```
tt()
```

Schedule

1. Review assignments: SVG
2. JavaScript deep dive
    1. Scope and return values
    2. Functional patterns
    3. .map().filter().reduce()
3. Logout

Schedule

1. **Review assignments**
2. JavaScript deep dive
    1. Scope and return values
    2. Functional patterns
    3. .map().filter().reduce()
3. Logout

Review

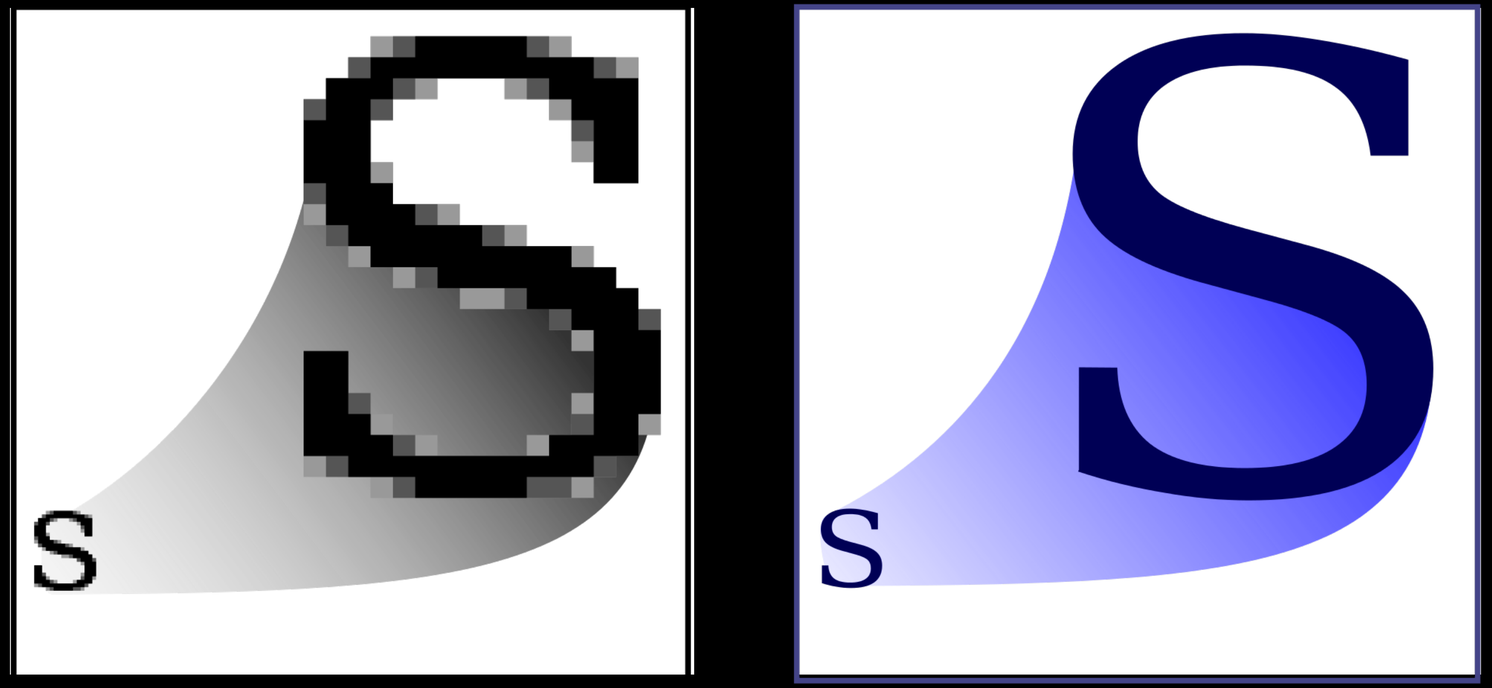
```
residen
work: {
    title: "Project Manag
    employer: "Clarify"
  }
}


* 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

SVG

Scalable **V**ector **G**raphics

SVG



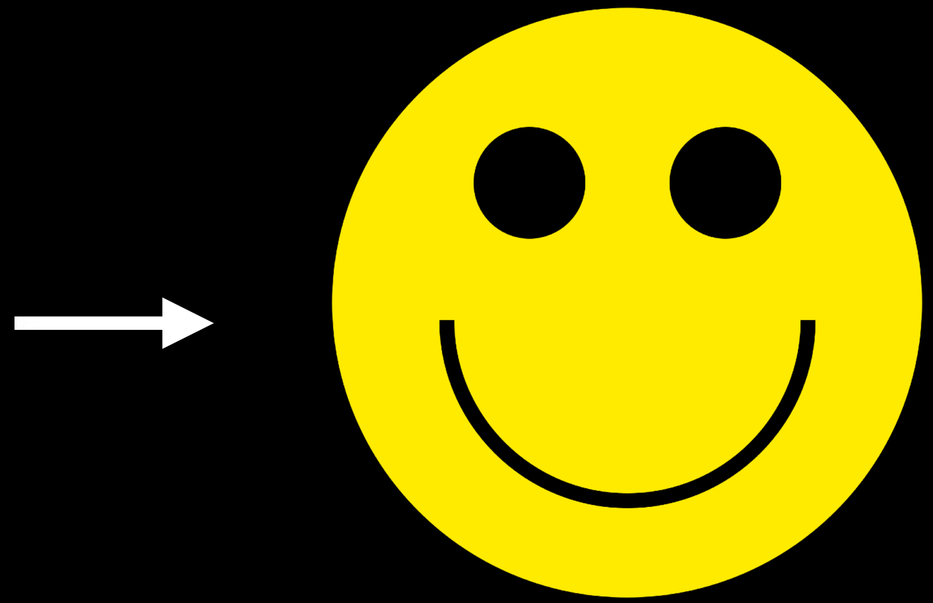
SVG

Smile.svg

Smile.svg

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <svg width="1080px" height="1080px" viewBox="0 0 1080 1080
   " version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:
   xlink="http://www.w3.org/1999/xlink">
3      <title>Smile</title>
4      <g id="Smile" stroke="none" stroke-width="1" fill="
       none" fill-rule="evenodd">
5          <circle id="Oval" stroke="#000000" stroke-width="
           20" fill="#FFEB00" cx="540" cy="540" r="406"></
           circle>
6          <circle id="Oval" fill="#000000" cx="409" cy="379"
            r="75"></circle>
7          <circle id="Oval-Copy" fill="#000000" cx="672" cy=
           "379" r="75"></circle>
8          <path d="M298,563.5 C298,697.429052
           406.570948,806 540.5,806 C674.429052,806
           783,697.429052 783,563.5" id="Path" stroke="
           #000000" stroke-width="20"></path>
9      </g>
10 </svg>
11
```

