



CODERS.BAY

# DATENSPEICHERUNG:

COOKIES – LOCALSTORAGE  
SESSIONSTORAGE - INDEXEDDB

- JS läuft aus Sicherheitsgründen in einer Sandbox = Programme können nur auf Ressourcen zurückgreifen können die ihnen der Browser explizit zur Verfügung stellt
- Dateisystem kann nicht verwendet werden
- Clientseitig – Daten werden nicht zum Server geschickt
- Probleme der vorhandenen Möglichkeiten
  - Datenmenge begrenzt
  - Nur string-Werte verarbeiten (Lösung: mit JSON arbeiten)
  - Mangel an Sicherheit -> Daten können ohne große Probleme ausgewertet werden
  - **(keine sensible Daten darin abspeichern!!)**

## COOKIES

**Nutzlast**  
max. 4KB

**max. Lebenszeit**  
praktisch unbegrenzt

**Geltungsbereich**  
Alle Browserfenster /  
Tabs

Haltbarkeitsdatum wird  
bei der Erzeugung des  
Cookies festgelegt

## SESSION STORAGE

**Nutzlast**  
5 bis 10MB

**max. Lebenszeit**  
bis Seite geschlossen  
wird

**Geltungsbereich**  
ein individuelles  
Browserfenster / Tab

Wird beim Schließen  
des Browserfensters  
automatisch gelöscht

## LOCAL STORAGE

**Nutzlast**  
5 bis 10MB

**max. Lebenszeit**  
praktisch unbegrenzt

**Geltungsbereich**  
Alle Browserfenster /  
Tabs

Wird nur von Javascript  
oder Löschen des  
Browser-Cache gelöscht

## INDEXEDDB

**Nutzlast**  
250MB

**max. Lebenszeit**  
praktisch unbegrenzt

**Geltungsbereich**  
Alle Browserfenster /  
Tabs

Wird nur von Javascript  
oder Löschen des  
Browser-Cache gelöscht

- Hinweis: keine Cookies von lokalen Seiten
- Key-Value Speicher
- Ohne Pfad setzt der Browser das Cookie immer für die aktuelle Seite
- **Anwendungsfall:** Login, Warenkorb
- Cookie in einer Funktion erstellen mit Ablaufdatum

```
// Cookies unter document.cookie erreichbar  
// neuen Wert zuweisen: Name des Cookies deklarieren und String setzen  
document.cookie = "meinCookie=Hier steht ein beliebiger Wert.";
```

```
function setCookie(cookieName, data, duration) {  
    // neues Objekt vom Typ Date  
    let date = new Date();  
    // gewünschtes Ablaufdatum festlegen  
    // setTime ermöglicht es, einen neuen Zeitpunkt vorzugeben  
    // getTime() = aktuelles Datum plus gewünschte Dauer in Millisekunden  
    // Da Dauer in Tagen sinnvoller => 24 h * 60 min * 60 sec * 1000 => Wert in Millisekunden  
    date.setTime(date.getTime() + (duration*24*60*60*1000));  
    // für gewünschtes Format toGMTString() Methode  
    // für Ablaufdatum expires vor Methode stellen  
    let expireDate = "expires=" + date.toGMTString();  
    document.cookie = cookieName + "=" + data + ";" + duration;  
}
```

```
function getCookie(cookieName) {
    cookieName += "=";
    /* um Inhalt des Cookie zu ermitteln = Eigenschaft document.cookie
     * um ev. vorhandene Umlaute oder Sonderzeichen richtig darzustellen Methode decodeURIComponent()
     * man erhält kompletten Cookie String mit Namen, Inhalt, Ablaufdatum und Pfadnamen falls vorhanden */
    let decCookie = decodeURIComponent(document.cookie);
    /* kompletten Cookie String in einer Zeichenkette
     * Methode split(';') zerteilt Zeichenkette und erzeugt ein Array dessen Felder die Inhalte
     * Der Bereiche zwischen den Semikolons enthalten */
    let arr = decCookie.split(";");

    for (let i = 0; i < arr.length; i++) {
        let data = arr[i];
        /* Für Überprüfung hilfreich, alle Leerzeichen, die eventuell am
         * Anfang des Array-Feldes vorhanden sind zu entfernen mit while und substring()-Methode */
        while (data.charAt(0) == ' ') {
            data = data.substring(1);
        }
        /* Wenn Variable mit cookieName existiert => Name des Cookies
         * Überprüfung mit indexOf() Methode ob dieser Ausdruck zu Beginn der aktuellen
         * Zeichenkette steht */
        if(data.indexOf(cookieName) == 0) {
            /* Wenn ja, gewünschten Inhalt mit substring() Methode extrahieren
             * Da Name des Cookies nicht zurückgegeben werden soll, wird als Startpunkt cookieName.length gewählt */
            return data.substring(cookieName.length);
        }
    }
    return "";
}
```

