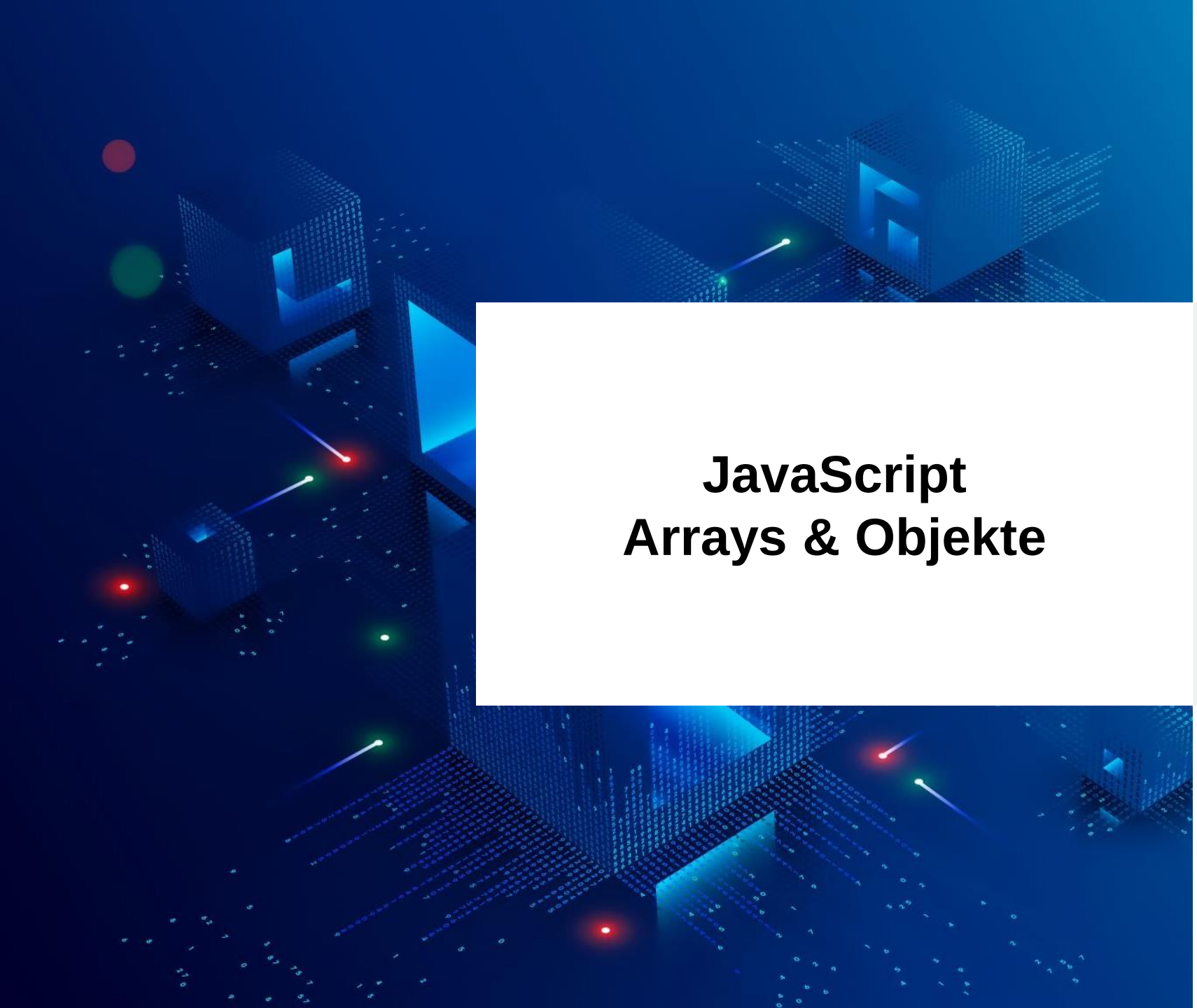


JavaScript
Arrays & Objekte

Übersicht

- Arrays
  - Grundlagen
  - Zugriff
  - Befüllen und verändern
  - Mehrdimensionale Arrays
- Array Methoden
  - Allgemeine Methoden
  - Iterationen
- Objekte
  - Grundlagen
  - Verändern & Erweitern
  - Objekte und Arrays kombinieren

