Cheatsheet: Arrays und Objekte in JavaScript

JavaScript Array und Objekte	Beschreibung	Codebeispiel
Array-Deklaration	Arrays in JavaScript sind geordnet, was bedeutet, dass die Elemente in einer bestimmten Reihenfolge gespeichert werden.	`const fruits = ["apple", "banana", "cherry"];`

Array-Indizierung	Arrays sind nullbasiert, was bedeutet, dass das erste Element den Index 0 hat, das zweite den Index 1 und so weiter.	`const fruits = ["apple", "banana", "cherry"];`
		`const firstFruit = fruits[0]; // "apple"`
		`const secondFruit = fruits[1]; // "banana"`

Array-Länge	Die Length-Eigenschaft wird verwendet, um die Anzahl der Elemente in einem Array zu bestimmen.	`const fruits = ["apple", "banana", "cherry"];`
		`const numFruits = fruits.length; // 3`
		`console.log(numFruits);`

Änderbarkeit von Arrays	Arrays in JavaScript sind veränderbar, was bedeutet, dass Sie Elemente nach der Erstellung des Arrays ändern, hinzufügen oder entfernen können.	`const fruits = ["apple", "banana", "cherry"];`
		`fruits[2] = "strawberry"; // Modifying an element`
		`fruits[3] = "Kiwi"; // Adding an element`

push-Methode	Fügt ein oder mehrere Elemente am Ende eines Arrays hinzu.	`const fruits = ["apple", "banana"];`
		`fruits.push("orange", "strawberry");`
		`console.log(fruits)`

pop Methode	Entfernt das letzte Element aus einem Array und gibt es zurück.	`const fruits = ["apple", "banana", "orange"];`
		`const removedFruit = fruits.pop();`
		`console.log('Fruits are',fruits)`
		`console.log('Removed fruits are',removedFruit)`

Verschiebemethoden	Entfernt das erste Element aus einem Array und gibt es zurück.	`Removes the first element from an array and returns it.`

unshift-Methode	Fügt ein oder mehrere Elemente am Anfang eines Arrays hinzu und gibt es zurück.	`const fruits = ["banana", "orange"];`
		`fruits.unshift("apple", "strawberry");`
		`console.log(fruits);`

splice-Methode	Ändert den Inhalt eines Arrays, indem Elemente an einer bestimmten Position entfernt, ersetzt oder hinzugefügt werden.	`const fruits = ["apple", "banana", "cherry"];`
		`fruits.splice(1, 1, "grape"); // Replace the second element with "grape"`
		`console.log(fruits)`

concat-Methode	Die concat-Methode in JavaScript-Arrays kombiniert Arrays in Folge und erstellt ein neues Array, das die Elemente der ursprünglichen Arrays in der Reihenfolge enthält, in der sie zusammengefügt wurden.	`const fruits = ["apple", "banana"];`
		`const additionalFruits = ["orange", "strawberry"];`
		`const combinedFruits = fruits.concat(additionalFruits);`
		`console.log('combinedFruits are', combinedFruits)`

slice-Methode	Gibt eine flache Kopie eines Teils eines Arrays in ein neues Array zurück.	`const fruits = ["apple", "banana", "cherry", "orange"];`
		`const slicedFruits = fruits.slice(1, 3); // Creates a new array with elements from index 1 to 2`
		`console.log('slicedFruits are',slicedFruits)`

indexOf-Methode	Diese Methode wird verwendet, um den Index eines bestimmten Elements innerhalb eines Arrays zu finden. Sie gibt den Index des ersten Vorkommens des Elements im Array zurück oder -1, wenn das Element nicht gefunden wird.	`const fruits = ["apple", "banana", "cherry", "banana"];`
		`const index = fruits.indexOf("banana"); // Returns 1 (the first occurrence of "banana")`
		`console.log('Index of banana is', index)`

reverse-Methode	Die reverse-Methode kehrt die Reihenfolge der Elemente in einem Array um und kehrt das Array somit an Ort und Stelle um.	`const fruits = ["apple", "banana", "cherry"];`
		`fruits.reverse(); // Reverses the order of the array`
		`console.log(fruits)`

Sortiermethode	Die Sortiermethode wird verwendet, um die Elemente eines Arrays vor Ort zu sortieren und gibt das sortierte Array zurück. Standardmäßig	`const numbers = [4, 2, 8, 6, 1,10];`
		`numbers.sort(); // Sorts as strings: [1,10, 2, 4, 6, 8]`
		`numbers.sort((a, b) => a - b); // Sorts as numbers: [1, 2, 4, 6, 8]`
		`console.log(numbers)`

	sortiert sie die Elemente als Strings und in lexikografischer Reihenfolge.	

