

## Celestra cheatsheet – v6.2.0 – <https://github.com/Serrin/Celestra/>

Core API		Type API	DOM API
constant(value);	asyncConstant(value);	typeof(value);	qsa(selector[,context]).forEach(callback);
identity(value);	asyncIdentity(value);	is(val[,expectedType[,Throw=false]]);	qs(selector[,context]);
noop();	asyncNoop();	isTypedCollection(iter,expectedType,Throw=false);	domReady(callback);
T();	asyncT();	isSameType(value1,value2);	domClear(element);
F();	asyncF();	isSameInstance(v1,v2,Constructor);	domCreate(type[,properties[,innerHTML]]);
VERSION;	BASE16;	isDeepStrictEqual(value1,value2);	domCreate(element descriptive object);
eq(value1, value2);	BASE32;	isCoercedObject(object);	domToElement(htmlString);
gt(value1, value2);	BASE36;	isEmptyValue(value);	domGetCSS(element[,property]);
gte(value1, value2);	BASE58;	isNull(value);	domSetCSS(element,property,value);
lt(value1, value2);	BASE62;	isUndefined(value);	domSetCSS(element,properties);
lte(value1, value2);	WORDSAFEALPHABET;	isNullish(value);	domFadeIn(element[,duration[,display]]);
extend([deep,]target,source1[,sourceN]);		isNotNullable(value);	domFadeOut(element[,duration]);
deleteOwnProperty(obj,prop[,Throw=false]);		isNotNullablePrimitive(value);	domFadeToggle(elem[,duration[,display]]);
sizeIn(object);		isNumeric(value);	domShow(element[,display]);
pick(object,keys);		isChar(value);	domHide(element);
omit(object,keys);		isPlainObject(value);	domToggle(element[,display]);
assoc(object,key,value);		isFunction(value);	domIsHidden(element);
delay(milisec).then(callback);		isCallable(value);	domScrollToTop();
bind(function,context);		isGeneratorFn(value);	domScrollToBottom();
unBind(function);		isAsyncFn(value);	domScrollToElement(element[,top=true]);
curry(function);		isAsyncGeneratorFn(value);	domSiblings(element);
compose(function1[,functionN]);		isProxy(value);	domSiblingsPrev(element);
pipe(function1[,functionN]);		isElement(value);	domSiblingsLeft(element);
once(function);		isRegex(value);	domSiblingsNext(element);
tap(function): function(value);		isArraylike(value);	domSiblingsRight(element);
randomBoolean();		isTypedArray(value);	domGetCSSVar(name);
randomUUIDv7(v4=false);		isIterator(value);	domSetCSSVar(name,value);
timestampID([size=21[,alphabet="123456789ABCDEFHGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"]]);		isIterable(value);	importScript(script1[,scriptN]);
nanoid([size=21[,alphabet="A-Za-z0-9_-"]]);		isAsyncIterable(value);	importStyle(style1[,styleN]);
createPolyfillMethod(object,prop,value);		isPropertyKey(value);	setFullscreenOn(selector);
createPolyfillProperty(object,prop,value);		toPropertyKey(value);	setFullscreenOn(element);
getUrlVars([str=location.search]);		isPrimitive(value);	setFullscreenOff();
obj2string(object);		toPrimitiveValue(value);	getFullscreen();
assert(condition[,message error]);		isObject(value); and toObject(value);	form2array(form);
		isIndex(value); and toIndex(value);	form2string(form);
		isLength(value); and toLength(value);	getDoNotTrack();
		toSafeString(value);	getLocation(success[,error]);
			createFile(filename,content[,dType]);

String API	Cookie API	Math API
b64Decode(string); b64Encode(string); strAt(string,index[,newChar]); strCapitalize(string); strCodePoints(string); strDownFirst(string); strFromCodePoints(iterator); strHTMLEscape(string); strHTMLRemoveTags(string); strHTMLUnEscape(string); strPropercase(string); strReverse(string); strSplice(string,index,count[,add]); strTitlecase(string); strTruncate(string); strUpFirst(string);	getCookie([name]); hasCookie(name); setCookie(name,value[,hours=8760[,path="/"[,domain[,secure[,SameSite="Lax"[,HttpOnly]]]]]]); setCookie(Options object: properties are the same as the parameters); removeCookie(name[,path="/"[,domain[,secure[,SameSite="Lax"[,HttpOnly]]]]]); removeCookie(Options object: properties are the same as the parameters); clearCookies([path="/"[,domain[,sec[,SameSite="Lax"[,HttpOnly]]]]]); clearCookies(Options object: properties are the same as the parameters);	sum(value1[,valueN]); avg(value1[,valueN]); product(value1[,valueN]); mod(value1,value2); rem(value1,value2); 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> castArray(value); compact(iterator); 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(superCollection, subCollection); setDifference(set1, set2); setIntersection(set1, set2); setSymmetricDifference(set1, set2); setUnion(iterator1[, iteratorN] ); </pre>	<pre> forEach(iterator, callback); map(iterator, callback); enumerate(iterator[, offset = 0]); size(iterator);  every(iterator, callback); some(iterator, callback); none(iterator, callback);  includes(collection, value[, comparator]); 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();  globalThis.AsyncFunction();  globalThis.AsyncGeneratorFunction();  globalThis.GeneratorFunction(); </pre>

How to import		
Celestra for browser: <i>celestra.browser.js</i>	Celestra for Node.js and Deno: <i>celestra.node.mjs</i>	
<pre> &lt;script type="module"&gt; // import the defaultExport object import defaultExport from "./celestra.browser.js"; globalThis.celestra = defaultExport; globalThis.CEL = defaultExport; &lt;/script&gt;  &lt;script type="module"&gt; // import with default with name import { default as celestra } from "./celestra.browser.js"; globalThis.celestra = celestra; globalThis.CEL = celestra; &lt;/script&gt;  &lt;script type="module"&gt; // import all into a new celestra object import * as celestra from "./celestra.browser.js"; globalThis.celestra = celestra; globalThis.CEL = celestra; &lt;/script&gt;  &lt;script type="module"&gt; // import some functions import { first, map } from "./celestra.browser.js"; globalThis.first = first; globalThis.map = map; &lt;/script&gt;  &lt;script type="module"&gt; // dynamic import const celestra = await import("./celestra.browser.js"); globalThis.celestra = celestra; globalThis.CEL = celestra; &lt;/script&gt; </pre>	<pre> // import the defaultExport object import defaultExport from "./celestra.node.mjs"; globalThis.celestra = defaultExport; globalThis.CEL = defaultExport;  // import with default with name import { default as celestra } from "./celestra.node.mjs"; globalThis.celestra = celestra; globalThis.CEL = celestra;  // import all into a new celestra object import * as celestra from "./celestra.node.mjs"; globalThis.celestra = celestra; globalThis.CEL = celestra;  // import some functions import { first, map } from "./celestra.node.mjs"; globalThis.first = first; globalThis.map = map;  // dynamic import const celestra = await import("./celestra.node.mjs"); globalThis.celestra = celestra; globalThis.CEL = celestra; </pre>	
	Removed APIs in the <i>celestra.node.mjs</i>	
	DOM API	Legacy AJAX API
		Cookie API

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(); 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();  <div>v5.9.0</div> BigInt.prototype.toJSON();