Arrays

Arrays

- Ein Array speichert eine Liste von mehreren beliebigen Werten.
- Die Werte eines Arrays können sich auch im Typ unterscheiden, das sollte man aber vermeiden.
- Ein Array wird mit Eckigen Klammern [ ] geschrieben, die einzelnen Werte mit Kommas , getrennt.

```
1 const grades = [1, 2, 3, 4, 5];
2
3 console.log(grades);
4
5 const mixed = [1, "A", true];
6
7 console.log(mixed);
```

Array Zugriff

- Mit .length lässt sich die Länge eines Arrays bestimmen.
- Mit [i] kann man auf eine bestimmten Stelle (Index) im Array zugreifen, wobei die Erste Stelle den Index **0** hat.
- Der Wert von .length ist also immer um 1 höher als die letzte Stelle, auf die man zugreifen kann.
- Greift man auf eine Stelle zu, die außerhalb des Wertebereichs liegt, so erhält man den Wert undefined. Dies ist in JS kein Laufzeitfehler!

```
1 const values = ["A", "B", "C"];
2
3 console.log(values.length); // => 3
4
5 console.log(values[0]);     // => "A"
6
7 console.log(values[3]);     // => undefined
```

Arrays befüllen / verändern

- Mit [i] und einer Zuweisung kann man den Wert der gewünschten Stelle überschreiben oder befüllen.
- Mit der Funktion .**push(wert),** die man auf die Variable anwendet, lässt sich ein Wert am Ende des Arrays anhängen, mit .**unshift(wert)** am Anfang.
- **Vorsicht**: Das geht auch dann, wenn das Array selbst als const deklariert wurde!
- const schützt die Variable nur davor, mit einem gänzlich neuen Array überschrieben zu werden.

```
1 const values = ["A", "B", "C"];
2
3 values[1] = "X";      // => ["A", "X", "C"]
4 values.push("Y");     // => ["A", "X", "C", "Y"]
5 values.unshift("Z"); // => ["Z", "A", "X", "C", "Y"]
6
7 console.log(values);
```

