ARRAY Iteration
forEach/map/filter/reduce

for (let elem in array)

Anstatt mit einer normalen For-Schleife, bei dem durch die Länge eines Arrays iteriert wird und der Schleifenzähler nach jedem Schleifendurchlauf um eins erhöht wird

for (let i = 0; i < array.length; i++) { Code }

kann durch alle Items eines Arrays auch mit einer einer For-Elem-In-Schleife durch iteriert werden.

for (let elem in array) { Code }

Da im Javascript alle Arrays auch Objekte sind, ist dies doe Anwendung der For-Key-Schleife, welche wir von Objekten kennen, auf Arrays, elem steht hier für die Indexnummer des einzelnen Arrays, die einzelnen Items werden mit array[elem] ausgelesen.

Da Strings auch Indexnummern haben, kann diese Schleife auch für Strings genutzt werden.

Im nachfolgenden Beispiel wird ein String durchlaufen, und das jeweilige Element wird in toUpperCase umgewandet, wenn das vorherige ein Leerzeichen war und gespeichert, ansonsten wird das ursprünglich Zeichen gespeichert.

```
let string = "In diesem string soll jedesmal nach einem leerzeichen ein großbuchstabe kommen!";
let newString = "";
for (let character in string) {
        if (string[character-1] === " ") {
                newString = newString + string[character].toUpperCase();
        } else {
                newString = newString + string[character];
        }
}

console.log(newString);
// gibt auf der Console aus:
```

In Diesem String Soll Jedesmal Nach Einem Leerzeichen Ein Großbuchstabe Kommen!

.forEach

Mit .forEach() können Arrays direkt durchlaufen werden. Der ursprüngliche Array wird nicht verändert, das Ergebnis muss auf einen neuen Array gespeichert werden

```
let array2 = [4768, 8542, 7341, 9742, 1284, 598];
let newArray = [];
array2.forEach(element => newArray.push(element - 1000));

console.log(newArray);
// gibt auf der Console aus:

(6) [3768, 7542, 6341, 8742, 284, -402]
```

Eine etwas ausführliche Schreibweise lautet wie folgt:

```
let array3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
array3.forEach(function (value, index, array) {
array[index] = value * 10 ;
})
```

Hier wird bei der Methode .forEach eine unbenannte Funktion aufgerufen, welche die Parameter value, index und array entgegen nimmt (immer in dieser Reihenfolge), dadurch ist es möglich den ursprünglichen Array durch Zuweisung neuer Werte für die Indexnummern zu verändern.

```
console.log(array3);
// gibt auf der Console aus:

(10) [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

.map()

Mit .map() wird ein Array direkt durchlaufen und Code ausgeführt, das Ergebnis muss in einem neuen Array zwischengespeichert werden oder im selben Array überschrieben werden.

```
let siebnerArray = [0,1,2,3,4,5,6,7,8,9,10]
siebnerArray=siebnerArray.map(value => value*7);
```

Hier wird der Array “siebnerArray“ mit .map() durchlaufen und das 7-Fache des Wertes berechnet und der ursprüngliche Array überschrieben.

```
console.log(siebnerArray);
// gibt auf der Console aus:

(11) [0, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
```

Die ausführliche Schreibweise lautet wie folgt:

```
let statusActive = [false, true, true, false, false, true, false];

statusActive.map(function (value, index, array) {
        if (!value) {
            array[index] = true;
        }
});
```

Hier wird der Array “statusActive“ durchlaufen, mit !value wird überprüft ob die einzelnen Werte der Items false sind. Wenn es so ist, dann wird im ursprünglichen Array der Wert des Items mit true überschrieben.
Dazu ist es nötig, der namenslosen funktion die Parameter value, index und array (in dieser Reihenfolge!) zu übergeben.
Achtung: Auch bei dieser Variante wird der ursprüngliche Array überschrieben!

```
console.log(statusActive);
```
// gbt auf der Console aus:

(7) [true, true, true, true, true, true, true]

.filter()

Mit .filter() wird ein Array durchlaufen, einzelnen Items werden überprüft und nur die Items, welche den Suchkriterien entsprechen, werden übernommen (also heraus gefiltert), der ursprüngliche Array wird nicht überschrieben! Das Ergebnis muss also in einem anderen Array zwischengespeichert werden.

