FRONTEND SCHULUNG

Team NWI

AGENDA

- Javascript Basics
- Objekte & Funktionen
- es5 & es6 Neuerungen
- vue, vue router, vuex

HAT JAVASCRIPT (JS) TYPEN?

Wikipedia:

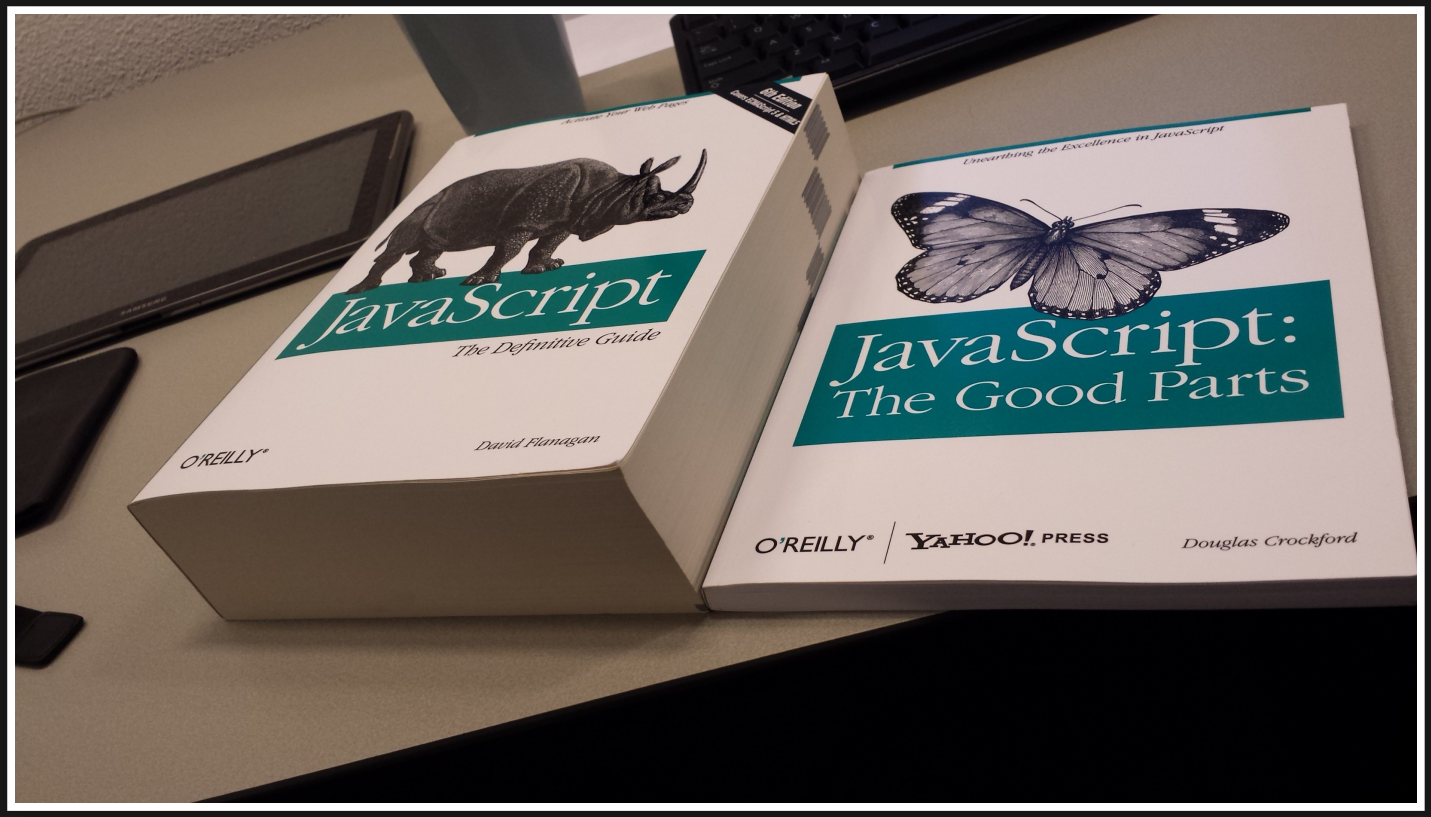
[JS ist] eine dynamisch typisierte, objektorientierte, aber klassenlose Skriptsprache.