Mehrdimensionale Arrays

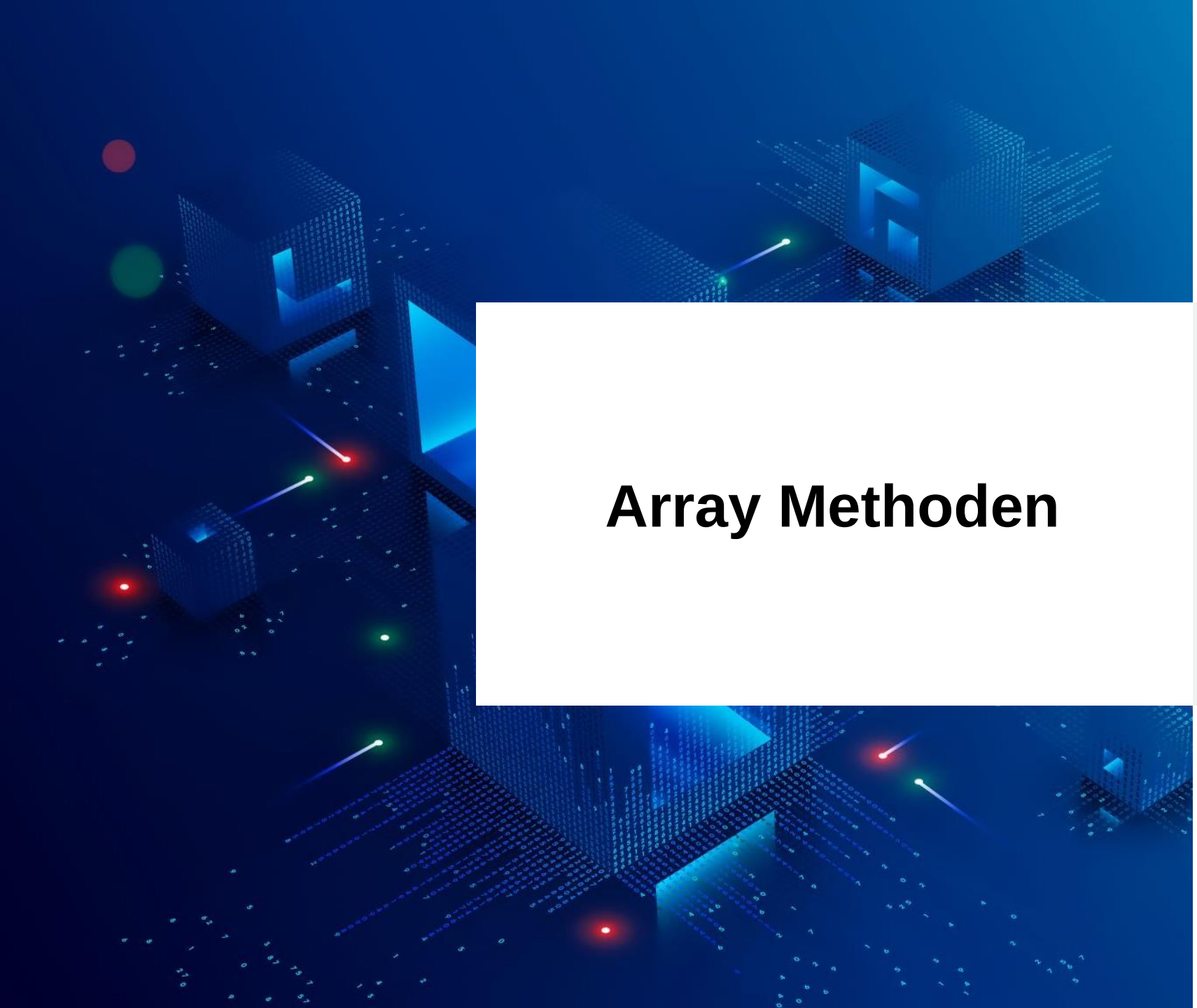
- Arrays können auch Arrays beinhalten, man spricht dann von Mehrdimensionalen Arrays.
- Ein Wert in einem Zweidimensionalen Array könnte beispielsweise die Zelle einer Tabelle oder einen Feld auf einem Schachbrett repräsentieren.
- Mit der ersten eckigen Klammer beim Zugriff erhält man ein inneres Array, mit der zweiten einen Wert im inneren Array. Bei mehr als zwei Dimensionen folgen weitere eckige Klammern...

```
1 const field = [
2     ["A", "B", "C"],
3     ["D", "E", "F"],
4     ["G", "H", "I"],
5 ];
6 console.log(field[1]);    // => ["D","E","F"]
7 console.log(field[1][0]); // => "D"
```

Übung zu Arrays „Wer bezahlt die Rechnung“

- Schreibe eine Funktion „whosPaying(names)“, die einen zufälligen Namen aus einer Liste von Namen auswählt. Die ausgewählte Person muss die Rechnung für das Essen aller bezahlen.
- Wichtig: Das Ergebnis sollte aus der Funktion zurückgegeben werden und über console.log geloggt werden.
- Das Ergebnis sollte genau mit dem Beispielergebnis übereinstimmen, einschließlich Groß- und Kleinschreibung sowie Interpunktion.
- Beispiel-Eingabe: ["Angela", "Ben", "Jenny", "Michael", "Chloe"]
- Beispiel-Ausgabe: Michael bezahlt heute das Mittagessen!„

```
function whosPaying(names)
```



