

WEBTECHNOLOGIEN

03 – JAVASCRIPT

NODEJS, CONST LET, OBJEKTE UND ARRAYS AUS OBJEKTEN

PROF. DR. MARKUS HECKNER

NODE.JS IST EIN FRAMEWORK ZUR ENTWICKLUNG VON ANWENDUNGEN IN JS

- Framework, das JS Code auf dem Server ausführen kann
- Rudimentäre Unterstützung von http-Kommunikation
- Erweiterbar durch packages (z.B. Express)
- Replit bietet Template für Node.JS Projekte

Node.js



Hello world from Node.js in replit:

```
hello_node.js x [Menu Icon] Console Shell
1 console.log("Hello world from NodeJS");
~/WebDevkidslecture0301JS0bjects$ node hello_node.js
Hello world from NodeJS
~/WebDevkidslecture0301JS0bjects$
```

CONST


- Ähnlich zu `let` (Blockscope)
- Redeklamieren und zuweisen neuer Werte nicht möglich
- D.h. Variablen, die mit `const` deklariert wurden sind constant (offensichtlich)

```
// String  
const greeting = "hello";
```

```
// Number  
const favoriteNum = 42;
```

```
// Boolean  
const isAwesome = true;
```

```
// Gives an error  
isAwesome = false;
```



```
isAwesome = false;  
           ^
```

```
TypeError: Assignment to constant variable.
```

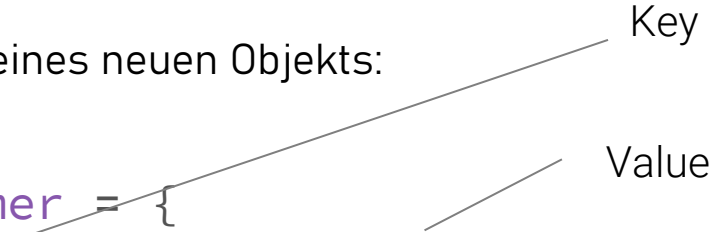
```
    at Object.<anonymous> (/home/runner/WebDevkidslecture0301JSobjects/  
const.js:11:11)
```

JAVASCRIPT EIGENE OBJEKTE

- Objekte sind “key-value-pairs”
- “key-value-pairs” sind die Attribute des Objekts
- Homer enthält: Zwei Strings und eine Number

Erzeugen eines neuen Objekts:

```
let homer = {  
  name: "Homer Simpson",  
  age: 39,  
  occupation: "low-level safety inspector"  
}
```



Zugriff auf die Attribute des Objekts:

```
console.log(homer.name);  
console.log(homer.age);  
console.log(homer.occupation);
```

OBJEKTE KÖNNEN AUCH FUNKTIONEN ENTHALTEN

- Auch Funktionen sind über Keys des Objekts aufrufbar (hier: say zeigt auf die Funktion say())
- this referenziert Variablen des Objekts

