Master Javascript Cheat Sheet

by xenaxon via cheatography.com/196679/cs/41372/

Data Types

String	a series of characters
Number	any numeric value up to 16 in length (check Number.MAX_SAFE_INTEGER)
Boolean	either true or false
Object	data structure of key-value pairs or class instances. Arrays are also objects in JS
Undefined	variable value not defined, type not defined
Null	used mainly for objects for referring absence of any object value and if any function or variable returns null, then we can infer that the object could not be created
Symbol	a data type that is a unique value. usage: Symbol() or Symbol(label) or Symbol.for(label). Youtube Explained: 4J5hnOCj69w
Bigint	used to store big integer values that are too big to be represented by Number type

Variables

var	- global scoped or function scoped - it is hoisted -can be used without being declared -it also binds to 'this'
var redeclaring	Because var is function scoped and not Block scoped, redeclaring a variable inside a block will also redeclare the variable outside the block
let	- block scoped variable
const	- block scoped variable -cannot be reassigned. -must be assigned a value - const doesnt mean it's a constant value. It is a constant REFERENCE to a value. -you can't reassign it a different value/object/array, but you can change it's object or array elements and properties Other than that, it behaves same as let

Variables (cont)

let and const	let and const have block scope. let and const can not be redeclared. let and const must be declared before use. let and const does not bind to this. let and const are not hoisted.
var VS let	- Variables declared by let are only available inside the block where they're defined. - Variables declared by var are available throughout the function in which they're declared. - Variables declared with var can be redeclared, ones with let canoot. they will give an error
global VS local	Global and local variables with the same name are different variables. Modifying one, does not modify the other.
Variable Lifetime	- While identifier is reachable, it will not be deleted from memory. - Global variables live until the page is discarded / tab closed / page changed / etc - Local variables are deleted when the function is completed - In CLOSURES, as long as the variable is still accessible, it will not be removed.

Functions

Function Declaration	function funcA (parameters) { statements; return value) If it starts with function keyword, it's a function declaration
Function Expression	const funcA = function (parameters) { statements; return value) If name of function is omitted, then it's a function expression

Master Javascript Cheat Sheet

by xenaxon via cheatography.com/196679/cs/41372/

Functions (cont)

Anonymous Function	(function () { ... }); An anonymous function is a function without a name. So the function inside a function expression is also an anonymous function
Arrow Function	const arrowFunc = (params) => statement It is a Shorthand for function expression. - Single Line arrow Functions can ommit return statement and just return the result of the statement directly!
Arrow Function (with multiple statements body)	const arrowFunc = (params) => {statement; return value} - If the anonymous function has multiple statements, it needs to be surrounded by {} and must contain a return statement in order to return a value
Self Invoking Function	(function(params){ body })()

Closure	const add = (function () {
	let counter = 0;
	return function () {
	counter += 1;
	return counter}
	})();
	- It makes it possible for a function to have "private" variables. The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

Functions (cont)

() Operator	The () Operator invokes (calls) a function. The function parameters are passed between the round braces. Example myFunc() or myFunc(param1, param2, etc...)
let myVar = myFunc vs myVar = myFunc()	myFunc reffers to the function Object myFunc() reffers to the function result after invoking it
let totalFields = function (rows=10, columns=2) { return rows*columns }	If we invoke totalFields, without parameters we will get back 20, because default values will be used
Function 'arguments' variable	Every function has a built in variable arguments that will contain an array of all the parameters passed upon invocation. - This can be used inside the function to create custom logic for parameters missing or of different value than expected, or can be used to permit the function to receive an unknown number of parameters. - $arguments can be iterated like a normal array - If arguments are not passed, then the arguments array will be empty
Function combined output func(a)(b)	fun(a)(b) will return a combined output by using both the parameter in the parent function and the parameter in the child function.

Native Array Methods

Array Creation Methods	
Array.from()	Creates a new array instance from an array-like or iterable object.
Array.isArray()	Returns true if the argument is an array.
Array.of()	Creates a new array instance with a variable number of arguments, regardless of number or type of the arguments.
Array Mutator Methods	
Array.prototype.copyWithin()	Copies part of an array to another location in the same array.
Array.prototype.fill()	Fills all the elements of an array from a start index to an end index with a static value.
Array.prototype.pop()	Removes the last element from an array and returns that element.
Array.prototype.push()	Adds one or more elements to the end of an array and returns the new length of the array.
Array.prototype.reverse()	Reverses the order of the elements of an array in place.
Array.prototype.shift()	Removes the first element from an array and returns that element.
Array.prototype.sort()	Sorts the elements of an array in place and returns the array.
Array.prototype.splice()	Adds and/or removes elements from an array.
Array.prototype.unshift()	Adds one or more elements to the beginning of an array and returns the new length of the array.
Accessor Methods	
Array.prototype.concat()	Merges two or more arrays.

