# Celestra cheatsheet – v5.4.2 – https://github.com/Serrin/Celestra/

The `celestra` and/or the `CEL` objects contain these functions, except the polyfills. Example: `CEL.qsa("p");`

| Core API | DOM | Type checking |
|---|---|---|
| signbit(<value>); | qsa(<selector>[,context]).forEach(<cb>); | isMap(<value>); *and* isWeakMap(<v>); |
| delay(<ms>).then(<callback>); | qs(<selector>[,context]); | isSet(<value>); *and* isWeakSet(<v>); |
| inherit(<subclass>,<superclass>); | | isNumber(<v>); *and* isNumeric(<v>); |
| randomInt([max] *or* <min>,<max>); | domReady(<callback>); | isFloat(<val>); *and* isBigInt(<v>); |
| randomFloat([max] *or* <min>,<max>); | domCreate(<type>[,properties[,innerHTML]]); | isString(<v>); *and* isChar(<val>); |
| randomBoolean(); | domCreate(<element descriptive object>); | isDate(<val>); *and* isError(<val>); |
| randomID([hyphens=true][,usedate=false]); | domToElement(<htmlString>); | isRegexp(<v>); *and* isSymbol(<v>); |
| randomString([length[,specChar=false]]); | domGetCSS(<element>[,property]); | isElement(<v>); *and* isObject(<v>); |
| inRange(<value>,<min>,<max>); | domSetCSS(<element>,<property>,<value>); | isDataView(<value>); |
| b64Encode(<string>); b64Decode(<string>); | domSetCSS(<element>,<properties>); | isBoolean(<value>); |
| javaHash(<data>[,hexa=false]); | domFadeIn(<element>[,duration[,display]]); | isNull(<value>); |
| getUrlVars([str=location.search]); | domFadeOut(<element>[,duration]); | isUndefined(<value>); |
| obj2string(<object>); | domFadeToggle(<elem.>[,duration[,display]]); | isNullOrUndefined(<value>); |
| classof(<variable>[,type[,throw=false]]); | domShow(<element>[,display]); | isNil(<value>); |
| extend([deep,]<target>,<source1>[,srcN]); | domHide(<element>); | isPlainObject(<value>); |
| sizeIn(<object>); | domToggle(<element>[,display]); | isTruthy(<value>); |
| forIn(<object>,<callback>); | domIsHidden(<element>); | isFalsy(<value>); |
| filterIn(<object>,<callback>); | | isFunction(<v>); + isCallable(<v>); |
| popIn(<object>,<property>); | domSiblings(<element>); | isConstructorFn(<value>); |
| strPropercase(<string>); | domSiblingsPrev(<element>); | isGeneratorFn(<value>); |
| strTitlecase(<string>); | domSiblingsLeft(<element>); | isAsyncGeneratorFn(<value>); |
| strCapitalize(<string>); | domSiblingsNext(<element>); | isAsyncFn(<value>); |
| strUpFirst(<string>); | domSiblingsRight(<element>); | isArraylike(<value>); |
| strDownFirst(<string>); | domGetCSSVar(<name>); | isTypedArray(<value>); |
| strHTMLRemoveTags(<string>); | domSetCSSVar(<name>,<value>); | isArrayBuffer(<value>); |
| strHTMLEscape(<string>); | | isPrimitive(<value>); |
| strHTMLUnEscape(<string>); | importScript(<script1>[,scriptN]); | isPromise(<value>); |
| strReverse(<string>); | importStyle(<style1>[,styleN]); | isIterator(<value>); |
| strAt(<string>,<index>); | | isIterable(<value>); |
| strCodePoints(<string>); | setFullscreenOn(<selector> *or* <element>); | isEmptyObject(<value>); |
| strFromCodePoints(<collection>); | setFullscreenOff(); | isEmptyArray(<value>); |
| bind(<fn>,<context>); *and* unBind(<fn>); | getFullscreen(); | isEmptyMap(<value>); |
| constant(<value>); *and* identity(<value>); | | isEmptySet(<value>); |
| noop(); *and* T(); *and* F(); | form2array(<form>); | isEmptyIterator(<value>); |
| assertEq(<msg>,<v1>,<v2>[,strict=true]); | form2string(<form>); | isSameObject(<object1>,<object2>); |
| assertNotEq(<m>,<v1>,<v2>[,strict=true]); | | isSameArray(<array1>,<array2>); |
| assertTrue(<msg>,<value>); | getDoNotTrack(); | isSameMap(<map1>,<map2>); |
| assertFalse(<msg>,<value>); | getLocation(<success>[,error]); | isSameSet(<set1>,<set2>); |
| noConflict(); *and* VERSION; | createFile(<filename>,<content>[,dType]); | isSameIterator(<iter1>,<iter2>); |

