

JavaScript cheatsheet – v7.0.1 – <https://github.com/Serrin/Celestra/>

Javascript data types			
Type	typeof value	Get real type	Subtypes (possible values)
undefined	"undefined"	typeof value === "undefined"	undefined
		const isUndefined=(value)=>typeof value==="undefined";	
null	" <i>object</i> "	value === null	null
		const isNull=(value)=>value===null;	
boolean	"boolean"	typeof value === "boolean"	true, false
		const isBoolean=(value)=>typeof value==="boolean";	
number	"number"	typeof value === "number"	integer (-4, 4), float (-3.14, 3.14), 0, -0, Infinity, -Infinity, NaN
		const isNumber=(value)=>typeof value==="number";	
bigint	"bigint"	typeof value === "bigint"	bigint (-4n, 0n, 4n)
		const isBigInt=(value)=>typeof value==="bigint";	
string	"string"	typeof value === "string"	text, example: "hello world"
		const isString=(value)=>typeof value==="string";	
symbol	"symbol"	typeof value === "symbol"	symbol
		const isSymbol=(value)=>typeof value==="symbol";	
function	"function"	typeof value === "function"	Function , AsyncFunction , GeneratorFunction , AsyncGeneratorFunction , ArrowFunction
		const isFunction=(value)=>typeof value==="function";	
object	" <i>object</i> "	value !== null && typeof value === "object"	Object and Standard built-in objects
		const isObject=(value)=>value!==null&&typeof value==="object";	
Array	" <i>object</i> "	Array.isArray (value)	
		value instanceof Array	
Error	" <i>object</i> "	Error.isError (value)	Error , EvalError , RangeError , ReferenceError , SyntaxError , TypeError , URIError
		value instanceof Error	
Other classes	" <i>object</i> "	value instanceof CurrentClass	Standard built-in objects : Map , Set , Iterator , etc.

Type helper functions from Celestra	Details
<code>const typeOf=(value)=>value===null?"null":typeof value;</code>	Returns the real type of the value as a string.
<code>const isNullish=(value)=>value==null;</code>	Checks if the value is null or undefined.
<code>const isNonNullable=(value)=>value!=null;</code>	Checks if the value is not null or undefined.
<code>const isPrimitive=(value)=>value==null (typeof value!=="object"&&typeof value!=="function");</code>	Checks if the value is not an object or function.
<code>const isNonNullablePrimitive=(value)=>value!=null&&typeof value!=="object"&&typeof value!=="function";</code>	Checks if the value is not null, undefined, object or function.
<code>const isPlainObject=(value)=>value!=null&& typeof value==="object"&& (Object.getPrototypeOf(value) ===Object.prototype object.getPrototypeOf(value)===null);</code>	Check if the value has been created from the Object constructor or null .
<code>const isAsyncFunction=(value)=>Object.getPrototypeOf(value).constructor===Object.getPrototypeOf(async function(){}).constructor;</code>	Checks if the value is an AsyncFunction .
<code>const isGeneratorFunction=(value)=>Object.getPrototypeOf(value).constructor===Object.getPrototypeOf(function*(){}).constructor;</code>	Checks if the value is a GeneratorFunction .
<code>const isAsyncGeneratorFunction=(value)=>Object.getPrototypeOf(value).constructor===Object.getPrototypeOf(async function*(){}).constructor;</code>	Checks if the value is an AsyncGeneratorFunction .
<code>function isArrowFunction(value){if(typeof value!=="function" ("prototype" in value&&value.prototype!==undefined) !(value.toString().includes(">"))){return false;}try{new value();return false;}catch(error){return true;}}</code>	Checks if the value is an ArrowFunction .
<code>const isTypedArray=(value)=>ArrayBuffer.isView(value)&&!(value instanceof DataView);</code>	Checks if the value is a TypedArray .
<code>const isIterable=(value)=>value!=null&&typeof value[Symbol.iterator]==="function";</code>	Checks if the value is an Iterable .
<code>const isAsyncIterable=(value)=>value!=null&&typeof value[Symbol.asyncIterator]==="function";</code>	Checks if the value is an Async Iterable .
<code>const isIterator=(value)=>value instanceof Iterator;</code>	Checks if the value is an Iterator .
<code>const isAsyncIterator=(value)=>value!= null&&typeof value.next==="function"&&typeof value[Symbol.asyncIterator]==="function";</code>	Checks if the value is an AsyncIterator .
<code>const isPropertyKey=(value)=>typeof value==="string" typeof value==="symbol";</code>	Checks if the value is a valid property key (string or symbol).
<code>const isRegex=(value)=>value instanceof RegExp;</code>	Checks if the value is a RegExp .
<code>const isSameType=(value1,value2)=>(value1==null value2==null)?(value1===value2):(typeof value1===typeof value2);</code>	Checks if the values are the same type values.
<code>const isSameInstance=(value1,value2,Constructor)=>value1 instanceof Constructor&&value2 instanceof Constructor;</code>	Checks if the values have been created from the same constructor.