DER BEWEIS

Note that variables may hold values that have different data types:

Variable	Explanation	Example
String	A sequence of text known as a string. To signify that the value is a string, you must enclose it in quote marks.	`let myVariable = 'Bob';`
Number	A number. Numbers don't have quotes around them.	`let myVariable = 10;`
Boolean	A True/False value. The words `true` and `false` are special keywords in JS, and don't need quotes.	`let myVariable = true;`
Array	A structure that allows you to store multiple values in one single reference.	`let myVariable = [1,'Bob','Steve',10];` Refer to each member of the array like this: `myVariable[0]`, `myVariable[1]`, etc.
Object	Basically, anything. Everything in JavaScript is an object, and can be stored in a variable. Keep this in mind as you learn.	`let myVariable = document.querySelector('h1');` All of the above examples too.

WIESO HAT JS DANN EINEN SO SCHLECHTEN RUF?



*[JS ist] eine **dynamisch typisierte**, objektorientierte, aber klassenlose Skriptsprache.*

AUTOMATISCHE TYPKONVERTIERUNG WTF'S

```
 1 let i = 1;
 2
 3 // some code
 4 i = i + ""; // automatisch number -> string
 5 i + 1;
 6 i - 1;
 7
 8 let j = "1";
 9 j++;
10
11 let k = "1";
12 k += 1;
13
14 [1,5,20,10].sort()
```

Solange man keinen Quatsch mit Zahlen und Strings macht funktioniert das aber meistens ganz gut.

caveat: Vergleiche immer mit === bzw. !== machen, dadurch wird auch der Typ auf Gleichheit geprüft!

MANUELLE TYPKONVERTIERUNG

```
1 const toBoolean = !!123
2
3 const toString = 123 + ""
4
5 const toInt = parseInt('123', 10)
6 const toFloat = parseFloat('12.34')
7 const toNumber = Number('123')
8 // Vorsicht vor NaN!
```

TRUTHINESS

Völlig gültiger JS Code:

```
1 const text = 'blub';
2 if ( text ) {
3     // code
4 }
```

WAS IST TRUTHY?

Alles, was nicht falsy ist.

ALSO:

```
// Falsy
false, 0, '', null, undefined, NaN

// Jeder andere Wert (!) ist truthy, z.B.:
[], {}, 1, '0', 'null' etc.
```

EXERCISE TIME

Step 0: devdocs.io im browser aufrufen

```
git clone https://github.com/muhkuhxy/js-schulung

code js-schulung

# Datei src/01-first-steps.js bearbeiten
# src/examples/01-js-basics.js enthält syntax Beispiele

npm run serve # Visuelles Feedback
# alternativ
npx vue-cli-service test:unit --watch # unit tests
```

OBJECTS, METHODS AND SUCH

functions können auf `this` zugreifen:

```
1 const myObject = {
2    name: 'Horst',
3    sayHello: function() {
4       return "Hi, I'm " + this.name
5    }
6 }
7
8 myObject.sayHello()
9 // -> Hi, I'm Horst
```

Was `this` ist, hängt aber davon ab, wie die Methode aufgerufen wird

```
 1 function thisTest() {
 2     return this;
 3 }
 4
 5 thisTest()
 6 // -> undefined bzw. window Objekt
 7
 8
 9 const myObject = {
10     thisTest: thisTest
11 }
12
13 myObject.thisTest()
14 // -> myObject
```

```
function thisTest() { return this }

const myObject = {
    thisTest: thisTest,
    wrapperFunction: function() {
        return thisTest()
    }
}

myObject.thisTest()
// -> myObject

myObject.wrapperFunction()
// -> undefined bzw. window
```

```
function thisTest() { return this }

const myObject = {
    thisTest: thisTest,
    wrapperFunction: function() {
        return this.thisTest()
    }
}

myObject.thisTest()
// -> myObject

myObject.wrapperFunction()
// -> myObject
```

Arrow functions übernehmen this vom lexikalischen Kontext

```
1 const thisTest = () => { return this }
2
3 const myObject = {
4    thisTest: thisTest
5 }
6
7 myObject.thisTest()
8 // -> undefined bzw. window
```

Funktionen mit denen `this` fest gezurrt werden kann

```
 1 function thisTest() { return this }
 2
 3 const anotherObject = { name: 'anotherObject' }
 4
 5 thisTest.call(anotherObject, 'here', 'can', 'be', 'args')
 6 // -> anotherObject
 7
 8
 9 const somethingElse = { name: 'somethingElse' }
10
11 const boundFunction = thisTest.bind(somethingElse)
12
13 boundFunction()
14 // -> somethingElse
```

OBACHT BEI VERWENDUNG VON THIS

Nebeninfo: Object keys sind *immer* strings

```
1 const myObject = {
2    123: 'hello'
3 }
4
5 myObject['123']
6 // -> hello
```

Am besten von vorneherein nur strings als keys benutzen.
Definitiv keine anderen Objekte, Arrays etc.!

