Komplette Arrays mit Callback-Functions transformieren

forEach()

eine übergebene Funktion für jedes Element eines Arrays ausführen

für: Strings, Arrays

Iteration über ein Array(-Objekt)

(Array).forEach(function callback(item [, index [, array]]) { //… });

- callback Funktion, die auf jedes Element angewendet
- item der Wert des aktuellen Elements
- index der Index des aktuellen Elements
- array das Array durchlaufene Array

forEach() führt übergebene Funktion für jedes Element eines Arrays aus

hat keinen Rückgabewert

map()

mit map (dt: abbilden) lässt sich komplettes Array transformieren

- map() benötigt Ausgangs-Array und Funktion als Argument
- jedes einzelne Element des Ausgangs-Arrays wird mit angegebener Funktion transformiert und im Ergebnis-Array aufgefangen
- erzeugt Ergebnis-Array

map() verwenden, wenn alle Elemente eines Arrays transformiert werden sollen

filter()

- filter() benötigt Ausgangs-Array und Funktion als Argument
- erzeugt Ergebnis-Array
- filter() erstellt neues Array mit Elementen, die einen über Funktion durchlaufenen Test bestehen

reduce()

reduce() reduziert Array auf einzelnen Wert
benötigt dazu als Argument Funktion, die angibt, wie aus jeweils zwei Werten einer wird

Sonderfälle bei reduce()

wenn zu reduzierendes Array nur ein Element enthält, gibt reduce das enthaltene Element zurück
wenn Array komplett leer ist, wirft Code einen Fehler:
TypeError: Reduce of empty array with no initial value

können reduce() als zweiten Parameter Initialwert übergeben
bei klassischer Summenbildung sollte das Zahl 0 sein

Initialwert kann beliebigen Datentyp haben – sollte natürlich zur übergebenen Funktion passen

every()

gibt true zurück, falls alle Elemente der angegebenen Bedingung entsprechen

some()

gibt true zurück, falls einige Elemente einer angegebenen Bedingung entsprechen

find()

findet das erste Element, das angegebene Bedingung erfüllt

findIndex()

gibt Index des ersten Elements zurück, das angegebene Bedingung erfüllt

Referenz

Name	**Parameter der Callback-Funktion**
map, filter, every, some, find, findIndex, forEach	1. Value: Der Wert des aktuell betrachteten Elements aus dem Array. 2. Index: Ein laufender Index-Wert, beginnend bei 0. 3. Array: Das komplette Array, auf dem Methode aufgerufen wird.
reduce, reduceRight (wie reduce nur von rechts nach links)	1. previousValue: Ergebnis der letzten Ausführung des Callbacks. Beim ersten Aufruf ist es der Initialwert von reduce. Falls es keinen Initialwert gibt, verwendet reduce stattdessen den ersten Wert des Arrays. 2. Value: Der Wert des aktuell betrachteten Elements aus dem Array. 3. Index: Ein laufender Index-Wert, beginnend bei 0. 4. Array: Das komplette Array, auf dem reduce aufgerufen wird
sort	1. & 2. element: Elemente, die miteinander zu vergleichen sind

Objekte

vordefinierte, browserunabhängige Objekte

Objekte besitzen:

- Eigenschaften (objektgebundene Variablen)
- Methoden (objektgebundene Funktionen)

Eigenschaften und Methoden werden mit einem Punkt an das Objekt angehängt.

Math.PI
Math.sin()

browserunabhängigen Objekte von JavaScript:

- Object
- Array
- Boolean
- Date
- Function
- Math
- Number
- RegExp
- String
- JSON

Object

- oberstes Objekt in JavaScript
- alle anderen Objekte stammen von ihm ab
- zum Erzeugen eigener Objekte

Objekte erstellen

Objektdefinition
1. Konstruktor
2: Hinzufügen

Objekt in Literal-Schreibweise erstellen

```
let myObject = {
  // eigenschaft: wert,
  // eigenschaft: wert,
  // methode: function() {  alert(this.eigenschaft); }
};

console.log(myObject.key);
myObject. methode();
```

Namen mit Leerzeichen, Bindestrich oder reservierten Wörtern in "" oder ''