Line 11, Column 1 Spaces: 4 XML



Schedule

1. Review assignments
2. **JavaScript deep dive**
    1. Scope and return values
    2. Functional patterns
    3. .map().filter().reduce()
3. Logout

Schedule

1. Review assignments
2. JavaScript deep dive
    1. **Scope and return values**
    2. Functional patterns
    3. .map().filter().reduce()
3. Logout

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 greet(name) {
    let greeting = “Goedemorgen, ”;
    console.log(greeting + name);
}
```

(A)greeting=NULL (B)greeting == “Goedemorgen, ” (C) greeting 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 greet(name) {
   let greeting = “Goedemorgen, ”+ name;
   return greeting
}


console.log(greet(“Laura”);
```

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Scope and return values
    2. **Functional patterns**
    3. .map().filter().reduce()
3. Logout

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

```
1  const kleuren = ["rood", "geel", "paars", "turquoise"];
2  const dieren = ["hond", "kat", "kip", "schildpad", "paard",
   "parkiet", "cavia"];
3
4
5  function telImpure() {
6    console.log(kleuren.length);
7
8  }
9
10 telImpure();
```

Pure functions

```
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
- C**an be reused** in different contexts

Functional programming

- Supports reuse through abstraction
- `console.log();` Only writes 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/respons
es')
        .then(res => res.json())
        .then(data => {↔});
    }
```

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Scope and return values
    2. Functional patterns
    3. **.map().filter().reduce()**
3. Logout

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

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

.map().filter().reduce() in JavaScript5:

Steven Luscher
@steveluscher

Map/filter/reduce in a tweet:

map([🌽, 🐮, 🐔], cook)
=> [🍿, 🍔, 🍳]

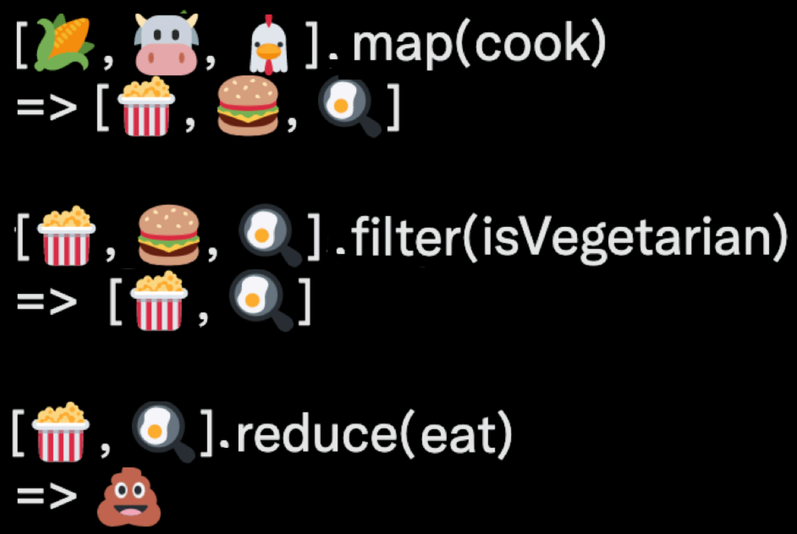
filter([🍿, 🍔, 🍳], isVegetarian)
=> [🍿, 🍳]

reduce([🍿, 🍳], eat)
=> 💩

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

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

Vrij naar Steven Luscher, in ES6:



