23.01.2020
Düsseldorf, Geektastic Connect

Per Anhalter durch JavaScript

INNOQ

Und der Petunientopf: „Oh nein, nicht schon wieder“

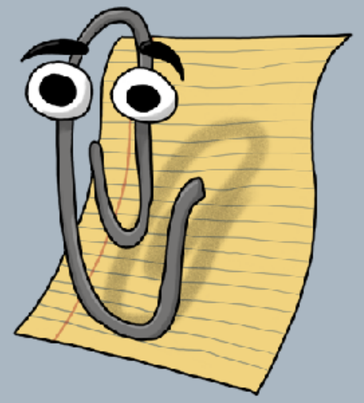
Design

1. Brendan mochte gerne Scheme und Self
2. Marketing-Abteilung wollte Ähnlichkeit zu Java
3. Es sollte einfach zu benutzen sein, auch für Nicht-Programmierer
4. Er hatte nur 10 Tage Zeit

Automatic Type Conversion

"The ECMAScript runtime system performs automatic type conversion as needed."

ECMAScript 2018 Language Specification, 7.1 Type Conversion



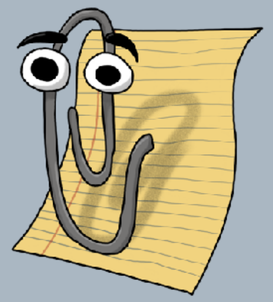
Es sieht so aus als würdest du einen String brauchen. Ich hab ihn für dich konvertiert.

```
'Hello' + 'World' // HelloWorld
'Hello' + 5 // Hello5
'Hello' - 1 // NaN
'' - 1 // -1
```

Klarer Fall, du willst eine Zahl*!

* 'hello' als Zahl ist NaN

'' als Zahl ist 0



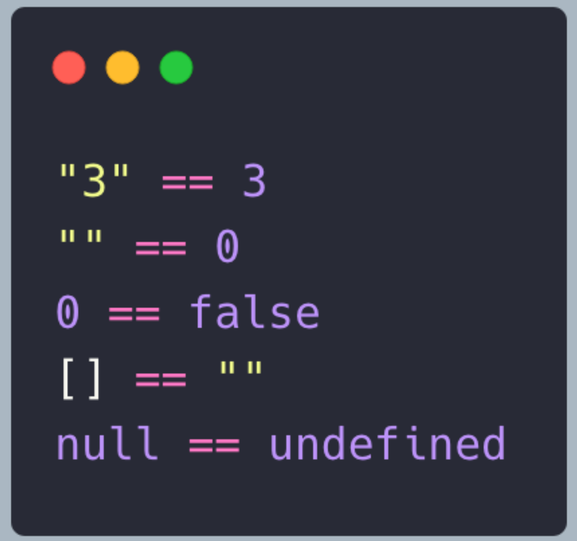
Klarer Fall! Du willst einen Boolean

```
if (something) {
  // do stuff
}
```

Die Wahrheit ist irgendwo da draußen

Alles wird zu true umgewandelt außer

`false, null, undefined, 0, NaN, ""`


"3" == 3
"" == 0
0 == false
[] == ""
null == undefined

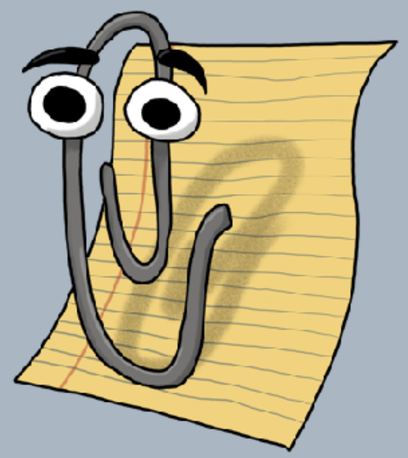
JavaScript versucht für dich Dinge in den richtigen Typen zu verwandeln.
Dabei stellt es sich nicht immer klug an.

Folglich

- Immer === statt == benutzen – hier passiert keine Typkonvertierung
- Bei Gleichheit zudem beachten: Primitive Werte werden anhand ihres Wertes verglichen, Objekte anhand ihrer Identiät
- Was ist ein Objekt? Dazu nun mehr.

Objekte

Kurzschreibweise
für
x["a"]



```
let lisa = { name: "Lisa" }
let lucas = { name: "Lucas" }

let cache = {}
cache[lisa] = "<h1>Lisa</h1>"
```

Clippy strikes again

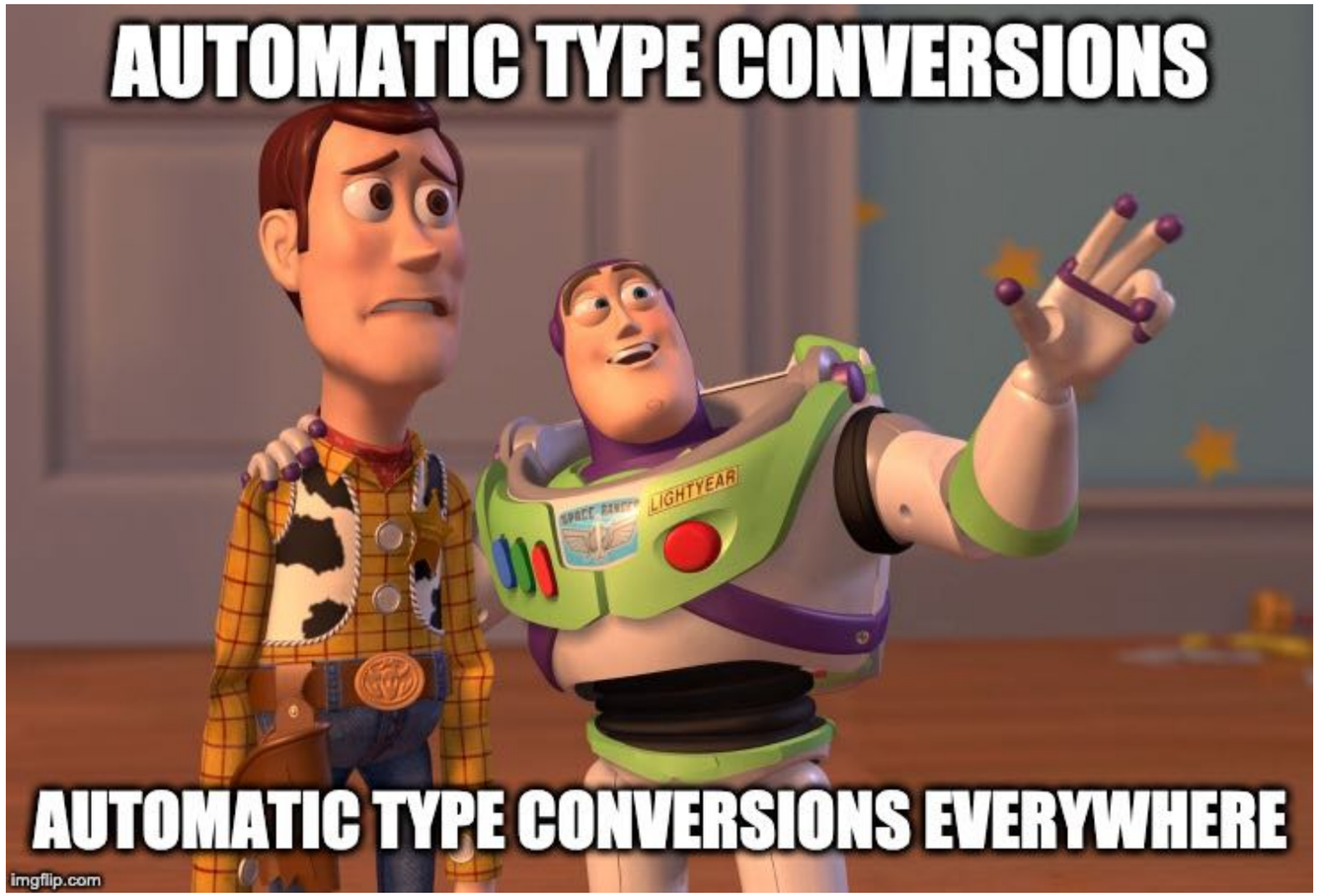
cache[lisa] ist nun <h1>Lisa</h1>
...aber cache[lucas] auch
Wait what?

