Celestra cheatsheet – v6.1.0 – https://github.com/Serrin/Celestra/

Core API

```
constant(value);
identity(value);
noop();
T();
F();
```

```
asyncConstant(value);
asyncIdentity(value);
asyncNoop();
asyncT();
asyncF();
```

```
BASE16; BASE32; BASE36; BASE58; BASE62;
WORDSAFEALPHABET;
```

```
extend([deep,]target,source1[,sourceN]);
deleteOwnProperty(obj,prop[,Throw=false]);
sizeIn(object);
pick(object,keys);
omit(object,keys);
assoc(object,key,value);
```

```
delay(milisec).then(callback);
bind(function,context);
unBind(function);
curry(function);
compose(function1[,functionN]);
pipe(function1[,functionN]);
once(function);
tap(function): function(value);
```

```
randomBoolean();
randomUUIDv7(v4=false);
timestampID([size=21[,alphabet="ABCDEFGHIJ
KLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
0123456789_-"]]);
nanoid([size=21[,alphabet="123456789ABCDEF
GHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxy
z"]]);
```

```
createPolyfillMethod(object,prop,value);
createPolyfillProperty(object,prop,value);
getUrlVars([str=location.search]);
obj2string(object);
toSafeString(value);
VERSION;
```

Type API

```
typeOf(value);
is(val[,expectedType[,Throw=false]]);
isSameType(value1,value2);
isSameInstance(v1,v2,Contructor);
isDeepStrictEqual(value1,value2);
isCoercedObject(object);
isEmptyValue(value);
isNull(value);
isUndefined(value);
isNullish(value);
isNumeric(value);
isChar(value);
isPlainObject(value);
isFunction(value);
isCallable(value);
isClass(value);
isGeneratorFn(value);
isAsyncFn(value);
isAsyncGeneratorFn(value);
isProxy(value);
isElement(value);
isRegexp(value);
isArraylike(value);
isTypedArray(value);
isIterator(value);
isIterable(value);
isAsyncIterable(value);
isIndex(value);
toIndex(value);
isLength(value);
toLength(value);
isPropertyKey(value);
toPropertyKey(value);
isPrimitive(value);
toPrimitiveValue(value);
isObject(value);
toObject(value);
```

DOM API

```
qsa(selector[,context]).forEach(callback);
qs(selector[,context]);
domReady(callback);
domClear(element);
domCreate(type[,properties[,innerHTML]]);
domCreate(element descriptive object);
domToElement(htmlString);
domGetCSS(element[,property]);
domSetCSS(element,property,value);
domSetCSS(element,properties);
domFadeIn(element[,duration[,display]]);
domFadeOut(element[,duration]);
domFadeToggle(elem[,duration[,display]]);
domShow(element[,display]);
domHide(element);
domToggle(element[,display]);
domIsHidden(element);
domScrollToTop(); and domScrollToBottom();
domScrollToElement(element[,top=true]);
domSiblings(element);
domSiblingsPrev(element);
domSiblingsLeft(element);
domSiblingsNext(element);
domSiblingsRight(element);
domGetCSSVar(name);
domSetCSSVar(name,value);
importScript(script1[,scriptN]);
importStyle(style1[,styleN]);
setFullscreenOn(selector);
setFullscreenOn(element);
setFullscreenOff();
getFullscreen();
form2array(form);
form2string(form);
getDoNotTrack();
getLocation(success[,error]);
createFile(filename,content[,dType]);
```

String API	Assertion 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);
	assert(condition[,message|error]);
	assertTrue(condition[,message|error]);
	
	assertFalse(condition[,message|error]);
	
	assertThrows(callback[,message|error]);
	
	assertFail(message|error);
	
	assertEqual(value1,value2[,message|error]);
	assertNotEqual(value1,value2[,message|error]);
	
	assertStrictEqual(value1,value2[,message|error]);
	assertNotStrictEqual(value1,value2[,message|error]);
	
	assertDeepEqual(value1,value2[,message|error]);
	assertNotDeepEqual(value1,value2[,message|error]);
	
	assertDeepStrictEqual(value1,value2[,message|error]);
	assertNotDeepStrictEqual(value1,value2[,message|error]);
	
	assertIs(value,exptectedType[,message|error]);
	assertIsNot(value,exptectedType[,message|error]);
	
	assertIsNullish(value[,message|error]);
	assertIsNotNullish(value[,message|error]);
	
	assertMatch(string,regexp[,message|error]);
	assertDoesNotMatch(string,regexp[,message|error]);
		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

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]);
	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 = ","]);
		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();

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]]]]]);
```

How to import

Celestra for browser: *celestra.browser.js*

```
<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>
```

Celestra for Node.js and Deno: *celestra.node.mjs*

```
// 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;
```

Removed APIs in the celestra.node.mjs

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.prototype.at();
Array.of();	Array.prototype.values();	
Array.prototype.copyWithin();	Array.prototype.includes();	Array.prototype.findLast();
Array.prototype.fill();		Array.prototype.findLastIndex();
Array.prototype.find();	ChildNode.after();	
Array.prototype.findIndex();	ChildNode.before();	Array.prototype.flat();
Object.create();	ChildNode.remove();	Array.prototype.flatMap();
String.fromCodePoint();	ChildNode.replaceWith();	
String.prototype.codePointAt();		Number.MIN_SAFE_INTEGER;
String.prototype.endsWith();	Element.prototype.closest();	Number.MAX_SAFE_INTEGER;
String.prototype.startsWith();	Element.prototype.getAttributeNames();	
Math.acosh();	Element.prototype.matches();	Object.fromEntries();
Math.asinh();	Element.prototype.toggleAttribute();	Object.is();
Math.atanh();		
Math.cbrt();	ParentNode.append();	String.prototype.at();
Math.clz32();		
Math.cosh();	ParentNode.prepend();	String.prototype.matchAll();
Math.expm1();		
Math.fround();	String.prototype[Symbol.iterator]();	String.prototype.padStart();
Math.hypot();	String.prototype.includes();	String.prototype.padEnd();
Math.imul();	String.prototype.repeat();	
Math.log1p();		String.prototype.replaceAll();
Math.log10();	NodeList.prototype.forEach();	
Math.log2();		String.prototype.trimStart();
Math.sign();	Object.assign();	String.prototype.trimLeft();
Math.sinh();	Object.entries();	
Math.tanh();		String.prototype.trimEnd();
Math.trunc();	Object.getOwnPropertyDescriptors();	String.prototype.trimRight();
Number.EPSILON;	Object.values();	
Number.isNaN();		Typedarray.prototype.at();
isNaN();	RegExp.prototype.flags;	
Number.isInteger();		TypedArray.prototype.findLast();
Number.isFinite();	window.screenLeft;	TypedArray.prototype.findLastIndex();
Number.isSafeInteger();	window.screenTop;	**v5.9.0**
Number.parseInt();		
Number.parseFloat();		BigInt.prototype.toJSON();