Web Storage api and JSON	element.dataset & data-* attributes	TypedArray
IE8+ localStorage: localStorage.length; localStorage.key(index); localStorage.getItem(key); localStorage.setItem(key, data); localStorage.removeItem(key); localStorage.clear(); sessionStorage: sessionStorage.length; sessionStorage.key(index); sessionStorage.getItem(key); sessionStorage.setItem(key, data); sessionStorage.removeItem(key); sessionStorage.clear(); hasItem: localStorage.getItem(key) !== null sessionStorage.getItem(key) !== null setJSON: localStorage.setItem(key, JSON.stringify(object)); sessionStorage.setItem(key, JSON.stringify(object)); getJSON: JSON.parse(localStorage.getItem(key)); JSON.parse(sessionStorage.getItem(key));	- IE11 compatible - element data-* attributes - no methods and events camelcase: element.data-name -> element.dataset.name element.data-first-second -> element.dataset.firstSecond set: element.dataset.name = "value"; element.dataset["name"] = "value"; element.setAttribute("data-name", "value"); element["data-name"] = "value"; get: element.dataset.name; element.dataset["name"]; element.getAttribute("data-name"); element["data-name"]; remove: element.removeAttribute("data- name"); check: element.hasAttribute("data-name");	new <TypedArray>(); ES2017 new <TypedArray>(length); new <TypedArray>(typedArray); new <TypedArray>(object); new <TypedArray>(buffer[,byteOffset[,len]]); <TypedArray>.from(arrayLike, mapFn); <TypedArray>.of(element1, /*...,*/ elementN); Int8Array(); -128 to 127, 1 byte, int8_t, Shortint Uint8Array(); 0 to 255, 1 byte, uint8_t, Byte Uint8ClampedArray(); 0 to 255, 1 byte, uint8_t, Byte Int16Array(); -32768 to 32767, 2 byte, int16_t, Smallint Uint16Array(); 0 to 65535, 2 byte, uint16_t, Word Int32Array(); -2147483648 to 2147483647, 4 byte, int32_t Uint32Array(); 0 to 4294967295, 4 byte, uint32_t, Longword BigInt64Array(); -2**63 to 2**63-1, 8 byte, int64_t, Int64 BigUint64Array(); 0 to 2**64-1, 8 byte, uint64_t, Qword Float16Array(); -65504 to 65504, 2 byte Float32Array(); 1.2x10-38 to 3.4x1038, 4 byte, float, Real Float64Array(); 5.0x10-324 to 1.8x10308, 8 byte, Double
DOM events		
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>); and target.type(); or target["type"]();		

element.classList	JSON
IE10+IE11 don't have support for classList on SVG or MathML elements. element.classList.add(String[,String]); IE10+11: yes (except the multiple arguments) element.classList.remove(String[,String]); IE10+11: yes (except the multiple arguments) - Removing a class that does not exist, does NOT throw an error. element.classList.contains(String); IE10+11: yes 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. element.classList.item(Number); IE10+11: yes element.classList.length; IE10+11: yes element.classList.replace(oldClass, newClass); IE10+11: No and the method isn't compatible with the Safari and mobile browsers too. Remove all classes: element.className = "";	IE8+ Valid Data Types - string - number - object (containing valid JSON values) - array - boolean - date - null Invalid Data Types - 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> JSON.stringify(value[,replacer[,space]]); Convert a JavaScript object to a JSON string. JSON.stringify({ a: 1, b: "2", c: true }); // -> '{"a":1,"b":"2","c":true}' JSON.stringify([1, 2, 3, 4, 5]); // -> "[1,2,3,4,5]" JSON.parse(text[,reviver]); Parses a JSON string and returns a JavaScript object. JSON.parse(JSON.stringify({a: 1, b: "2", c: true})); // -> Object { a: 1, b: "2", c: true } JSON.parse(JSON.stringify([1, 2, 3, 4, 5])); // -> Array(5) [1, 2, 3, 4, 5]