- Der User gibt seinen Namen beim Laden der Seite ein z.B. MAX
- Der Name wird 180 Tage gespeichert
- Innerhalb dieser 180 Tage wird der User mit „Hallo MAX“ begrüßt

```
<body onload="checkCookie()">
<button type="button" onclick="setCookie('user','',-1)">Cookie
löschen</button>
<script>
  function setCookie(cookieName, data, duration) {
    let date = new Date();
    date.setTime(date.getTime() + (duration*24*60*60*1000));
    let expireDate = "expires=" + datum.toGMTString();
    document.cookie = cookieName + "=" + data + ";"
      + duration;
  }
  function getCookie(cookieName) {
    cookieName += "=";
    let decCookie = decodeURIComponent(document.cookie);
    let arr = decCookie.split(";");
    for (let i = 0; i < arr.length; i++) {
      let data = arr[i];
      while (inhalt.charAt(0) == ' ') {
        data = data.substring(1);
      }
    }
  }
}
```

```
    if(data.indexOf(cookieName) == 0) {
      return data.substring(cookieName.length);
    }
  }
  return "";
}
function checkCookie() {
  let user = getCookie(„user“);
  if(user != "") {
    alert("Hallo " + user + "!");
  } else {
    user = prompt("Gib deinen Namen ein:");
    alert("Hallo " + user + "!");
    if (user != "" && user != null) {
      setCookie("user", user, 180);
    }
  }
}
</script>
</body>
```

- Key-Value Speicher
- Existiert nur im Tab im aktuellen Browser
- Einfache Anwendung
- **Anwendungsfall:** Sprachauswahl speichern

```
// setzen eines Key-Value-Pairs
sessionStorage.setItem("key", "value");

// gespeicherte Daten holen
sessionStorage.getItem("key");

// löschen der Daten
sessionStorage.removeItem("key");
```

# LOCAL STORAGE

- Key-Value Speicher der die Werte als String speichert
- Einfache Anwendung
- **Anwendungsfall:** Userbezogene Daten speichern

```
// setzen eines Key-Value-Pairs  
localStorage.setItem("key", "value");  
  
// gespeicherte Daten holen  
localStorage.getItem("key");  
  
// löschen der Daten  
localStorage.removeItem("key");
```



# DATEN MIT LOCAL STORAGE SPEICHERN

- LocalStorage ist ein Objekt, deshalb kann für Ausgabe von allen Elementen eine for-in-Schleife verwendet werden

```
<body>
<p>Bezeichnung: <input id="key" value=""></p>
<p>Inhalt: <input id="value" value=""></p>
<button type="button" onclick="save()">Eingabe</button>
<button type="button" onclick="print()">Werte ausgeben</button>
<button type="button" onclick="delete()">Werte im localStorage
löschen</button>
<p id="outputP"></p>
<script>
  function save() {
    let key = key.value;
    let value = value.value;
    // Wert wird im localStorage gespeichert
    localStorage.setItem(key, value);
  }
  function print() {
    let output = "";
    let i = 0;
    for (let key in localStorage) {
      // mit getItem() wird der Wert aus
      // dem localStorage abgerufen
      output += key + ": " + localStorage.getItem(key);
      output += "<br>";
    }
  }
  function delete() {
    // Werte werden gelöscht
    localStorage.clear();
  }
</script>
</body>
```

```
i++;
// Um weitere Attribute und Methoden des Objektes nicht
// auszugeben = zusätzlicher Zähler der die Länge des
// Eintrages überprüft
if(i == localStorage.length) {
  break;
}
}
outputP.innerHTML = output;
}
function delete() {
  // Werte werden gelöscht
  localStorage.clear();
}
</script>
</body>
```

Bezeichnung:

Inhalt:

ersten Wert abspeichern: im localStorage  
 length: null  
 clear: null  
 getItem: null  
 key: null  
 removeItem: null  
 setItem: null

- Kann Objekte und Key-Value-Pairs speichern
- Asynchron (im Gegensatz zu Session und Local Storage)
  - Durch ein Event getriggert
- SQL Statements möglich
- **Anwendungsfall:** große Menge von Objekten speichern - WebApps

```
// Kundendaten
const customerData = [
  { cnumber: "17844", lastname: "Maier", firstname: "Laura", email: "laura@maier.at" },
  { cnumber: "17845", lastname: "Huber", firstname: "Franz", email: "franz@huber.at" }
];

// gespeicherte Daten holen
const customerDB = "Customer DB";

// Datenbank Öffnen, 2 Parameter ist die Version der DB
let request = indexedDB.open(customerDB, 1);

request.onerror = function(event) {
  // Error behandeln
}

request.onupgradeneeded = function(event) {
  // Datenbank holen
  let db = event.target.result;

  //ObjectStore erstellen wo die Daten gespeichert werden, cnumber ist der primary-key
  let objectStore = db.createObjectStore("Customers", {keyPath: "cnumber" });

  // Werte speichern
  for(let i in customerData) {
    objectStore.add(customerData[i]);
  }
}
```



## AUFGABE



1. Erstelle eine Seite, die beim Laden überprüft, ob im LocalStorage bereits ein Name und die Lieblingsfarbe des Besuchers vorhanden sind.
  - Trifft dies zu, wird der User begrüßt:
    - Hallo „Name des Users“, schön dass du da bist
    - Und der body bekommt die Farbe der Lieblingsfarbe (gebt am besten Hexadeximale ein)
2. Ist das nicht der Fall, soll das Programm mit zwei prompt-Befehlen die entsprechenden Werte abfragen. Speichere diese daraufhin in zwei LocalStorage-Keys.
3. Mit Klick auf einen Button können die Daten des Users geändert werden
4. Mit Klick auf einen anderen Button werden die Daten gelöscht



CODERS.BAY

**ENDE**

QUELLE: JAVASCRIPT  
PROGRAMMIEREN FÜR EINSTEIGER  
ISBN: 978-3-96645-016-4

[MEDIUM.COM](https://medium.com)  
[DEVELOPER.MOZILLA.ORG](https://developer.mozilla.org)