Contents

Chapter 1

Algorithmen und Datenstrukturen in ES6+

1.1 Warum in ES6+

- ES is eating the world
    - meaningful understanding
    - I spent most of my professional life writing in JS and I think I know most about it

Und, um sicher zu sein, dass ich nicht versehentlich ein Beispiel für etwas erzeugt, was gar nicht der Problemfall ist, wird der Problemfall jetzt erzwungen:

asldf jlwefjlwkejf

1.2 Nicht behandelte Themen

- Promises - async/await - webAPI - Browser spezifische Unterschiede

Chapter 2

Entwicklungsumgebung

JS ist eine interpretierte Sprache. Sie läuft auf einer JS Engine. Die JS Engine läuft auf jeden Browser. Man kann direkt im Browser den Code ausführen.

Sobald man den Browser aber neu lädt ist unser Code weg. Besser ist es den Code entweder in nodejs auszuführen.

File speichern und mit node [filename] ausführen

Chapter 3

JavaScript Paradigmen

JS ist eine sog. Multiparadigmen Programmiersprache. JS ist imperativ, objektorientiert und funktional.

3.1 Objekte und Object-Orientiertes Programmieren

3.2 Funktionales Programmieren

Chapter 4

JavaScript Konstrukte

4.1 Entscheidungskonstrukte

In JS gibt es if Statements, ternary und switch statement. if if else if else if ternary operator

Return

statement vs. expression switch

4.2 Wiederholungen

for loop, while loop, symbol iterator; Generator; yields

4.3 Funktionen

4.4 Scope

4.5 Closures

Chapter 5

Datenstrukturen

5.1 Array

JS hat im Vergleich zu Java oder C/C++ nur sehr wenige Datenstrukturen. Eines ihrer wichtigsten Datenstrukturen ist der Array. Das Array werden wir später dazu benutzen, um alle anderen komplexeren Datenstrukturen zu implementieren. Der Unterschied zu JSs Arrays im Vergleich zu anderen Programmiersprachen ist, dass Arrays in JS keine fixe Länge haben. Durch das Hinzufügen und Entfernen von Elementen verändert sich die Array-Länge dynamisch mit. Bei der Initialisierung muss man dem Array dadurch auch keine bestimmte Länge mitgeben werden.

Arrays können in JS auf zwei Arten erstellt werden: Mit dem Array Konstruktor oder mit Array Literal:

```
1 var myArrayConstructor = new Array();
2 var myArrayLiteral = [];
```

Listing 5.1: Array Konstruktor

Der Array Konstruktur wird mit `new` eingeleitet und darauf folgt `Array()`. Beim Array Literal wird nur eine eckige Klammer `[]` benötigt. Beide Möglichkeiten erstellen einen Array. Jedoch wird angeraten zur Erstellung eines Arrays das Array Literal zu nehmen. Nicht nur ist er kürzer und auch schneller, er ist auch syntaktisch eindeutiger. Denn mit dem Array Konstruktur kann man auch die Länge des Arrays definieren als auch initialisieren. Die Syntax von beiden Konstrukten sind sich ähnlich, sodass es zu Verwirrung kommen kann, wenn man den Array mit Zahlen initialisiert:

```
1 var myArrayLength = new Array(3);
2 var myArrayInit = new Array(3,2,1);
```

Listing 5.2: Array Konstruktor

Die Länge eines Arrays wird als Zahl (hier: 3) in den Konstruktur reingeschrieben. Damit hat das Array in unserem Beispiel eine Länge von 3, die man mit `length`

überprüfen kann. Die (hier 3) Elemente sind noch `undefined`, da sie noch nicht initialisiert sind. Bei der Initialisierung gibt man die Elemente ebenfalls in den Konstruktur mit ein, jeweils getrennt durch einen Komma.

```
var myArrayLength = new Array(3);
console.log(myArrayLength);
// [undefined, undefined, undefined]
console.log(myArrayLenth.length);
// 3

var myArrayInit = new Array(3,2,1);
console.log(myArrayInit);
// [3, 2, 1]
console.log(myArrayInit.length);
// 3
```

Listing 5.3: Array Konstruktor

JS ist nicht static typed. D.h. ein Array kann Elemente nicht nur eines Typen gleichzeitig aufnehmen, sondern auch verschiedene. Ein Array in JS kann damit auch Zahlen, Strings, Bool und Objekte gleichzeitig aufnehmen:

```
var myArray = [1, "42", true, "hi", {"hello": "world"}];
```

Listing 5.4: Array Konstruktor

Intern werden die Elemente in einen String gecastet. Dadurch sind Arrays in JS langsamer als in anderen Sprachen. Um auf ein Array-Element zuzugreifen benutzen wir die eckige Klammer `[]`. Ein Array ist Index basiert und fängt mit `0` an. Um auf das zweite Element in einen Array zuzugreifen, schreiben wir also `myArray[1]`:

```
var myArray = [1, "42", true, "hi", {"hello": "world"}];
console.log(myArray[1]);
// "42"
```

Listing 5.5: Array Konstruktor

JS bieten viele Funktionen zur Manipulationen von Arrays an. Zum Hinzufügen am Ende wird `push()` benutzt. Um ein Element am Anfang des Arrays hinzuzufügen, wird `unshift()` verwendet:

```
var myArray = [1, 2, 3];
myArray.push(4);
console.log(myArray);
// [1, 2, 3, 4]

myArray.unshift(0);
console.log(myArray);
// [0, 1, 2, 3, 4]
```

Listing 5.6: Array Konstruktor

Für das Entfernen am Ende des Arrays gibt es `pop()`. Die Funktion `pop()` entfernt das letzte Element und gibt das entfernte Element zurück. Für das Entfernen am Anfang des Arrays verwendet man `shift()`.

```
var myArray = [0, 1, 2, 3, 4];
console.log(myArray.pop());
// 4
console.log(myArray);
// [0, 1, 2, 3]

console.log(myArray.shift());
// 0
console.log(myArray);
// [1, 2, 3]
```