| Collections | | Polyfills |
|---|---|---|
| arrayCreate([length=0]); | forEach(<collection>,<callback>); | |
| arrayDeepClone(<array>); | map(<collection>,<callback>); | Array.prototype.at(); |
| arrayMerge(<target>,<source1>[,sourceN]); | enumerate(<collection>[,offset=0]); | Array.prototype.findLast(); |
| arrayUnique(<collection>); | entries(<collection>[,offset=0]); | Array.prototype.findLastIndex(); |
| arrayAdd(<array>,<value>); | size(<collection>); | Array.prototype.flat(); |
| arrayClear(<array>); | | Array.prototype.flatMap(); |
| arrayRemove(<array>,<value>[,all=false]); | every(<collection>,<callback>); | Array.prototype.groupBy(); |
| arrayRemoveBy(<array>,<callback>[,all=false]); | some(<collection>,<callback>); | Array.prototype.groupByToMap(); |
| arrayRange([start=0[,end=99[,step=1]]]); | none(<collection>,<callback>); | |
| arrayCycle(<collection>[,n=100]); | | crypto.randomUUID(); |
| arrayRepeat(<value>[,n=100]); | includes(<collection>,<value>); | |
| | contains(<collection>,<value>); | globalThis; |
| iterRange([start=0[,step=1[,end=Infinity]]]); | | |
| iterCycle(<iter>[,n=Infinity]); | find(<collection>,<callback>); | Number.MIN_SAFE_INTEGER; |
| iterRepeat(<value>[,n=Infinity]); | findLast(<collection>,<callback>); | Number.MAX_SAFE_INTEGER; |
| | | |
| arrayUnion(<collection1>[,collectionN]); | filter(<collection>,<callback>); | Object.fromEntries(); |
| arrayIntersection(<collection1>,<collection2>); | reject(<collection>,<callback>); | Object.hasOwn(); |
| arrayDifference(<collection1>,<collection2>); | partition(<collection>,<callback>); | Object.is(); |
| arraySymmetricDifference(<collec1>,<collec2>); | groupBy(<collec.>,<cb>[,map=false]); | |
| setUnion(<collection1>[,collectionN]); | | String.prototype.at(); |
| | shuffle(<collection>); | String.prototype.matchAll(); |
| setIntersection(<set1>,<set2>); | min(<value1>[,valueN]); | String.prototype.replaceAll(); |
| setDifference(<set1>,<set2>); | max(<value1>[,valueN]); | String.prototype.trimStart(); |
| setSymmetricDifference(<set1>,<set2>); | sort(<collection>[,numbers=false]); | String.prototype.trimLeft(); |
| | reverse(<collection>); | String.prototype.trimEnd(); |
| isSuperset(<superCollection>,<subCollection>); | | String.prototype.trimRight(); |
| | zip(<collection1>[,collectionN]); | |
| slice(<collection>[,begin=0[,end=Infinity]]); | unzip(<collection>); | TypedArray.prototype.at(); |
| withOut(<collection>,<filterCollection>); | zipObj(<collection1>,<collection2>); | TypedArray.prototype.findLast(); |
| reduce(<collection>,<callback>[,initialvalue]); | item(<collection>,<index>); | TypedArray.prototype.findLastIndex(); |
| | nth(<collection>,<index>); | |
| take(<collection>[,n=1]); | first(<collection>); | **Non-standard polyfills** |
| takeWhile(<collection>,<callback>); | head(<collection>); | |
| takeRight(<collection>[,n=1]); | last(<collection>); | |
| takeRightWhile(<collection>,<callback>); | initial(<collection>); | BigInt.prototype.toJSON(); |
| drop(<collection>[,n=1]); | tail(<collection>); | |
| dropWhile(<collection>,<callback>); | flat(<collection>); | AsyncFunction(); |
| dropRight(<collection>[,n=1]); | concat(<collection1>[,collectionN]); | GeneratorFunction(); |
| dropRightWhile(<collection>,<callback>); | join(<collection>[,separator=","]); | |

| Math plugin (with celestra-math.min.js) | | Abstract functions |
|---|---|---|
| sum(<value1>[,valueN]);<br>avg(<value1>[,valueN]);<br>product(<value1>[,valueN]);<br><br>clamp(<value>,<min>,<max>);<br><br>isEven(<value>);<br>isOdd(<value>); | toFloat32(<value>);<br><br>toInt8(<value>);<br>toUInt8(<value>);<br>toInt16(<value>);<br>toUInt16(<value>);<br>toInt32(<value>);<br>toUInt32(<value>);<br>toBigInt64(<value>);<br>toBigUInt64(<value>);<br><br>isInt8(<value>);<br>isUInt8(<value>);<br>isInt16(<value>);<br>isUInt16(<value>);<br>isInt32(<value>);<br>isUInt32(<value>);<br>isBigInt64(<value>);<br>isBigUInt64(<value>); | getIn(<object>,<property>);<br>getInV(<object>,<property>);<br>hasIn(<object>,<property>);<br>setIn(<object>,<property>,<value>);<br><br>toIndex(<value>);<br>toPropertyKey(<value>);<br>toInteger(<value>);<br>toArray(value);<br>toObject(<value>);<br><br>isIndex(<value>);<br>isPropertyKey(<value>);<br>isSameValue(<value1>,<value2>);<br>isSameValueZero(<value1>,<value2>);<br>isSameValueNonNumber(<value1>,<value2>);<br><br>type(<value>);<br>createDataProperty(<object>,<property>,<value>);<br>createMethodProperty(<object>,<property>,<value>); |

**Cookie**

getCookie([name]);
hasCookie(<name>);
setCookie(<Options object>);
setCookie(<name>,<value>[,hours=8760[,path="/"[,domain[,secure[,SameSite="Lax"[,HttpOnly]]]]]]);
removeCookie(<Options object>);
removeCookie(<name>[,path="/"[,domain[,secure[,SameSite="Lax"[,HttpOnly]]]]]);
clearCookies(<Options object>);
clearCookies([path="/"[,domain[,sec[,SameSite="Lax"[,HttpOnly]]]]]);

**AJAX and CORS**

getText(<url>,<success>);
getJson(<url>,<success>);
ajax(<Options object>);
**Options object properties (* = default value)**: url: *string*, data: *string*, queryType: **"ajax"/"cors"*, type: **"get"/"post"*, success: *function*, error: *function*, format: **"text"/"json"/"xml"*, user: *string*, password: *string*