Datenstrukturen: Map und Set in JavaScript

- **Lernziele:**
  - Wiederholung: Arrays
  - Einführung in `Map` und `Set`
  - Unterschiede und Gemeinsamkeiten
  - Wichtige Methoden und Beispiele
  - Praktische Beispiele und Übungen

Wiederholung: Arrays

- **Was ist ein Array?**
  - Eine geordnete Liste von Werten, zugänglich über nummerische Indizes.
  - Kann beliebige Datentypen speichern.

```
let fruechte = ['Apfel', 'Banane']; // Ein einfaches Array von Früchten
console.log(fruechte[0]); // Ausgabe: Apfel
fruechte.push('Orange'); // Eine weitere Frucht hinzufügen

// Über das Array iterieren und jedes Element ausgeben
for (let i = 0; i < fruechte.length; i++) {
  console.log(fruechte[i]);
}
```

- **Limitierungen von Arrays:**
  - Nur nummerische Indizes.
  - Nicht optimal für Schlüssel-Wert-Paare.

Einführung: Map und Set

- **Warum `Map` und `Set`?**
  - `Map`: Speicherung von Schlüssel-Wert-Paaren mit flexiblen Schlüsseln.
  - `Set`: Speicherung von eindeutigen Werten ohne Duplikate.
- **Wann sind sie nützlich?**
  - Bei der Speicherung komplexerer Datenstrukturen und Beziehungen.

Wichtige Methoden für `Map`

- `.set(key, value)` : Fügt ein Schlüssel-Wert-Paar hinzu oder aktualisiert es.
- `.get(key)` : Gibt den Wert für den angegebenen Schlüssel zurück.
- `.has(key)` : Überprüft, ob ein Schlüssel existiert.
- `.delete(key)` : Entfernt das Schlüssel-Wert-Paar.
- `.size` : Gibt die Anzahl der Einträge zurück.

```
let benutzerMap = new Map();
benutzerMap.set('name', 'Jacob');
benutzerMap.set('alter', 27);
console.log(benutzerMap.get('name')); // Ausgabe: Jacob
console.log(benutzerMap.has('alter')); // Ausgabe: true
benutzerMap.delete('alter'); // Entfernt den Eintrag 'alter'
console.log(benutzerMap.size); // Ausgabe: 1
```

Wichtige Methoden für `Set`

- `.add(value)`: Fügt einen Wert hinzu (nur, wenn er nicht bereits existiert).
- `.has(value)`: Überprüft, ob ein Wert im Set existiert.
- `.delete(value)`: Entfernt den Wert aus dem Set.
- `.size`: Gibt die Anzahl der Elemente zurück.

```
let farbenSet = new Set();
farbenSet.add('Rot');
farbenSet.add('Blau');
console.log(farbenSet.has('Blau')); // Ausgabe: true
farbenSet.delete('Blau');
console.log(farbenSet.size); // Ausgabe: 1
```

Praktisches Beispiel: `Map` (Produkte)

- **Anwendungsszenario:**
  - Speichern von Informationen über verschiedene Produkte.

```
let produkteMap = new Map();
produkteMap.set('Laptop', {preis: 1000, kategorie: 'Elektronik'});
produkteMap.set('Schreibtisch', {preis: 150, kategorie: 'Möbel'});

// Zugriff auf Produktinformationen
console.log(produkteMap.get('Laptop').preis); // Ausgabe: 1000
```

Praktisches Beispiel: `Set` (Eindeutige Besucher)

- **Anwendungsszenario:**
  - Speichern einer Liste von einzigartigen Besuchern einer Webseite.

```
let besucherSet = new Set();
besucherSet.add('Anna');
besucherSet.add('Tom');
besucherSet.add('Anna'); // Duplikat, wird ignoriert

console.log(besucherSet.size); // Ausgabe: 2
```

Unterschiede und Gemeinsamkeiten

- **Gemeinsamkeiten:**
  - Beide sind iterierbare Datenstrukturen.
  - Beide bieten Methoden wie `.size`.
- **Unterschiede:**
  - `Map` speichert Schlüssel-Wert-Paare.
  - `Set` speichert nur Werte.

`for...of` vs. `.forEach()`

- `for...of`
  - Iteriert über Werte von iterierbaren Datenstrukturen (z.B. Arrays, Sets, Maps).
  - **Vorteile:**
    - Einfach und lesbar.
    - Schleife kann mit `break` oder `continue` unterbrochen werden.
  - **Beispiel:**

```
let fruechte = ['Apfel', 'Banane', 'Orange'];
for (let frucht of fruechte) {
  console.log(frucht);
  if (frucht === 'Banane') break; // Beendet die Schleife
}
```

`.forEach()`

- **Wann verwenden?**
  - Iteriert über jedes Element einer Datenstruktur ohne Möglichkeit zur Unterbrechung.
  - Zugriff auf zusätzlichen Kontext wie den Index (bei Arrays).
- **Vorteile:**
  - Kürzer und funktionaler Stil.
- **Beispiel:**

```
let fruechte = ['Apfel', 'Banane', 'Orange'];
fruechte.forEach((frucht, index) => {
  console.log(`Frucht ${index}: ${frucht}`);
});
```