

JavaScript cheatsheet – v5.5.2 – <https://github.com/Serrin/Celestra/>

Web Storage api and JSON	element.dataset & data-* attributes	TypedArray
<p>IE8+</p> <p>localStorage: localStorage.length; localStorage.key(index); localStorage.getItem(key); localStorage.setItem(key, data); localStorage.removeItem(key); localStorage.clear();</p> <p>sessionStorage: sessionStorage.length; sessionStorage.key(index); sessionStorage.getItem(key); sessionStorage.setItem(key, data); sessionStorage.removeItem(key); sessionStorage.clear();</p> <p>hasItem: localStorage.getItem(key) !== null sessionStorage.getItem(key) !== null</p> <p>setJSON: localStorage.setItem(key, JSON.stringify(object)); sessionStorage.setItem(key, JSON.stringify(object));</p> <p>getJSON: JSON.parse(localStorage.getItem(key)); JSON.parse(sessionStorage.getItem(key));</p>	<p>- IE11 compatible - element data-* attributes - no methods and events</p> <p>camelcase: element.data-name -> element.dataset.name element.data-first-second -> element.dataset.firstSecond</p> <p>set: element.dataset.name = "value"; element.dataset["name"] = "value"; element.setAttribute("data-name", "value"); element["data-name"] = "value";</p> <p>get: element.dataset.name; element.dataset["name"]; element.getAttribute("data-name"); element["data-name"];</p> <p>remove: element.removeAttribute("data-name");</p> <p>check: element.hasAttribute("data-name");</p>	<p>IE10+11 compatible</p> <p>new <TypedArray>(); <i>ES2017</i> new <TypedArray>(length); new <TypedArray>(typedArray); new <TypedArray>(object); new <TypedArray>(buffer[,byteOffset[,len]]);</p> <p>Int8Array(); -128 to 127, 1 byte, int8_t, Shortint</p> <p>Uint8Array(); 0 to 255, 1 byte, uint8_t, Byte</p> <p>Uint8ClampedArray(); - not in IE10-11 0 to 255, 1 byte, uint8_t, Byte</p> <p>Int16Array(); -32768 to 32767, 2 byte, int16_t, Smallint</p> <p>Uint16Array(); 0 to 65535, 2 byte, uint16_t, Word</p> <p>Int32Array(); -2147483648 to 2147483647, 4 byte, int32_t</p> <p>Uint32Array(); 0 to 4294967295, 4 byte, uint32_t, Longword</p> <p>BigInt64Array(); - not in IE10-11 -2**63 to 2**63-1, 8 byte, int64_t, Int64</p> <p>BigUint64Array(); - not in IE10-11 0 to 2**64-1, 8 byte, uint64_t, QWord</p> <p>Float32Array(); 1.2x10-38 to 3.4x1038, 4 byte, float, Real</p> <p>Float64Array(); 5.0x10-324 to 1.8x10308, 8 byte, Double</p>
DOM events		
<p>target.addEventListener(<type>,<listener>[,useCapture]); or target.addEventListener(<type>,<listener>[,options]); target.removeEventListener(<type>,<listener>[,useCapture]); or target.removeEventListener(<type>,<listener>[,options]); target.dispatchEvent(<event>); target.type(); or target["type"]();</p>		

element.classList	JSON
<p>IE10+IE11 don't have support for classList on SVG or MathML elements.</p> <p>element.classList.add(String[,String]); IE10+11: yes (except the multiple arguments)</p> <p>element.classList.remove(String[,String]); IE10+11: yes (except the multiple arguments) - Removing a class that does not exist, does NOT throw an error.</p> <p>element.classList.contains(String); IE10+11: yes</p> <p>element.classList.toggle(String[,force]); IE10+11: yes (except the second argument) - When only one argument is present: Toggle class value; if class exists then remove it and return false, if not, then add it and return true. - When a second argument is present: If the second argument evaluates to true, add specified class value, and if it evaluates to false, remove it.</p> <p>element.classList.item(Number); IE10+11: yes</p> <p>element.classList.length; IE10+11: yes</p> <p>element.classList.replace(oldClass, newClass); IE10+11: No and the method isn't compatible with the Safari and mobile browsers too.</p> <p>Remove all classes: element.className = "";</p>	<p>IE8+</p> <p>Valid Data Types</p> <ul style="list-style-type: none"> - string - number - object (containing valid JSON values) - array - boolean - date - null <p>Invalid Data Types</p> <ul style="list-style-type: none"> - function - Symbol - NaN, Infinity, undefined - <i>will be "null"</i> - an object with method(s) (functions) - Map, Set, WeakMap, WeakSet - <i>fix: convert to array</i> - BigInt - <i>fixed in Celestra - BigInt.prototype.toJSON();</i> <p>JSON.stringify(value[,replacer[,space]]); Convert a JavaScript object to a JSON string.</p> <p>JSON.stringify({ a: 1, b: "2", c: true }); // -> '{"a":1,"b":"2","c":true}'</p> <p>JSON.stringify([1, 2, 3, 4, 5]); // -> "[1,2,3,4,5]"</p> <p>JSON.parse(text[,reviver]); Parses a JSON string and returns a JavaScript object.</p> <p>JSON.parse(JSON.stringify({a: 1, b: "2", c: true})); // -> Object { a: 1, b: "2", c: true }</p> <p>JSON.parse(JSON.stringify([1, 2, 3, 4, 5])); // -> Array(5) [1, 2, 3, 4, 5]</p>

