3.5 – JSA: Sets & Maps

Einleitung

In JavaScript stehen neben Arrays auch die **Set**- und **Map**-Objekte zur Verfügung, um strukturierte Daten zu speichern. Sie erlauben spezielle Formen der Datenspeicherung und -abfrage:

- **Set** speichert einzigartige Werte ohne Duplikate.
- **Map** speichert Daten in Form von **Schlüssel-Wert-Paaren**, wobei die Schlüssel jeden beliebigen Datentyp annehmen können.

Beide Strukturen sind iterierbar, bieten komfortable Methoden zur Verwaltung ihrer Inhalte und eignen sich gut für spezialisierte Aufgaben wie Lookups, Mengenoperationen oder semantisch eindeutige Zuordnungen.

---

1. Set

1.1 Grundlagen und Konstruktor

Beschreibung: Ein Set ist eine Sammlung einzigartiger Werte. Jeder Wert darf nur einmal enthalten sein. Die Einfüge-Reihenfolge wird gespeichert, ist aber semantisch meist irrelevant.

Syntax:

```
let s = new Set([iterable]);
```

Beispiele:

```
let emptySet = new Set();
console.log(emptySet.size); // 0

let petsSet = new Set(["cat", "dog", "cat"]);
console.log(petsSet);       // Set {"cat", "dog"}
console.log(petsSet.size);  // 2
```

1.2 `add()`

Fügt ein Element zum Set hinzu. Doppelte Einträge werden ignoriert.

```
petsSet.add("hamster");
petsSet.add("dog"); // wird ignoriert
```

1.3 `has()`

Prüft, ob ein Element enthalten ist.

```
petsSet.has("cat");    // true
petsSet.has("shark");  // false
```

1.4 `delete()` und `clear()`

```
petsSet.delete("dog"); // true
petsSet.delete("dog"); // false
petsSet.clear();        // leert das Set
```

1.5 Iteration über Set

a) `forEach()`

```
petsSet.forEach((value, key) => {
  console.log(`${key}:${value}`); // beide identisch
});
```

b) Iteratoren: `values()` und `keys()`

```
let it = petsSet.values();
let result = it.next();
while (!result.done) {
  console.log(result.value);
  result = it.next();
}
```

c) Spread

```
let array = [...petsSet];
```

---

2. Map

2.1 Grundlagen und Konstruktor

Beschreibung: Eine Map speichert Paare aus **Schlüssel und Wert**. Die Schlüssel **müssen eindeutig sein** – ein neuer Eintrag mit demselben Schlüssel überschreibt den vorherigen Wert. Im Gegensatz zu Objekten:

- dürfen die Schlüssel beliebige Typen sein (auch Objekte oder Funktionen),
- behält Map die Einfügereihenfolge der Einträge,
- ist direkt iterierbar (z. B. in `for...of`),
- speichert den Schlüssel nicht als String, sondern über Referenzvergleich (===).

Wichtig: Zwei verschiedene leere Objekte (`{}`) sind nicht gleich – sie referenzieren verschiedene Speicherorte. Daher ist folgender Code korrekt, aber verwirrend:

```
const map = new Map();
map.set({}, "a");
map.set({}, "b");
console.log(map.size); // 2 → weil die Objekte unterschiedlich sind
```

Syntax:

```
let m = new Map([[key1, value1], [key2, value2]]);
```

Beispiel:

```
let petsMap = new Map([
  ["cats", 1],
  ["dogs", 2],
  ["hamsters", 5]
]);
```

2.2 `set()` / `get()`

```
petsMap.set("cats", 3);
petsMap.get("cats"); // 3
```

2.3 `has()`, `delete()`, `clear()`

```
petsMap.has("dogs");    // true
petsMap.delete("cats"); // true
petsMap.clear();         // leert die Map
```

2.4 Iteration über Map

a) `forEach()`

```
petsMap.forEach((value, key) => {
  console.log(`${key} → ${value}`);
});
```

b) `keys()`, `values()`, `entries()`

```
for (let key of petsMap.keys()) console.log(key);
for (let val of petsMap.values()) console.log(val);
for (let [key, val] of petsMap.entries()) console.log(`${key}:${val}`);
```

c) Spread

```
let arr = [...petsMap];
// → [["cats", 3], ["dogs", 2], ...]
```

---

3. Vergleich: Map vs. Objekt

Merkmal	Map	Objekt
Schlüsseltyp	beliebig (inkl. Objekte, Funktionen etc.)	nur Strings oder Symbole
Reihenfolge	garantiert (Einfügereihenfolge)	historisch unzuverlässig
Iteration	`for...of`, `.entries()` etc.	`for...in`, `Object.keys()` etc.
Konstruktion	`new Map([...])`	Literale oder `Object.create()`
Spezialfunktionen	`set`, `get`, `has`, `entries`	direkte Zugriffssyntax
Performance	optimiert für häufige Hinzufügung/Entfernung	bei kleinen, flachen Objekten schneller

Wann sollte man Map statt Object verwenden?

- Wenn Schlüssel nicht-primitive Typen sind (z. B. DOM-Elemente, Objekte, Arrays).
- Wenn Einfügereihenfolge entscheidend ist.
- Wenn häufige dynamische Manipulation von Key-Value-Paaren erforderlich ist.
- Wenn Iteration mit klarer Kontrolle gebraucht wird.

Objekte sind vorzuziehen, wenn:

- feste, bekannte Properties modelliert werden,
- JSON-kompatible Strukturen gebraucht werden,
- eine einfache Datenstruktur ausreicht.

---

4. Vergleich: Set vs. Array

Merkmal	Set	Array
Werte mehrfach?	Nein	Ja
Ordnung	nicht relevant	wichtig (indexbasiert)
Zugriff	indirekt (Iterator)	direkt per Index

---

5. Übungsaufgaben

1. Erstelle ein Set mit fünf Tieren, darunter ein Duplikat. Gib die finale Anzahl aus.
2. Prüfe mit `has()` ob ein bestimmtes Tier enthalten ist.
3. Entferne gezielt ein Element und nutze `forEach` zur Ausgabe.
4. Erstelle eine Map mit Produktnamen als Schlüssel und Preisen als Werte.
5. Aktualisiere einen Preis und gib alle Einträge mit `for...of` aus.
6. Verwandle eine Map in ein Array und prüfe den Typ.
7. Erstelle ein Objekt und iteriere mit `Object.entries()`.

---

6. Micro-Projekt: Kategorien-Manager

Ziel: Verwaltung von Kategorien mit zugewiesenen Elementen (z. B. Tags, Labels).

Datenstruktur:

```
const categories = new Map();
categories.set("Tiere", new Set(["Hund", "Katze"]));
categories.set("Farben", new Set(["Blau", "Rot"]));
```

Funktionen:

- `addToCategory(cat, item)` – legt Kategorie an oder erweitert sie
- `removeFromCategory(cat, item)` – entfernt gezielt Eintrag
- `listCategory(cat)` – listet alle Einträge (z. B. per Spread)

Beispielimplementierung:

```
function addToCategory(cat, item) {
  if (!categories.has(cat)) {
    categories.set(cat, new Set());
  }
  categories.get(cat).add(item);
}

function listCategory(cat) {
  if (categories.has(cat)) {
    console.log([...categories.get(cat)].join(", "));
  }
}

addToCategory("Tiere", "Maus");
listCategory("Tiere"); // → Hund, Katze, Maus
```

Dieses Projekt bereitet auf modulare Datenverwaltung im Web vor – z. B. für dynamische Tag-Systeme, Filter oder Nutzergruppen.