YES, JAVASCRIPT IS A LISP

`function` in `object`: OK!

```
1 const myObject = {
2     myFunction: function() {
3         // ...
4     }
5 }
```

`function` als Parameter?

`function` als Parameter: OK!

```
1  function myMap( array, mapper ) {
2     const result = []
3     for (const x of array) {
4        const mapped = mapper( x )
5        result.push( mapped )
6     }
7     return result
8  }
9
10 function square( x ) { return x * x }
11 myMap( [ 1, 2, 3 ], square )
12 // -> [ 1, 4, 9 ]
```

`function` als Rückgabewert?

`function` als Rückgabewert: OK!

```
1 function repeater( text ) {
2     return function( times ) {
3         return text.repeat( times )
4     }
5 }
6
7 const helloRepeater = repeater( 'hello' )
8 helloRepeater( 10 )
9 // -> 'hellohellohellohellohellohellohellohellohellohello'
```

FUNCTIONS SIND AUCH NUR WERTE

EXERCISE TIME

Datei 02-objects-and-functions.js bearbeiten

NEUERUNGEN VON ES5 UND ES6

- let/const
- modules
- arrow functions
- desctructuring
- object property shorthand
- method properties
- rest/spread operator
- template strings
- async/await
- default parameter values

Außerdem:

- Promises
- Map/Set Typen
- Classes
- Getters/Setters
- Symbols
- Generators

OBJECT PROPERTY SHORTHAND

Vorher

```
1 function foo() { ... }
2
3 const adviser = {
4     foo: foo
5 }
```

Nachher

```
1 function foo() { ... }
2
3 const adviser = {
4     foo
5 }
```

METHOD PROPERTIES

Vorher

```
1 const adviser = {
2     foo: function() {
3         ...
4     }
5 }
```

Nachher

```
1 const adviser = {
2     foo() {
3         ...
4     }
5 }
```

DESTRUCTURING

Vorher

```
1 const adviser = {
2    name: 'Adam Adviser',
3    clients: [
4       { name: 'Lady Brittany' }
5    ]
6 }
7
8 const adviserName = adviser.name;
9 const adviserClientName = adviser.clients[ 0 ].name
```

Nachher

```
1 const { name, clients } = adviser
2
3 const [ firstClient, secondClient ] = clients
```

Klappt auch bei function parameters.

Vorher

```
1 function adviser( names ) {
2     const firstName = names[ 0 ]
3     const lastName = names[ 1 ]
4     ...
5 }
```

Nachher

```
1 function adviser( [ firstName, lastName ] ) {
2     ...
3 }
```

VUE STORE BEISPIEL

Vorher

```
1 const actions = {
2     doSomething( context ) {
3         context.commit( 'myMutation' )
4     }
5 }
```

Nachher

```
1 const actions = {
2     doSomething( { commit } ) {
3         commit( 'myMutation' )
4     }
5 }
```

Umbenennung möglich

```
1 // client ist im aktuellen scope schon vergeben
2 const client = {}
3
4 const apiResult = {
5    client: ...
6 }
7
8 const { client: clientResult } = apiResult
9 // clientResult definiert
```

Beliebig Schachtelbar

```
1 const adviser = {
2    name: 'Adam Adviser',
3    clients: [
4       { name: 'Lady Brittany' }
5    ]
6 }
7
8 const { name, clients: [ first ] } = adviser
9 // name und first definiert, clients aber nichts
```

MODULES

Named Exports

```
// file: library.js

function foo() {
    // ...
}

function bar() {
    // ...
}

export {
    foo,
    bar
}
```

Named Imports

```
1  // file: app.js
2
3  import { foo, bar } from './library'
4
5  foo()
6  bar()
7
8  // oder
9
10 import * as lib from './library'
11
12 lib.foo()
13 lib.bar()
```