Fetch	Fetch POST
<pre> Firefox, Firefox for Android 39 Chrome, Chrome Android, WebView Android 42 Edge 14 Opera 29 Safari 10.1, Safari on iOS 10.3 Samsung Internet 4.0 Deno 1, Node.js 18 // 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.call(target, thisArgument, argumentsList);</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>	

Array.fromAsync();	Set methods
<pre>Array.fromAsync(<object>[,mapFn[,thisArg]]) .then((resultArray) => /* todo with resultArray */);</pre> Object types: async iterable, iterable (Array, Map, Set, NodeList, etc.), array-like mapfn parameters: element, index	Chrome, Chrome Android, Edge, WebView Android v122 Firefox, Firefox for Android v127 Safari, Safari on iOS, WebView on iOS v17 Opera v108, Opera Android v81 Samsung Internet v26.0 Deno 1.42, Node.js 22.0.0
<pre>async function* asyncIterable () { for (let i = 0; i < 5; i++) { await new Promise((resolve)=> setTimeout(resolve,50*i)); yield i; } }</pre> <pre>Array.fromAsync(asyncIterable()) .then((res) => console.log("asyncIterable1: "+res)); // asyncIterable1: 0,1,2,3,4</pre> <pre>Array.fromAsync(asyncIterable(), (x) => x*2) .then((res) => console.log("asyncIterable2: "+res)); // asyncIterable2: 0,2,4,6,8</pre> <pre>Array.fromAsync([4,5,6,7,8]) .then((res) => console.log("[4,5,6,7,8]: "+res)); // [4,5,6,7,8]: 4,5,6,7,8</pre> <pre>Array.fromAsync([4,5,6,7,8], (x) => x*2) .then((res) => console.log("[4,5,6,7,8] + fn: "+res)); // [4,5,6,7,8] + fn: 8,10,12,14,16</pre> <pre>Array.fromAsync(new Set([4,5,6,6,10])) .then((res) => console.log("Set: "+res)); // Set: 4,5,6,10</pre> <pre>Array.fromAsync(new Set([4,5,6,6,10]), (x) => x*2) .then((res) => console.log("Set + fn: "+res)); // Set + fn: 8,10,12,20</pre> <pre>Array.fromAsync({"0": 3, "1": 4, "2": 5, length: 3}) .then((res) => console.log("arraylike: "+res)); // arraylike: 3,4,5</pre> <pre>Array.fromAsync({"0": 3, "1": 4, "2": 5, length: 3}, (x) => x*2).then((res) => console.log("arraylike + fn: "+res)); // arraylike + fn: 6,8,10</pre>	<pre>Set.prototype.intersection(other): Set Set.prototype.union(other): Set Set.prototype.difference(other): Set Set.prototype.symmetricDifference(other): Set Set.prototype.isSubsetOf(other): Boolean Set.prototype.isSupersetOf(other): Boolean Set.prototype.isDisjointFrom(other): Boolean</pre> <pre>var setA = new Set([1]); var setB = new Set([1,2]); var setC = new Set([2,3]);</pre> <pre>console.log(setB.intersection(setC)); // Set [2] console.log(setB.union(setC)); // Set(3) [1, 2, 3] console.log(setB.difference(setC)); // Set [1] console.log(setB.symmetricDifference(setC)); // Set [1, 3]</pre> <pre>console.log(setB.isSupersetOf(setA)); // true console.log(setA.isSupersetOf(setC)); // false console.log(setA.isSubsetOf(setB)); // true console.log(setA.isSubsetOf(setC)); // false</pre> <pre>console.log(setA.isDisjointFrom(setC)); // true - there are no common elements console.log(setA.isDisjointFrom(setB)); // false - there are common elements</pre>