Schlüsselwort this bezieht sich immer auf das entsprechende Objekt

Leere Objekte

können Objekte wie Arrays auch komplett leer anlegen
Attribute werden später schrittweise ergänzt

```
let myObject = {};

myObject.eigenschaft = "Wert";
myObject['bezeichner'] = 'Wert';
```

Eigenschaft löschen:
delete myObject.eigenschaft

Objekt mit Konstruktor-Funktion erstellen:

```
let myObject = new Object();
myObject.eigenschaft = 'Wert'
myObject.methode = function(){
  alert(this.eigenschaft);
};
```

Objektschreibweise erlaubt, Eigenschaften zu gruppieren, die zusammengehörig ein Objekt beschreiben

innerhalb der geschweiften Klammern {} werden Eigenschaften (Properties) gruppiert
hierzu werden sogenannte Key/Value-Pairs (dt: Name/Wert-Paare) genutzt

Name der Eigenschaft (key): Wert der Eigenschaft (value)

Key/Value-Pairs innerhalb des Objektes durch Komma getrennt
nach letztem Key/Value-Pair ist Komma optional

```
{
key1: value1,
key2: value2
}
```

kann von außerhalb eines Objektes auf seine Property zugreifen:
myObject.eigenschaft // "Wert"

mit console.dir(myObject) oder console.table(myObject) tabellarische Auflistung der einzelnen Properties und ihrer Werte

Änderung von Eigenschaften durch Zuweisung:
myObject.eigenschaft = "Neuer Wert";

statt Punktschreibweise auch Property-Access möglich: myObject['bezeichner']

auch Zuweisung funktioniert mit Property-Access: myObject['bezeichner'] = "Neuer Wert";

ist sinnvoller, kürzere Punkt-Schreibweise zu verwenden
nur falls Key als string vorliegt oder zur Laufzeit ermittelt wird ist es nötig, Property-Access zu verwenden - z. B.:
const fieldToZeroOut = "price";
product[fieldToZeroOut] = 0;

im Beispielcode soll Feld auf Wert 0 gesetzt werden - welches Feld das ist, ergibt sich aber erst bei Programmausführung - Wert von fieldToZeroOut könnte aus Usereingabe stammen

Hinzufügen neuer Properties

Properties in JavaScript voll dynamisch
jederzeit möglich, zu bestehenden Objekten neue Properties hinzufügen, oder bestehende wieder zu entfernen

Objekt sortiert neue Eigenschaft nicht an spezieller Stelle ein
einzelne Eigenschaftsnamen (keys) haben untereinander keine Reihenfolge

bei Ausgabe sortiert console.log bzw. console.dir sie üblicherweise alphabetisch

typische Funktion in Tiefen des Shops könnte so aussehen:

```
let cartProduct = (customerLastname, customerFirstname, customerAddress, productName,
productPrice, productCategory, availableSince, numberInStock) => {
  // Statements
};
```

Funktionen mit derart vielen Parametern in Praxis unhandlich
führen auch zu schwer lesbarem Code und sind fehleranfällig (falsche Reihenfolge oder Anzahl der Argumente…) - **gilt als schlechte Praxis, Funktionen mit mehr als drei Argumenten zu befüttern!**

Lösung: Objekte ermöglichen, zusammengehörige Werte zu gruppieren und an Funktion zu übergeben

Schlüsselwort this

über this spricht man innerhalb von Objektmethode (oder Konstruktor) jeweilige Objektinstanz an - das aktuelle Objekt, »dieses« Objekt - dieses Objekt = Kontextobjekt

this = impliziter Parameter einer Funktion, die bei Aufruf mit Wert des Objekts belegt wird, auf dem sie aufgerufen wird

this referenziert das Objekt in dessen Kontext es aufgerufen wird

im globalen Raum bezieht sich this auf globalen Kontext (dort im Strikten Modus keine Referenz, also Rückgabe undefined , sonst window-Objekt als globales Objekt

Wann normale Funktionen und wann Pfeil-Funktionen nutzen:

- auf globaler Ebene im Code -> normale Funktionen
- in Call-Back-Funktionen -> Pfeil-Funktionen
- in Methoden von Objekten -> Pfeil-Funktionen
- allgemein immer beim Kontakt mit dem Keyword this -> Pfeil-Funktionen