Native Array Methods (cont)

Array.prototype.includes()	Determines whether an array includes a certain value among its entries.
Array.prototype.indexOf()	Returns the first index at which a given element can be found in the array.
Array.prototype.join()	Joins all elements of an array into a string.
Array.prototype.lastIndexOf()	Returns the last index at which a given element can be found in the array.
Array.prototype.slice()	Returns a shallow copy of a portion of an array into a new array object.
Array.prototype.toString()	Returns a string representing the array and its elements.
Array.prototype.toLocaleString()	Returns a localized string representing the array and its elements.
Iteration Methods	
Array.prototype.entries()	Returns a new array iterator object that contains the key/value pairs for each index in the array.
Array.prototype.every()	Tests whether all elements in the array pass the test implemented by the provided function.
Array.prototype.filter()	Creates a new array with all elements that pass the test implemented by the provided function.
Array.prototype.find()	Returns the value of the first element in the array that satisfies the provided testing function.
Array.prototype.findIndex()	Returns the index of the first element in the array that satisfies the provided testing function.
Array.prototype.forEach()	Executes a provided function once for each array element.

Native Array Methods (cont)

Array.prototype.keys()	Returns a new array iterator that contains the keys for each index in the array.
Array.prototype.map()	Creates a new array with the results of calling a provided function on every element in the calling array.
Array.prototype.reduce()	Executes a reducer function on each element of the array, resulting in a single output value.
Array.prototype.reduceRight()	Executes a reducer function on each element of the array, from right to left, resulting in a single output value.
Array.prototype.some()	Tests whether at least one element in the array passes the test implemented by the provided function.
Array.prototype.values()	Returns a new array iterator object that contains the values for each index in the array.
Array.prototype.flat()	Creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.
Array.prototype.flatMap()	Maps each element using a mapping function, then flattens the result into a new array.
Array.prototype[@@iterator]()	Returns the default iterator for an array, which is the same as the `values()` method.

Arithmetic & Assignment Operators

+	Addition
x + y (numbers)	Adds x and y together and returns the sum x and y are called Operands and + is called Operator
x + y (strings)	Concatenation Operator (joins two strings into one)

Arithmetic & Assignment Operators (cont)

x+y (string + number)	Using the + operator with strings and numbers at same time will result in string concatenation. Example: "Hello"+5 = "Hello5"{{nl}}-In an operation, if multiple number operands preceed a string, the numbers will be added together until the js interpreter reaches the string, then it concatenates the sum with the string. Example: 16 + 4 + "Volvo" = "20Volvo"
-	Subtraction
*	Multiplication
**	Exponentiation (raise to power) Ex: 2**4
/	Division
%	Modulus (division remainder) ex: 10%4=2 (10/4=2.5, (0.5*4 is remainder) so whatever is not a full number is added, and the sum is what is called "remainder")
++	Increment ++x will add 1 and return x x++ will return x and then add 1 to it Example: let x = 0; let count = x++ will set count to 0 and x to 1.
--	Decrement - same behavior as ++x and x++ subtracts instead.
=	Assignment Operator
x+=y	Shorthand for: x = x + y (same for all other operators)

Comparison Operators

==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to

Comparison Operators (cont)

<=	less than or equal to
?	ternary operator

Logical Operators

&&	logical and
||	logical or
!	logical not
TODO	ADD LOGICAL ASSIGNMENT OPERATORS https://www.w3schools.com/js/js_assignment.asp

The Logical Operators are used in logical statements to test multiple conditions alternatively, simultaneously or to negate a condition.
Example:
if (1<5 && 6<10) console.log(true)
if (!(6<5)) console.log(true)

Negation operator must be used in combination with round braces over the expression which needs to be negated. Otherwise for example: (!0<5) will test if 0 is not true. not if 0<5 is not true.

Bitwise Operators

TODO: I need to add this later

Objects

Key Value Pairs	Objects are key: value pairs that can have: - any valid value as key - can contain any data type as value
Properties & Methods	Objects can contain: -properties (variables,constants) -methods (local functions, stored in properties as well, as function definitions)
Classes (ES6)	- Classes where introduced in ECMAScript2015 (ES6) - Objects can be instances of a class
Arrays	In Javascript, arrays are not a primitive data type. Arrays are Objects as well with index based key:value pairs.
Associ-ative Arrays	Objects can be used as Associative Arrays, by using key value pairs and accessing them using the object-Name["key"] syntax

Objects (cont)

Built-In Objects	Javascript has several Built-In Objects including: - Date -JSON -Math - and other (check: https://developer.mozilla.org/en-US/-docs/Web/JavaScript/Reference/Global_Objects)
Object Property Access	Object Properties can be accessed in 2 ways: - objectName["propertyName"] objectName.propertyName - objectName["propertyName"] by this way we can access properties with STRING key names. as opposed to using "." (dot) with which we can only access properties with keys that have valid variable names. As said above, this way we can use objects as associative arrays and even more complex, multidimensional arrays or combinations of arrays/objects on multiple dimensions.
this	This is a keyword in javascript which refers to a specific object depending on the context in which it is used. More on this in the section dedicated to this.

this

this	In JavaScript, the this keyword refers to an object.
Inside Object Method	In an object method, this refers to the object.
Inside Function	In a function, this refers to the global object.
In Function (Strict Mode)	In a function, in strict mode, this is undefined.
Inside Event Handler	In an event, this refers to the element that received the event.

this (cont)

Special Object Methods	Methods like call(), apply(), and bind() can refer this to any object. Example: person1.fullName.call(person2); will call fullName as if we did person2.fullName with the function definition in person1
This Immutable	This is not a variable, it is a keyword that refers to an object. This cannot be changed or reassigned
Global this	When used alone, this refers to the global object. -In a browser window the global object is [object Window]: