3.3 – JSA: Das `Date`-Objekt und Zeitverarbeitung

Hintergrund und Zielsetzung

Zeit- und Datumsangaben sind in nahezu allen Webanwendungen von zentraler Bedeutung: ob für ToDo-Listen, Buchungen, Kalender, Logdaten oder API-Kommunikation. JavaScript bietet dafür das eingebaute Objekt `Date`, das intern einen **Zeitstempel (timestamp)** speichert – also die Anzahl der Millisekunden seit dem **1. Januar 1970, 00:00:00 UTC**.

Ziel dieses Kapitels ist es, den Umgang mit `Date` vollständig zu verstehen – von der Erstellung über den Zugriff auf Zeitkomponenten bis zur Formatierung und Umrechnung. Besonderes Augenmerk liegt auf **lokaler Zeit vs. UTC**, den **Konstruktorvarianten**, den **`get*()`/`set*()`-Methoden** sowie den Formatmethoden wie `toISOString()` und `toLocaleString()`.

---

1. Grundlagen: Timestamps & Zeitursprung

Zeitbasis: Unix-Zeitstempel

```
let epoch = new Date(0);
console.log(epoch.toUTCString()); // Thu, 01 Jan 1970 00:00:00 GMT
```

Erzeugung per Timestamp (Millisekunden)

```
let ms = 1000 * 60 * 60 * 10; // 10 Stunden
let date = new Date(ms);
console.log(date.toUTCString()); // Thu, 01 Jan 1970 10:00:00 GMT
```

> MDN: Date()

---

2. Konstruktorvarianten

2.1 Ohne Argument → aktuelle Zeit

```
let now = new Date();
console.log(now.toLocaleString());
```

2.2 Timestamp als Argument (Millisekunden)

```
let t = Date.now(); // timestamp
let d = new Date(t);
```

2.3 ISO 8601 Zeichenkette

```
let d = new Date("2020-02-02T20:20:00.000Z");
```

2.4 Einzelkomponenten: Jahr, Monat, Tag, Stunde, Minute, Sekunde, ms

```
let d = new Date(2020, 6, 8, 10, 30, 0); // 8. Juli 2020, 10:30:00 (Achtung: Monat 6 = Juli)
```

---

3. Ausgabeformate und Timezones

3.1 `toLocaleString()` – Lokale Darstellung

```
let d = new Date("2020-07-08T10:20:00");
console.log(d.toLocaleString());
```

> MDN: toLocaleString()

3.2 `toISOString()` – UTC, ISO 8601

```
d.toISOString(); // z. B. "2020-07-08T08:20:00.000Z"
```

3.3 `toUTCString()` – Menschlich lesbare UTC-Ausgabe

```
d.toUTCString(); // z. B. "Wed, 08 Jul 2020 08:20:00 GMT"
```

3.4 `toLocaleDateString()` & `toLocaleTimeString()`

```
d.toLocaleDateString(); // z. B. "08.07.2020"
d.toLocaleTimeString(); // z. B. "10:20:00"
```

---

4. Einzelne Zeitkomponenten lesen

Methode	Bedeutung
`getFullYear()`	Jahr
`getMonth()`	Monat (0–11)
`getDate()`	Tag des Monats (1–31)
`getDay()`	Wochentag (0=So, 6=Sa)
`getHours()`	Stunde (0–23)
`getMinutes()`	Minuten (0–59)
`getSeconds()`	Sekunden (0–59)
`getMilliseconds()`	Millisekunden (0–999)

Beispiel

```
let d = new Date("2020-07-08T10:20:00");
console.log(d.getFullYear()); // 2020
console.log(d.getMonth());    // 6
console.log(d.getDate());     // 8
console.log(d.getDay());      // 3
```

---

5. Komponenten setzen

Alle `get*()`-Methoden existieren auch als `set*()`-Version:

```
d.setFullYear(2024);
d.setHours(12);
```

> Beachte: Auch UTC-Versionen existieren: `getUTCFullYear()`, `setUTCMonth()` usw.

---

6. Zeitunterschiede & Timestamps

6.1 `getTime()` / `valueOf()`

```
let t1 = new Date("2020-07-08T10:00:00");
let t2 = new Date("2020-07-09T10:00:00");
console.log(t2.getTime() - t1.getTime()); // 86400000 (1 Tag in ms)
```

6.2 Stoppuhr (Performance-Messung)

```
let start = Date.now();
for (let i = 0; i < 1e6; i++) {}
let end = Date.now();
console.log(`Benötigte Zeit: ${end - start} ms`);
```

---

7. String-Parsing & Problemfälle

7.1 ISO empfohlen

```
new Date("2020-03-02T10:00:00Z"); // UTC-Zeit
```

7.2 Inoffizielle Formate vermeiden:

```
new Date("3.2.2020"); // lokal, unzuverlässig
new Date("2020, 10:00"); // missverständlich
```

> MDN: Date parsing

---

8. Vergleich mit Python

Konzept	JavaScript	Python `datetime`
Now	`new Date()`	`datetime.now()`
Timestamp	`Date.now()`	`datetime.timestamp()`
ISO	`toISOString()`	`isoformat()`
Konstruktion	`new Date(...)`	`datetime(2020, 7, 8, 10)`
Differenz	`getTime()` diff	`timedelta`

---

9. Übungsaufgaben

1. Erstelle ein Datum für den 1. Januar 2000 um 12:00 Uhr.
2. Hole Jahr, Monat und Tag aus einem `Date`-Objekt.
3. Berechne die Differenz zwischen heute und dem 1.1.1970 in Tagen.

4. Baue eine Funktion `getDayName(date)` mit Wochentagsnamen.
5. Schreibe eine Funktion `formatDate(d)`, die `DD.MM.YYYY` zurückgibt.
6. Nutze `Date.now()` zur Messung der Zeitdauer eines Array-Sortierens.
7. Erstelle ein Date-Objekt mit einem ISO-String mit und ohne `Z`.
8. Erkläre den Unterschied zwischen `toISOString()` und `toLocaleString()`.

---

10. Micro-Projekt: Deadline-Manager

Ziel

Ein Mini-Tool zur Verwaltung und Anzeige von Aufgaben mit Deadlines im lokalen Zeitformat.

Anforderungen

- Eingabe: Aufgabenbeschreibung + ISO-Zeitstempel
- Ausgabe: Formatierte Zeitangabe mit verbleibender Zeit
- Automatische Umrechnung von UTC zu Lokalzeit

Beispielcode

```
function showTaskInfo(task, deadlineIso) {
  const deadline = new Date(deadlineIso);
  const now = new Date();
  const diffMs = deadline - now;
  const minsLeft = Math.floor(diffMs / 60000);
  console.log(`Aufgabe: ${task}`);
  console.log(`Deadline: ${deadline.toLocaleString()}`);
  console.log(`Verbleibende Minuten: ${minsLeft}`);
}

showTaskInfo("Abgabe Modul 3", "2024-06-01T15:00:00Z");
```

Erweiterungsideen

- Aufgabenliste als Array
- Farbe je nach Zeitrest (z. B. rot < 60 min)
- Option zum Setzen neuer Deadlines über Formulardaten

Hinweis: In modernen Webanwendungen wird oft mit Bibliotheken wie **Luxon** oder **date-fns** gearbeitet. Für die JSA-Prüfung sind aber ausschließlich die eingebauten Funktionen relevant.