

Cheatsheet: Arrays and Objects in JavaScript

| JavaScript Array and Objects | Description | Code Example |
|------------------------------|---|---|
| Array declaration | Arrays in JavaScript are ordered, meaning that the elements are stored in a specific sequence. | <pre>const fruits = ["apple", "banana", "cherry"];</pre> |
| Array Indexing | Arrays are zero-indexed, meaning the first element is at index 0, the second at index 1, and so on. | <pre>const fruits = ["apple", "banana", "cherry"]; const firstFruit = fruits[0]; // "apple" const secondFruit = fruits[1]; // "banana"</pre> |
| Array Length | The length property is used to determine the number of items present in an array. | <pre>const fruits = ["apple", "banana", "cherry"]; const numFruits = fruits.length; // 3 console.log(numFruits);</pre> |
| Array Mutability | Arrays in JavaScript are mutable, meaning you can change, add, or remove elements after the array is created. | <pre>const fruits = ["apple", "banana", "cherry"]; fruits[2] = "strawberry"; // Modifying an element fruits[3] = "Kiwi"; // Adding an element</pre> |
| push method | Adds one or more elements to the end of an array. | <pre>const fruits = ["apple", "banana"]; fruits.push("orange", "strawberry"); console.log(fruits)</pre> |
| pop method | Removes the last element from an array and returns it. | <pre>const fruits = ["apple", "banana", "orange"]; const removedFruit = fruits.pop(); console.log('Fruits are',fruits) console.log('Removed fruits are',removedFruit)</pre> |
| shift methods | Removes the first element from an array and returns it. | <pre>Removes the first element from an array and returns it.</pre> |

| | | |
|------------------------|--|--|
| unshift method | Adds one or more elements to the beginning of an array and returns it. | <pre>const fruits = ["banana", "orange"]; fruits.unshift("apple", "strawberry"); console.log(fruits);</pre> |
| splice method | Changes the contents of an array by removing, replacing, or adding elements at a specified position. | <pre>const fruits = ["apple", "banana", "cherry"]; fruits.splice(1, 1, "grape"); // Replace the second element with "grape" console.log(fruits)</pre> |
| concat method | The concat method in JavaScript arrays combines arrays in sequence, creating a new array containing the elements of the original arrays in the order they were concatenated. | <pre>const fruits = ["apple", "banana"]; const additionalFruits = ["orange", "strawberry"]; const combinedFruits = fruits.concat(additionalFruits); console.log('combinedFruits are', combinedFruits)</pre> |
| slice method | Returns a shallow copy of a portion of an array into a new array. | <pre>const fruits = ["apple", "banana", "cherry", "orange"]; const slicedFruits = fruits.slice(1, 3); // Creates a new array with elements from index 1 to 2 (not including 3). console.log('slicedFruits are',slicedFruits)</pre> |
| indexOf method | This method is used to find the index of a specified element within an array. It returns the index of the first occurrence of the element in the array, or -1 if the element is not found. | <pre>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)</pre> |
| reverse method | The reverse method reverses the order of elements in an array, effectively reversing the array in place. | <pre>const fruits = ["apple", "banana", "cherry"]; fruits.reverse(); // Reverses the order of the array console.log(fruits)</pre> |
| sort method | The sort method is used to sort the elements of an array in place and returns the sorted array. By default, it sorts elements as strings and in lexicographic order. | <pre>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)</pre> |
| Array iteration | A for loop can be used to iterate through the elements | <pre>const fruits = ['apple', 'banana', 'cherry', 'date']; for (let i = 0; i < fruits.length; i++) { console.log(fruits[i]); }</pre> |

| | | |
|---------------|--|---|
| | of an array to access and manipulate each item in the array. | |
| forEach | The forEach method iterates through an array and applies a provided function to each element. | <pre>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); });</pre> |
| map method | The map method creates a new array by applying a provided function to each element in the original array. | <pre>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}`); });</pre> |
| filter method | The filter method creates a new array containing elements that pass a specified condition. It's useful for extracting specific data from an array. | <pre>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}`); });</pre> |
| reduce method | The reduce method allows you to reduce an array to a single value by applying a function to each element. It's excellent for aggregating data. | <pre>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)</pre> |
| find method | The find method returns the first element in an array that satisfies a specified condition. It's useful for searching for specific data. | <pre>const employees = [{ id: 1, name: 'Alice', Eid: 'EMP001', 'Contact details': 'alice@example.com', Role: 'Manager', Designation: 'Software Engineer' }, { id: 2, name: 'Bob', Eid: 'EMP002', 'Contact details': 'bob@example.com', Role: 'Engineer', Designation: 'Software Engineer' }, { id: 3, name: 'Charlie', Eid: 'EMP003', 'Contact details': 'charlie@example.com', Role: 'Analyst', Designation: 'Software Engineer' },]; const employee = employees.find((e) => e.id === 2); console.log(`Details of the employee\nname: \${employee.name}\nEid: \${employee.Eid}\nContact details: \${employee['Contact details']}\nRole: \${employee.Role}\nDesignation: \${employee.Designation}`);</pre> |

