

Celestra cheatsheet – v5.8.0 – <https://github.com/Serrin/Celestra/>

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

Core API	Type API	DOM API
<code>noConflict();</code> <code>VERSION;</code> <code>BASE16; BASE32; BASE36; BASE58; BASE62;</code> <code>WORDSAFEALPHABET;</code> <code>javaHash(data[, hexadecimal = false]);</code> <code>b64Encode(string);</code> <code>b64Decode(string);</code> <code>extend([deep,]target, source1[, sourceN]);</code> <code>sizeIn(object);</code> <code>popIn(object, property);</code> <code>forIn(object, callback);</code> <code>filterIn(object, callback);</code> <code>delay(milisec).then(callback);</code> <code>sleep(milisec).then(callback);</code> <code>createPolyfillMethod(object, prop, value);</code> <code>createPolyfillProperty(object, prop, value);</code> <code>randomBoolean();</code> <code>randomUUIDv7();</code> <code>timestampID([size=21[, alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_-"]]);</code> <code>nanoid([size=21[, alphabet="123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"]]);</code> <code>getUrlVars([str=location.search]);</code> <code>obj2string(object);</code> <code>classof(value[, class[, throw=false]]);</code> <code>getType(value[, class[, throw=false]]);</code> <code>toObject(value);</code> <code>bind(function, context);</code> <code>unBind(function);</code> <code>constant(value);</code> <code>identity(value);</code> <code>noop();</code> <code>T();</code> <code>F();</code>	<code>type(value);</code> <code>isSameType(value1, value2);</code> <code>isSameClass(value1, value2);</code> <code>isSameInstance(v1, v2, Constructor);</code> <code>isNull(value);</code> <code>isUndefined(value);</code> <code>isNil(value);</code> <code>isPrimitive(value);</code> <code>isNumeric(value);</code> <code>isChar(value);</code> <code>isFunction(value);</code> <code>isCallable(value);</code> <code>isConstructorFn(value);</code> <code>isClass(value);</code> <code>isGeneratorFn(value);</code> <code>isAsyncFn(value);</code> <code>isAsyncGeneratorFn(value);</code> <code>isObject(value);</code> <code>isPlainObject(value)</code> <code>isProxy(value);</code> <code>isElement(value);</code> <code>isRegex(value);</code> <code>isArraylike(value);</code> <code>isTypedArray(value);</code> <code>isIterator(value);</code> <code>isIterable(value);</code> <code>isCoercedObject(object);</code> <code>isDeepStrictEqual(value1, value2);</code> <code>isEmptyValue(value);</code> <code>toPrimitiveValue(value);</code> <code>isIndex(value);</code> <code>toIndex(value);</code> <code>isLength(value);</code> <code>toLength(value);</code> <code>isPropertyKey(value);</code> <code>toPropertyKey(value);</code>	<code>qsa(selector[, context]).forEach(callback);</code> <code>qs(selector[, context]);</code> <code>domReady(callback);</code> <code>domClear(element);</code> <code>domCreate(type[, properties[, innerHTML]]);</code> <code>domCreate(element descriptive object);</code> <code>domToElement(htmlString);</code> <code>domGetCSS(element[, property]);</code> <code>domSetCSS(element, property, value);</code> <code>domSetCSS(element, properties);</code> <code>domFadeIn(element[, duration[, display]]);</code> <code>domFadeOut(element[, duration]);</code> <code>domFadeToggle(element[, duration[, display]]);</code> <code>domShow(element[, display]);</code> <code>domHide(element);</code> <code>domToggle(element[, display]);</code> <code>domIsHidden(element);</code> <code>domScrollToTop(); and domScrollToBottom();</code> <code>domScrollToElement(element[, top=true]);</code> <code>domSiblings(element);</code> <code>domSiblingsPrev(element);</code> <code>domSiblingsLeft(element);</code> <code>domSiblingsNext(element);</code> <code>domSiblingsRight(element);</code> <code>domGetCSSVar(name);</code> <code>domSetCSSVar(name, value);</code> <code>importScript(script1[, scriptN]);</code> <code>importStyle(style1[, styleN]);</code> <code>setFullscreenOn(selector);</code> <code>setFullscreenOn(element);</code> <code>setFullscreenOff();</code> <code>getFullscreen();</code> <code>form2array(form);</code> <code>form2string(form);</code> <code>getDoNotTrack();</code> <code>getLocation(success[, error]);</code> <code>createFile(filename, content[, dType]);</code>