Named Exports: Direktroute

```
1 // file: library.js
2
3 export function foo() { ... }
4
5 export function bar() { ... }
6
7 export const ANSWER = 42
```

Import wie gehabt

Default Export

```
1 // file: library.js
2
3 export default function veryImportantFunction() { ... }
4
5
6
7 // file: app.js
8
9 import nameCanBeDifferentForDefaultExports from './library'
```

SPREAD

Vorher

```
function oldVarArgFunction( first, second ) {
   const third = arguments[2]
   const rest =
      Array.prototype.slice.call(arguments, 2)
   rest.map( ... )
}

myVarArgFunction( 1, 2, 3, 4 )
// in der function ist third = 3
// rest = [3,4]
```

Nachher

```
function myEs6VarArgFunction( first, second, ...rest ) {
   // rest ist ein array
   rest.map( ... )
}
```

In Verbindung mit destructuring

```
1 const clients = api.getClients( ... )
2
3 const [ first, second, ...rest ] = clients
4 // rest ist ein array mit den restlichen werten
```

Bei objects werden die verbliebenen keys gespreadet

```
const resource = {
   main: ...,
   spouse: ....,
   furtherInformation: ...,
   anotherKey: ...
}

const { main, spouse, ...foo } = resource
// foo = {
//    furtherInformation,
//    anotherKey
// }
```

Funktioniert auch andersrum

```
1 const foo = [ 1, 2, 3 ]
2 const bar = [ 4, 5, 6 ]
3
4 const qux = [ ...foo, ...bar ]
5 // qux = [1, 2, 3, 4, 5, 6]
```

Auch bei objects

```
1  // in Component.vue
2  updateItem( { commit, actions, state },
3     { index: 3, furtherInformation: 'text' } )
4
5  // in store.js
6  async updateItem( { commit, ...context },
7     { index, ...changes } ) {
8
9     commit( 'setFields', {
10       _itemsKey: 'pensions',
11       _index: index,
12       ...changes } );
13
14    savePensions( context );
15 },
```

DEFAULT PARAMETER VALUES

Vorher

```
function greet( name ) {
    if( name == null ) {
        ...
    }
}
```

Nachher

```
function greet( name = 'Adviser' ) {
    return `Hello, ${name}`
}
```

TEMPLATE STRINGS

Vorher

```
const error = 'Error occured while performing ' + action +
    ' against URL ' + url + ' resulting in status code ' +
    statusCode

// syntax fehler
const multiline = 'foo
    bar'
```

Nachher

```
// multiline ist kein syntax fehler
const error = `Error occured while performing ${action}
    against URL ${url} resulting in status code ${statusCode}
```

ARROW FUNCTIONS

Vorher

```
1 [ 1, 2, 3 ].map( function( x ) { return x * x } )
```

Nachher

```
 1 // function weg, dafür => hinter parameter liste
 2
 3 [ 1, 2, 3 ].map( ( x ) => { return x * x } )
 4
 5 // bei genau einem parameter können () weg
 6
 7 [ 1, 2, 3 ].map( x => { return x * x } )
 8
 9 // wenn body genau ein statement ist können {} weg
10 // und return ist implizit
11
12 [ 1, 2, 3 ].map( x => x * x )
```

ASYNC/AWAIT

Vorher

```
1 loadPensions( { commit }, { sessionId } ) {
2    api.loadPensions().then( result => {
3       // success, do something with result
4    }, error => {
5       // handle error
6    } ).then( ... )
7 }
```

Nachher

```
1 async loadPensions( { commit }, { sessionId } ) {
2    const pensions = await api.loadPensions( sessionId );
3    ...
4 }
```

EXERCISE TIME

Durch die bisher bearbeiteten Dateien gehen und nach Möglichkeiten suchen, die Syntax zu verkürzen

eslint Kommentare am Anfang der Dateien entfernen, die verhindern, dass eslint selbst Vorschläge macht

VUE

BEGRIFFLICHKEITEN

vue	aixigo
component	control / widget

Grundgerüst Single File Component

```
1  // MyFirstComponent.vue
2  <template>
3     <div>
4        <!-- html -->
5     </div>
6  </template>
7
8  <script>
9     import './MyFirstComponent.scss'
10    // js code
11    export default {
12       // vue component object
13    }
14 </script>
```

Declarative Rendering

```
<template>
    <div>
        {{ greeting }} {{ name }}
    </div>
</template>
<script>
    export default {
        props: {
            name: String
        },
        data() {
            return {
                greeting: 'Hello'
            }
        }
```

