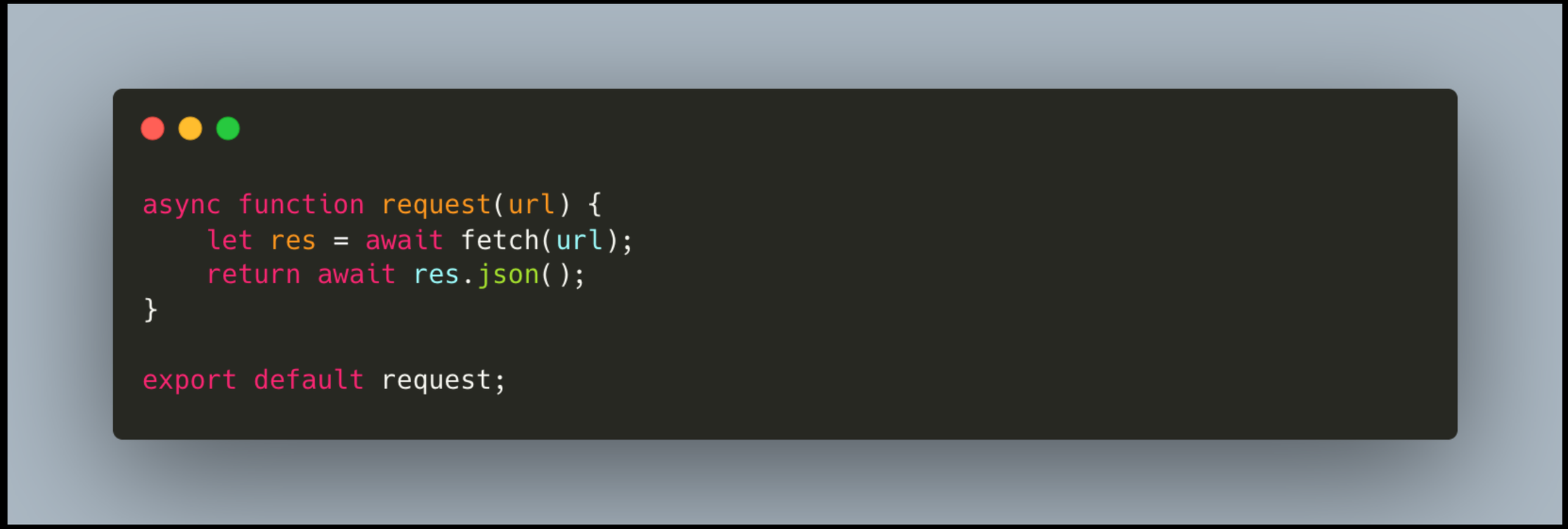
- You can export a function or variable from any file
- There are two types of exports, named and default

Named exports

```
// Individually

export const name = "Robert";
export const age = 29;

// All at once as an object

const name = "Robert";
const age = 29;

export { name, age }
```

Exporting as an object, due to the {}

Default exports

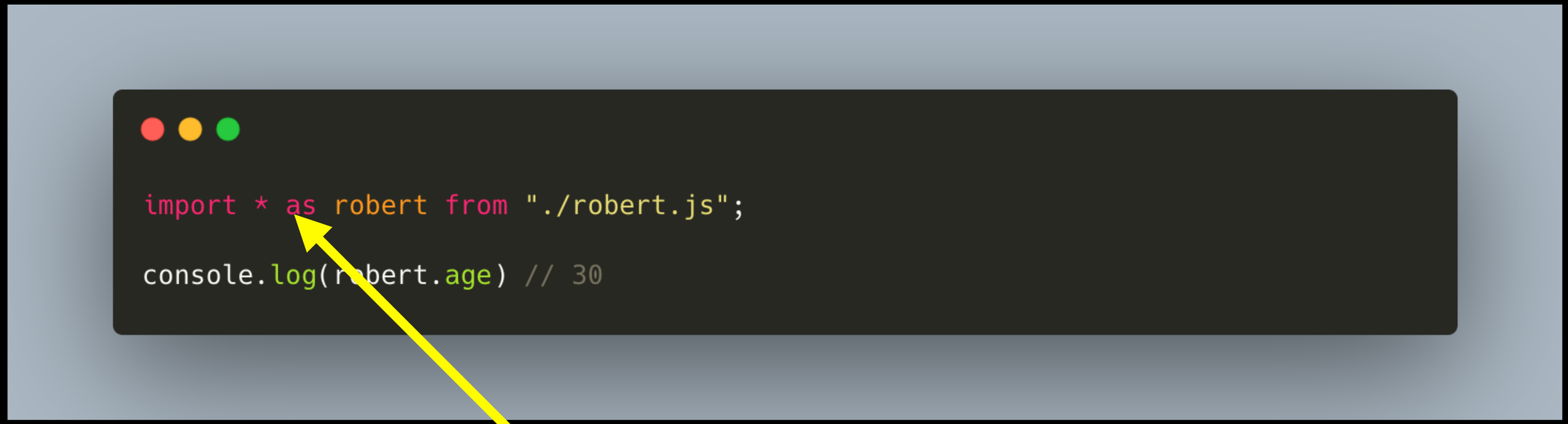


```
async function request(url) {
    let res = await fetch(url);
    return await res.json();
}

export default request;
```

Named imports

Importing as an object, due to the {}

Named imports



Why?