```
let zahlenreihe = [7, 13, 14, 66, 21, 231, 77, 89, 2, 5];
let teilbarDurch7 = zahlenreihe.filter(value => !(value%7));
```

Hier wird der Array “zahlenreihe“ durchlaufen, alle überprüften Items, welche dem Prüfkriterium !(value%7) (das bedeutet, welches Item keinen Rest hat, wenn es durch 7 geteilt wird) entsprechen, werden in das neue Array “teilbarDurch7“ geschrieben.

```
console.log(teilbarDurch7);
```
// gibt auf der Console aus:

(5) [7, 14, 21, 231, 77]

Hinweis:

Bei der Methode .filter() wird nur der ursprüngliche Wert der Items zurück gegeben, der Wert kann nicht noch dabei bearbeitet werden, dies muss dann in einem extra Schritt erfolgen.

Die ausführliche Schreibweise nimmt bei der unbenannten Funktion den Parameter value entgegen.

```
let tiere = [`Katze`, `Eichhörnchen`, `Hund`, `Goldfisch`, `Amsel`, `Kuh`, `Vogelspinne`];

let haustiere = tiere.filter(function (value) {
	switch (value) {
		case `Katze`:
			return value;
		case `Hund`:
			return value;
		case `Goldfisch`:
			return value;
		case `Vogelspinne`:
			return value;
		default:
			break;
	}
})
```

Hier wird de Array “tiere“ durchlaufen, alle Items die einem der Prüfkriterien ensprechen, werden in das neue Array “haustiere“ geschrieben.

```
console.log(haustiere);
// gibt auf der Console aus:

(4) ["Hund", "Goldfisch", "Kaninchen", "Vogelspinne"]
```

.reduce()

Mit .reduce() werden alle Werte eines Arrays auf einen einzigen Wert reduziert, d.h. es gibt einen einzigen Wert zurück. Wird dieser Wert in einer Variablen gespeichert, so richtet sich der Typ der Variablen nach dem zurückgegebenen Wert. Soll der Wert also in einem leeren Array gespeichert werden und der Wert ist boolean, so wird die Variable zu einer boolean!

```
let werte = [4, true, "Alpha", undefined];
let stimmtNicht = [];

stimmtNicht = werte.reduce(function () {
	return false
});
```

Hier werden alle Items des Arrays “werte“ (egal welchen Typ sie vorher haben) auf den Wert false reduziert, der leere Array “stimmtNicht“ wird zu einer einfachen Variablen Typs Boolean.

```
console.log(stimmtNicht);
```
// gibt auf der Console aus:
false

```
console.log(typeof stimmtNicht);
```
// gibt auf der Console aus:
boolean

Bei der Methode .reduce() können der unbenannten Funktion folgende Parameter übergeben werden: vorherigerWert, aktuellerWert, index, array

Einsatzzweck kann zum Beispiel eine Summen- oder Durchschnittsbildung sein.

```
let zahlenZumAddieren = [5, 9, 12, 7];
let summe = zahlenZumAddieren.reduce(function (vorherigerWert, aktuellerWert, index, array) {
        return vorherigerWert + aktuellerWert;
});
```

Hier wird der Array “zahlenZumAddieren“ durchlaufen, der jeweils aktuelle Wert wird dem vorherigen Return-Wert zu addiert. Deshalb werden hier die Parameter vorherigerWert und aktuellerWert benötigt, die weiteren Parameter index und array sind überflüssig.
Am Ende wird ein Wert zurückgegeben, die Summe aller Items des ursprünglichen Arrays.

```
console.log(summe);
```
// gibt auf der Console aus:
33

```
let alleNoten = [2, 2, 1, 5, 3, 4, 2, 3, 3, 1, 3, 2];
let durchschnitt = (alleNoten.reduce(function (vor, akt) {
return (vor + akt);
})) / alleNoten.length;
```

Hier wird mit der Methode .reduce() der Durchschnittswert aller Werte des Arrays “alleNoten“ berechnet.

```
console.log(durchschnitt);
```
// gibt auf der Console aus:
2.5833333333333335

Questions? You‘re welcome!

holger.zerbe@digitalcareerinstitute.org