Javascript Equality comparisons and sameness					
X	Y	<u>loose equality</u>	<u>strict equality</u>	<u>Same-value</u>	<u>Same-value-zero</u>
		X == Y	X === Y	Object.is(X, Y)	<u>[X].includes(Y)</u>
					X === Y (X !== X && Y !== Y)
undefined	undefined	✓ true	✓ true	✓ true	✓ true
null	null	✓ true	✓ true	✓ true	✓ true
true	true	✓ true	✓ true	✓ true	✓ true
false	false	✓ true	✓ true	✓ true	✓ true
"foo"	"foo"	✓ true	✓ true	✓ true	✓ true
0	0	✓ true	✓ true	✓ true	✓ true
+0	-0	✓ true	✓ true	✗ false	✓ true
+0	0	✓ true	✓ true	✓ true	✓ true
-0	0	✓ true	✓ true	✗ false	✓ true
0n	-0n	✓ true	✓ true	✓ true	✓ true
0	false	✓ true	✗ false	✗ false	✗ false
""	false	✓ true	✗ false	✗ false	✗ false
""	0	✓ true	✗ false	✗ false	✗ false
"0"	0	✓ true	✗ false	✗ false	✗ false
"17"	17	✓ true	✗ false	✗ false	✗ false
[1, 2]	"1,2"	✓ true	✗ false	✗ false	✗ false
new String("foo")	"foo"	✓ true	✗ false	✗ false	✗ false
null	undefined	✓ true	✗ false	✗ false	✗ false
null	false	✗ false	✗ false	✗ false	✗ false
undefined	false	✗ false	✗ false	✗ false	✗ false
{foo: "bar"}	{foo: "bar"}	✗ false	✗ false	✗ false	✗ false
new String("foo")	new String("foo")	✗ false	✗ false	✗ false	✗ false
0	null	✗ false	✗ false	✗ false	✗ false
0	NaN	✗ false	✗ false	✗ false	✗ false
"foo"	NaN	✗ false	✗ false	✗ false	✗ false
NaN	NaN	✗ false	✗ false	✓ true	✓ true

<u>StructuredClone();</u>	<u>StructuredClone(); Supported types</u>	
Firefox, Firefox for Android 94 Chrome, Chrome Android, WebView Android 98 Edge 98 Opera 84 Opera Android 68 Safari, Safari for iOS, WebView on iOS 15.4 Samsung Internet 18 Deno 1.14, Node.js 17 Supported in all major browsers. The structuredClone() function creates a deep clone of a given value using the structured clone algorithm. Usage: <pre>structuredClone(value) structuredClone(value, options)</pre> Options: transfer: An array of transferable objects that will be moved rather than cloned to the returned object. Example: <pre>let x = { "a": [1, 2], "b": "lorem ipsum" }; let y = structuredClone(x); console.log(x === y); // false console.log(x.a === y.a); // false console.log(x.a[0] === y.a[0]); // true console.log(x.a[1] === y.a[1]); // true console.log(x.b === y.b); // true</pre>	Primitive types, except Symbol: Null Undefined Boolean Number BigInt String Object types: Array ArrayBuffer Boolean DataView Date Map Number Object objects: but only plain objects (e.g., from object literals). RegExp (lastIndex is not preserved) Set String TypedArray Error objects: AggregateError (cloning not supported in every JS interpreter) Error EvalError RangeError ReferenceError SyntaxError TypeError URIError	Web/API types: AudioData Blob CropTarget CryptoKey DOMException: browsers must serialize the properties name and message. Other attributes may also be serialized/cloned. DOMMatrix DOMMatrixReadOnly DOMPoint DOMPointReadOnly DOMQuad DOMRect DOMRectReadOnly EncodedAudioChunk EncodedVideoChunk FencedFrameConfig File FileList FileSystemDirectoryHandle FileSystemFileHandle FileSystemHandle GPUCompilationInfo GPUCompilationMessage GPUPipelineError ImageBitmap ImageData RTCCertificate RTCEncodedAudioFrame RTCEncodedVideoFrame VideoFrame WebTransportError