String API	Assertion API	Math API
strPropercase(string); strTitlecase(string); strCapitalize(string); strTruncate(string); strUpFirst(string); strDownFirst(string); strReverse(string); strCodePoints(string); strFromCodePoints(iterator); strAt(string, index[, newChar]); strSplice(string, index, count[, add]); strHTMLRemoveTags(string); strHTMLEscape(string); strHTMLUnEscape(string);	assert(condition[, message]); assertTrue(condition[, message]); assertFalse(condition[, message]); assertThrows(callback[, message]); assertFail(message OR error); assertEqual(value1, value2[, message]); assertNotEqual(value1, value2[, message]); assertStrictEqual(value1, value2[, message]); assertNotStrictEqual(value1, value2[, message]); assertDeepEqual(value1, value2[, message]); assertNotDeepEqual(value1, value2[, message]); assertDeepStrictEqual(value1, value2[, message]); assertNotDeepStrictEqual(value1, value2[, message]); assertTypeOf(value, type[, message]); assertNotTypeOf(value, typeString[, message]); assertInstanceOf(value, constructor[, message]); assertNotInstanceOf(value, constructor[, message]); assertIsNil(value[, message]); assertIsNotNil(value[, message]); assertMatch(string, regexp, message); assertDoesNotMatch(string, regexp, message);	sum(value1[, valueN]); avg(value1[, valueN]); product(value1[, valueN]); clamp(value, min, max); minmax(value, min, max); inRange(value, min, max); signbit(value); randomInt([max]); randomInt(min, max); randomFloat([max]); randomFloat(min, max); isEven(value); isOdd(value); isInt8(value); isInt16(value); isUInt32(value); isUInt8(value); isUInt16(value); isInt32(value); isBigInt64(value); isBigUInt64(value); isFloat16(value); isFloat(value); toInteger(value); toIntegerOrInfinity(value); toInt8(value); toInt16(value); toInt32(value); toUInt8(value); toUInt16(value); toUInt32(value); toBigInt64(value); toBigUInt64(value); toFloat16(value); toFloat32(value);

Collections API		Polyfills
<pre> arrayCreate([length = 0]); arrayDeepClone(array); arrayMerge(target, source1[, sourceN]); arrayAdd(array, value); arrayClear(array); arrayRemove(array, value[, all = false]); arrayRemoveBy(array, callback[, all=false]); arrayRange([start=0[, end = 99[, step = 1]]]); iterRange([start=0[, step=1[, end=Infinity]]]); arrayCycle(iterator[, n = 100]); iterCycle(iterator[, n = Infinity]); arrayRepeat(value[, n = 100]); iterRepeat(value[, n = Infinity]); unique(iterator[, resolver]); slice(iterator[, begin=0[, end = Infinity]]); without(iterator, filterIterator); reduce(iterator, callback[, initialValue]); count(iterator, callback); take(iterator[, n = 1]); takeWhile(iterator, callback); takeRight(iterator[, n = 1]); takeRightWhile(iterator, callback); drop(iterator[, n = 1]); dropWhile(iterator, callback); dropRight(iterator[, n = 1]); dropRightWhile(iterator, callback); isSuperset(); arrayDifference(); arrayIntersection(); arraySymmetricDifference(); arrayUnion(); setDifference(); setIntersection(); setSymmetricDifference(); setUnion(); </pre>	<pre> forEach(iterator, callback); map(iterator, callback); enumerate(iterator[, offset = 0]); entries(iterator[, offset = 0]); size(iterator); every(iterator, callback); some(iterator, callback); none(iterator, callback); includes(iterator, value); contains(iterator, value); find(iterator, callback); findLast(iterator, callback); filter(iterator, callback); reject(iterator, callback); partition(iterator, callback); zip(iterator1[, iteratorN]); unzip(iterator); zipObj(iterator1, iterator1); shuffle(iterator); min(value1[, valueN]); max(value1[, valueN]); sort(iterator[, numbers = false]); reverse(iterator); item(iterator, index); nth(iterator, index); first(iterator); head(iterator); last(iterator); initial(iterator); tail(iterator); flat(iterator); concat(iterator1[, iteratorN]); join(iterator[, separator = ", "]); </pre>	<pre> Array.fromAsync(); Array.prototype.toReversed(); Array.prototype.toSorted(); Array.prototype.toSpliced(); Array.prototype.with(); crypto.randomUUID(); Error.isError(); globalThis; Map.groupBy(); Math.sumPrecise(); Object.groupBy(); Object.hasOwn(); TypedArray.prototype.toReversed(); TypedArray.prototype.toSorted(); TypedArray.prototype.with(); </pre>
		Non-standard polyfills
		<pre> BigInt.prototype.toJSON(); window.AsyncFunction(); window.GeneratorFunction(); </pre>