Listing 5.7: Array Konstruktor

Um Elemente hinzuzufügen, zu ersetzen oder zu entfernen, die sich in der Mitte des Arrays befinden, verwendet man `splice()`. `splice()` nimmt als ersten Parameter den Index, an den das neue Element man hinzufügen will. Als zweiten Parameter wieviele Elemente danach ersetzt wird. Alle darauffolgenden Paramter die hinzuzufügenden Elemente.

```
var myArray = [1, 3, 4];
myArray.splice(1,0,2);
// adds a new element, 2, at index 1

myArray.splice(3,0,5,6,7);
// adds new elements, 5,6, and 7, at index 3

myArray.splice(5, 1);
// removes one element at index 5

myArray.splice(2, 3);
// removes 3 elements starting at index 2

myArray.splice(3, 1, 99);
// replaces one element at index 3 with the new element 99

myArray.splice(3, 2, 42);
// replaces 2 elements starting at index 3 with the new element 42

myArray.splice(999,0,2);
// if the index (first parameter) is larger than the array length,
    then it will just adds a new element to the end
```

Listing 5.8: Array Konstruktor

Die Funktionen `push()`, `unshift()`, `pop()`, `shift()`, `splice()`, verändert den originären Array. Es gibt auch Funktionen, die das Original nicht ändert, sondern stattdessen einen neuen Array zurückliefert.

Die `map((currentValue, index, array)=> {})` Methode geht durch die Elemente durch und wendet dabei eine ihr mitgegebene Methode auf einzeln jeden Element im Array an. Die ihr mitgegebene Methode nimmt als erstes Argument das

aktuelle Element auf. Das zweite Element ist das momentane Index im Array. Das dritte Argument das ursprüngliche Array.

```
var myArray = [1, 2, 3, 4];
function multiplyBy2 = item => item * 2;
var doubleArray = myArray.map(multiplyBy2);
console.log(doubleArray);
// [2, 4, 6, 8]
```

Listing 5.9: Array Konstruktor

`filter()` `reduce()` `flat()` `flatMap()` `every()` `some()` Die `slice()` Methode `concat()` Array kopieren.

Array manipulieren: Hinzufügen (push, unshift), entfernen (des letzten pop, des ersten shift), Zugriff die Elemente. Subarray erstellen: splice (mit mutation), slice (ohne). Kopieren von Arrays (slice(0)), usw..... dieser Abschnitt eng in Abstimmmung mit der Implementierung der komplexen Datenstrukturen machen.

- arrays are only objects- values are converted to Strings- is slower than in other JS- creating an Array with Array literal or constructor - go with the array literal because it's faster and is more clear. Array constructor can be weird....e.g.: new Array(10); // creates new array of size 10 new Array(10, 9,8,7,6); // creates new array with the elements 10, 9, 8, etc.- different objects, e.g. strings, int, bool can be in one array- accessing array elements- writing into array- can grow dynamicallly beyond the specified size- creating arrays from strings with split()- Assign array to another, shallow copy- How to copy array- searching for an value, indexOf(), return the first element, lastIndexOf(), returns the last element- exists value, includes()- return string from an array: toString(), join()- creating new arrays from existing ones: concat(), splice()- mutator functions: shift(), push(), pop(), - reordering reverse(), sort(); (sort doesn't work well with numbers!)- adding and removeing elements from the middle of an array: splice()- iterator functions: forEach, every(), some(), map(), filter(), reduce(), reduceRight(), flat(), flatMap()- multidimensional Array

5.2 Liste

asldfjlkewjf Lists are convenient if the order doesn't matter or if you don't have to search for a certain item implement a listlistSize (property) Number of elements in list pos (property) Current position in list length (property) Returns the number of elements in list clear (function) Clears all elements from list toString (function) Returns string representation of list getElement (function) Returns element at current position insert (function) Inserts new element after existing element append (function) Adds new element to end of list remove (function) Removes element from list front (function) Sets current position to first element of list end (function) Sets current position to last element of list prev (function) Moves current position back one element next (function) Moves current position forward one element currPos (function) Returns the current

position in list moveTo (function) Moves the current position to specified position

While writing also test your impelmentation with console.assert()

5.3 Stack

5.4 Queue

5.5 Dequeue

5.6 Priority Queue

5.7 LinkedList

5.8 Zirkulare LinkedList

5.9 Zweifach verknüpfte LinkedList

5.10 Dictionary

5.11 Hashing

5.12 HashMap

5.13 MapTree

5.14 LinkedMap

5.15 Sets

5.16 Binäre Bäume

5.17 Graphen

5.18 AVL Tree

Chapter 6

Algorithmen

Sortier Algorithmen und Such Algorithmen

6.1 Breitensuche

6.2 Tiefensuche

6.3 Bubble Sort

6.4 Selection Sort

6.5 Shellsort

6.6 Mergesort

6.7 Quicksort

6.8 Sequential Suche

6.9 Binäres Suchen

6.10 Suchen nach Minimum und Maximum

6.11 Rucksackproblem

6.12 Greedy Algorithm

```
Name.prototype = {
  methodName: function(params){
    var doubleQuoteString = "some text";
    var singleQuoteString = 'some more text';
    // this is a comment
    if(this.confirmed != null && typeof(this.confirmed) == Boolean
        && this.confirmed == true){
      document.createElement('h3');
      $('#system').append("This looks great");
      return false;
    } else {
      throw new Error;
    }
  }
}
```

Listing 6.1: My Javascript Example

laskd flwkejf wlekjf ew