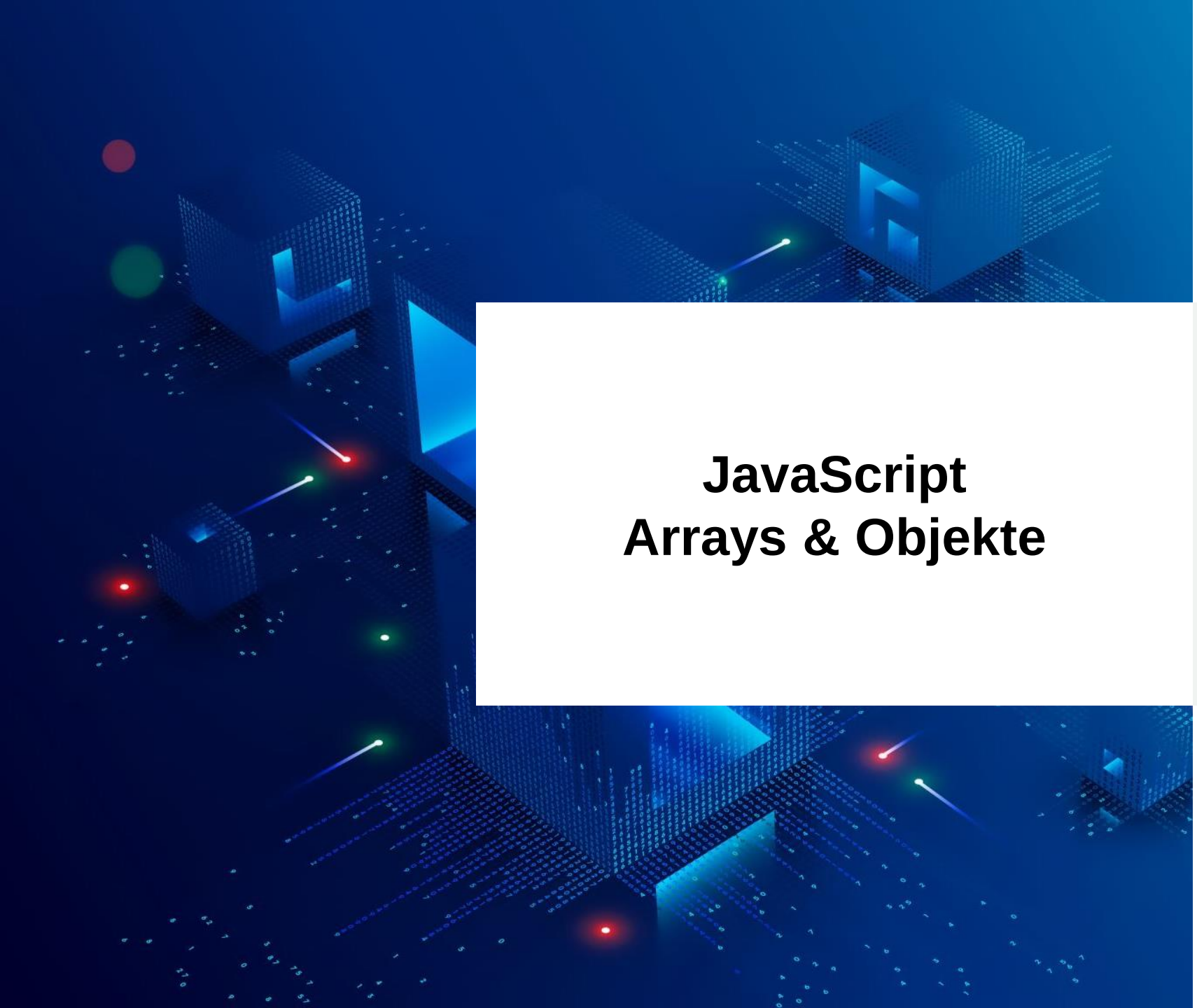

JavaScript
Arrays & Objekte

Übersicht

- Arrays
  - Grundlagen
  - Zugriff
  - Befüllen und verändern
  - Mehrdimensionale Arrays

- 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)
```

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!