Array-Iteration	Eine for-Schleife kann verwendet werden, um durch die Elemente eines Arrays zu iterieren, um auf jedes Element im Array zuzugreifen und es zu manipulieren.	`const fruits = ['apple', 'banana', 'cherry', 'date'];`
		`for (let i = 0; i < fruits.length; i++) {`
		`  console.log(fruits[i]);`
		`}`

forEach	Die forEach-Methode durchläuft ein Array und wendet eine bereitgestellte Funktion auf jedes Element an.	`function sendWelcomeEmail(email) {`
		`  console.log(`Welcome email sent to ${email}`);`
		`}`
		`const users = [`
		`  { name: 'Alice', email: 'alice@example.com' },`
		`  { name: 'Bob', email: 'bob@example.com' },`
		`  { name: 'Charlie', email: 'charlie@example.com' },`
		`];`
		`users.forEach((user) => {`
		`  sendWelcomeEmail(user.email);`
		`});`

map-Methode	Die map-Methode erstellt ein neues Array, indem eine bereitgestellte Funktion auf jedes Element im ursprünglichen Array angewendet wird.	`const products = [`
		`  { name: 'Laptop', price: 1000 },`
		`  { name: 'Smartphone', price: 500 },`
		`  { name: 'Tablet', price: 300 },`
		`];`
		`products.map((product) => {`
		`  console.log(`The price of ${product.name} is $${product.price}`);`
		`});`

Filtermethode	Die Filtermethode erstellt ein neues Array, das Elemente enthält, die eine bestimmte Bedingung erfüllen. Sie ist nützlich, um spezifische Daten aus einem Array zu extrahieren.	`const products = [`
		`  { name: 'Laptop', price: 1000 },`
		`  { name: 'Smartphone', price: 500 },`
		`  { name: 'Tablet', price: 300 },`
		`  { name: 'Monitor', price: 250 },`
		`  { name: 'Keyboard', price: 50 },`
		`];`
		`function filterProductsByPriceRange(products, minPrice, maxPrice) {`
		`  return products.filter((product) => product.price >= minPrice && product.price <= maxPrice)`
		`}`
		`const minPrice = 100;  // Minimum price threshold`
		`const maxPrice = 500;  // Maximum price threshold`
		`const filteredProducts = filterProductsByPriceRange(products, minPrice, maxPrice);`
		`filteredProducts.forEach((product) => {`
		`  console.log(`${product.name} is of $${product.price}`);`
		`});`

reduce-Methode	Die reduce-Methode ermöglicht es Ihnen, ein Array auf einen einzelnen Wert zu reduzieren, indem Sie eine Funktion auf jedes Element anwenden. Sie	`const orderPrices = [50, 30, 25, 40, 15];`
		`const totalOrderValue = orderPrices.reduce((total, price) => total + price, 0);`
		`console.log('The total value of order is ', totalOrderValue)`

