Cheatsheet: Einführung in die JavaScript-Entwicklung

JavaScript-Tag und Begriffe	Beschreibung	Codebeispiel
<script>	Wird verwendet, um den erforderlichen JavaScript-Code in Ihr HTML-Dokument einzufügen.	`<body>` `  <p id="showname"></p>` `  <script>` `  document.getElementById('showname').innerHTML='Peter';` `  </script>` `</body>`
<script src>	Wird verwendet, um die erforderlichen JavaScript-Dateien in Ihrem HTML-Dokument zu verlinken.	`<script src="script.js"></script>`
var	var ist ein Schlüsselwort, das verwendet wird, um Variablen zu deklarieren.	`var num1=10;` `var num2=11;`
var & Umfang	var hat einen funktionalen Umfang, der den Zugriff auf die Variable nur innerhalb der Funktion erlaubt.	`<!DOCTYPE html>` `<html lang="en">` `<head>` `  <meta charset="UTF-8">` `  <meta name="viewport" content="width=device-width, initial-scale=1.0">` `  <title>Document</title>` `</head>` `<body>` `  <p id="showname"></p>` `  <script>` `  function show() {` `  var name = 'Peter';` `  document.getElementById('showname').innerHTML = name;` `  }` `  </script>` `</body>` `</html>`
let	let ist ein Schlüsselwort, das verwendet wird, um Variablen zu deklarieren.	`let num1=20;` `let num2=21;`
let & Gültigkeitsbereich	let hat einen Block-Gültigkeitsbereich, der es der Variable ermöglicht, auf den Block, die Anweisung oder den Ausdruck	`<!DOCTYPE html>` `<html lang="en">` `<head>` `  <meta charset="UTF-8">` `  <meta name="viewport" content="width=device-width, initial-scale=1.0">` `  <title>Document</title>` `</head>` `<body>`

	beschränkt zu sein, in dem sie definiert ist, und eine erneute Deklaration im selben Gültigkeitsbereich zu verhindern.	`<p id="showemail"></p>` `<script>` `{` `let emailId = 'test@example.com';` `document.getElementById('showemail').innerHTML = emailId;` `}` `</script>` `</body>` `</html>`
const	const ist ein Schlüsselwort, das verwendet wird, um Variablen zu deklarieren.	`const employeeId=120;` `cont employeeId=121;`
const & Scope	Es erstellt eine Konstante, deren Wert nicht neu zugewiesen oder neu deklariert werden kann.	`<!DOCTYPE html>` `<html lang="en">` `<head>` `<meta charset="UTF-8">` `<meta name="viewport" content="width=device-width, initial-scale=1.0">` `<title>Document</title>` `</head>` `<body>` `<p id="showeEId"></p>` `<script>` `{` `const employeeId = 120';` `document.getElementById('showeEId').innerHTML = employeeId;` `}` `</script>` `</body>` `</html>`
Arithmetische Operatoren	Arithmetische Operatoren führen mathematische Berechnungen wie Addition, Subtraktion, Multiplikation, Division und Modulo durch.	`let x = 15;` `let y = 3;` `let sum = x + y; // Addition` `console.log(sum)  //the answer is 8` `let difference = x - y; // Subtraction` `console.log(difference)  //the answer is 2` `let product = x * y; // Multiplication` `console.log(product)  //the answer is 8` `let quotient = x / y; // Division` `console.log(quotient)  //the answer is 8` `let remainder = x % y; // Modulus` `console.log(remainder)  //the answer is 0`
Vergleichsoperatoren	Vergleichsoperatoren vergleichen Werte und geben basierend auf dem Vergleich wahr/falsch zurück.	`let a = 5;` `let b = 7;` `let isEqual = a == b; // Equality` `let isNotEqual = a != b; // Inequality` `let isStrictEqual = a === b; // Strict equality` `let isGreaterThan = a > b; // Greater than`