| | | |
|-----------------------|--|---|
| | | |
| 2D Array | A 2D array can be created by initializing an array of arrays. | <pre>const grid = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];</pre> |
| Access 2D Array | To access a specific element in a 2D array, you need to provide both row and column indices. | <pre>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]}`); } }</pre> |
| 2D array to book seat | You can create a booking system using 2D array. | <pre><!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); } </script></pre> |

| | | |
|----------------------|--|--|
| | | <pre> } 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] === '0') { 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.');</pre> <pre> } } </script> </body> </html></pre> |
| Classes | Classes are a way to create blueprint or templates for objects. They define the structure and behavior of objects of that class. | <pre>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"</pre> |
| Constructor Objects | Objects are instances of classes or can be created as standalone objects without a class. They can have properties and methods. | <pre>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."</pre> |
| Object Literals | Object literals are a way to create ad-hoc objects without defining a class. | <pre>const person = { firstName: "Alice", lastName: "Johnson", getFullName: function() { return `\${this.firstName} \${this.lastName}`; } }; console.log(person.getFullName()); // Output: "Alice Johnson"</pre> |
| Function Constructor | A function constructor is a regular JavaScript function that is used to create and initialize objects. It's a convention to name function constructors | <pre>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);</pre> |

| | | |
|--|--|--|
| | with an initial capital letter. | |
| . (Dot) Notation | Dot notation is a way to access object properties. | <pre>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</pre> |
| Bracket Notation | Bracket notation is a way to access object properties, especially useful when property names contain special characters or spaces. | <pre>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</pre> |
| Arrays of Objects | An array of objects in JavaScript is a collection of multiple objects stored within a single array container. | <pre>const students = [{ name: "Alice", age: 25 }, { name: "Bob", age: 22 }, { name: "Charlie", age: 28 }];</pre> |
| Access Array of Objects | You can access elements within an array of objects using the array index and using dot notation. | <pre>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</pre> |
| Iterating Through an Array of Objects | Iteration of objects through arrays include for loops and array methods. | <pre>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); }</pre> |
| Adding Objects | You can add new objects to the array using the push method. | <pre>//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</pre> |

| | | |
|---|---|--|
| | | <pre> console.log('After using push method '); console.log(students); </pre> |
| Removing Objects | <p>You can remove objects using the pop method.</p> | <pre> //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); </pre> |
| Filtering and Mapping Arrays of Objects | <p>You can filter and transform arrays of objects using array methods like filter and map.</p> | <pre> 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 older console.log('Using Filter Method'); const studentNames = students.map(student => student.name); // Create an array of student names console.log(adults); console.log('Using Map Method'); console.log(studentNames); </pre> |
| Mapping Arrays of Objects | <p>You can traverse and transform arrays of objects using array method like map.</p> | <pre> const employees = [{ name: "Alice", age: 35 }, { name: "Bob", age: 32 }, { name: "Charlie", age: 38 }]; const employee = employees.map((employee) => { return employee}); console.log(employee); </pre> |
| Searching for Objects | <p>You can search for objects within an array of objects using array methods like find.</p> | <pre> 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); </pre> |
| Nested Array of objects | <p>An array of objects is used to store and organize data in a way that allows you to access and manipulate the information easily.</p> | <pre> 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'], </pre> |

| | | |
|--------------------------------|--|---|
| | | <pre> 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' }] };</pre> |
| Access Nested Array-Code Above | Using . dot operator elements of nested array can be accesed, which has been described in just above code. | <pre>// 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</pre> |
| Strings | Strings are data type in JavaScript used to represent text. They can contain letters, numbers, symbols, and whitespace characters. | <pre>const message = "This is a message.";</pre> |
| Strings | Strings are data type in JavaScript used to represent text. They can contain letters, numbers, symbols, and whitespace characters. | <pre>const message = "This is a message.";</pre> |
| template literals | Template literals in JavaScript are strings allowing embedded expressions, denoted by backticks (), enabling easy multiline strings and interpolation of variables using <code>\${}</code> . | <pre>const fullName = `\${firstName} \${lastName}`;</pre> |