- Using modules allow us to work in components
- Working in components allows us to:
  - Re-use snippets of code (DRY)
  - Write cleaner code
  - 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. Imports & Exports
    2. **Functional patterns**
    3. .map().filter().reduce()
3. All together!

Functional patterns

```
function makeHtml(obj, src) {
    createHeaders(Object.keys(obj), src);
    createContent(Object.entries(obj), src);
}

function createHeaders(th, src) {
    let rowEl = document.createElement('tr');
    let htmlRow = src.appendChild(rowEl);

    th.forEach(heading => {
        let headingEl = document.createElement('th');
        headingEl.innerHTML = heading;
        htmlRow.appendChild(headingEl);
    })
}

function createContent(tr, src) {
    let rowEl = document.createElement('tr');
    let htmlRow = src.appendChild(rowEl);

    tr.forEach(entry => {
        let headingEl = document.createElement('td');
        headingEl.innerHTML = entry[1];
        htmlRow.appendChild(headingEl);
    })

}

export default makeHtml;
```

Functional patterns

Just because your program contains functions does not necessarily mean that you are doing 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 patterns

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.

Functional patterns

```
let heightRequirement = 46;

// Impure because it relies on a mutable (reassignable) variable.
function canRide(height) {
  return height >= heightRequirement;
}

// Impure because it causes a side-effect by logging to the console.
function multiply(a, b) {
  console.log('Arguments: ', a, b);
  return a * b;
}
```

Functional patterns

```
let heightRequirement = 46;

function canRide(height) {
  return height >= heightRequirement;
}

// Every half second, set heightRequirement to a random number between 0 and
200.
setInterval(() => heightRequirement = Math.floor(Math.random() * 201), 500);

const mySonsHeight = 47;

// Every half second, check if my son can ride.
// Sometimes it will be true and sometimes it will be false.
setInterval(() => console.log(canRide(mySonsHeight)), 500);
```

Functional patterns

- We don’t need to use pure functions, you don’t need to use the functional programming paradigm.
- We offer you a way of **structuring your code** which we think is clean, concise and self-explanatory.

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Imports & Exports
    2. Functional patterns
    3. **.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

Before we continue, we should explore the **return** in JavaScript to understand what is happening

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

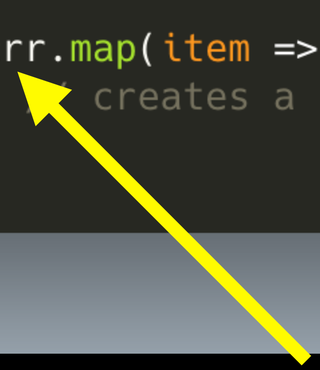
Returns

Map, filter and reduce are **array methods**, so they are able to **return** something. They create a **array or object.**

This is what differentiates them from a .forEach()

Returns

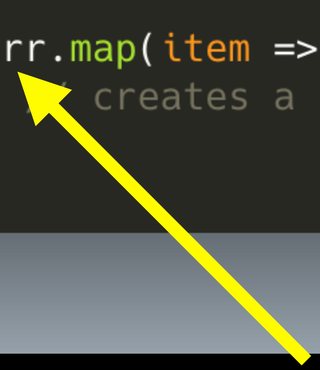
```
const arr = [1,2,3];

arr.forEach(item => {
  console.log(item) // forEach _does_ something with each item
})

const newArray = arr.map(item => {
  return item * 2; // creates a new array, we can save this
})
```



This map _also_ returns something,
we have two returns here,

Returns

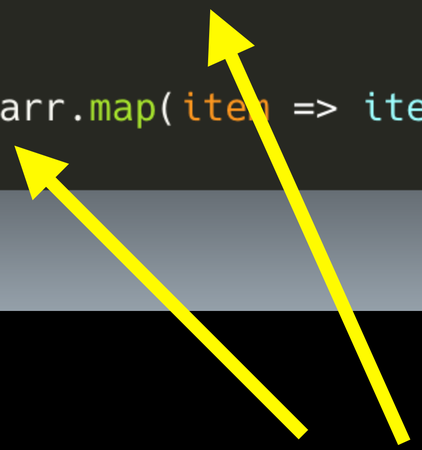
```
const arr = [1,2,3];

arr.forEach(item => {
  console.log(item) // forEach _does_ something with each item
})

const newArray = arr.map(item => {
  return item * 2; // creates a new array, we can save this
})
```



This map _also_ returns something,
we have two returns here,

Implicit vs explicit

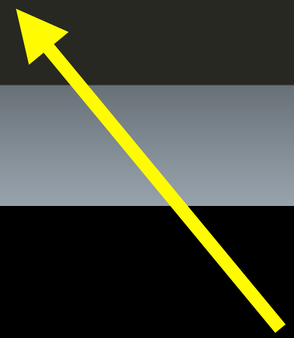
```
const arr = [1,2,3];

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?

Chaining

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

const newArray = arr.map(item => {
  return item * 2; // creates a new array, we can save this
}).map(item => {
  item = item - 1; // Now remove 1 from the new values
})
```

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

Livecode!

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Imports & Exports
    2. Functional patterns
    3. .map().filter().reduce()
3. **All together!**

All together!

Live example from the editor combining everything…

Uncaught SyntaxError
Unexpected end of input