Logische Operatoren	Logische Operatoren kombinieren mehrere Bedingungen und geben ein boolesches Ergebnis zurück.	`let hasPermission = true;` `let isMember = false;` `let canAccessResource = hasPermission && isMember; // Logical AND` `let canViewPage = hasPermission || isMember; // Logical OR` `let isDenied = !hasPermission; // Logical NOT`
Zuweisungsoperatoren	Zuweisungsoperatoren weisen Variablen Werte zu. Zum Beispiel =, +=, -=.	`let x = 10; // Assigns the value 10 to the variable x` `x += 5; // Equivalent to x = x + 5` `x -= 5; //Equivalent to x = x + 5`
Unäre Operatoren	Unäre Operatoren wirken auf einen einzelnen Operanden und führen Operationen wie Negation oder Inkrementierung durch.	`let count = 5;` `count++; // Increment count by 1 (count is now 6)` `count--; // Decrement count by 1 (count is now 5 again)`
typeof-Operator	Der typeof-Operator gibt den Datentyp einer Variablen oder eines Ausdrucks als Zeichenkette zurück.	`let num1 = 42;` `console.log(typeof(num1)); //the awnswer is Number` `let name = 'John';` `console.log(typeof(name)); //the awnswer is String`
if-Anweisung	Die if-Anweisung wird verwendet, um einen Block von Code auszuführen, wenn die gegebene Bedingung wahr ist.	`let age = 25;` `if (age >= 18) {` `console.log("You are an adult.");` `} else {` `console.log("You are a minor.");` `}`
else if-Anweisung	Sie ermöglicht es Ihnen, mehrere Bedingungen nacheinander zu testen. Wenn die Bedingung wahr ist, wird der if-Anweisungsblock ausgeführt, andernfalls wird der else-Anweisungsblock ausgeführt.	`<!DOCTYPE html>` `<html lang="en">` `<head>` `<meta charset="UTF-8">` `<meta name="viewport" content="width=device-width, initial-scale=1.0">` `<title>Document</title>` `</head>` `<body>` `<p id="seasonmessage"></p>` `<script>` `let Seasonmonth = 'March to May';` `if (Seasonmonth == 'March to May') {` `document.getElementById("seasonmessage") = 'It is spring season';` `}` `else if (Seasonmonth == 'June to August') {` `document.getElementById("seasonmessage") ='It is summer season';` `}` `else if (Seasonmonth =='September to November') {` `document.getElementById("seasonmessage") = 'It is autumn season';`

		`` } else { document.getElementById("seasonmessage") = 'It is winter season'; } </script> </body> </html>```
Verschachtelte if else-Anweisung	Diese Anweisung ermöglicht es Ihnen, mehrere Bedingungen zu testen und unterschiedliche Codeblöcke basierend auf den Ergebnissen dieser Bedingungen auszuführen.	```const temperature = 30; const isRaining = true; if (temperature > 30) { if (isRaining) { console.log("It's hot and raining. Stay inside."); } else { console.log("It's hot, but not raining. Enjoy the sunshine."); } } else { if (isRaining) { console.log("It's not so hot, but it's raining. Take an umbrella."); } else { console.log("It's not hot, and it's not raining. Have a nice day."); } }```
switch-Anweisung	Die switch-Anweisung wird für mehrere bedingte Verzweigungen verwendet und ermöglicht die Ausführung verschiedener Codeblöcke basierend auf dem Wert eines Ausdrucks.	```let month = "December"; switch (day) { case "December": console.log("It's Christmas month."); break; case "November": console.log("It's Thanksgiving month"); break; default: console.log("It's a regular month."); }```
Terner Operator	Der terner Operator ist die einfachste Möglichkeit, bedingte Anweisungen wie if-else-Bedingungen zu schreiben.	```let age = 20; let canVote = age >= 18 ? "Yes" : "No";```
for Schleife	Eine for Schleife ist eine Kontrollstruktur, die es ermöglicht, einen Codeblock wiederholt eine bestimmte Anzahl von Malen auszuführen, bis eine bestimmte Bedingung erfüllt ist.	```for (let i = 1; i <= 5; i++) { console.log(i); }```
While-Schleife	Eine While-Schleife ist eine Kontrollstruktur, die es ermöglicht, einen Codeblock wiederholt auszuführen, solange	```let limit = 50; let a = 0; let b = 1; while (a <= limit) { console.log(a); let temp = a + b;```