| | | |
|-----------------------------|---|--|
| String Concatenation | The concatenation operator + in JavaScript is used to combine (join) two or more strings together to create a single, longer string. | <pre>const firstName='Peter'; const greeting = 'Hello, ' + firstName + '!'; console.log(greeting);</pre> |
| String Length | To determine the length of a string, length property can be used. | <pre>const message1 = "This is a message."; const Stringlength1 = message1.length; const message2 = "Thisisamessage"; const Stringlength2 = message2.length; console.log(Stringlength1); console.log(Stringlength2)</pre> |
| Accessing Characters | Individual characters within a string can be accessed using bracket notation and a zero-based index. | <pre>const text = "JavaScript"; const firstCharacter = text[0];</pre> |
| toLowerCase and toUpperCase | JavaScript provides methods to change the case of a string into lowercase and uppercase. | <pre>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);</pre> |
| indexOf() method | indexOf returns the index of the first occurrence of a specified substring within a string. It returns -1 if the substring is not found. | <pre>const sentence = "The quick brown fox jumps over the lazy dog."; const indexOfFox = sentence.indexOf("fox"); // 16 console.log(indexOfFox);</pre> |
| includes() method | includes returns a boolean indicating whether a specified substring is found within a string, returning true if found and false if not. | <pre>const sentence = "The quick brown fox jumps over the lazy dog."; const hasFox = sentence.includes("fox"); // true console.log(hasFox);</pre> |
| substring() methods | substring extracts characters from a string between two specified indices. It means extracting a substring from the text starting at index 0 and ending at index 5 (excluding index 5). | <pre>const text = "Hello, World!"; const subText1 = text.substring(0, 5); // "Hello" console.log(subText1);</pre> |
| slice() method | slice extracts a section of a string | <pre>const text = "Hello, World!"; const subText2 = text.slice(7); // "World!"</pre> |

| | | |
|---|---|--|
| | and returns it as a new string, specifying the start and end positions. It means extracting a substring from the text starting at index 7 until the end of the string. | <pre>console.log(subText2);</pre> |
| substr() method | substr extracts a specified number of characters from a string, starting at a specified index.It means extracting a substring from the text starting at the 7th index and including 5 characters. | <pre>const text = "Hello, World!"; const subText3 = text.substr(7, 5); // "World" console.log(subText3);</pre> |
| Replacing Substrings | The replace method allows you to replace substrings with new values. | <pre>const text = "Hello, World!"; const updatedText = text.replace("World", "Universe"); console.log(updatedText);</pre> |
| Splitting Strings | You can split a string into an array of substrings using the split method. | <pre>const csvData = "Alice,25,New York;Bob,30,Los Angeles;Charlie,28,Chicago"; const peopleArray = csvData.split(';'); console.log(peopleArray);</pre> |
| trim()method | The trim method removes leading and trailing whitespace from a string. | <pre>const text = " Trim me! "; console.log(text.length); const trimmedText = text.trim(); console.log(trimmedText.length);</pre> |
| round(), ceil() and floor() Math Methods | round() rounds a number to the nearest integer. ceil() rounds a number up to the nearest integer. floor() rounds a number down to the nearest integer. | <pre>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</pre> |
| pow(), sqrt() and log() Math Methods | pow() raises a number to a specified exponent. sqrt() returns the square root of a number. log() returns the natural logarithm (base e) of a number. | <pre>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</pre> |
| random() Method | The random() method in JavaScript generates a pseudo-random floating- | <pre><!DOCTYPE html> <html> <head> <title>Random Quote Generator</title> </head></pre> |

| | | |
|--|--|---|
| | point number between 0 (inclusive) and n (exclusive). | <pre> <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 Frost", "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 greatest accomplishment."]; function generateRandomQuote() { const randomIndex = Math.floor(Math.random() * quotes.length); // Generate a random index const randomQuote = quotes[randomIndex]; // Get a random quote document.getElementById("quoteDisplay").textContent = randomQuote; } </script> </body> </html> </pre> |
| Date Object | Date objects are used to represent specific moments in time. | <pre> 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 </pre> |
| Retrieving Date | Date objects provide access to individual components of a date, such as year, month, day, and hour. | <pre> 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) </pre> |
| toLocaleDateString() and toLocaleTimeString() | toLocaleDateString() to converts a date to a string representing the date portion according to the locale's formatting conventions. toLocaleTimeString() to converts a date to a string representing the time portion according to the locale's formatting conventions. | <pre> const date = new Date(); const formattedDate = date.toLocaleDateString(); // "11/15/2023" const formattedTime = date.toLocaleTimeString(); // "1:30:45 PM" </pre> |
| Date Arithmetic | Date objects allow for various date arithmetic operations, including adding and subtracting time intervals. | <pre> 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 </pre> |
| setTimeout() Method | The setTimeout function schedules the execution of a function after a | <pre> setTimeout(function() { console.log("This message appears after a delay."); }, 2000); // Displayed after a 2-second delay </pre> |

specified delay in milliseconds:

setInterval repeatedly executes a function at a specified interval.

```
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.
```



Skills Network