	ist hervorragend geeignet, um Daten zu aggregieren.	
find-Methode	Die find-Methode gibt das erste Element in einem Array zurück, das eine bestimmte Bedingung erfüllt. Sie ist nützlich, um nach spezifischen Daten zu suchen.	`const employees = [{ id: 1, name: 'Alice', Eid: 'EMP001', 'Contact details': 'alice@example.com', Role: 'Manag { id: 2, name: 'Bob', Eid: 'EMP002', 'Contact details': 'bob@example.com', Role: 'Engineer' { id: 3, name: 'Charlie', Eid: 'EMP003', 'Contact details': 'charlie@example.com', Role: 'A]; const employee = employees.find((e) => e.id === 2); console.log(`Details of the employee\nname: ${employee.name}\nEid: ${employee.Eid}\nContact det`
2D Array	Ein 2D-Array kann erstellt werden, indem ein Array von Arrays initialisiert wird.	`const grid = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ];`
Zugriff auf 2D-Array	Um auf ein bestimmtes Element in einem 2D-Array zuzugreifen, müssen Sie sowohl die Zeilen- als auch die Spaltenindizes angeben.	`for (let i = 0; i < grid.length; i++) { for (let j = 0; j < grid[i].length; j++) { console.log(`Element at (${i}, ${j}): ${grid[i][j]}`); } }`
2D-Array zur Buchung von Sitzplätzen	Sie können ein Buchungssystem mit einem 2D-Array erstellen.	`<!DOCTYPE html> <html> <head> <style> /* CSS for styling the seats */ .seating-chart { display: grid; grid-template-columns: repeat(3, 70px); gap: 10px; justify-content: center; } .seat { width: 70px; height: 40px; text-align: center; line-height: 40px; border: 1px solid #ccc; cursor: pointer; } .booked { background-color: #FF0000; /* Red */ cursor: not-allowed; color: white; /* Set the text color to white for booked seats */ } .available { background-color: #7FFF00; /* Light Green */ } .select-button { width: 100%; padding: 10px; margin: 10px; background-color: #007BFF; /* Blue */ color: white; border: none; cursor: pointer; } </style> </head> <body> <h2>Movie Theater Seating</h2> <div id="seating-chart" class="seating-chart"> <div class="seat available" onclick="bookSeat(0, 0)">A1</div> <div class="seat available" onclick="bookSeat(0, 1)">A2</div> <div class="seat available" onclick="bookSeat(0, 2)">A3</div>`

```
        <div class="seat available" onclick="bookSeat(1, 0)">B1</div>
        <div class="seat available" onclick="bookSeat(1, 1)">B2</div>
        <div class="seat available" onclick="bookSeat(1, 2)">B3</div>
        <div class="seat available" onclick="bookSeat(2, 0)">C1</div>
        <div class="seat available" onclick="bookSeat(2, 1)">C2</div>
        <div class="seat available" onclick="bookSeat(2, 2)">C3</div>
    </div>
    <button class="select-button" onclick="bookRandomSeat()">Select Random Seat</button>
    <script>
        // JavaScript for booking seats
        const theaterSeats = [
            ['X', 'O', 'X'],
            ['O', 'X', 'O'],
            ['X', 'O', 'X']
        ];
        function bookSeat(row, col) {
            if (theaterSeats[row][col] === 'O') {
                theaterSeats[row][col] = 'X';
                updateSeatStatus(row, col, 'booked');
                alert(`Seat ${String.fromCharCode(65 + row)}${col + 1} is booked.`);
            } else {
                alert(`Seat ${String.fromCharCode(65 + row)}${col + 1} is already taken.`);
            }
        }
        function updateSeatStatus(row, col, status) {
            const seats = document.getElementsByClassName('seat');
            const index = row * 3 + col;
            seats[index].classList.remove('available', 'booked');
            seats[index].classList.add(status);
        }
        function bookRandomSeat() {
            const availableSeats = [];
            for (let row = 0; row < theaterSeats.length; row++) {
                for (let col = 0; col < theaterSeats[row].length; col++) {
                    if (theaterSeats[row][col] === 'O') {
                        availableSeats.push({ row, col });
                    }
                }
            }
            if (availableSeats.length > 0) {
                const randomIndex = Math.floor(Math.random() * availableSeats.length);
                const { row, col } = availableSeats[randomIndex];
                bookSeat(row, col);
            } else {
                alert('All seats are already booked.');
            }
        }
    </script>
</body>
</html>
```

Klassen	Klassen sind eine Möglichkeit, Blaupausen oder Vorlagen für Objekte zu erstellen. Sie definieren die Struktur und das Verhalten der Objekte dieser Klasse.	`class Person { constructor(firstName, lastName) { this.firstName = firstName; this.lastName = lastName; } getFullName() { return `${this.firstName} ${this.lastName}`; } } // Creating an instance of the Person class const person1 = new Person("John", "Doe"); console.log(person1.getFullName()); // Output: "John Doe"`
Konstruktorobjekte	Objekte sind Instanzen von Klassen oder können als eigenständige Objekte ohne eine Klasse erstellt werden. Sie können Eigenschaften und Methoden haben.	`class Car { constructor(make, model, year) { this.make = make; this.model = model; this.year = year; } startEngine() { console.log(`The ${this.make} ${this.model}'s engine is running.`); } } const myCar = new Car("Toyota", "Camry", 2022); myCar.startEngine(); // Output: "The Toyota Camry's engine is running."`

