Weiterführende Aufgaben zu Javascript

Die folgenden Erklärungen und Aufgaben dienen für einen tieferen Einstieg in Javascript. Diese Aufgaben sind für den **28.10.2019** vorzubereiten. Die Aufgaben behandeln Datenstrukturen in Javascript und einige Besonderheiten, wie die Sichtbarkeit von Variablen oder Arrow Functions.

Arrays

Die Datenstruktur "Arrays" findet sich in fast jeder Programmiersprache und weicht auch in Javascript nicht von dem Konzept anderer Sprachen ab. Arrays werden auf folgende Weise initialisiert und verwendet: Dabei spielt es keine Rolle, ob einfache oder doppelte Anführungszeichen verwendet werden.

```
let cars = ['Audi', 'Tesla', 'BMW'];

console.log(cars[0]); // Audi
console.log(cars[cars.length - 1]); // BMW
```

Um mit Arrays zu arbeiten existieren eine ganze Reihe von Methoden. Die wichtigen Methoden sind dabei die Folgenden:

```
cars.forEach(function(item, index, array) {
        console.log(item, index)
        / *
          * Audi 0
          * Tesla 1
          * BMW 2
          */
});

cars.push('Mercedes'); // fügt 'Mercedes' am Ende des Arrays hinzu

cars.pop(); // entfernt 'Mercedes' wieder (immer das letzte Element)

cars.shift(); // entfernt immer das erste Element ('Audi')

cars.unshift('Audi') // fügt dieses Element am Anfang des Arrays hinzu

cars.indexOf('Tesla') // gibt den Index des Elements 'Tesla' zurück => 1

cars.splice(pos, 1); // Entfernt ein Element aus der Liste, "pos" ist ein Wert von indexOf()
```

Unabhängig von der ersten Methode kann man auch mit einer einfachen for- oder while-Schleife über ein Array iterieren. Eine ausführliche Dokumentation zu Arrays ist hier zu finden [1].

Objekte

Auch Javascript bietet die Möglichkeit objekt-orientiert zu arbeiten. Objekte lassen sich in Javascript auf verschiedene Weise erzeugen. Die am meisten verwendeten Arten sehen Sie in dem folgenden Beispiel:

```
let car = new Object();
car.manufacturer = "Tesla";
car.color = "red";
car.name = "Model 3";

let car2 = {};
car2.manufacturer = "Audi";
car2.color = "black";
car2.name = "A3";

let car3 = { manufacturer: "BMW", color: "blue" }; // …
```

In jedem dieser Fälle wird ein neues Objekt mit eigener Referenz erzeugt. Die ersten beiden Fälle sind dabei sogar komplett identisch. Bei der ersten Variante wird ein Objekt mit einer sogenannten Konstruktorfunktion erzeugt. Dabei handelt es sich lediglich um eine Funktion, welche ein Objekt zurück gibt. Hier handelt es sich um das Objekt "Object" das von Javascript selbst definiert wird und als universelle Grundlage dient. Der zweite Fall führt im Hintergrund den gleichen Code aus, allerdings mit einer kürzeren Schreibeweise. Außerdem erlaubt diese Schreibweise bereits die konkrete Definition von Objekteigenschaften, wie im letzten Beispiel zu sehen ist. Objekteigenschaften werden über Objekt.dieEigenschaft aufgerufen und sind case-sensitive.

Neben einfachen Eigenschaften lassen sich auch Methoden mit Hilfe von Funktionen definieren. Hierzu ein einfaches Beispiel:

```
let car = {
        manufacturer: "Tesla",
        color: "red",
        name: "Model 3",
        speed: 130, // Meilen pro Stunde
        kmh: function() {
                return 130 * 1.609344; // speed in kilometer pro stunde
        }
};

console.log(car.kmh()); // 209.21472
```

Auch in Javascript existiert die this Referenz auf das instanziierte Objekt. So können auch hier Eigenschaften und Methoden über this angesprochen werden. Wichtig dabei ist, das diese Referenz auch in Funktionen existiert. Dazu folgen nun zwei Beispiele: das erste Beispiel zeigt das

vorhergehende Beispiel mit der this Referenz und das zweite Beispiel die Erzeugung des Objekts mit einer Konstruktorfunktion.

```
let car = {
        // …
        speed: 130, // Meilen pro Stunde
        kmh: function() {
                return this.speed * 1.609344; // speed in kilometer pro stunde
        }
};
```

```
function Car(manufacturer, name, color, speed) {
        this.manufacturer = manufacturer;
        this.name = name;
        this.color = color;
        this.speed = speed;
};

let car = new Car("Tesla", "Model 3", "red");
```

Welche Art von Objekterzeugung Sie verwenden ist Ihnen überlassen. Weiterführende Informationen zu Objekten und deren Anwendung können Sie hier finden [2].

