

Celestra cheatsheet – v6.1.1 – <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();	isSameType(value1,value2);	domReady(callback);
T();	asyncT();	isSameInstance(v1,v2,Constructor);	domClear(element);
F();	asyncF();	isDeepStrictEqual(value1,value2);	domCreate(type[,properties[,innerHTML]]);
VERSION;	BASE16;	isCoercedObject(object);	domCreate(element descriptive object);
eq(value1, value2);	BASE32;	isEmptyValue(value);	domToElement(htmlString);
gt(value1, value2);	BASE36;	isNull(value);	domGetCSS(element[,property]);
gte(value1, value2);	BASE58;	isUndefined(value);	domSetCSS(element,property,value);
lt(value1, value2);	BASE62;	isNullish(value);	domSetCSS(element,properties);
lte(value1, value2);	WORDSAFEALPHABET;	isNumeric(value);	domFadeIn(element[,duration[,display]]);
extend([deep,]target,source1[,sourceN]);		isChar(value);	domFadeOut(element[,duration]);
deleteOwnProperty(obj,prop[,Throw=false]);		isPlainObject(value);	domFadeToggle(elem[,duration[,display]]);
sizeIn(object);		isFunction(value);	domShow(element[,display]);
pick(object,keys);		isCallable(value);	domHide(element);
omit(object,keys);		isClass(value);	domToggle(element[,display]);
assoc(object,key,value);		isGeneratorFn(value);	domIsHidden(element);
		isAsyncFn(value);	domScrollToTop();
delay(milisec).then(callback);		isAsyncGeneratorFn(value);	domScrollToBottom();
bind(function,context);		isProxy(value);	domScrollToElement(element[,top=true]);
unBind(function);		isElement(value);	domSiblings(element);
curry(function);		isRegex(value);	domSiblingsPrev(element);
compose(function1[,functionN]);		isArraylike(value);	domSiblingsLeft(element);
pipe(function1[,functionN]);		isTypedArray(value);	domSiblingsNext(element);
once(function);		isIterator(value);	domSiblingsRight(element);
tap(function): function(value);		isIterable(value);	domGetCSSVar(name);
randomBoolean();		isAsyncIterable(value);	domSetCSSVar(name,value);
randomUUIDv7(v4=false);		isIndex(value);	importScript(script1[,scriptN]);
timestampID([size=21[,alphabet="123456789ABCDEFHGJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"]]);		toIndex(value);	importStyle(style1[,styleN]);
nanoid([size=21[,alphabet="A-Za-z0-9_-"]]);		isLength(value);	setFullscreenOn(selector);
createPolyfillMethod(object,prop,value);		toLength(value);	setFullscreenOn(element);
createPolyfillProperty(object,prop,value);		isPropertyKey(value);	setFullscreenOff();
getUrlVars([str=location.search]);		toPropertyKey(value);	getFullscreen();
obj2string(object);		isPrimitive(value);	form2array(form);
		toPrimitiveValue(value);	form2string(form);
		isObject(value);	getDoNotTrack();
		toObject(value);	getLocation(success[,error]);
		toSafeString(value);	createFile(filename,content[,dType]);

String API	Assertion API	Math API
b64Decode(string);	assert(condition[,message error]);	sum(value1[,valueN]);
b64Encode(string);	assertTrue(condition[,message error]);	avg(value1[,valueN]);
strAt(string,index[,newChar]);	assertFalse(condition[,message error]);	product(value1[,valueN]);
strCapitalize(string);	assertThrows(callback[,message error]);	clamp(value,min,max);
strCodePoints(string);	assertFail(message error);	minmax(value,min,max);
strDownFirst(string);	assertEqual(value1,value2[,message error]);	inRange(value,min,max);
strFromCodePoints(iterator);	assertNotEqual(value1,value2[,message error]);	signbit(value);
strHTMLEscape(string);	assertStrictEqual(value1,value2[,message error]);	randomInt([max]);
strHTMLRemoveTags(string);	assertNotStrictEqual(value1,value2[,message error]);	randomInt(min,max);
strHTMLUnEscape(string);	assertDeepEqual(value1,value2[,message error]);	randomFloat([max]);
strPropercase(string);	assertNotDeepEqual(value1,value2[,message error]);	randomFloat(min,max);
strReverse(string);	assertDeepStrictEqual(value1,value2[,message error]);	isEven(value);
strSplice(string,index,count[,add]);	assertNotDeepStrictEqual(value1,value2[,message error]);	isOdd(value);
strTitlecase(string);	assertIs(value,exptectedType[,message error]);	isInt8(value);
strTruncate(string);	assertIsNot(value,exptectedType[,message error]);	isInt16(value);
strUpFirst(string);	assertIsNullish(value[,message error]);	isUInt32(value);
	assertIsNotNullish(value[,message error]);	isUInt8(value);
	assertMatch(string,regexp[,message error]);	isUInt16(value);
	assertDoesNotMatch(string,regexp[,message error]);	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>

AJAX and CORS API

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

Options object properties (* = default value):

Property	Value
url	string
data	string
contentType	*"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]]]]]);
```

How to import			
Celestra for browser: <i>celestra.browser.js</i>		Celestra for Node.js and Deno: <i>celestra.node.mjs</i>	
<pre> <script type="module"> // import the defaultExport object import defaultExport from "./celestra.browser.js"; globalThis.celestra = defaultExport; globalThis.CEL = defaultExport; </script> <script type="module"> // import with default with name import { default as celestra } from "./celestra.browser.js"; globalThis.celestra = celestra; globalThis.CEL = celestra; </script> <script type="module"> // import all into a new celestra object import * as celestra from "./celestra.browser.js"; globalThis.celestra = celestra; globalThis.CEL = celestra; </script> <script type="module"> // import some functions import { first, assert } from "./celestra.browser.js"; globalThis.first = first; globalThis.assert = assert; </script> </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, assert } from "./celestra.node.mjs"; globalThis.first = first; globalThis.assert = assert; // 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	AJAX and CORS 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();
		v5.9.0
		BigInt.prototype.toJSON();