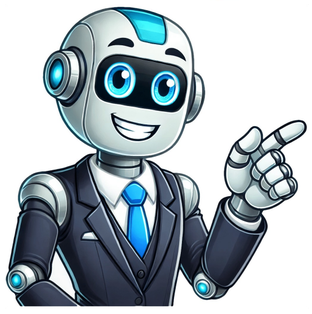
Continue



Given article text here JavaScript is a cornerstone of modern web development, offering flexibility to create dynamic user experiences. This comprehensive javascript cheatsheet covers the basics to advanced concepts, providing quick reference and actionable insights for both beginners and experienced developers. Use constructor functions to create objects with a blueprint for inheritance. You can use extends to inherit properties from other classes. Making HTTP requests is easy with fetch(url).then(res => res.json());. This approach supports modern promise-based syntax. For data persistence, you can utilize LocalStorage or SessionStorage. When working with objects, use Object.keys(obj), Object.values(obj) to retrieve keys and values effectively. You can also create new Map() or Set() instances for efficient key-value pair management or unique value storage. To validate patterns in strings, use Regular Expressions like /pattern/.test(string). For more precise control over function execution, consider using Debouncing (const fn = debounce(func, delay)) to limit the frequency of a function's calls. Alternatively, Throttling (const fn = throttle(func, delay)) ensures a function runs at most once per specified interval. Understanding how JavaScript handles async operations is crucial. The Event Loop explains the single-threaded model with non-blocking asynchronous APIs. It's essential to grasp tasks, microtasks, and priorities in JavaScript. When working with callbacks, remember that arrow functions + This retain this from enclosing contexts, making them useful in such scenarios. For safely accessing nested properties, use Optional Chaining (obj?.prop?.nestedProp). Avoid runtime errors on null or undefined values by utilizing this approach. Lazy loading of modules can be achieved through Dynamic Imports (import('./module.js').then(module => { })), which is useful for performance optimization. In function calls, take advantage of Optional Parameters to set default values (function fn(a, b = 10) { }). This feature provides flexibility in how functions are called. When using SetTimeout, remember you can pass additional arguments to the callback function with setTimeout(fn, ms, arg1, arg2). GlobalThis offers a standardized way to access the global object across different environments like browsers and Node.js. The Intl API is another powerful tool for internationalization, enabling formats of numbers, dates, and strings. For optimizing memory usage, WeakMap and WeakSet can hold objects without preventing garbage collection, making them ideal for private properties and data storage. Symbol Data Type provides unique and immutable values useful for creating custom object properties. Proxy Objects allow intercepting and customizing operations on objects. In JavaScript, the BigInt type represents integers with arbitrary precision, suitable for calculations beyond Number.MAX_SAFE_INTEGER. Dynamic Property Keys can be created dynamically with const obj = { [keyName]: value };, enabling flexibility in object creation. JavaScript is a versatile language that supports both front-end and back-end development. The cheat sheet covers modern JavaScript features, such as template literals, arrow functions, and the spread operator, which are introduced in ES6 and later versions. Common array methods like .map(), .filter(), and .reduce() are also included to help with data manipulation. In addition to front-end development, JavaScript is used for back-end development with frameworks like Node.js and mobile app building with tools like React Native. The Intl.NumberFormat and Intl.DateTimeFormat APIs can be utilized to format numbers and dates according to different locales. JavaScript provides efficient ways to handle asynchronous tasks using Promises, Async/Await, and the Fetch API. For basic syntax and introduction, HTML CSS JS includes a guide on how to use JavaScript in web development. The provided code examples demonstrate various JavaScript concepts, including variables, operators, functions, loops, conditional statements, and object-oriented programming. The examples cover topics such as: * Variables and data types: strings, numbers, booleans, arrays, and regular expressions * Operators: arithmetic, comparison, logical, assignment, and bitwise * Functions: defining and calling functions, arguments, and return values * Loops: for loops, while loops, and do-while loops * Conditional statements: if-else statements and switch statements * Object-oriented programming: creating objects and using prototypes Let z = 'zzz'; "use strict"; x = 1; false, true // boolean, number, string, undefined, null, Infinity special a = b + c - d; // addition, subtraction a = b * (c / d); // multiplication, division x = 100 % 48; // modulo. 100 / 48 remainder = 4 a++; b--; // postfix increment and decrement & AND 5 & 1 | OR 5 | 1 ^ XOR 5 ^ 1 ~ NOT ~ 5 ^ 10 > right shift 5 >>> zero fill right shift 5 person.age // member person[age] // member !(a == b) // logical not a != b // not equal typeof a // type (number, object, function...) x > 3 // minary shifting a = b // assignment a == b // equals a != b // unequal a === b // strict equal a !