Objekte

- Ein Objekt ist ein Dictionary
- Die Schlüssel sind immer Strings
- Die Werte können beliebige Typen haben
- Ist der Schlüssel kein String, wird toString() darauf aufgerufen:
  - `lisa.toString() === '[object Object]'`
  - `lucas.toString() === '[object Object]'`
  - `lisa.toString() === lucas.toString()`

```
let list = ["a", "b", "c"]
list[1] // "b"
list["1"] // "b"
```

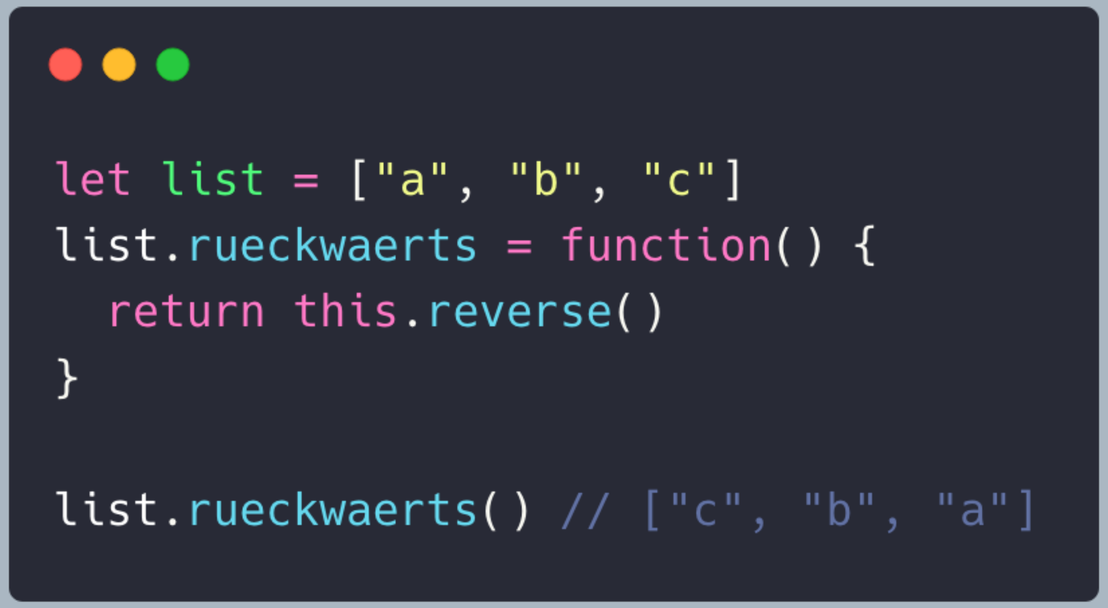
Arrays sind auch Objekte


AUTOMATIC TYPE CONVERSIONS
LIGHTYEAR
AUTOMATIC TYPE CONVERSIONS EVERYWHERE
imgflip.com

Fast alles ist ein Objekt

- Primitive Values: Undefined, Null, Boolean, Number, String, Symbol
- Alles andere sind Objekte, zB.:
  - Arrays
  - Reguläre Ausdrücke
  - Promises
  - Funktionen
  - ...

```
let list = ["a", "b", "c"]
list.name = "Meine Liste"
list.name // "Meine Liste"

let fn = a => a * 2
fn.name = "Verdoppeln"
fn.name // "Verdoppeln"
```



```
let list = ["a", "b", "c"]
list.rueckwaerts = function() {
  return this.reverse()
}

list.rueckwaerts() // ["c", "b", "a"]
```

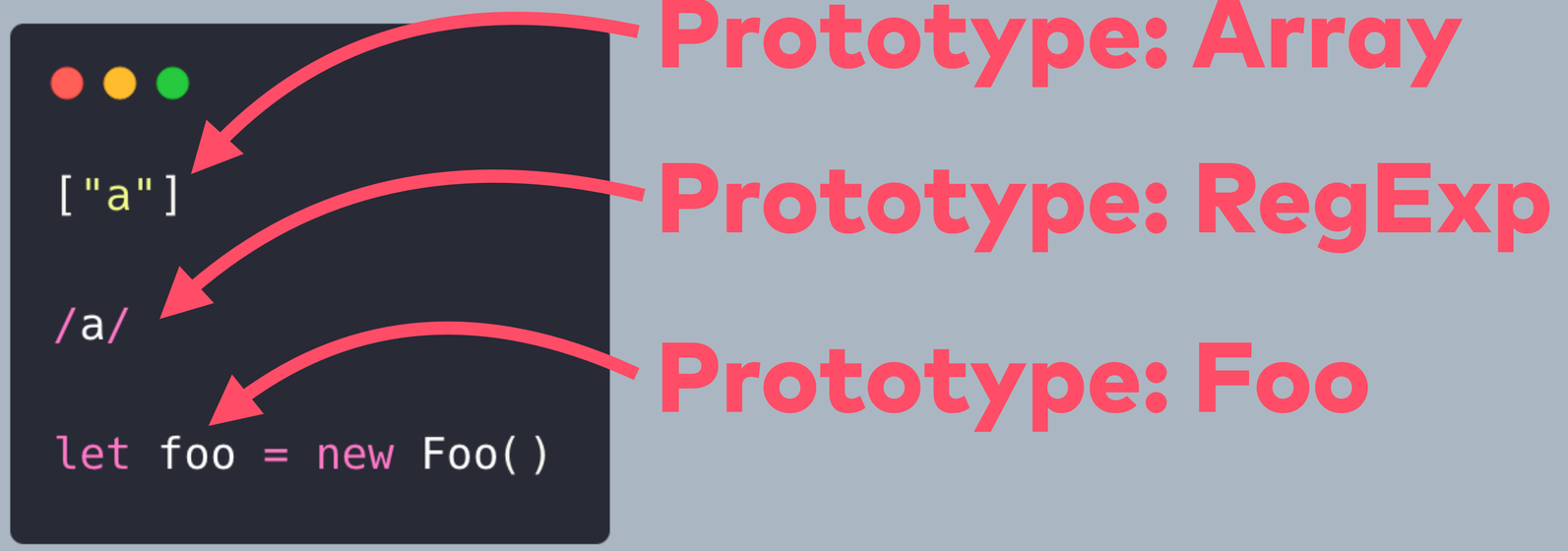
Das fügt die Methode aber nur dieser Liste hinzu

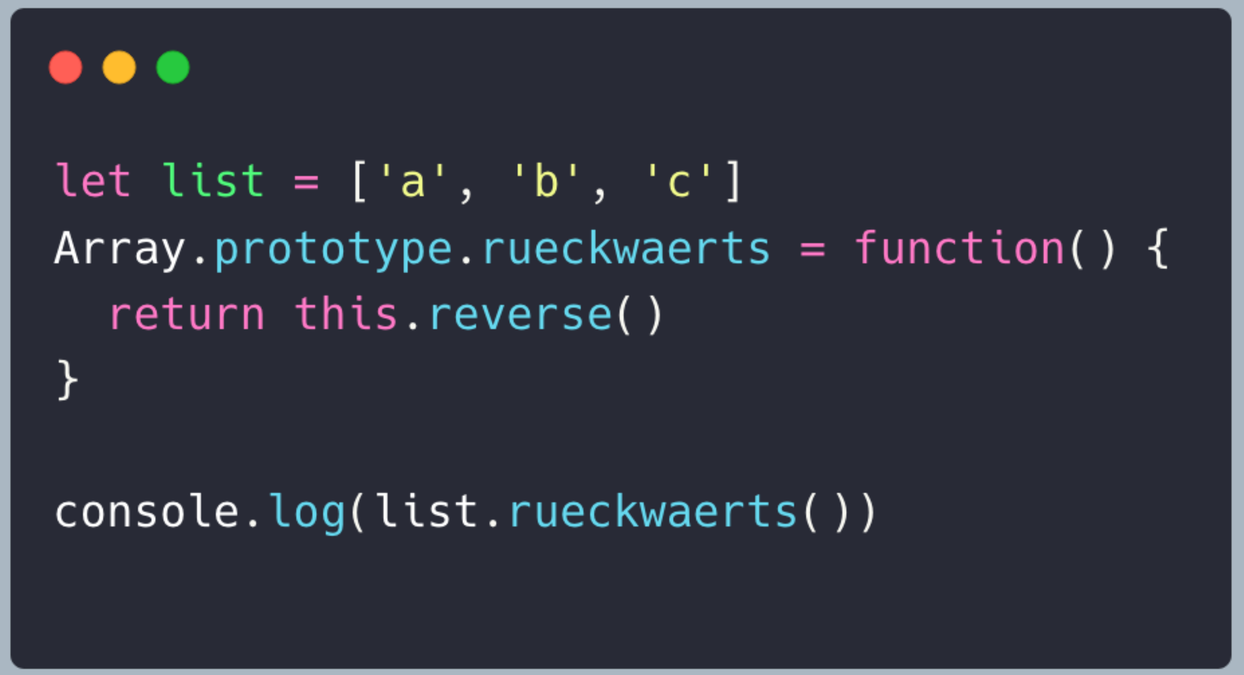
```
let person = {
  age: function() {
    return 2020 - this.birthYear;
  }
}

let person1 = Object.create(person);
person1.birthYear = 1988;
console.log(person1.age()); // 32
```

person ist der Prototyp von person1

You can think of a prototype as a "hidden" property on every object that determines "where to look next". So if there's no taste property on iceCream, JavaScript will look for a taste property on its prototype, then on that object's prototype, and so on, and will only give us undefined if it reaches the end of this "prototype chain" without finding .taste