Fetch	Fetch POST
<pre> Firefox 39, Chrome 42, Edgel4, Opera29 , Safari 10.1, Samsung I. 4.0 // Example GET method implementation with TEXT: fetch("https://api.coindesk.com/v1/bpi/currentprice.json") .then(response => response.text()) .then(data => console.log(data)) .catch(error => console.log(error)); // Example GET method implementation with JSON: fetch("https://api.coindesk.com/v1/bpi/currentprice.json") .then(response => response.json()) .then(data => console.log(data.bpi.USD.rate)) .catch(error => console.log(error)); // Example GET method implementation with TEXT and JSON: fetch("https://api.coindesk.com/v1/bpi/currentprice.json") .then(response => response.text()) .then(text => console.log(JSON.parse(text).bpi.USD.rate+"\n"+text)) .catch(error => console.log(error)); // Example POST method implementation with upload JSON data: const data = { username: "example" }; fetch("https://example.com/profile", { method: "POST", // or "PUT" headers: { "Content-Type": "application/json", }, body: JSON.stringify(data), }) .then(response => response.json()) .then(data => { console.log("Success:", data); }) .catch((error) => { console.error("Error:", error); }); </pre>	<pre> // Example POST method implementation: // Default options are marked with * async function postData(url = "url", data = {}) { const response = await fetch(url, { method: "POST", // *GET, POST, PUT, DELETE, etc. mode: "cors", // no-cors, *cors, same-origin cache: "no-cache", // *default, no-cache, reload, force-cache, only-if-cached credentials: "same-origin", // include, *same-origin, omit headers: {"Content-Type": "application/json"}, // "Content-Type": "application/x-www-form-urlencoded" redirect: "follow", // manual, *follow, error referrerPolicy: "no-referrer", // no-referrer, *no-referrer-when-downgrade, origin, origin-when-cross-origin, same-origin, strict-origin, strict-origin-when-cross-origin, unsafe-url body: JSON.stringify(data) // body data type must match "Content-Type" header }); return response.json(); // parses JSON response into native JavaScript objects } postData("https://example.com/answer", { answer: 42 }) .then(data => { console.log(data); }); // JSON data parsed by `data.json()` call </pre>

Nullish coalescing operator <code>x ?? y</code>	Logical nullish assignment <code>x ??= y</code>	Logical AND assignment <code>x &&= y</code>	Logical OR assignment <code>x = y</code>
FF 72, Chrome and Edge 80, Safari 13.1, Safari on iOS 13.4, Samsung Internet 13	FF 79, Chrome and Edge 85, Safari 14, Samsung Internet 14		
The nullish coalescing operator (??) is a logical operator that returns its right-hand side operand when its left-hand side operand is null or undefined, and otherwise returns its left-hand side operand.	The logical nullish assignment operator only assigns if x is nullish (null or undefined).	The logical AND assignment operator only assigns if x is truthy.	The logical OR assignment operator only assigns if x is falsy. (false, 0, -0, 0n, "", '', ``, null, undefined, NaN)
<pre>const nullValue = null; const emptyText = ""; // falsy const someNumber = 42; const valA = nullValue ?? "defaultA"; // "defaultA" const valB = emptyText ?? "default B"; // "" (empty string is not null or undefined) const valC = someNumber ?? 0; // 42</pre>	<pre>function config (options) { options.duration ??= 100; options.speed ??= 25; return options; } config({duration: 125}); // {duration: 125, speed: 25} config({}); // {duration: 100, speed: 25}</pre>	<pre>let x = 0; let y = 1; x &&= 0; // 0 x &&= 1; // 0 y &&= 1; // 1 y &&= 0; // 0</pre>	<pre>const a = { duration: 50, title: "" }; a.duration = 10; // 5 a.title = "title is empty."; // "title is empty"</pre>
<pre>let count = 0; let text = ""; let qty = count 42; // 42 let message = text "hi!"; // "hi!"</pre>	<pre>const a = { duration: 50 }; a.duration ??= 10; // 50 a.speed ??= 25; // 25</pre>	<pre>let a = 1; let b = 0; a &&= 2; // 2 b &&= 2; // 0</pre>	
		equivalent	not equivalent
Nullish coalescing operator (??)	<code>x ?? y</code>	<code>(x !== null) ? x : y</code>	
Logical nullish assignment (??=)	<code>x ??= y</code>	<code>x ?? (x = y);</code>	<code>x = x ?? y;</code>
Logical AND assignment (&&=)	<code>x &&= y</code>	<code>x && (x = y);</code>	<code>x = x && y;</code>
Logical OR assignment (=)	<code>x = y</code>	<code>x (x = y);</code>	<code>x = x y;</code>