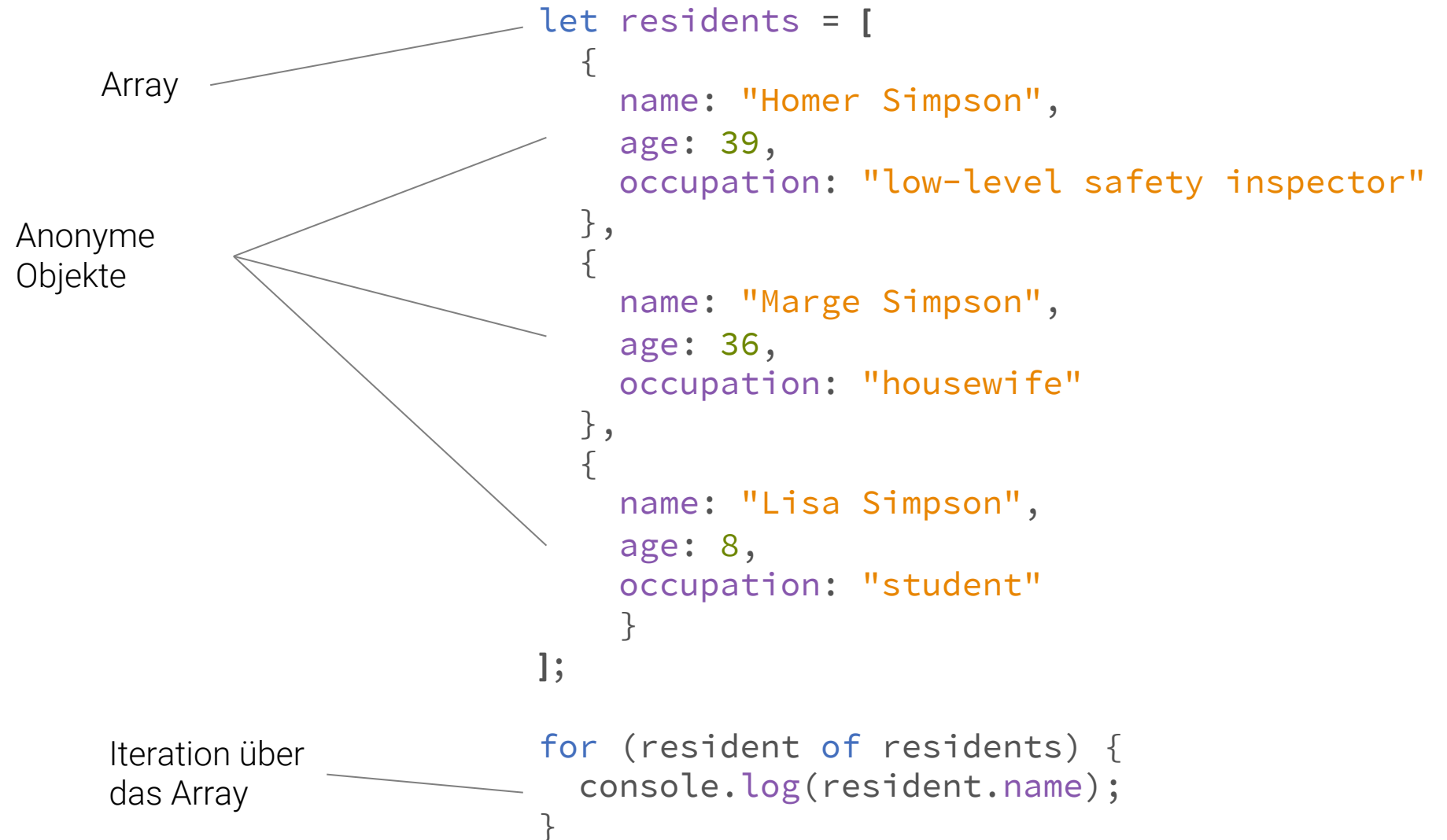
```
let homer = {  
  name: "Homer Simpson",  
  age: 39,  
  occupation: "low-level safety inspector",  
  
  say: function hello() {  
    console.log("Hello from " + this.name);  
  }  
}  
  
homer.say();
```

OBJEKTE: KURZSCHREIBWEISE FÜR FUNKTIONEN

- say: function kann weggelassen werden

```
let homer = {  
  name: "Homer Simpson",  
  age: 39,  
  occupation: "low-level safety inspector",  
  
  say() {  
    console.log("Hello from " + this.name);  
  }  
}  
  
homer.say();
```

ANONYME OBJEKTE KÖNNEN ZUR ERZEUGUNG VON ARRAYS VERWENDET WERDEN



OBJEKTE ALS MODULE DIE VON
CLIENT CODE VERWENDET WERDEN

residentStore ist das
Modul

residents Array als
Attribut des Objekts
residentStore

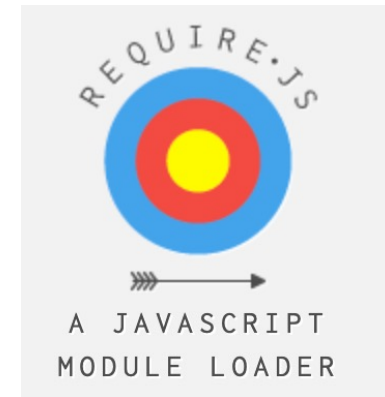
residentStore bietet
zwei Funktionen für Clients
an