Objektliterale	Objektliterale sind eine Möglichkeit, ad-hoc Objekte zu erstellen, ohne eine Klasse zu definieren.	`const person = { firstName: "Alice", lastName: "Johnson", getFullName: function() { return `${this.firstName} ${this.lastName}`; } }; console.log(person.getFullName()); // Output: "Alice Johnson"`
Funktionskonstruktor	Ein Funktionskonstruktor ist eine reguläre JavaScript-Funktion, die verwendet wird, um Objekte zu erstellen und zu initialisieren. Es ist üblich, Funktionskonstruktoren mit einem Großbuchstaben zu benennen.	`function Car(make, model) { this.make = make; this.model = model; } const car1 = new Car("Toyota", "Camry"); const car2 = new Car("Honda", "Civic"); console.log('Car1 details are', car1); console.log('Car2 details are', car2);`
. (Punkt) Notation	Die Punktnotation ist eine Möglichkeit, auf Eigenschaften von Objekten zuzugreifen.	`const person = { firstName: "John", lastName: "Doe", age: 30 }; console.log(person.firstName); // Output: "John" console.log(person.lastName); // Output: "Doe" console.log(person.age); // Output: 30`
Bracket-Notation	Die Bracket-Notation ist eine Methode, um auf Eigenschaften von Objekten zuzugreifen, besonders nützlich, wenn Eigenschaftsnamen Sonderzeichen oder Leerzeichen enthalten.	`const person = { "first name": "John", "last name": "Doe", age: 30 }; console.log(person["first name"]); // Output: "John" console.log(person["last name"]); // Output: "Doe" console.log(person["age"]); // Output: 30`
Arrays von Objekten	Ein Array von Objekten in JavaScript ist eine Sammlung mehrerer Objekte, die innerhalb eines einzigen Array-Containers gespeichert sind.	`const students = [ { name: "Alice", age: 25 }, { name: "Bob", age: 22 }, { name: "Charlie", age: 28 } ];`

Zugriff auf Array von Objekten	Sie können auf Elemente innerhalb eines Arrays von Objekten zugreifen, indem Sie den Array-Index und die Punktnotation verwenden.	`const students = [`
		`  { name: "Alice", age: 25 },`
		`  { name: "Bob", age: 22 },`
		`  { name: "Charlie", age: 28 }`
		`];`
		`console.log(students[0].name); // Output: "Alice"`
		`console.log(students[2].age);  // Output: 28`

Durchlaufen eines Arrays von Objekten	Das Durchlaufen von Objekten durch Arrays umfasst for-Schleifen und Array-Methoden.	`const students = [`
		`  { name: "Alice", age: 25 },`
		`  { name: "Bob", age: 22 },`
		`  { name: "Charlie", age: 28 }`
		`];`
		`for (let i = 0; i < students.length; i++) {`
		`  console.log(students[i].name);`
		`}`

Objekte hinzufügen	Sie können neue Objekte mit der Methode push zum Array hinzufügen.	`//Adding Elements`
		`const students = [`
		`  { name: "Alice", age: 25 },`
		`  { name: "Bob", age: 22 },`
		`  { name: "Charlie", age: 28 }`
		`];`
		`students.push({ name: "David", age: 20 });  // Add a new student`
		`console.log('After using push method ');`
		`console.log(students);`

Objekte entfernen	Sie können Objekte mit der pop-Methode entfernen.	`//Removing Elements`
		`const students = [`
		`  { name: "Alice", age: 25 },`
		`  { name: "Bob", age: 22 },`
		`  { name: "Charlie", age: 28 }`
		`];`
		`const removedStudent = students.pop();      // Remove the last student`
		`console.log('After using pop method ');`
		`console.log(students);`

Filtern und Abbilden von Arrays von Objekten	Sie können Arrays von Objekten mit Methoden wie filter und map filtern und transformieren.	`const students = [`
		`  { name: "Alice", age: 25 },`
		`  { name: "Bob", age: 22 },`
		`  { name: "Charlie", age: 28 }`
		`];`
		`const adults = students.filter(student => student.age >= 23);  // Filter students who are 18 or`
		`const studentNames = students.map(student => student.name);   // Create an array of student nam`
		`console.log(‘Using Filter Method’);`
		`console.log(adults);`
		`console.log(‘Using Map Method’`
		`console.log(studentNames);`