== b // strict unequal a < b a > b // less and greater than a = b // less or equal, greater or eq a += b // a = a + b (works with - * %...) a && b // logical and a || b // logical or We show ads to support this free website. JavaScript variables and objects var age = 18; var name = "Jane"; var name = {first:"Jane", last:"Doe"}; var truth = false; var sheets = ["HTML","CSS","JS"]; var a; typeof a; var a = null; var student = {name firstName:"Jane", lastName:"Doe", age:18, height:170, fullName : function() { return this.firstName + " " + this.lastName; }}; student.age = 19; student[age]++; name = student.fullName(); Selectors specify which elements are targeted with a style var abc = "abcdefghijklmnopqrstuvwxyz"; var esc = 'I don\'t know'; var len = abc.length; abc.indexOf("lmno"); abc.lastIndexOf("lmno"); abc.slice(3, 6); abc.replace("abc","123"); abc.toUpperCase(); abc.toLowerCase(); abc.concat(" ", str2); abc.charAt(2); abc[2]; abc.charCodeAt(2); abc.split(","); abc.split(""); 128.toString(16); Event handlers in JavaScript. Click here **Event Handlers and Math Operations** JavaScript provides various event handlers that can be triggered by user interactions, such as `onpagehide`, `onscroll`, and `onsubmit`. These events can be used to create dynamic web applications. When working with numbers in JavaScript, there are several built-in functions available. The `Number` function can convert a value to a number, while the `parseInt` and `parseFloat` functions can parse strings as integers or floating-point numbers, respectively. The `Math` object provides various mathematical constants, such as `PI`, `E`, and `SQRT2`. It also offers methods for performing calculations, including exponentiation, logarithms, and trigonometric functions. **Working with Dates** JavaScript's `Date` object allows you to work with dates and times. You can create a new date by passing a string in the format `YYYY-MM-DD` or `YYYY-MM-DDTHH:MM:SSZ`. The `Date` constructor also supports various methods for getting and setting date values, such as `getFullYear()`, `getMonth()`, `getDate()`, and `setDate()`. Additionally, you can use methods like `getTime()` to get the number of milliseconds since 1970. **Key Concepts** * Event handlers: functions that are triggered by user interactions * Math operations: calculations using built-in functions like `Math.pow` and `Math.log` * Date object: working with dates and times in JavaScript * Constants: mathematical constants like `PI` and `E` Note that I've tried to maintain the original structure and organization of the text, while making it more concise and easier to read. arrays: multiple values in a single variable var dogs = ["Bulldog", "Beagle", "Labrador"]; var dogs = new Array("Bulldog", "Beagle", "Labrador"); // declaration alert(dogs[1]); // access value at index, first item being [0] dogs[0] = "Bull Terier"; // change the first item for (var i = 0; i < dogs.length; i++) { // parsing with array.length console.log(dogs[i]); } dogs.toString(); // convert to string: results "Bulldog,Beagle,Labrador" dogs.join(" * "); // join: "Bulldog * Beagle * Labrador" dogs.pop(); // remove last element dogs.push("Chihuahua"); // add new element to the end dogs[dogs.length] = "Chihuahua"; // the same as push dogs.shift(); // remove first element dogs.unshift("Chihuahua"); // add new element to the beginning delete dogs[0]; // change element to undefined (not recommended) dogs.splice(2, 0, "Pug", "Boxer"); // add elements (where, how many to remove, element list) var animals = dogs.concat(cats,birds); // join two arrays (dogs followed by cats and birds) dogs.slice(1,4); // elements from [1] to [4-1] dogs.sort(); // sort string alphabetically dogs.reverse(); // sort string in descending order x.sort(function(a, b){return a - b}); // numeric sort x.sort(function(a, b){return b - a}); // numeric descending sort highest = x[0]; // first item in sorted array is the lowest (or highest) value x.sort(function(a, b){return 0.5 - Math.random()}); // random order sort concat, copyWithin, every, fill, filter, find, findIndex, forEach, indexOf, isArray, join, lastIndexOf, map, pop, push, reduce, reduceRight, reverse, shift, slice, some, sort, splice, toString, unshift, valueOf Common JS functions. eval(); // executes a string as if it was script code String(23); // return string from number (23).toString(); // return string from number Number("23"); // return number from string decodeURI(enc); // decode URI. Result: "my page.asp" encodeURI(uri); // encode URI. Result: "my%page.asp" decodeURIComponent(enc); // decode a URI component encodeURIComponent(uri); // encode a URI component isFinite(); // is variable a finite, legal number isNaN(); // is variable an illegal number parseFloat(); // returns floating point number of string parseInt(); // parses a string and returns an integer JavaScript regular expression cheatsheet. var a = str.search(/CheatSheet/i); \ Escape character \d find a digit \s find a whitespace character \b find match at beginning or end of a word n+ contains at least one n n* contains zero or more occurrences of n n? contains zero or one occurrences of n ^ Start of string $ End of string \uxxxx find the Unicode character . Any single character (a|b) a or b (...) Group section [abc] In range (a, b or c) [0-9] any of the digits between the brackets [^abc] Not in range \s White space a? Zero or one of a a* Zero or more of a a*? Zero or more, ungreedy a+ One or more of a a+? One or more, ungreedy a{2} Exactly 2 of a a{2,} 2 or more of a a{,5} Up to 5 of a a{2,5} 2 to 5 of a a{2,5}? 2 to 5 of a, ungreedy [:punct:] Any punctuation symbol [:space:] Any We're showing ads here to keep this website free. JavaScript error handling involves using try, catch, throw, and finally blocks to manage errors. Try block: try { undefinedFunction(); // attempt code execution } Catch block: catch(err) { console.log(err.message); // display error message } Throwing an error: throw "My error message"; // send a custom error text Error handling for input values: var x = document.getElementById("mynum").value; try { if(x == "") throw "empty"; if(isNaN(x)) throw "not a number"; x = Number(x); if(x > 10) throw "too high"; } catch(err) { document.write("Input is " + err); // display error console.error(err); // write error to console } Finally block: finally { document.write(" Done"); // execute regardless of try/catch result } Which help in executing one or more statements up to a desired number of times are Find the "for" and "while" loop syntax in this section If - Else statements – Conditional statements are used to perform different actions based on different conditions Variables can use variables such as numbers strings and arrays etc. and learn the operators