Reflect object		
ES6, no IE11 support - FF 42, Chrome 49, Edge 12, Safari 10, Safari on iOS 10, Samsung Internet 5.0		
Function	Description	equivalent
<code>Reflect.apply(target, thisArgument, argumentsList);</code>	Calls a target function with arguments as specified by the argumentsList parameter.	<code>Function.prototype.apply();</code>
<code>Reflect.construct(target, argumentsList [,newTarget]);</code>	The new operator as a function.	<code>new target(...argumentsList);</code>
<code>Reflect.defineProperty(target,propertyKey, attributes);</code>	Similar to <code>Object.defineProperty()</code> . Returns a boolean that is true if the property was successfully defined.	<code>Object.defineProperty(target, propertyKey, attributes);</code>
<code>Reflect.deleteProperty(target,propertyKey);</code>	The delete operator as a function.	<code>delete target[propertyKey];</code>
<code>Reflect.get(target,propertyKey[,receiver]);</code>	Returns the value of the property of the object.	<code>target[propertyKey];</code>
<code>Reflect.getOwnPropertyDescriptor(target, propertyKey);</code>	Returns a property descriptor of the given property if it exists on the object, undefined otherwise.	<code>Object.getOwnPropertyDescriptor(target, propertyKey);</code>
<code>Reflect.getPrototypeOf(target);</code>	<code>Object.getPrototypeOf(target);</code>	<code>Object.getPrototypeOf(target);</code>
<code>Reflect.has(target,propertyKey);</code>	Returns a boolean whether the target has the property.	<code>propertyKey in target;</code>
<code>Reflect.isExtensible(target);</code>	Returns a boolean that is true if the target is extensible.	<code>Object.isExtensible(target);</code>
<code>Reflect.ownKeys(target);</code>	Returns an array of the target object's own (not inherited) property keys.	<code>Object.getOwnPropertyNames(target).concat(Object.getOwnPropertySymbols(target));</code>
<code>Reflect.preventExtensions(target);</code>	Prevents new properties from ever being added to an object. Similar to <code>Object.preventExtensions()</code> .	<code>Object.preventExtensions(target);</code>
<code>Reflect.set(target,propertyKey,value [,receiver]);</code>	Assigns values to properties. Returns a boolean that is true if the update was successful.	<code>target[propertyKey] = value;</code>
<code>Reflect.setPrototypeOf(target,prototype);</code>	Sets the prototype of an object. Returns a boolean that is true if the update was successful.	<code>Object.setPrototypeOf(target,prototype);</code>

Map Object	Set Object helper functions
<pre> var myMap = new Map([iterable]); // The Map objects are iterable. for (let [key, value] of myMap) { console.log(`\${key} = \${value}`); } var cloneMap = new Map(myMap); Map.prototype.size; Map.prototype.get(<key>); -> value/undefines Map.prototype.set(<key>,<value>); -> Map object Map.prototype.has(<key>); -> boolean Map.prototype.delete(<key>); -> boolean Map.prototype.clear(); -> undefined Map.prototype.forEach(function (value,key,map)); -> undefined Map.prototype.keys(); -> iterator of keys Map.prototype.values(); -> iterator of values Map.prototype.entries(); -> iterator of [key, value] </pre>	<pre> function isSuperset(set, subset) { for (const elem of subset) { if (!set.has(elem)) { return false; } } return true; } function union(setA, setB) { const _union = new Set(setA); for (const elem of setB) { _union.add(elem); } return _union; } function intersection(setA, setB) { const _intersection = new Set(); for (const elem of setB) { if (setA.has(elem)) { _intersection.add(elem); } } return _intersection; } function difference(setA, setB) { const _difference = new Set(setA); for (const elem of setB) { _difference.delete(elem); } return _difference; } function symmetricDifference(setA, setB) { const _d = new Set(setA); for (const e of setB) { if (_d.has(e)) { _d.delete(e); } else { _d.add(e); } } return _d; } const setA = new Set([1,2,3,4]), setB = new Set([2, 3]), setC = new Set([3, 4, 5, 6]) isSuperset(setA, setB); // true union(setA, setC); // Set {1, 2, 3, 4, 5, 6} intersection(setA, setC); // Set {3, 4} difference(setA, setC); // Set {1, 2} symmetricDifference(setA, setC); // Set {1, 2, 5, 6} </pre>
Set Object	
<pre> var mySet = new Set([iterable]); // The Set objects are iterable. for (const item of mySet) { console.log(item); } var cloneSet = new Set(mySet); Set.prototype.size; Set.prototype.add(<value>); -> Set object Set.prototype.has(<value>); -> boolean Set.prototype.delete(<value>); -> boolean Set.prototype.clear(); -> undefined Set.prototype.forEach(function (value,value,set)); -> undefined Set.prototype.keys(); -> iterator of values Set.prototype.values(); -> iterator of values Set.prototype.entries(); -> iterator of [value, value] </pre>	