Arrays von Objekten abbilden	Sie können Arrays von Objekten durch Methoden wie map durchlaufen und transformieren.	`const employees = [ { name: "Alice", age: 35 }, { name: "Bob", age: 32 }, { name: "Charlie", age: 38 } ]; const employee = employees.map((employee) => { return employee}); console.log(employee);`
Nach Objekten suchen	Sie können in einem Array von Objekten mit Methoden wie find nach Objekten suchen.	`const employees = [ { name: "Alice", age: 35 }, { name: "Bob", age: 32 }, { name: "Charlie", age: 38 } ]; const employee = employees.find(employee => employee.name === "Charlie"); console.log(employee.age);`
Verschachteltes Array von Objekten	Ein Array von Objekten wird verwendet, um Daten zu speichern und zu organisieren, sodass Sie die Informationen einfach abrufen und bearbeiten können.	`let arrayOfObjects = [{ name: 'John', age: 25, hobbies: ['Reading', 'Traveling'], address: { street: '123 Main St', city: 'New York', zip: '10001' } }, { name: 'Alice', age: 30, skills: ['JavaScript', 'React', 'Node.js'], projects: [{ title: 'Project A', completed: true }, { title: 'Project B', completed: false }] }, { title: 'Special Object', data: [1, 2, 3], metadata: { key: 'value' } }, { // An object with no specific properties }, { anotherObject: true, nestedArrays: [[1, 2, 3], ['a', 'b', 'c']], additionalProperty: 'Extra' }];`
Zugriff auf das verschachtelte Array - Code oben	Mit dem Punktoperator können Elemente des verschachtelten Arrays zugegriffen werden, was im obenstehenden Code beschrieben wurde.	`// Accessing properties of the first object console.log(arrayOfObjects[0].name); // Output: John console.log(arrayOfObjects[0].hobbies[0]); // Output: Reading // Accessing properties of the second object console.log(arrayOfObjects[1].skills[2]); // Output: Node.js console.log(arrayOfObjects[1].projects[0].title); // Output: Project A // Accessing properties of the third object console.log(arrayOfObjects[2].metadata.key); // Output: value // Accessing properties of the fourth object console.log(arrayOfObjects[3]); // Output: {} // Accessing properties of the fifth object console.log(arrayOfObjects[4].anotherObject); // Output: true console.log(arrayOfObjects[4].additionalProperty); // Output: Extra`

Strings	Strings sind ein Datentyp in JavaScript, der verwendet wird, um Text darzustellen. Sie können Buchstaben, Zahlen, Symbole und Leerzeichen enthalten.	`const message = "This is a message.";`
Strings	Strings sind ein Datentyp in JavaScript, der verwendet wird, um Text darzustellen. Sie können Buchstaben, Zahlen, Symbole und Leerzeichen enthalten.	`const message = "This is a message.";`
Template-Literale	Template-Literale in JavaScript sind Strings, die eingebettete Ausdrücke erlauben, gekennzeichnet durch Backticks (), und ermöglichen einfache mehrzeilige Strings sowie die Interpolation von Variablen mit ${}`.	`const fullName = `${firstName} ${lastName}`;`

String-Verkettung	Der Verkettungsoperator + in JavaScript wird verwendet, um zwei oder mehr Strings zusammenzuführen, um einen einzelnen, längeren String zu erstellen.	`const firstName='Peter';`
		`const greeting = 'Hello, ' + firstName + '!';`
		`console.log(greeting);`

Zeichenlänge	Um die Länge eines Strings zu bestimmen, kann die Length-Eigenschaft verwendet werden.	`const message1 = "This is a message.";`
		`const Stringlength1 = message1.length;`
		`const message2 = "Thisisamessage";`
		`const Stringlength2 = message2.length;`
		`console.log(Stringlength1);`
		`console.log(Stringlength2)`