Dan Abramov


["a"]
/a/
let foo = new Foo()
Prototype: Array
Prototype: RegExp
Prototype: Foo

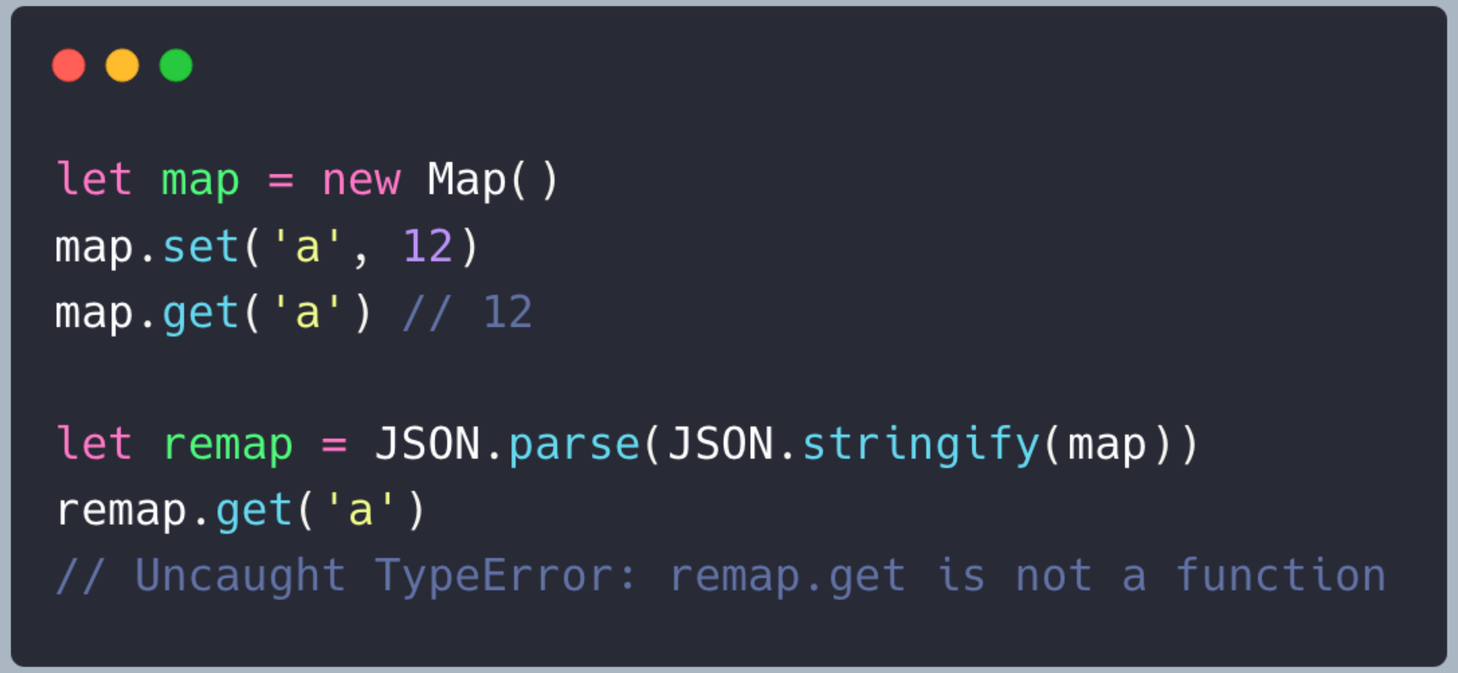


```
let list = ['a', 'b', 'c']
Array.prototype.rueckwaerts = function() {
  return this.reverse()
}

console.log(list.rueckwaerts())
```

(don't do this at home)

class ist nur neue Syntax

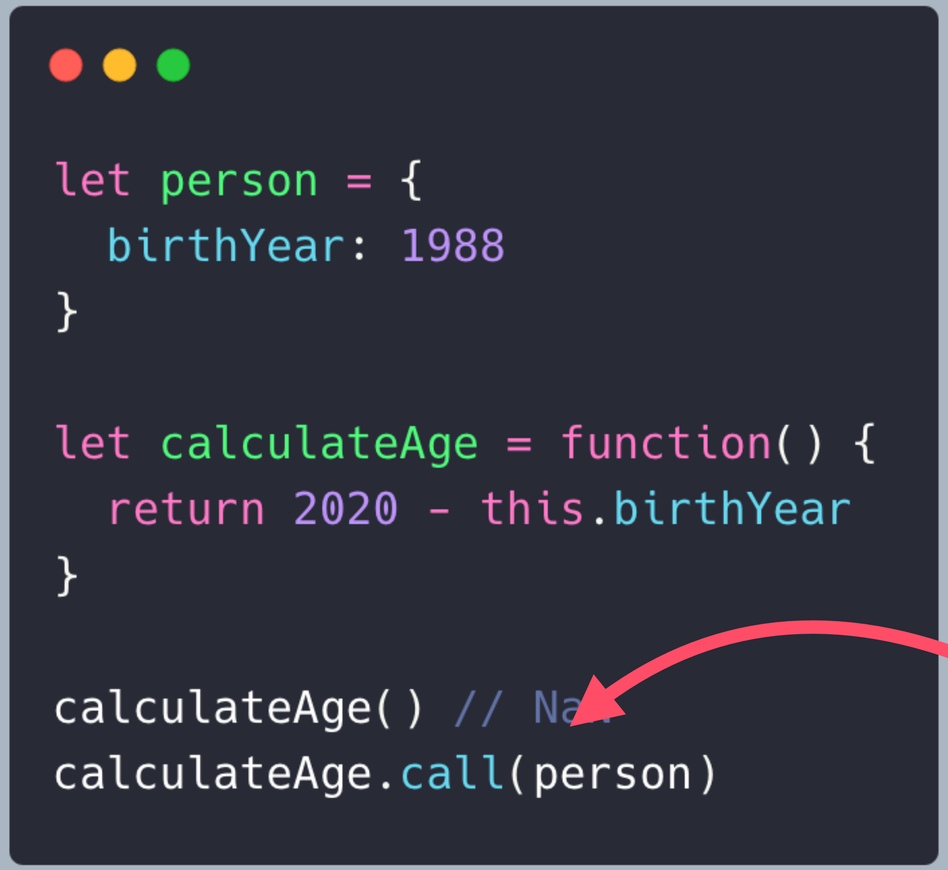
für diese Funktionalität

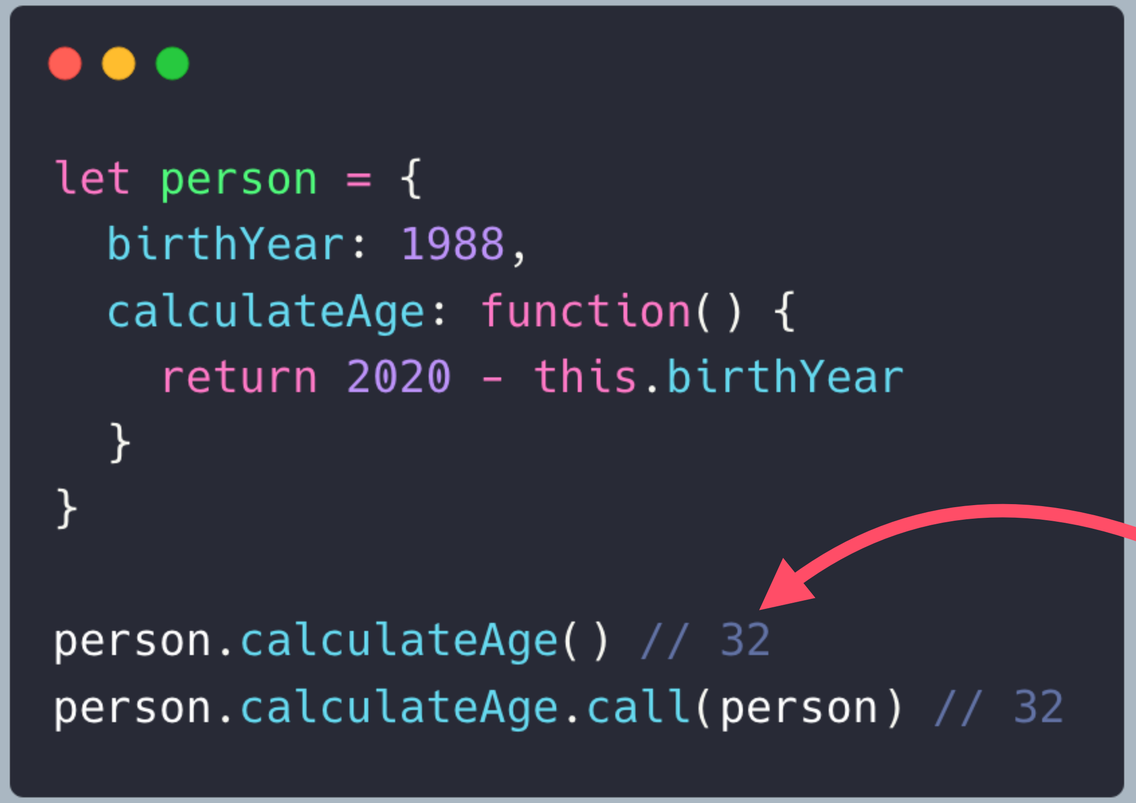


```
let map = new Map()
map.set('a', 12)
map.get('a') // 12

let remap = JSON.parse(JSON.stringify(map))
remap.get('a')
// Uncaught TypeError: remap.get is not a function
```