Array Methoden

Array Methoden

- Javascript bietet eine Vielzahl an Methoden um Arrays zu bearbeiten
- Beispiele dafür:
  - **length:** Gibt die Anzahl der Elemente in einem Array zurück.
  - **toString():** Konvertiert alle Elemente eines Arrays zu einer Zeichenkettenrepräsentation und gibt diese als neuen String zurück.
  - **pop():** Entfernt das letzte Element aus einem Array und gibt dieses Element zurück.
  - **push():** Fügt ein oder mehrere Elemente am Ende eines Arrays hinzu und gibt die neue Länge des Arrays zurück.
  - **shift():** Entfernt das erste Element aus einem Array und gibt dieses Element zurück.
  - **unshift():** Fügt ein oder mehrere Elemente am Anfang eines Arrays hinzu und gibt die neue Länge des Arrays zurück.
  - **join():** Erstellt eine Zeichenkette, indem alle Elemente eines Arrays mit einem angegebenen Trennzeichen verbunden werden.
  - **delete():** Löscht ein Element an einem bestimmten Index in einem Array, hinterlässt jedoch eine Lücke an dieser Stelle.
  - **concat():** Kombiniert zwei oder mehr Arrays, indem es ein neues Array erstellt, das die Elemente der kombinierten Arrays enthält.
  - **flat():** Erstellt ein neues Array mit allen Subarrays, die rekursiv in eine angegebene Tiefe abgeflacht sind.
  - **splice():** Ändert den Inhalt eines Arrays durch Hinzufügen oder Entfernen von Elementen an einer bestimmten Position.
  - **slice():** Gibt eine flache Kopie eines Teils eines Arrays zurück, ohne das Original-Array zu ändern.
- Ressourcen: 7. Javascript/Codebeispiele/Arrays/arrayMethods.js

```
let fruits = ['Apfel', 'Banane', 'Orange'];
let removedFruit = fruits.shift();
console.log(removedFruit); // Ausgabe: Apfel
console.log(fruits); // Ausgabe: ['Banane', 'Orange']
```

```
let fruits = ['Apfel', 'Banane'];
let newLength = fruits.push('Orange');
console.log(newLength); // Ausgabe: 3
console.log(fruits); // Ausgabe: ['Apfel', 'Banane', 'Orange']
```

Array Iterationen

- Um über jedes Element eines Arrays zu iterieren können folgende built-in Methoden verwendet werden:
  - **forEach():** Iteriert über jedes Element im Array und führt eine bereitgestellte Funktion aus.
  - **Map():** Erstellt ein neues Array, indem eine bereitgestellte Funktion auf jedes Element im Array angewendet wird.
  - **Some():** Überprüft, ob mindestens ein Element im Array eine bestimmte Bedingung erfüllt.
  - **Every():** Überprüft, ob alle Elemente im Array eine bestimmte Bedingung erfüllen.
  - **Filter():** Filtert Elemente basierend auf einer bestimmten Bedingung und erstellt ein neues Array.
  - **Reduce():** Reduziert das Array auf einen einzelnen Wert durch Anwendung einer akkumulierten Funktion auf jedes Element.
  - **IndexOf():** Gibt den Index des ersten Vorkommens eines Elements im Array zurück
  - **Find():** Gibt das erste Element im Array zurück, das eine bestimmte Bedingung erfüllt.
  - **Values():** Durchläuft das Array und gibt ein neues Array mit den Werten zurück.
  - **Entries():** Gibt ein neues Array mit den Schlüssel-Wert-Paaren des Arrays zurück.
  - **Keys():** Durchläuft das Array und gibt ein neues Array mit den Schlüsseln (Indizes) zurück.