Erzeugen eines neuen
Objekts (newResident) und
verwenden der Funktionen
von residentStore

```
const residentStore = {  
  residents: [  
    {  
      name: "Homer Simpson",  
      age: 39,  
      occupation: "low-level safety inspector"  
    },  
    {  
      name: "Marge Simpson",  
      age: 36,  
      occupation: "housewife"  
    },  
  ],  
  addResident(resident) {  
    this.residents.push(resident);  
  },  
  getResidents() {  
    return this.residents;  
  }  
}  
  
let newResident = { name: "Lisa Simpson", age: 8,  
  occupation: "student"};  
residentStore.addResident(newResident);  
let residents = residentStore.getResidents();  
for (resident of residents) {  
  console.log(resident.name);  
}
```

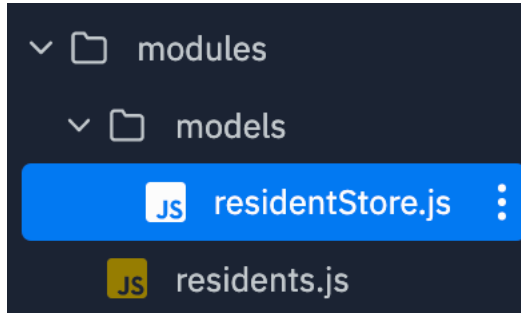

DER GANZE CODE IN EINER DATEI WIRD UNÜBERSICHTLICH UND IST IRGENDWANN NICHT MEHR WARTBAR

- Um eine Applikation sinnvoll zu strukturieren wird diese auf verschiedene JS-Dateien aufgeteilt
- Objekte die in diesen Dateien erzeugt werden müssen,
 - Von einer Datei exportiert werden
 - Von einer anderen Datei importiert werden
- Jedes Modul sollte sich dabei um eine eigene Aufgabe kümmern
- Mechanismus: require.js (kann in Node verwendet werden)



RequireJS is a JavaScript file and module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments, like Rhino and Node. Using a modular script loader like RequireJS will improve the speed and quality of your code.

DEFINITION EINES MODULS UND EXPORTIEREN DER FUNKTIONALITÄT



residentStore.js

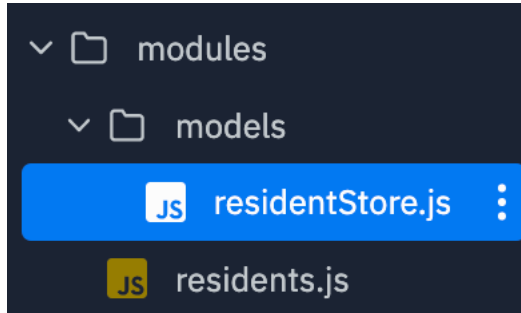
```
const residentStore = {  
  
  residents: [  
    ...  
  ],  
  
  addResident(resident) {  
    this.residents.push(resident);  
  },  
  getResidents() {  
    return this.residents;  
  }  
}
```

Stelle das Objekt
residentStore für
andere Dateien zur
Verfügung

module.exports = residentStore;

VERWENDEN DES CODES AUS DEM MODUL

JAVASCRIPT



Importieren des Moduls
residentStore

Verwenden des Codes aus
dem Modul

residents.js

```
const residentStore =  
require("../models/residentStore.js");  
  
let newResident = { name: "Lisa Simpson", age: 8,  
occupation: "student"};  
residentStore.addResident(newResident);
```

FAZIT

- Objekte in JavaScript haben Attribute, Attribute können Variablen oder Funktionen sein (auch wenn Funktionen strenggenommen auch Variablen sind)
- `this` zeigt auf das aktuelle Objekt
- Arrays lassen sich aus anonymen Objekten erzeugen
- Größere Applikationen lassen sich mit `require.js` modularisieren (eine Datei exportiert (`module.exports`) die Funktionalität, eine andere Datei importiert die Funktionalität (`require`))

QUELLEN

- Imbert, T. (2013). A JavaScript Refresh. Online verfügbar: <http://typedarray.org/JavaScript-refresh/>. Letzter Zugriff: 11.08.2015.
- Mozilla Developer Network. (2015b). A re-introduction to JavaScript (JS tutorial). Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript
- Mozilla Developer Network. (2015b). Introduction to Object Oriented JavaScript. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript#JavaScript_object_oriented_programming. Letzter Zugriff: 13.08.2015