Objekte sind Dictionaries
die Strings auf beliebige Objekte abbilden
und alle nicht definierten Schlüssel an die
Prototypenkette weitergeben.
Wird nichts gefunden, wird undefined
zurückgegeben.

this


let person = {
birthYear: 1988
}
let calculateAge = function() {
return 2020 - this.birthYear
}
calculateAge() // Na
calculateAge.call(person)
person ist this


let person = {
birthYear: 1988,
calculateAge: function() {
return 2020 - this.birthYear
}
}
person.calculateAge() // 32
person.calculateAge.call(person) // 32
Zauberei!

```
let person = {
  birthYear: 1988,
  calculateAge: function() {
    return 2020 - this.birthYear
  }
}

let calculateAge = person.calculateAge
calculateAge() // NaN
calculateAge.call(person) // 32
```

bind benutzen

this funktioniert nicht wie in anderen Sprachen:
Der Wert von this hängt vom Aufruf ab.

„Seht mich an. Ein Hirn von der Größe eines Planeten,... und man schickt mich, um euch in die Kommandozentrale zu bringen. Nennt man das vielleicht berufliche Erfüllung? Also ich nicht.“

Destructuring

Grundlage

```
const result = {
  answer: 42,
  isCalculatedByDeepThought: true
};
const { answer, isCalculatedByDeepThought } = result;

console.log(answer);                      // 42
console.log(isCalculatedByDeepThought); // true
```

Umbenennen

```
const {name: newName} = fromObject
```

Default-Wert

```
const {name = 'default'} = fromObject
```

Kombination

```
const {name: newName = 'default'} = fromObject
```

„Cooleres“ Destructuring

```
const twitterProfile = {
  id: 'teapot4181',
  displayName: 'Teapot-418',
  fullName: {
    firstName: 'Lisa',
    lastName: 'Moritz'
  }
};

function whois({id, displayName, fullName: {firstName: name}}) {
  return `${displayName} (${id}) is ${name}`;
}

console.log(whois(user));  // "Teapot-418 (teapot4181) is Lisa"
```

Arrays sind Objekte...

```
const [a, ...b] = [1, 2, 3];
console.log(a); // 1
console.log(b); // [2, 3]
```

„Der Rest“

```
function f() {
  return [1, 2, 3];
}

const [a, , b] = f();
console.log(a); // 1
console.log(b); // 3
```

„Ignorieren“

Hoisting

Definition

Mit **Hoisting** (engl., Hochziehen) bezeichnet man das Verhalten des JavaScript-Interpreters bei der Deklaration von Variablen und Funktionen.

Wikipedia
21.01.20, https://de.wikipedia.org/wiki/Hoisting

WIKIPEDIA
Die freie Enzyklopädie

Funktionsaufruf ohne Hoisting

```
function getTwitterProfileURL(id) {
  return `https://twitter.com/${id}`;
}

getTwitterProfileURL('teapot4181');

// https://twitter.com/teapot4181
```

Funktionsaufruf mit Hoisting

```
getTwitterProfileURL('teapot4181');

function getTwitterProfileURL(id) {
  return `https://twitter.com/${id}`;
}

// https://twitter.com/teapot4181
```

Hoisting von Variablen

```
console.log(theAnswer); // undefined
var theAnswer;
theAnswer = 42;
```

```
theAnswer = 42;
console.log(theAnswer); // 42
var theAnswer;
```