Zugriff auf Zeichen	Einzelne Zeichen innerhalb eines Strings können mit Hilfe der Klammernotation und einem nullbasierten Index zugegriffen werden.	`const text = "JavaScript";` `const firstCharacter = text[0];`

toLowerCase und toUpperCase	JavaScript bietet Methoden an, um die Groß- und Kleinschreibung eines Strings in Klein- und Großbuchstaben zu ändern.	`const text = "Hello, World!";`
		`const lowercaseText = text.toLowerCase(); // "hello, world!"`
		`const uppercaseText = text.toUpperCase(); // "HELLO, WORLD!"`
		`console.log('The lowercase  for text is ',lowercaseText);`
		`console.log('The uppercase  for text is ',uppercaseText);`

indexOf() Methode	indexOf gibt den Index des ersten Vorkommens eines angegebenen Teilstrings innerhalb eines Strings zurück. Es wird -1 zurückgegeben, wenn der Teilstring nicht gefunden wird.	`const sentence = "The quick brown fox jumps over the lazy dog.";`
		`const indexOfFox = sentence.indexOf("fox"); // 16`
		`console.log(indexOfFox);`

includes() Methode	includes gibt einen Boolean zurück, der angibt, ob ein bestimmter Teilstring innerhalb eines Strings gefunden wird, und gibt true zurück, wenn er gefunden wird, und false, wenn nicht.	`const sentence = "The quick brown fox jumps over the lazy dog.";`
		`const hasFox = sentence.includes("fox"); // true`
		`console.log(hasFox);`

substring() Methoden	substring extrahiert Zeichen aus einem String zwischen zwei angegebenen Indizes. Das bedeutet, einen Teilstring aus dem Text zu extrahieren, der bei Index 0 beginnt und bei Index 5 endet (Index 5 ausgeschlossen).	`const text = "Hello, World!";`
		`const subText1 = text.substring(0, 5); // "Hello"`
		`console.log(subText1);`

slice() Methode	slice extrahiert einen Abschnitt eines Strings und gibt ihn als neuen String zurück, wobei die Start- und Endpositionen angegeben werden. Das bedeutet, einen Teilstring aus dem Text zu extrahieren, der bei Index 7 beginnt und bis zum Ende des Strings reicht.	`const text = "Hello, World!";`
		`const subText2 = text.slice(7);  // "World!"`
		`console.log(subText2);`

substr() Methode	substr extrahiert eine bestimmte Anzahl von Zeichen aus einer Zeichenkette, beginnend an einem bestimmten Index. Das bedeutet, einen Teilstring aus dem Text zu extrahieren, der am 7. Index beginnt und 5 Zeichen umfasst.	`const text = "Hello, World!";`
		`const subText3 = text.substr(7, 5);  // "World"`
		`console.log(subText3);`

Ersetzen von Teilstrings	Die Methode replace ermöglicht es Ihnen, Teilstrings durch neue Werte zu ersetzen.	`const text = "Hello, World!";`
		`const updatedText = text.replace("World", "Universe");`
		`console.log(updatedText);`

Strings teilen	Sie können einen String mit der Methode split in	`const csvData = "Alice,25,New York;Bob,30,Los Angeles;Charlie,28,Chicago";`
		`const peopleArray = csvData.split(';');`
		`console.log(peopleArray);`

	ein Array von Teilstrings aufteilen.	

trim() Methode	Die trim Methode entfernt führende und nachfolgende Leerzeichen aus einem String.	`const text = "   Trim me!   ";`
		`console.log(text.length);`
		`const trimmedText = text.trim();`
		`console.log(trimmedText.length);`

round(), ceil() und floor() Mathematikmethoden	round() rundet eine Zahl auf die nächste ganze Zahl. ceil() rundet eine Zahl auf die nächste ganze Zahl nach oben. floor() rundet eine Zahl auf die nächste ganze Zahl nach unten.	`const number = 3.6;`
		`const rounded = Math.round(number); // Round to nearest integer: 4`
		`const ceil = Math.ceil(number);   // Round up: 4`
		`const floor = Math.floor(number);   // Round down: 3`

pow(), sqrt() und log() Mathematikmethoden	pow() hebt eine Zahl auf einen bestimmten Exponenten. sqrt() gibt die Quadratwurzel einer Zahl zurück. log() gibt den natürlichen Logarithmus (Basis e) einer Zahl zurück.	`const base = 2;`
		`const exponent = 3;`
		`const power = Math.pow(base, exponent); // Power: 8`
		`const squareRoot = Math.sqrt(base);     // Square Root: 1.41421356237`
		`const naturalLog = Math.log(base);      // Natural Logarithm: 0.69314718056`

random() Methode	Die random() Methode in JavaScript erzeugt eine pseudo-zufällige Fließkommazahl zwischen 0 (einschließlich) und n (ausschließlich).	`<!DOCTYPE html>`
		`<html>`
		`<head>`
		`  <title>Random Quote Generator</title>`
		`</head>`
		`<body>`
		`  <h1>Random Quote Generator</h1>`
		