Arrow functions

Während der letzten Aufgaben wurden Funktionen in Javascript vorgestellt. Die dort gezeigte Syntax ist je nach Version von Javascript veraltet und hängt davon ab, ob sie in Node.js oder im Browser verwendet wird. Node.js unterstützt hingegen die neue Syntax, welche sich "Arrow Functions" nennt [3]. Als Beispiel hier nochmal eine Funktion mit der alten Syntax:

```
const rating = function(oldrating, newVote) {
        return (oldrating + newVote) / 2;
};
```

Diese Funktion lässt sich auch wie folgt schreiben:

```
const rating = (oldrating, newVote) => (oldrating + newVote) / 2;
```

Hierbei wird schnell deutlich, wie viel weniger Schreibaufwand damit verbunden ist. Die Schlüsselwörter function und return fallen weg und auch die Klammern für den Funktionsblock sind nicht mehr zwingend notwendig. Dies ist aber nur der Fall, wenn der Funktionsblock aus einer Zeile besteht. Bei diesem Beispiel sind hingegen wieder Klammern erforderlich:

```
const rating = (oldrating, newVote) =>  {
        let newRating = (oldrating + newVote) / 2;
        return newRating;
};
```

Auch die Klammern bei der Parameterliste sind nicht immer notwendig. Verwendet man nur einen einzigen Parameter, dann können diese ebenfalls weg gelassen werden. Hier eine Funktion die versucht aus einem vollständigen Namen (bspw. Frank Müller) den Vor- und Nachnamen zu ermitteln und diesen im Anschluss als Objekt zurück zu geben.

```
const parseName = name => {
        let split = name.split(" ");
        return {
                firstname: split[0],
                lastname: split[1]
        };
} ;
```

Sofern gar kein Parameter verwendet wird müssen die Klammern ohne Inhalt angegeben werden:

```
const noParams = () => console.log("Wir brauchen keine Parameter!");
```

Neben der kürzeren Syntax besteht ein wichtiger Unterschied im Zusammenhang mit dem this Schlüsselwort. Hier folgt noch einmal das Beispiel der Konstruktorfunktion, allerdings mit einem wichtigen Unterschied. Hier wurde eine zusätzliche Methode innerhalb der Funktion und als Teil des Objekts definiert. Dies funktioniert allerdings nicht, da die zweite Funktion die Eigenschaft speed nicht kennt.

```
function Car(manufacturer, name, color, speed) {
	this.manufacturer = manufacturer;
	this.name = name;
	this.color = color;
	this.speed = speed;
	this.kmh = function() {
		return this.speed * 1.609344; // speed in kilometer pro stunde
	};
};
```

Um mit der bisherigen Syntax zu arbeiten müsste folgende Änderung durchgeführt werden:

```
function Car(manufacturer, name, color, speed) {
	// …
	let that = this;
	this.kmh = function() {
		return that.speed * 1.609344; // speed in kilometer pro stunde
	};
};
```

Oder mit Arrow Functions:

```
function Car(manufacturer, name, color, speed) {
	// …
	this.kmh = () => {
		return this.speed * 1.609344; // speed in kilometer pro stunde
	};
};
```

Das Problem dabei ist, das jede Funktion ihren eigene Sichtbarkeit hat, was die this Referenz betrifft. Daher ist bei verschachtelten Funktionen der Trick über die that Variable notwendig gewesen, da wir dadurch eine Referenz auf die umschließende Funktion erhalten. Mit Arrow Functions ist dies nicht notwendig, da sie eine Eigenschaft besitzen, welche sich "lexical this" nennt. Dadurch kennt das this der Arrow Function auch die Werte der umgebenden Funktion. Diese können daraufhin in der Arrow Function verwendet werden.

Aufgaben

Aufgabe 1 - Array

Um das Programm von letzter Woche zu erweitern, speichern Sie den Namen der Bewertung, die Anzahl der abgegebenen Bewertungen und die zuletzt eingetragene Bewertung in einem Array ab. Zudem soll dann die Länge des Arrays und die zuletzt eingetragene Bewertung auf der Konsole ausgegeben werden.

Aufgabe 2 - Object

Um verschiedene Bewertungen adäquater zu speichern, wird das Programm durch Objekte erweitert. Legen Sie dafür ein Objekt mit dem Namen „ratings“ an. Dieses Objekt enthält dann den Namen der Bewertung, die Anzahl der Abstimmungen und das letzte Ergebnis. Geben Sie nun den Namen der Bewertung auf der Konsole aus. Wie könnte man nun mehrere Bewertungen mit unterschiedlichen Namen abspeichern?

Aufgabe 3 - this

Der Durchschnitt einer Bewertung soll nun mit einer Methode innerhalb des Objektes ermittelt werden. Dafür können, wenn nötig, weitere Eigenschaften angelegt werden. Zudem soll das Keyword „this“ verwendet werden, um auf die vorhandenen Eigenschaften des angesprochenen Objektes zuzugreifen. Geben Sie mit return das Ergebnis zurück. Geben Sie das Ergebnis auf der Konsole aus.

Aufgabe 4 – Arrow function

Die vorher erstellte Funktion soll nun durch eine „arrow function“ verkürzt werden. Überprüfen Sie durch die Konsolenausgabe, ob das Ergebnis unverändert ist.

Aufgabe 5 - Scope

Legen Sie eine globale Variable mit const hello = "hello" an. Deklarieren Sie zusätzlich zwei Funktionen mit selbstgewählten und aussagekräftigen Namen. Bei der ersten Funktion soll eine lokale Variable mit const world = " World" angelegt und mit der globalen Variable hello konkateniert werden. Bei der zweiten Funktion versuchen Sie nun die beiden eben erstellten Variablen hello und world in anderer Reihenfolge zu konkatenieren. Versuchen Sie Ergebnisse der Funktionen auf der Konsole auszugeben. Was fällt Ihnen dabei auf? Versuchen Sie es so abzuändern, dass das gewünschte Ergebnis erscheint.

Quellen und Links

[1] https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Array
[2] https://developer.mozilla.org/de/docs/Web/JavaScript/Guide/Mit_Objekten_arbeiten
[3] https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Functions/Pfeilfunktionen