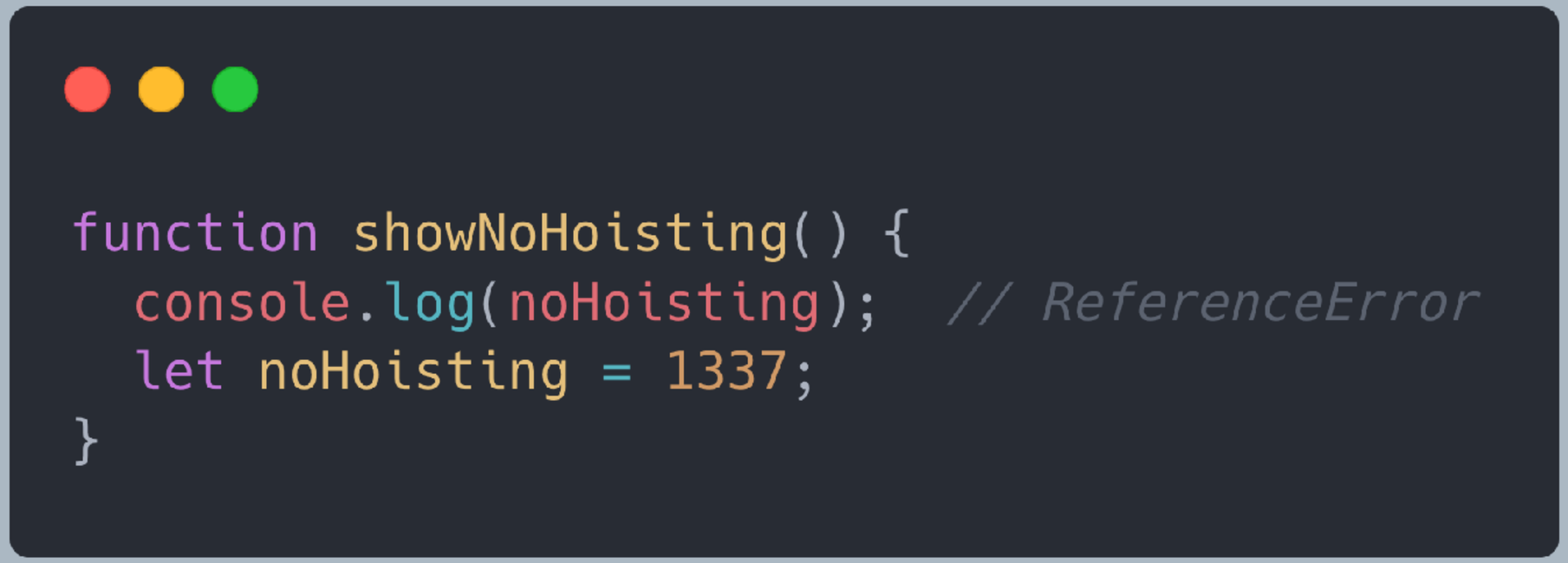
let -> kein Hoisting, Text

In ECMAScript 2015, werden Deklarationen mit *let* nicht an den Anfang des Blocks verschoben (hoist).

MDN

22.01.20, https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Statements/let

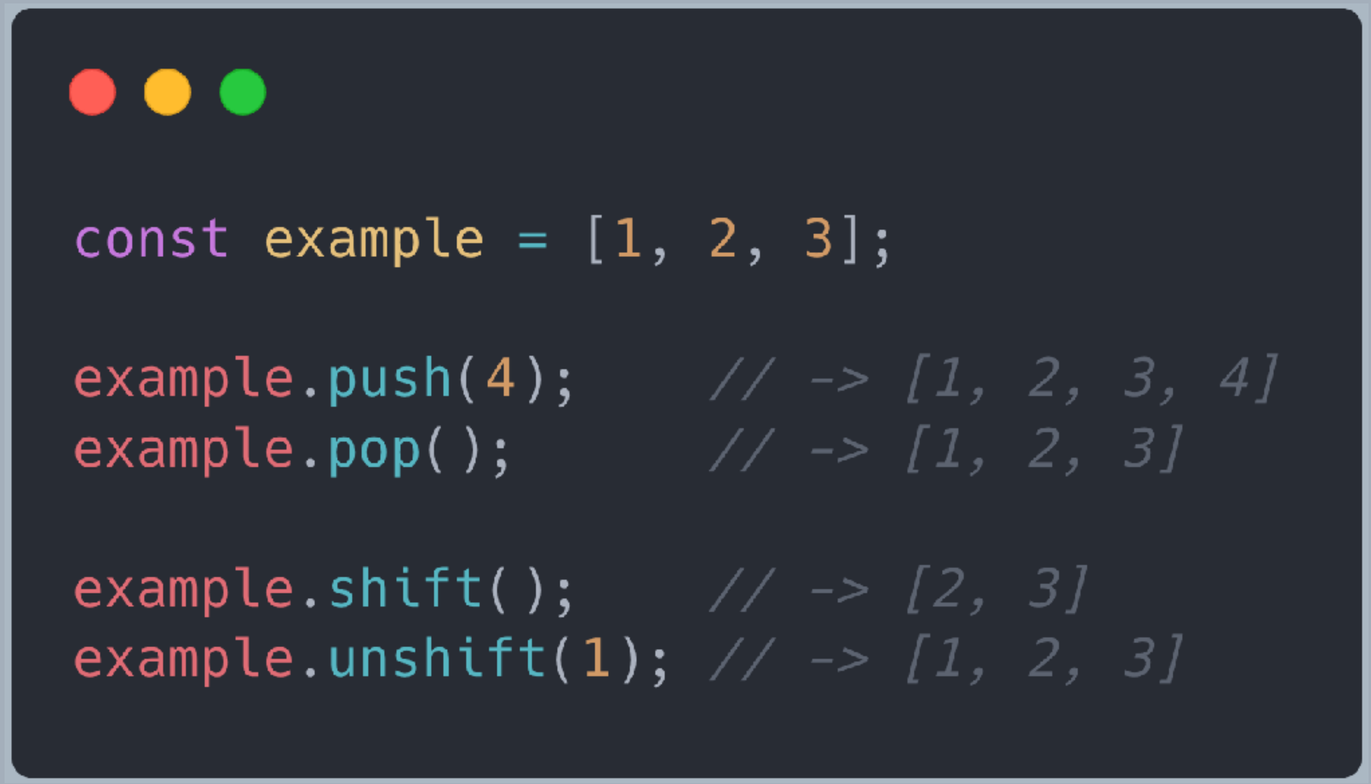
let -> kein Hoisting, Beispiel



```
function showNoHoisting() {
  console.log(noHoisting);  // ReferenceError
  let noHoisting = 1337;
}
```