	eine festgelegte Bedingung wahr ist.	``` a = b; b = temp; } ```
do while Schleife	Eine "do...while" Schleife ermöglicht es Ihnen, einen Codeblock wiederholt auszuführen, solange eine festgelegte Bedingung wahr ist, und garantiert, dass der Codeblock mindestens einmal ausgeführt wird, selbst wenn die Bedingung anfänglich falsch ist.	``` let roll = 1; do { console.log("Rolled a " + roll); roll++; } while (roll < 7); ```
Funktionsdeklaration und -aufruf	Eine Funktion ist ein wiederverwendbarer Codeblock, der so oft definiert und ausgeführt werden kann, wie es nötig ist.	``` function sayHello() { console.log("Hello!"); } //function declaration sayHello(); //function call ```
Nicht-parametrisierte Funktionen	Die Funktionen, die keine Parameter benötigen, um zu funktionieren.	``` function greet() { const greeting = "Hello, World!"; console.log(greeting); } // Call the non-parameterized function greet(); // This will print "Hello, World!" to the console ```
Parametrisierte Funktionen	Die Funktion, die einen oder mehrere Werte akzeptiert, die Eingabedaten für die Funktion bereitstellen. Diese Werte in der Deklaration der Funktion werden Parameter genannt, und beim Aufruf der Funktion werden sie Argumente genannt.	``` <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0"> <title>Document</title> </head> <body> <p id="functiondata1"></p> <script> function add(a, b) { return a + b; } document.getElementById('functiondata1').innerHTML = add(3, 4); </script> </body> </html> ```
Benannte Funktion	Die Funktionen mit einem spezifischen Namen, die unter diesem Namen aufgerufen werden können.	``` const add = function(a, b) { console.log(a+b); } //name of the function is add add(2, 3); ```

IIFE	Eine sofort aufgerufene Funktionsausdruck ist eine Funktion in JavaScript, die definiert und sofort nach ihrer Erstellung ausgeführt wird.	`(function sayWelcome() {` `  console.log("Welcome!");` `})();`
Pfeilfunktion	Pfeilfunktionen in JavaScript sind eine prägnante Möglichkeit, Funktionsausdrücke mit der => Syntax zu schreiben.	`const arrowFunc = (a, b) => a + b;` `console.log(arrowFunc(5, 3));`
return	Die Rückgabeanweisung in JavaScript wird verwendet, um die Ausführung einer Funktion zu beenden und den Wert anzugeben, den die Funktion an den Aufrufer zurückgeben soll.	`<!DOCTYPE html>` `<html lang="en">` `<head>` `  <meta charset="UTF-8">` `  <meta name="viewport" content="width=device-width, initial-scale=1.0">` `  <title>Document</title>` `</head>` `<body>` `  <p id="showmessage"></p>` `  <script>` `  function multiply(message) {` `  return message; // Returns the product of a and b` `  }` `  document.getElementById('showmessage').innerHTML = multiply('Hard work is the key');` `  </script>` `</body>` `</html>`
Funktionsschluss	Ein Funktionsschluss in JavaScript ermöglicht es einer Funktion, auf Variablen aus ihrem äußeren Gültigkeitsbereich zuzugreifen und diese zu merken, selbst nachdem dieser Gültigkeitsbereich abgeschlossen ist.	`function outerFunction() {` `  const outerVar = "I am from the outer function";` `  function innerFunction() {` `  console.log(outerVar); // innerFunction can access outerVar` `  }` `  return innerFunction;` `}` `const closure = outerFunction();` `closure(); // This will log "I am from the outer function"`
Funktions-Hoisting	Funktions-Hoisting bedeutet, dass Funktionsdeklarationen während der Kompilierungsphase an den Anfang ihres umschließenden Geltungsbereichs verschoben werden, sodass sie verwendet werden können, bevor sie im Code deklariert sind.	`sayHello(); // This works even though the function is called before it's declared` `function sayHello() {` `  console.log("Hello!");` `}`