- Ressourcen: 7. Javascript/Codebeispiele/Arrays/arrayIterations.js

Array Sortierung

- Um ein Array alphabetisch zu sortieren kann die Funktion **sort()** verwendet werden
- Um Arrays numerisch zu sortieren, verwendet man ein eine **Vergleichsfunktion**
  - Wenn das Ergebnis negativ ist, wird a vor b sortiert.
  - Wenn das Ergebnis positiv ist, wird b vor a sortiert.
  - Wenn das Ergebnis 0 ist, gibt es keine Änderungen in der Sortierreihenfolge der beiden Werte.
- Die Funktion reverse() kehrt das Array um (Elemente werden von hinten nach vorne ausgelesen)
- Ressourcen: 7. Javascript/Codebeispiele/Arrays/arrayIterations.js

```
const array2 = [10, 2, 5, 1, 20];
array2.sort((a, b) => a - b);
console.log(array2); // Output: [1, 2, 5, 10, 20]
```

Übung zu Array Methoden „Rangliste“

- Verwende die forEach-Methode, um jeden Teilnehmer und seine Punktzahl auszugeben.
- Verwende die map-Methode, um ein neues Array zu erstellen, das nur die Namen der Teilnehmer enthält.
- Verwende die filter-Methode, um ein neues Array zu erstellen, das nur die Teilnehmer enthält, die mehr als 100 Punkte haben.
- Verwende die sort-Methode, um die Liste der Teilnehmer nach ihren Punktzahlen zu sortieren (aufsteigend).
- Verwende die reverse-Methode, um die Liste in absteigender Reihenfolge zu sortieren.
- Gib die sortierte Rangliste aus, die sowohl den Namen als auch die Punktzahl jedes Teilnehmers enthält.

```
const participants = [
  { name: 'Alice', points: 120 },
  { name: 'Bob', points: 80 },
  { name: 'Charlie', points: 95 },
  { name: 'David', points: 110 },
  { name: 'Eva', points: 60 }
];
```

Objekte

Objekte Grundlagen

- Mit Geschwungenen Klammern { } kann man Objekte definieren, die Wertepaare speichern können.
- Darin gibt man mit Doppelpunkt **:** getrennt den Schlüssel (Key) und den Wert (Value) an
- Mehrere Wertepaare werden durch Komma **,** getrennt
- Die Werte können von beliebigem Typ sein!
- Zugriff auf einen Wert erfolgt mit Punkt **.** und dem jeweiligen Schlüssel (Key).
- Hat man den Schlüssel nur als String, kann man auch mit [ ] darauf zugreifen!

```
1 const car = {
2     brand: "Suzuki",
3     model: "Jimny",
4     horsePower: 102,
5 };
6 console.log(car.brand);     // => "Suzuki"
7 console.log(car["brand"]); // => "Suzuki"
```

Objekte verändern

- Greift man auf einen Wert mit **.** und dem Schlüssel (Key) auf das Objekt zu, so kann´man durch eine Zuweisung den gespeicherten Wert verändern.
- Benutzt man einen Schlüssel, der noch nicht im Objekt definiert war, so erweitert man das Objekt um diesen.
- Auch hier schützt das const nicht das Objekt vor Veränderung

```
1 const car = {
2     brand: "Suzuki",
3     model: "Jimny",
4     horsePower: 102,
5 };
6
7 car.model = "Ignis";
8 car.doors = 2;
9 console.log(car.model); // => "Ignis"
```

Objekte & Arrays kombinieren

- Objekte und Arrays lassen sich beliebig miteinander kombinieren
- ein Array kann als Wert in einem Objekt liegen, und ein Objekt kann in einem Array liegen.
- Auch Objekte können in Objekten liegen und Arrays in Arrays.
- So können theoretisch beliebig tief verschachtelte Strukturen entstehen.

```
const javaScript = {
    difficulty: "Easy",
    variableTypes: [
        "String",
        "Number",
        "Boolean",
        "Array",
        "Object",
    ],
    attributes: {
        isScriptLanguage: true,
        isTypeSafe: false,
    },
};

const languages = [
    {
        name: "JavaScript",
        difficulty: "Easy",
    },
    {
        name: "Java",
        difficulty: "Moderate",
    },
];
```