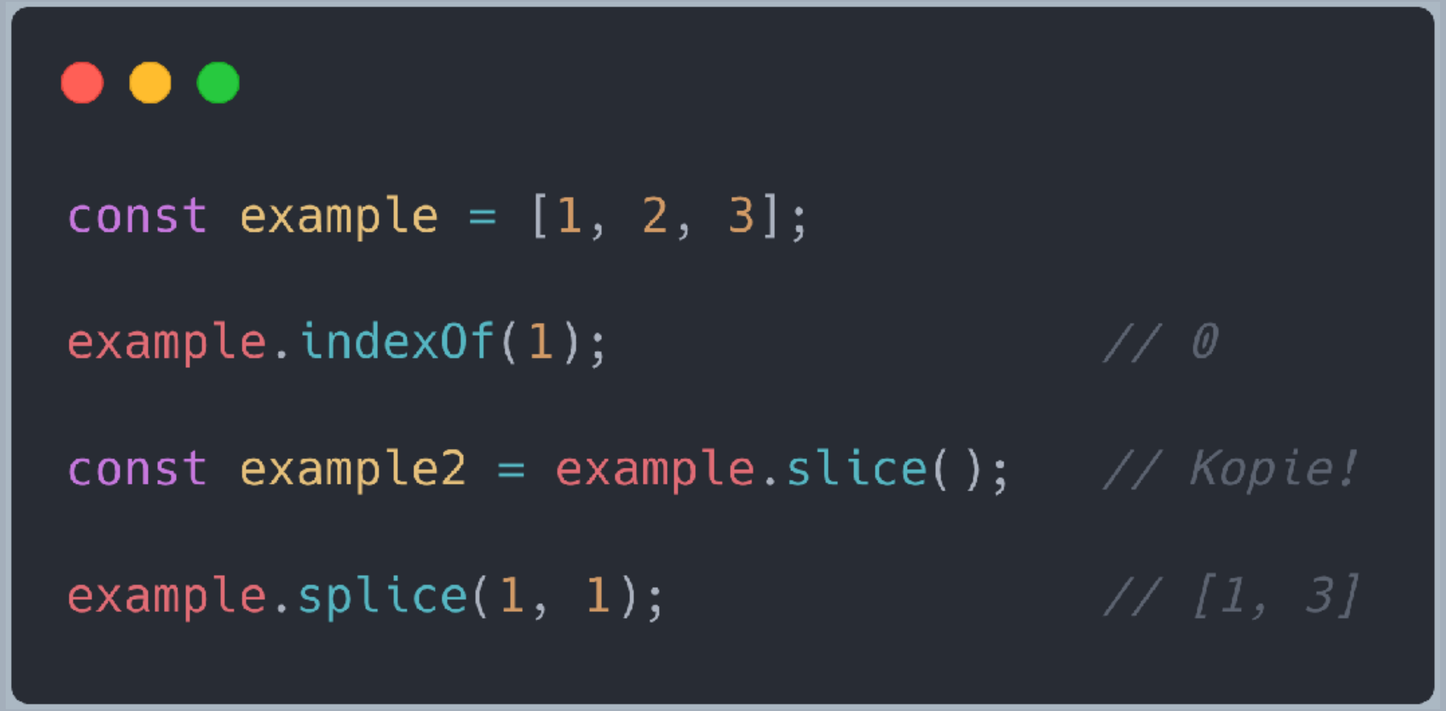
Array-Methoden

Die bekannten Klassiker, 1



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

example.push(4);     // -> [1, 2, 3, 4]
example.pop();       // -> [1, 2, 3]

example.shift();     // -> [2, 3]
example.unshift(1); // -> [1, 2, 3]
```

Die bekannten Klassiker, 2



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

example.indexOf(1);                    // 0

const example2 = example.slice();      // Kopie!

example.splice(1, 1);                  // [1, 3]
```

Schleifen -> forEach

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

allResults.forEach((result, index) => {
  console.log(`${index}: ${result}`);
})

/*
0: 1
1: 2
2: 3
*/
```

Schleifen -> for ... of

Wartet bei await

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

for (const result of allResults) {
  await persistInDatabase(result);
  console.log(`Waited for persisting ${result}`);
}
```

includes

Coole Array-Methoden

```
const httpCodes = [200, 201, 203, 418, 500];

if (httpCodes.includes(418)) {
  console.log('Teapot spotted');
}
```

filter

Coole Array-Methoden

```
const myNumbers = [1, 2, 3, 4, 5];

const evenNumbers = myNumbers.filter(number => number % 2 === 0);

// evenNumbers: [2, 4]
```

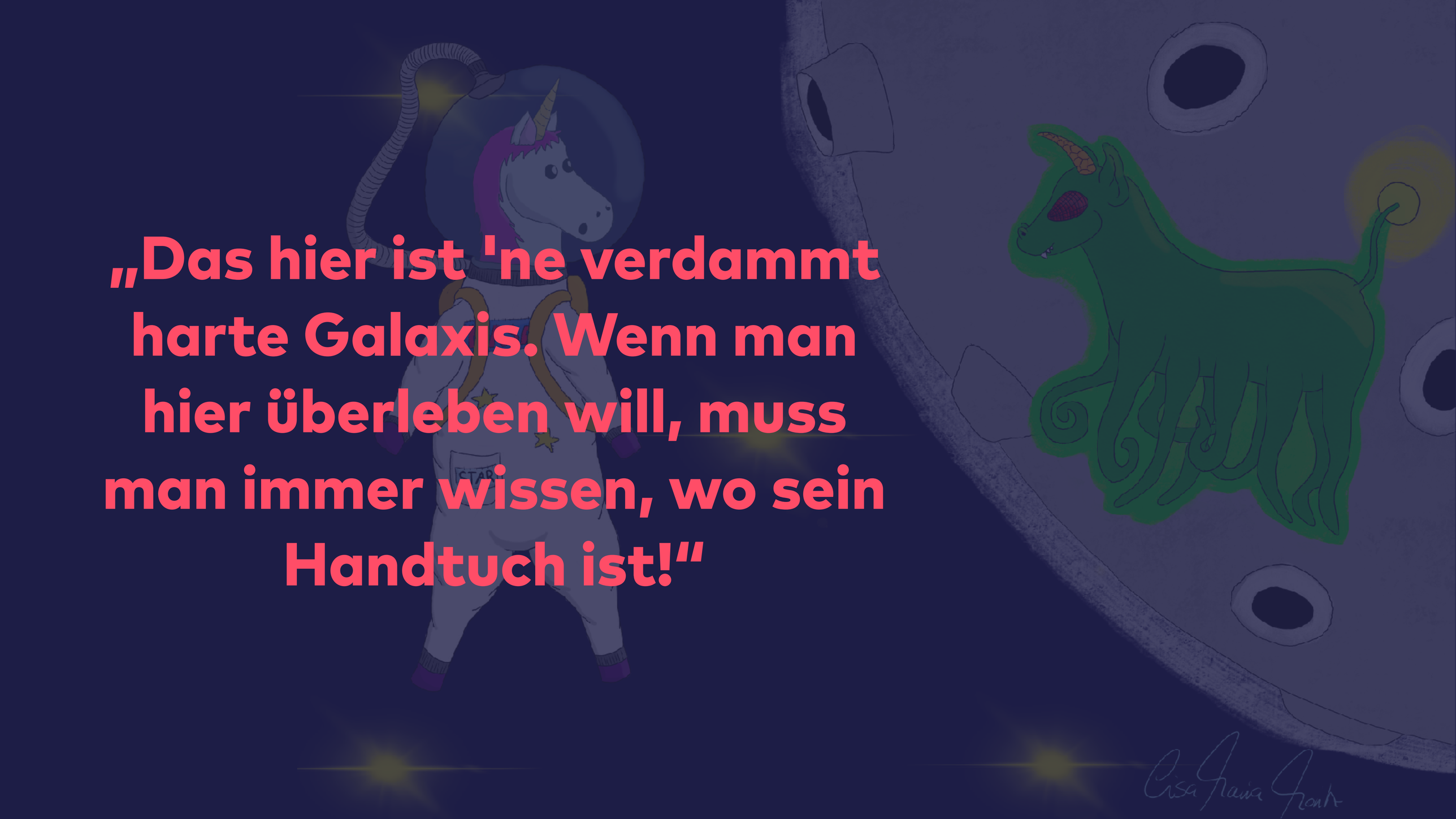
Coole Array-Methoden

```
const myNumbers = [1, 2, 3, 4, 5];

const myNumbers2 = myNumbers.map(number => 2 * number);

// evenNumbers: [2, 4, 6, 8, 10]
```