Funktions-Hoisting für Funktionsausdrücke	Funktionsausdrücke, bei denen eine Funktion einer Variablen zugewiesen wird, zeigen kein Hoisting-Verhalten.	`greet(); // This will result in an error` `const greet = function() {` `  console.log("Greetings!");` `};`
addEventListener	addEventListener ist eine JavaScript-Methode, die verwendet wird, um eine Funktion zuzuweisen, die ausgeführt wird, wenn ein bestimmtes Ereignis auf einem Element im DOM auftritt.	`<!DOCTYPE html>` `<html lang="en">` `<head>` `  <meta charset="UTF-8">` `  <meta name="viewport" content="width=device-width, initial-scale=1.0">` `  <title>Document</title>` `</head>` `<body>` `  <p id="btnclick"></p>` `  <button id="btn">Click Me</button>` `  <script>` `  // Get the element by its ID` `  const button = document.getElementById('btn');` `  // Add an event listener for the 'click' event` `  button.addEventListener('click', () => {` `  document.getElementById('btnclick').innerHTML = 'Button clicked!';` `  });` `  </script>` `</body>` `</html>`
onclick-Ereignis	Eine Möglichkeit, eine Funktion direkt einem HTML-Element zuzuweisen, die ausgeführt wird, wenn es angeklickt wird.	`<!DOCTYPE html>` `<html lang="en">` `<head>` `  <meta charset="UTF-8">` `  <meta name="viewport" content="width=device-width, initial-scale=1.0">` `  <title>Document</title>` `</head>` `<body>` `  <button onclick="myFunction()">Click me</button>` `  <script>` `  function myFunction() {` `  alert('Button clicked!');` `  }` `  </script>` `</body>` `</html>`
Mouseover-Ereignis	Das Mouseover-Ereignis wird ausgelöst, wenn der Mauszeiger ein Element betritt.	`<!DOCTYPE html>` `<html lang="en">` `<head>` `  <meta charset="UTF-8">` `  <meta name="viewport" content="width=device-width, initial-scale=1.0">` `  <title>Document</title>` `</head>` `<body>` `  <div id="myDiv" style="width: 200px; height: 200px; background-color: lightblue;"></div>` `  <script>` `  const myDiv = document.getElementById('myDiv');` `  // Adding a mouseover event listener` `  myDiv.addEventListener('mouseover', () => {` `  myDiv.style.backgroundColor = 'lightgreen';` `  });` `  </script>` `</body>` `</html>`

mouseout Ereignis	Das mouseout-Ereignis in JavaScript wird ausgelöst, wenn der Mauszeiger aus einem Element herausbewegt wird, was anzeigt, dass die Maus sich nicht mehr über diesem bestimmten Element befindet.	`<!DOCTYPE html>` `<html lang="en">` `<head>` `<meta charset="UTF-8">` `<meta name="viewport" content="width=device-width, initial-scale=1.0">` `<title>Document</title>` `</head>` `<body>` `<div id="myDiv" style="width: 200px; height: 200px; background-color: lightblue;"></div>` `<script>` `const myDiv = document.getElementById('myDiv');` `// Adding a mouseover event listener` `myDiv.addEventListener('mouseover', () => {` `myDiv.style.backgroundColor = 'lightgreen';` `});` `myDiv.addEventListener('mouseout', () => {` `myDiv.style.backgroundColor = 'lightcoral';` `});` `</script>` `</body>` `</html>`
Keydown-Ereignis	Das Keydown-Ereignis wird ausgelöst, wenn eine Taste auf der Tastatur gedrückt wird.	`<!DOCTYPE html>` `<html>` `<head>` `<title>Keydown Event Handling</title>` `</head>` `<body>` `<input type="text" id="myInput">` `<p id="output"></p>` `<script>` `const input = document.getElementById("myInput");` `const output = document.getElementById("output");` `input.onkeydown = function(event) {` `output.textContent = `Key pressed: ${event.key}`;` `};` `</script>` `</body>` `</html>`
Änderungsereignis	Das Änderungsereignis wird ausgelöst, wenn sich der Wert eines Eingabeelements ändert. Typischerweise wird es für Formularelemente wie Textfelder oder Dropdowns verwendet.	`<!DOCTYPE html>` `<html>` `<head>` `<title>Change Event Handling</title>` `</head>` `<body>` `<input type="text" id="myInput">` `<p id="output"></p>` `<script>` `const input = document.getElementById("myInput");` `const output = document.getElementById("output");` `input.onchange = function() {` `output.textContent = `Value changed to: ${input.value}`;` `};` `</script>` `</body>` `</html>`
onsubmit-Ereignis	Das onsubmit-Ereignis in HTML tritt auf, wenn ein Formular über einen Senden-Button oder durch Aufrufen von submit() gesendet wird.	`<!DOCTYPE html>` `<html>` `<head>` `<title>Form Submission Example</title>` `</head>` `<body>` `<form id="myForm" onsubmit="validateForm()">` `<label for="name">Name:</label>` `<input type="text" id="name" name="name">  ` `<label for="email">Email:</label>`

```
    <input type="email" id="email" name="email"><br><br>
    <input type="submit" value="Submit">
  </form>
  <script>
    function validateForm() {
      // Prevent the default form submission
      event.preventDefault();
      // Retrieve form values
      const name = document.getElementById('name').value;
      const email = document.getElementById('email').value;
      // Perform validation (for example, checking if fields are filled)
      if (name === '' || email === '') {
        alert('Please fill in all fields.');
        return false; // Prevent form submission if validation fails
      }
      // If validation passes, continue with form submission
      alert('Form submitted successfully!');
    }
  </script>
</body>
</html>
```

Skills Network