Peter Wolf
<Software Developer & Advisor />

Javascript Grundkurs

TERMINAL

mkdir => neuer Ordner bauen
ls => auflistung daten in dem ordner
cd => ordner aufrufen
cd ./nameOrdner => zum ordner gehen
cd .. => einen schritt in der ordnerstruktur nach oben gehen

Daten Typen

Primitives
numbers
string
float
boolean

non-primititives
function
object
array

Loops und Iteration

For loop
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

map
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map
hinweis: map methode ist für die transformation von daten und nicht für das filtern.

ForEach
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

find
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

find returns an object after frist

filter
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

Hinweis: filter Return ist array

inlcudes
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/includes

Statements

if / else
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else

Hinweis: if/else läuft permanent. Auch wenn das ergebnis zutrifft, werden die anderen anfragen geprüft. Performance ist hierbei schlechter, wenn millionen von daten abgerufen werden.

switch
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch

Hinweis: break bricht die switch() an dem punkt ab, wo das Ergebnis erzielt wurde. Die anderen Abfragen werden ignoiert. Bessere performance da ab erfolgreichem Ergebnis, abgebrochen wird.

Peter Wolf
<Software Developer & Advisor />

SPREAD Operator

Der **Spread-Operator** in JavaScript (dargestellt durch drei Punkte `...`) ist eine sehr nützliche Syntax, die es erlaubt, **Elemente eines Arrays oder Eigenschaften eines Objekts zu "entpacken"**, um sie an eine andere Stelle zu kopieren, kombinieren oder weiterzuverarbeiten.

Durch die … notierung, wird eine „shallow Copy“ der Datei erstellt.
Diese kann dann mit neuen parametern ergänzt werden.

Spread auf Objekten

const user = {username: „Peter“}

um das user object mit dem age – key zu erweitern schreibt man folgendes:

const user2 = {...user, age: 45}

somti ist user2 dann {username: „Peter, age: 45}

Peter Wolf
<Software Developer & Advisor />

Funktionen

In JavaScript gibt es mehrere Arten von Funktionen, die je nach Anwendungsfall eingesetzt werden können. Hier sind die häufigsten Funktionsarten:

1. Normale (deklarierte) Funktionen

```
function greet(name) {
    return `Hallo, ${name}!`;
}
```

- Werden mit dem Schlüsselwort function deklariert.
- Haben einen eigenen Namen und können vor ihrer Definition aufgerufen werden (Hoisting).

2. Anonyme Funktionen

```
const greet = function(name) {
    return `Hallo, ${name}!`;
};
```

- Haben keinen Namen.
- Werden häufig in Variablen gespeichert oder als Argumente übergeben.

3. Arrow-Funktionen (Pfeilfunktionen)

```
const greet = (name) => `Hallo, ${name}!`;
```

- Kürzere Syntax für Funktionen.
- Vererben this vom umgebenden Kontext (kein eigenes this).
- Ideal für Callback- und Kurzfunktionen

4. Methoden

```
const person = {
    greet(name) {
        return `Hallo, ${name}!`;
    }
};
```

- Funktionen, die als Methoden eines Objekts definiert werden.
- Werden direkt auf dem Objekt aufgerufen.

5. Konstruktor-Funktionen

```
function Person(name) {
    this.name = name;
    this.greet = function() {
        return `Hallo, ${this.name}!`;
    };
}

const person = new Person('Max');
```

- Werden mit new aufgerufen.
- Dienen der Erstellung von Objekten.

6. Generator-Funktionen

```
function* numberGenerator() {
    let i = 0;
    while (true) {
        yield i++;
    }
}

const gen = numberGenerator();
console.log(gen.next().value); // 0
console.log(gen.next().value); // 1
```

- Verwenden das function*-Schlüsselwort.
- Geben mit yield Werte nacheinander zurück

7. Asynchrone Funktionen

```
async function fetchData() {
    const response = await fetch('https://api.example.com');
    return await response.json();
}
```

- Verwenden das async-Schlüsselwort.
- Arbeiten mit await für asynchrone Operationen.

8. Callback-Funktionen

```
function doTask(callback) {
    console.log("Aufgabe wird ausgeführt...");
    callback();
}

doTask(() => console.log("Callback wird aufgerufen!"));
```

- Werden als Argumente an andere Funktionen übergeben.
- Häufig verwendet bei asynchronen oder ereignisbasierten Aufgaben.

9. Selbstaufrufende Funktionen (IIFE - Immediately Invoked Function Expressions)

```
(function() {
    console.log("Diese Funktion ruft sich sofort auf!");
})();
```

- Direkt nach ihrer Definition aufgerufen.
- Häufig verwendet, um einen eigenen Scope zu schaffen.

Peter Wolf
<Software Developer & Advisor />

Fetch-Methode

Die fetch()-Methode ist eine native JavaScript-Funktion, die verwendet wird, um asynchrone HTTP-Anfragen an einen Server zu senden. Sie ermöglicht das Abrufen von Daten von einem Server (z. B. APIs) und ist eine modernere Alternative zu XMLHttpRequest.

Die fetch()-Methode basiert auf Promises, was sie lesbarer und einfacher zu verwenden macht, insbesondere im Vergleich zu älteren Ansätzen.

fetch(URL).then(response => console.log(response))

die Ausgabe ist folgendes:

Hierbei ist es lediglich wichtig, dass das property ok mit TRUE matched.

Falls nicht, ist ein Fehler beim ziehen der Daten aufgetreten.

Peter Wolf
<Software Developer & Advisor />

Events in JavaScript

In JavaScript sind Events Aktionen oder Vorkommnisse, die während der Interaktion mit einer Webseite auftreten, wie das Klicken auf einen Button, das Bewegen der Maus, das Laden einer Seite oder das Drücken einer Taste. Sie ermöglichen die Interaktivität von Webseiten.

Wichtige Eigenschaften:

1. **Event Listener**: Ein Mechanismus, der auf ein bestimmtes Ereignis wartet und darauf reagiert.
    - Beispiel: addEventListener("click", function) fügt einem Element einen Listener für Klicks hinzu.

2. **Event Typen:**
    - Mausereignisse: click, dblclick, mouseover, mouseout, mousemove
    - Tastaturereignisse: keydown, keyup, keypress
    - Formularereignisse: submit, focus, blur, change
    - Fensterereignisse: load, resize, scroll, unload

Beispiel:

```
const button = document.querySelector('button');
button.addEventListener('click', (event) => {
    alert('Button wurde geklickt!');
    console.log(event); // Details zum Ereignis
});
```

Peter Wolf
<Software Developer & Advisor />

REST VS SOAP

https://aws.amazon.com/compare/the-difference-between-soap-rest/

Server Dialog