reduce

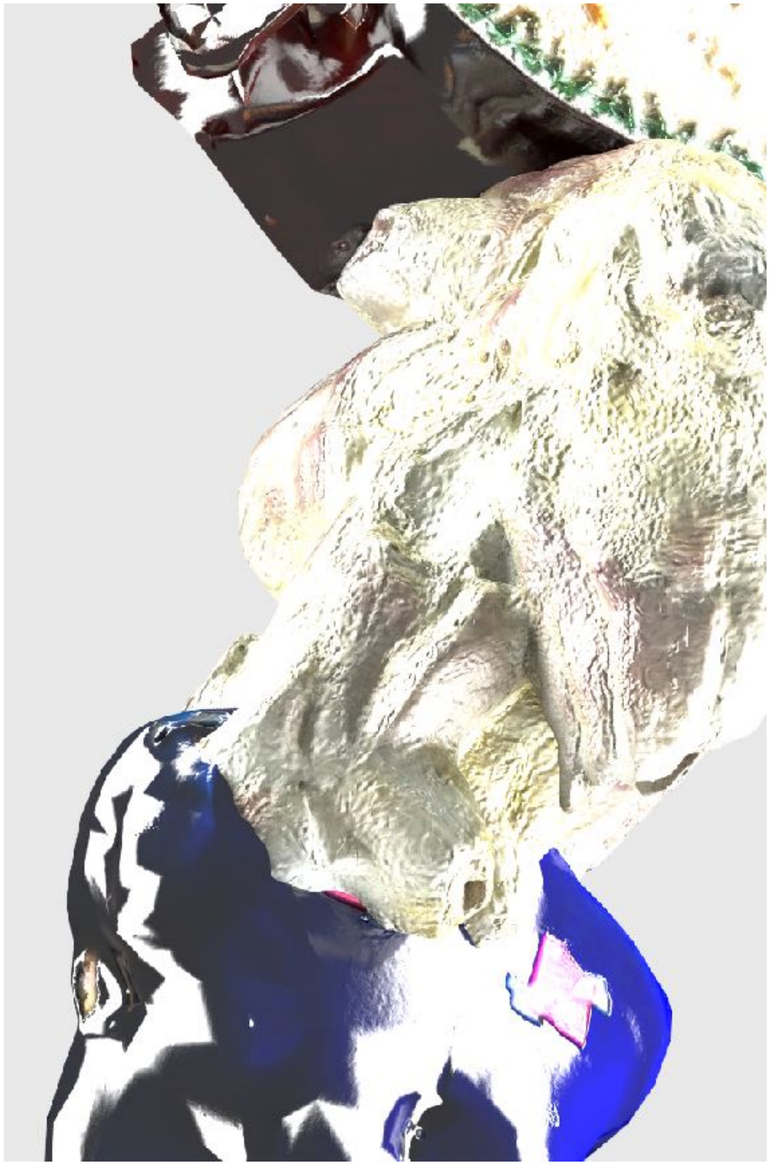
Coole Array-Methoden

```
const myNumbers = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) => accumulator + currentValue;

// 1 + 2 + 3 + 4
console.log(myNumbers.reduce(reducer));     // 10
```


„Das hier ist 'ne verdammt harte Galaxis. Wenn man hier überleben will, muss man immer wissen, wo sein Handtuch ist!“

Projekt aufsetzen

1. editorconfig
2. prettier
3. ESLint
4. Testing
5. NPM scripts
6. Ein paar Tipps zum Client/Server



editorconfig

- Plug-in für die meisten Editoren/IDEs
- Konfiguriert einige grundlegende Dinge über eine Config Datei

```
[*]
charset = utf-8
end_of_line = lf
insert_final_newline = true
trim_trailing_whitespace = true
max_line_length = 80
indent_style = tab
indent_size = 2
```

prettier

- Plug-in für die meisten Editoren/IDEs
- Automatisches Code Formatting (zB. beim Speichern)
  - auch für TypeScript, (S)CSS…
- Keine* Config, beachtet die editorconfig

```
 const promise = new MyPromise(
(resolve,      reject) => {
   setTimeout(() =>
      resolve(42),
        1000)
  })
```

⇒

```
const promise = new MyPromise((resolve, reject) => {
  setTimeout(() => resolve(42), 1000)
})
```

ESLint

- Plug-in für die meisten Editoren/IDEs
- Feedback zu häufigen Fehlern
- Hochkonfigurierbar
- Kann mit Prettier zusammenarbeiten

```
{
  "extends": "eslint:recommended",
  "plugins": ["prettier"],
  "rules": {
    "prettier/prettier": "error"
  },
  "parserOptions": {
    "ecmaVersion": 2018
  },
  "env": {
    "browser": true,
    "node": true
  }
}
```

Testing

- Eine riesige Auswahl an Testing Tools
- Die meisten sind okay
- In jedem Projekt ein neues Testing Tool…

NPM scripts

- Der einfachste Task Runner:
- Name, Shell Script
- npm run test (bzw. npm t)
- Kann auf CI ausgeführt werden

Client:
Asset Pipeline

FAUCET

Server:
dotenv
Logging
express

Danke! Fragen?

INNOQ
www.innoq.com

Lucas Dohmen
lucas.dohmen@innoq.com
+49 151 75062496
moonbeamlabs

Lisa Maria Moritz
lisa.moritz@innoq.com
+49 176 646 300 28
teapot4181

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany
+49 2173 3366-0

Ludwigstr. 180E
63067 Offenbach
Germany
+49 2173 3366-0

Kreuzstr. 16
80331 München
Germany
+49 2173 3366-0

Hermannstrasse 13
20095 Hamburg
Germany
+49 2173 3366-0

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 0116