		`  <p id="quoteDisplay"></p>`
		
		`  <button onclick="generateRandomQuote()">Get Quote</button>`
		
		`  <script>`
		`    const quotes = [`
		`      "Life is what happens when you're busy making other plans. - John Lennon",`
		`      "The only way to do great work is to love what you do. - Steve Jobs",`
		`      "In three words, I can sum up everything I've learned about life: it goes on. - Robert Fr`
		`      "Don't count the days, make the days count. - Muhammad Ali",`
		`      "The only thing we have to fear is fear itself. - Franklin D. Roosevelt",`
		`      "To be yourself in a world that is constantly trying to make you something else is the gr`
		`    ];`
		
		`    function generateRandomQuote() {`
		`      const randomIndex = Math.floor(Math.random() * quotes.length); // Generate a random inde`
		`      const randomQuote = quotes[randomIndex]; // Get a random quote`
		
		`      document.getElementById("quoteDisplay").textContent = randomQuote;`
		`    }`
		`  </script>`
		`</body>`
		`</html>`

Datum Objekt	Datum-Objekte werden verwendet, um spezifische Momente in der Zeit darzustellen.	`const currentDate = new Date(); // Current date and time`
		`const specificDate = new Date(2023, 0, 15); // January 15, 2023`
		`const fromMilliseconds = new Date(1672569600000); // From milliseconds since the epoch`

Abfragedatum	Datum-Objekte bieten Zugriff auf einzelne Komponenten eines Datums, wie Jahr, Monat, Tag und Stunde.	`const date = new Date();`
		`const year = date.getFullYear();  // Current year`
		`const month = date.getMonth();  // Current month (0-11)`
		`const day = date.getDate();  // Day of the month (1-31)`
		`const hours = date.getHours();  // Hours (0-23)`
		`const minutes = date.getMinutes();  // Minutes (0-59)`
		`const seconds = date.getSeconds();  // Seconds (0-59)`

toLocaleDateString() und toLocaleTimeString()	toLocaleDateString() konvertiert ein Datum in einen String, der den Datumsanteil gemäß den Formatierungsrichtlinien der jeweiligen Region darstellt. toLocaleTimeString() konvertiert ein Datum in einen String, der den Zeitanteil gemäß den Formatierungsrichtlinien der jeweiligen Region darstellt.	`const date = new Date();`
		`const formattedDate = date.toLocaleDateString(); // "11/15/2023"`
		`const formattedTime = date.toLocaleTimeString(); // "1:30:45 PM"`

Datumsarithmetik	Datumobjekte ermöglichen verschiedene Datumsarithmetik-Operationen, einschließlich des Hinzufügens und Subtrahierens von Zeitintervallen.	`const date = new Date();`
		`date.setFullYear(2024); // Set the year to 2024`
		`date.setDate(date.getDate() + 7); // Add 7 days`
		`const futureDate = new Date();`
		`futureDate.setDate(futureDate.getDate() + 30); // Date 30 days from now`

setTimeout() Methode	Die setTimeout-Funktion plant die Ausführung einer Funktion nach einer bestimmten Verzögerung in Millisekunden:	`setTimeout(function() {`
		`  console.log("This message appears after a delay.");`
		`}, 2000); // Displayed after a 2-second delay`

setInterval	setInterval führt eine Funktion wiederholt in einem festgelegten Intervall aus.	`let count = 0;`
		`const intervalId = setInterval(function() {`
		`  console.log("Count: " + count);`
		`  count++;`
		`  if (count > 5) {`
		`  clearInterval(intervalId); // Stop after 6 iterations`
		`  }`
		`}, 1000); // Displayed every second.`

Änderungsprotokoll

Datum	Version	Geändert von	Änderungsbeschreibung
2023-08-12	1.1	Richa	Cheatsheet erstellt
2025-06-25	1.2	Nikesh Kumar	Tippfehler in Zeile 164 von “onst” zu “const” korrigiert; Beschreibung für “unshift-Methode” aktualisiert