AJAX and CORS API

```
getText(url, success);  
getJSON(url, success);  
ajax(Options object);
```

Options object properties (* = default value):

Property	Value
url	string
data	string
queryType	*"ajax"/"cors"
type	*"get"/"post"
success	function
error	function
format	*"text"/"json"/"xml"
user	string
password	string

Cookie API

```
getCookie([name]);  
  
hasCookie(name);  
  
setCookie(Options object: properties are the same as the parameters);  
setCookie(name,value[,hours=8760[,path="/"[,domain[,secure[,SameSite="Lax"[,HttpOnly]]]]]);  
  
removeCookie(Options object: properties are the same as the parameters);  
removeCookie(name[,path="/"[,domain[,secure[,SameSite="Lax"[,HttpOnly]]]]]);  
  
clearCookies(Options object: properties are the same as the parameters);  
clearCookies([path="/"[,domain[,sec[,SameSite="Lax"[,HttpOnly]]]]]);
```

Removed Polyfills - Available in celestra-polyfills.dev.js and celestra-polyfills.min.js		
v3.1.0	v3.8.0	v5.6.0
Array.from(); Array.of(); Array.prototype.copyWithin(); Array.prototype.fill(); Array.prototype.find(); Array.prototype.findIndex(); Object.create(); String.fromCodePoint(); String.prototype.codePointAt(); String.prototype.endsWith(); String.prototype.startsWith(); Math.acosh(); Math.asinh(); Math.atanh(); Math.cbrt(); Math.clz32(); Math.cosh(); Math.expm1(); Math.fround(); Math.hypot(); Math.imul(); Math.log1p(); Math.log10(); Math.log2(); Math.sign(); Math.sinh(); Math.tanh(); Math.trunc(); Number.EPSILON; Number.isNaN(); and isNaN(); Number.isInteger(); Number.isFinite(); Number.isSafeInteger(); Number.parseInt(); Number.parseFloat();	Array.prototype.values(); Array.prototype.includes(); ChildNode.after(); ChildNode.before(); ChildNode.remove(); ChildNode.replaceWith(); Element.prototype.closest(); Element.prototype.getAttributeNames(); Element.prototype.matches(); Element.prototype.toggleAttribute(); ParentNode.append(); ParentNode.prepend(); String.prototype[Symbol.iterator](); String.prototype.includes(); String.prototype.repeat(); NodeList.prototype.forEach(); Object.assign(); Object.entries(); Object.getOwnPropertyDescriptors(); Object.values(); RegExp.prototype.flags; window.screenLeft; window.screenTop;	Array.prototype.at(); Array.prototype.findLast(); Array.prototype.findLastIndex(); Array.prototype.flat(); Array.prototype.flatMap(); Number.MIN_SAFE_INTEGER; Number.MAX_SAFE_INTEGER; Object.fromEntries(); Object.is(); String.prototype.at(); String.prototype.matchAll(); String.prototype.padStart(); String.prototype.padEnd(); String.prototype.replaceAll(); String.prototype.trimStart(); String.prototype.trimLeft(); String.prototype.trimEnd(); String.prototype.trimRight(); Typedarray.prototype.at(); TypedArray.prototype.findLast(); TypedArray.prototype.findLastIndex();