3.2 – JSA: Strings in JavaScript

Hintergrund und Zielsetzung

Zeichenketten („Strings“) zählen zu den häufigsten Datentypen in der Webentwicklung. Sie werden für Nutzereingaben, Kommunikation mit APIs, HTML-Manipulation, Formate, Fehlermeldungen und vieles mehr verwendet. JavaScript bietet mit dem `String`-Objekt eine Vielzahl leistungsstarker Methoden zur Analyse, Umwandlung und Bearbeitung von Zeichenketten.

Dieses Kapitel bietet eine umfassende Übersicht über die wichtigsten Methoden des `String`-Prototyps, inklusive aller Parameter, Beispiele und typischer Anwendungsfehler. Alle Methoden sind direkt über den Wrapper `String.prototype` verfügbar, funktionieren aber auch bei primitiven Strings dank Autoboxing.

---

1. Grundlagen: String-Literale und Template-Literals

1.1 Literalformen

```
let a = "Doppelquote";
let b = 'Einzelquote';
let c = `Backticks`;
```

1.2 Template-Literals (Backticks mit ${})

```
let name = "Alex";
let waitTime = 5;
let order = `Hallo ${name}, dein Tisch ist in ${waitTime} Minuten bereit.`;
console.log(order);
```

> MDN: Template literals

1.3 Escaping

```
let quote = 'Er sagte: "Hallo!"';
let path = "C:\\Users\\Name";
```

---

2. Zugriff auf Zeichen

2.1 Zeichenindexierung (Bracket Notation)

```
let text = "Alien";
console.log(text[0]); // "A"
```

2.2 `charAt(index)`

```
text.charAt(0); // "A"
```

> MDN: charAt()

---

3. Länge und Autoboxing

3.1 `length`

```
let name = "Anna";
console.log(name.length); // 4
```

3.2 Autoboxing

Auch primitive Strings können Methoden aufrufen:

```
let primitive = "Hallo";
console.log(primitive.toUpperCase()); // "HALLO"
```

---

4. Kopieren von Teilstrings

4.1 `slice(start, end)`

- Negative Werte zählen vom Ende

```
let txt = "Javascript";
txt.slice(0, 4); // "Java"
txt.slice(-6);   // "script"
```

> MDN: slice()

4.2 `substring(start, end)`

- Negative Werte werden als 0 interpretiert

```
txt.substring(4, 10); // "script"
```

> MDN: substring()

4.3 `substr(start, length)`

- Nicht mehr empfohlen, aber noch verbreitet

```
txt.substr(0, 4); // "Java"
```

> MDN: substr()

---

5. Groß-/Kleinschreibung

5.1 `toUpperCase()` und `toLowerCase()`

```
"Hallo".toUpperCase(); // "HALLO"
"TEST".toLowerCase(); // "test"
```

> MDN: toUpperCase()

---

6. Suche in Strings

6.1 `includes(substring)`

```
"Hausnummer".includes("nummer"); // true
```

6.2 `indexOf(substring)` / `lastIndexOf(substring)`

```
"eins zwei eins".indexOf("eins"); // 0
"eins zwei eins".lastIndexOf("eins"); // 10
```

> MDN: includes()

---

7. Ersetzen von Text

7.1 `replace(search, replacement)` – ersetzt nur das erste Vorkommen

```
"abc abc".replace("abc", "xyz"); // "xyz abc"
```

7.2 `replaceAll(search, replacement)` – ersetzt alle Vorkommen

```
"abc abc".replaceAll("abc", "xyz"); // "xyz xyz"
```

> MDN: replaceAll()

---

8. Teilen von Strings

8.1 `split(separator)`

```
"1,2,3".split(","); // ["1", "2", "3"]
```

> MDN: split()

---

9. Padding von Strings

9.1 `padStart(length, pad)` / `padEnd(length, pad)`

```
"42".padStart(5, "0"); // "00042"
"42".padEnd(6, "."); // "42...."
```

> MDN: padStart()

---

10. Whitespace entfernen

10.1 `trim()`, `trimStart()`, `trimEnd()`

```
" Test ".trim();       // "Test"
" Test ".trimStart();  // "Test "
" Test ".trimEnd();    // " Test"
```

> MDN: trim()

---

11. String-Vergleiche

11.1 Vergleich mit `<`, `>` – alphabetisch (Unicode)

```
"abc" < "bcd"; // true
```

11.2 `localeCompare()` – sprachspezifisch

```
"Österreich".localeCompare("Sundsvall", "de"); // -1
```

> MDN: localeCompare()

---

12. Vergleich mit Python

Funktion	JavaScript	Python
Slicing	slice()	`text[0:4]`
Split	split()	`str.split()`
Replace	replaceAll	`str.replace()`
Pad	padStart	`str.zfill()`
Trim	trim()	`str.strip()`

---

13. Übungsaufgaben

1. Gib alle Zeichen eines Strings einzeln aus (per Schleife und `charAt`).
2. Extrahiere aus einem Pfad (z. B. "/users/profile.html") den Dateinamen.
3. Ersetze alle Kommas in einem String durch Semikola.
4. Nutze `padStart`, um eine einstellige Zahl auf zweistellig zu bringen.
5. Suche mit `includes`, ob ein Wort in einem Satz vorkommt.
6. Erstelle eine eigene Implementierung von `trim()` mit `slice()` und `while()`.
7. Wandle ein Array von Zahlen in einen CSV-String.
8. Extrahiere die Domain aus einer E-Mail-Adresse (z. B. "user@mail.com" → "mail.com").
9. Zähle, wie oft ein Wort in einem Satz vorkommt (Tipp: `split` + Schleife).
10. Erstelle eine Vergleichsfunktion für zwei Begriffe mit `localeCompare`.

---

14. Micro-Projekt: String-Analyse-Tool für Formulareingaben

Ziel

Entwicklung eines Tools zur dynamischen Verarbeitung, Prüfung und Transformation von Nutzereingaben aus einem Webformular (z. B. Name, E-Mail, Stadt).

Aufgaben

- Eingaben werden vor der Speicherung automatisch **getrimmt**, **kleingeschrieben** und validiert.
- Leerzeichen am Anfang/Ende werden entfernt (via `trim()`)
- Umlaute werden korrekt sortiert (`localeCompare`)
- Eine Live-Vorschau zeigt das normalisierte Format (z. B. `"Max " → "max"`)

Beispielcode (ohne HTML)

```
function normalizeName(input) {
  return input.trim().toLowerCase();
}

function isValidEmail(email) {
  return email.includes("@") && email.includes(".");
}

function preview(name, email) {
  console.log(`Name: ${normalizeName(name)}`);
  console.log(`E-Mail gültig: ${isValidEmail(email)}`);
}

preview(" Max ", "max@example.com");
```

Erweiterungsideen

- Alle Zeichenketten via `replace()` filtern (z. B. verbotene Zeichen)
- Nutzung von `padStart` zur Darstellung von Postleitzahlen
- Analyselänge von Textfeldern mit `length` + `split`