<u>Map.prototype.getOrInsert(key, defaultValue);</u>	<u>WeakMap.prototype.getOrInsert(key, defaultValue);</u>
<pre> const map1 = new Map([["DNA", "42"]]); console.log(map1.getOrInsert("DNA", "54")); // 42 console.log(map1.getOrInsert("DNA2", "default")); // "default" console.log(map1.get("DNA2")); // "default" </pre>	<pre> const wm1 = new WeakMap(); const obj1 = {}; console.log(wm1.get(obj1)); // undefined console.log(wm1.getOrInsert(obj1, 42)); // 42 console.log(wm1.get(obj1)); // 42 console.log(wm1.getOrInsert(obj1, "default")); // 42 console.log(wm1.get(obj1)); // 42 </pre>
<u>Map.prototype.getOrInsertComputed(key, callback);</u>	<u>WeakMap.prototype.getOrInsertComputed(key, callback);</u>
<pre> const map1 = new Map([["DNA", 42]]); const helperFN1 = (prop) => 54; console.log(map1.get("DNA")); // 42 console.log(map1.getOrInsertComputed("DNA", helperFN1)); // 42 console.log(map1.get("DNA")); // 42 console.log(map1.getOrInsert("DNA2", helperFN1("DNA2"))); // 54 console.log(map1.getOrInsertComputed("DNA2", helperFN1)); // 54 console.log(map1.get("DNA2")); // 54 </pre>	<pre> const weakmap1 = new WeakMap(); const obj1 = {}; const obj2 = {}; const helperFN1 = (prop) => 54; console.log(weakmap1.get(obj1)); // undefined console.log(weakmap1.getOrInsert(obj1, 42)); // 42 console.log(weakmap1.getOrInsertComputed(obj1, helperFN1)); // 42 console.log(weakmap1.get(obj1)); // 42 console.log(weakmap1.get(obj2)); // undefined console.log(weakmap1.getOrInsertComputed(obj2, helperFN1)); // 54 console.log(weakmap1.get(obj2)); // 54 </pre>

Iterator methods	Iterator methods samples
Firefox, Firefox for Android 131 Chrome, Edge, Webview Android 122 and Opera 108 Safari, Safari in iOS, Webview on iOS 18.4 Samsung Internet 26.0 and Node.js 22 and Deno 1.42	const A1 = [1, 2, 3, 4]; const A2 = [5, 6, 7, 8];
Iterator.from (object);	Iterator.from(A1); // Iterator<any> { 1, 2, 3, 4 }
Iterator.concat (it1, it2, /* ..., */ itN);	Iterator.concat(A1.values(), A2.values()); // Iterator<any> { 1, 2, 3, 4, 5, 6, 7, 8 }
Iterator.zip (iterables[, options object]); options object: "mode": "shortest" (default), "longest", "strict" "padding": padding[i] (iterator converting to array)	Iterator.zip([A1.values(), A2.values()]); // Iterator<Array> { [1, 5], [2, 6], [3, 7], [4, 8] }
Iterator.zipKeyed (iterables[, options object]); options object: same as Iterator.zip() ; options object	Iterator.zipKeyed([A1.values(), A2.values()]); // Iterator<Object> {{0:1, 1:5}, {0:2, 1:6}, {0:3, 1:7}, {0:4, 1:8}}
Iterator.prototype.drop (limit);	A1.values().drop(2); // Iterator<any> { 3, 4 }
Iterator.prototype.every (callbackFn(Element, index));	A1.values().every((x) => x > 0); // true
Iterator.prototype.filter (callbackFn(Element, index));	A1.values().filter((x) => x > 2); // Iterator<any> { 3, 4 }
Iterator.prototype.find (callbackFn(Element, index));	A1.values().find((x) => x > 2); // 3
Iterator.prototype.flatMap (callbackFn(Element, index));	[A1, A2].values().flatMap((x) => x); // Iterator {1,2,3,4,5,6,7,8} new Map([new Map([["a", 1], ["b", 2]]), new Map([["c", 3], ["d", 4]])]).values().flatMap((x) => x); // Map(4) { a → 1, b → 2, c → 3, d → 4 }
Iterator.prototype.forEach (callbackFn(Element, index));	A1.values().forEach((e) => console.log(e)); // 1 2 3 4
Iterator.prototype.map (callbackFn(Element, index));	A1.values().map((x) => x * 2); // Iterator<any> { 2, 4, 6, 8 }
Iterator.prototype.reduce (callbackFn(accumulator, currentValue, currentIndex), initialValue: Optional);	A1.values().reduce((ac, it) => (ac + it), 0); // 10
Iterator.prototype.some (callbackFn(Element, index));	A1.values().some((x) => x > 2); // true
Iterator.prototype.take (limit);	A1.values().take(2); // Iterator<any> { 1, 2 }
Iterator.prototype.toArray (); Same as Array.from(iterator) ; and [...iterator] ;	A1.values().toArray(); // Array(4) [1, 2, 3, 4]
Iterator.prototype[Symbol.dispose] ();	
Iterator.prototype[Symbol.iterator] ();	