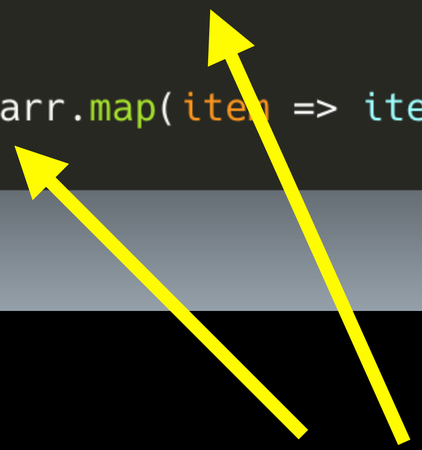
.map().filter().reduce()

Map, filter (and reduce) are methods of **Array**
that **return** something: an **Array or value.**

They help us to write **robust** code through **abstraction**

Implicit vs explicit

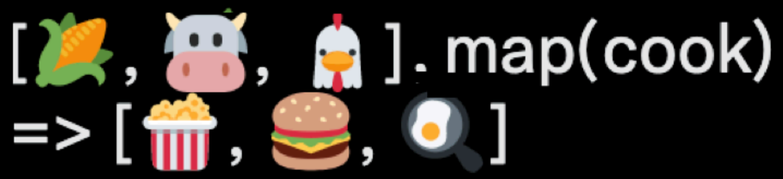
```
const arr = [1,2,3];

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

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



This code does the same

.map()

.map() is a method of **Array**
applies to **arrays**
requires a **function** as parameter



returns a new **array** where the **function**
is applied to each element of the array

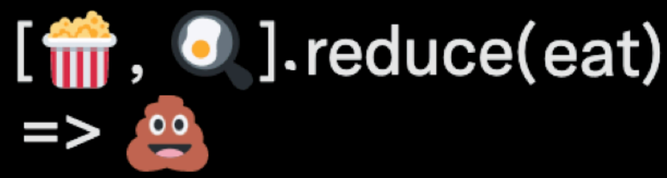
.filter()

.filter() is a method of **Array**
applies to **arrays**
requires a **function** as parameter that
returns a **boolean**

returns the same **array** whit the **elements**
that comply to the test-condition in the function

.reduce()

.reduce() is a method of **Array**
applies to **arrays**
requires (an accumulator value and)
a **function** as parameter
that returns a **value**



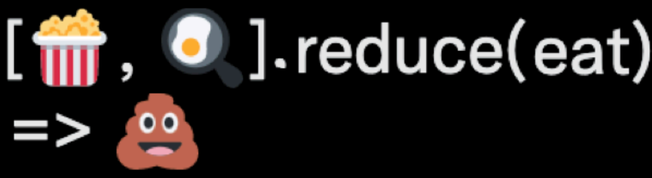
returns a **value**

.map().filter().reduce() The devil in the details

.**map()** does not affect the original Array.
It returns a **new** one

.**filter()** **changes** the original Array.

.**reduce()** returns a **value**, generally a Number



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

Hands-on

ObjectsMapFilter	(practise with Map, Filter)
ObjectsMapFilterReduce	(Add Reducet)
CountriesMapFilterReduce	(practise Map, Filter, reduce with a medium-size dataset)

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

Code lezen

Ok, hij doet het nu. Weet je ook, wat er staat (en wat “hij” doet?)

Schedule

1. Review assignments
2. JavaScript deep dive
    1. Scope and return values
    2. Functional patterns
    3. .map().filter().reduce()
3. **Logout**

Logout

Vrijdag 18: Landelijk actiedag tegen de langstudeerboete. Wie gaat daarheen?

Van FDND

Woensdag de 16e, om 13:30, komt Kevin Lewis van Directus een gastles geven over het maken van keuzes voor een tech-stack in Sprint 15: Choices Choices bij FDND op de 4e etage in het TTH.

Ook geeft hij wat meer insights over Directus (een Headless CMS) en maakt hij ruimte voor een Q&A.

Uncaught SyntaxError
Unexpected end of input