Funktionen innerhalb von Objekten

- Man kann sogar Funktionen in Objekten definieren
- dabei kann man Anonyme funktionen oder Arrow-Functions nutzen.
- So kann man beispielsweise Funktionen, die thematisch zusammen passen, gruppieren.
- Funktionen die innerhalb eines Objektes definiert sind, werden „Methoden“ genannt

```
const calc = {
    add: function (a, b) {
        return a + b;
    },
    max: (a, b) => {
        return a > b ? a : b;
    },
    pow: (a, b) => a ** b,
};

console.log(calc.add(1, 2)); // => 3

console.log(calc.max(2, 3)); // => 5

console.log(calc.pow(3, 2)); // => 9
```

Übungen zu Arrays & Objekte

1. Definiere die Wochentage in einem **Array**, lese vom Nutzer eine Zahl von 1-7 ein und gib den entsprechenden Wochentag aus dem Array aus.

2. Definiere ein **Objekt** mit diversen Daten eures lieblings Fahrzeugs (Auto, Motorrad, Fahrrad, ...), dabei sollten verschiedene Datentypen verwendet werden. Gebt die Daten mit einem Template Literal schön formatiert aus.

3. Definiere die abgebildete Tabelle als **zweidimensionales Array** und lass den Benutzer eine Zeilen und Spaltennummer angeben, gib den gewünschten Wert aus.

4. Lege die selbe Tabelle als **Array von Objekten** an, wobei die Spaltennamen die Schlüssel in den Objekten darstellen. Lass den Nutzer eine Zeilennummer und den Namen einer Spalte angeben, gib den gewünschten Wert aus.

Vorname	Nachname	Alter
Hans	Müller	22
Georg	Huber	37
Fritz	Mayr	19

Konstruktorfunktion

- Konstruktorfunktionen werden verwendet, um Objekte mit gleicher Struktur und Verhalten zu erstellen.
- Sie verwenden das 'new'-Schlüsselwort und 'this', um Eigenschaften und Methoden dem erstellten Objekt zuzuweisen.
- Konventionell sind Konstruktorfunktionen mit einem Anfangsbuchstaben in Großbuchstaben benannt.

```
// Definition einer einfachen Konstruktorfunktion für ein Auto
function Auto(marke, modell, baujahr) {
  this.marke = marke;
  this.modell = modell;
  this.baujahr = baujahr;
  this.getDetails = function() {
    return `Auto: ${this.marke} ${this.modell}, Baujahr ${this.baujahr}`;
  };
}

// Erzeugen von Instanzen der Auto-Klasse
let auto1 = new Auto("Toyota", "Corolla", 2020);
let auto2 = new Auto("Tesla", "Model 3", 2022);

// Aufruf der Methode getDetails für jedes Auto
console.log(auto1.getDetails());
console.log(auto2.getDetails());
```

Factory Pattern

- Das Fabrikmuster ermöglicht die Erstellung von Objekten ohne Verwendung des 'new'-Schlüsselworts
- Es verwendet eine Funktion, die ein Objekt mit Eigenschaften und möglicherweise Methoden zurückgibt
- Dieses Muster ist flexibler und erlaubt mehr Kontrolle über den Erstellungsprozess.
- Es erleichtert die Erstellung von Objekten mit unterschiedlichen Eigenschaftswerten basierend auf den übergebenen Parametern.

```
function createAuto(marke, modell, baujahr) {
  return {
    marke: marke,
    modell: modell,
    baujahr: baujahr,
    getDetails: function() {
      return `Auto: ${this.marke} ${this.modell}, Baujahr ${this.baujahr}`;
    }
  };
}
let auto = createAuto("Toyota", "Corolla", 2020);
```

Viel Erfolg beim Entwickeln!