Verwendung in einer anderen Komponent

```
<template>
    <div>
        <Comp name="Horst" />
    </div>
</template>
<script>
    import Comp from '@/components/MyFirstComponent'
    export default {
        components: {
            Comp
        }
    }
</script>
```

Verwendung in einer anderen Komponent

```
<template>
    <div>
        <MyFirstComponent :name="someName" />
    </div>
</template>
<script>
    export default {
        ...
        data() {
            return {
                someName: 'Horst'
            }
        }
    }
</script>
```

Computed properties

```
<script>
    export default {
        props: {
            firstName: String,
            lastName: String
        }
        computed: {
            fullName() {
                return `${this.firstName} ${this.lastName}`
            }
        }
    }
</script>
```

Computed **properties**, nicht methods()

```
// template
<div>{{ fullName }}</div>

<script>
    export default {
        computed: {
            fullName() { ... },
            greeting() {
                return 'Hello, ' + this.fullName
            }
        }
    }
</script>
```

Jetzt aber methods

```
1  // template
2  <div>{{ fullName( 'Sir' ) }}</div>
3
4  <script>
5     export default {
6        methods: {
7           fullName( title ) {
8              return `${title} ${this.firstName} ${this.lastN
9           }
10       }
11    }
12 </script>
```

EXERCISE TIME

Datei components/CaseSelectionMVP.vue bearbeiten

VUE ROUTER

vue router kümmert sich um das mapping der URL auf die anzuzeigenden components

```
1  // router.js
2  const routes = [
3      {
4          path: '/',
5          name: 'home',
6          component: Home
7      },
8      {
9          path: '/case-selection-mvp',
10         name: 'ex4',
11         component: CaseSelectionMVP
12     },
13     {
14         path: '/exercise1',
15         name: 'ex1',
```

Navigation deklarativ

```
<!-- Home.vue -->
<router-link :to="{ name : 'ex1' }">
   Exercise 1 - JS Basics
</router-link>
<!-- ... -->
<router-link :to="{ name: 'ex4' }">
   Exercise 4 - Vue Basics
</router-link>
```

Navigation programmatisch

```
// template
<button @click="navigate()">Navigate</button>

// js code
export default {
    methods: {
        navigate() {
            this.$router.push( { name: 'routeName' } )
        }
    }
}
```

route params

```
routes: [
    // dynamic segments start with a colon
    { path: '/user/:id', component: User }
]

// in user component: this.$route.params.id
<div>your id is {{ $route.params.id }}</div>
```

EXERCISE TIME

Datei components/FactFindMVP.vue bearbeiten

VUEX STORE

In unserem MVP kümmert sich jede Komponente um ihre eigenen Daten.

Was aber, wenn dieselben Daten an verschiedenen Stellen im Komponentenbaum gebraucht werden?

- Daten nahe der Wurzel laden, per props nach unten reichen
- Datenhaltung zentralisieren

vuex macht letzteres

At the center of every Vuex application is the store. A "store" is basically a container that holds your application state.

https://vuex.vuejs.org/guide/

store example

```
1  // store.js
2  export default new Vuex.Store({
3     state: {
4       count: 0
5     },
6     mutations: {
7       increment( state ) {
8         state.count++
9       }
10    }
11 })
12
13 // in components
14 store.commit( 'increment' )
```

asynchrone actions

```
state: {
  clients: []
},
mutations: {
  clientsLoaded( state, clients ) {
    state.clients = clients
  }
},
actions: {
   async loadClients( { commit } ) {
      const clientsResult = await api.loadClients()
      commit( 'clientsLoaded', clientsResult )
   }
}
```

getters

```
state: {
    currentAdviser: {
        firstName: '...',
        lastName: '...'
    }
},
getters: {
    fullName( state ) {
        return `${state.currentAdviser.firstName} ...`
    }
}
```

Verwendung in components

```
1  import { mapGetters, mapActions } from 'vuex'
2
3  methods: {
4     myOwnAction() { ... },
5     ...mapActions( [ 'loadClients' ] )
6  },
7  computed: {
8     myComputedProp() { ... },
9     ...mapGetters( [ 'fullName' ] )
10 }
11
12 // this.fullName und this.loadClients() definiert
```

EXERCISE TIME

Datei views/CaseSelectionView.vue bearbeiten