Celestra cheatsheet – v6.7.0 – https://github.com/Serrin/Celestra/

Core API

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

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

```
eq(value1, value2);
gt(value1, value2);
gte(value1, value2);
lt(value1, value2);
lte(value1, value2);
```

```
VERSION;

BASE16 = "0123456789ABCDEF";
BASE32 = "234567ABCDEFGHIJKLMNOPQRSTUVWXYZ";
BASE36 =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
BASE58 =
"123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghij
kmnopqrstuvwxyz";
BASE62 =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefg
hijklmnopqrstuvwxyz";

WORDSAFEALPHABET =
"23456789CFGHJMPQRVWXcfghjmpqvwx";
```

```
delay(milisec).then(callback);

bind(function,context);

unBind(function);

curry(function);

once(function);

tap(function): function(value);

compose(function1[,functionN]);

pipe(function1[,functionN]);
```

```
deepAssign(target,source1[,sourceN]);
sizeIn(object);

pick(object,keys);
omit(object,keys);

assoc(object,key,value);
```

```
RandomBoolean();

randomUUIDv7(v4=false);

nanoid([size=21[,alphabet="ABCDE
FGHIJKLMNOPQRSTUVWXYZabcdefghijk
lmnopqrstuvwxyz0123456789_-"]]);

timestampID([size=21[,alphabet="
123456789ABCDEFGHJKLMNPQRSTUVWXY
Zabcdefghijkmnopqrstuvwxyz"]]);
```

```
assert(condition[,message|error]);
```

DOM API

```
qsa(selector[,context]);
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(element[,duration[,display]]);
domShow(element[,display]);
domHide(element);
domToggle(element[,display]);
domIsHidden(element);
domScrollToTop();
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]);
```

Polyfills	String API	Math API	

crypto.randomUUID();
 	
Error.isError();
 	
globalThis;
 	
Math.sumPrecise();
 	
globalThis.AsyncFunction();
 	
globalthis.AsyncGeneratorFunction();
 	
globalThis.GeneratorFunction();
	b64Decode(string);
	b64Encode(string);
	
	strCodePoints(string);
	strFromCodePoints(iterator);
	
	strAt(string,index[,newChar]);
	
	strCount(string, substring);
	
	strSplice(str,index,count[,add]);
	
	strTruncate(string);
	
	strReverse(string);
	
	strUpFirst(string);
	strDownFirst(string);
	strCapitalize(string);
	strPropercase(string);
	strTitlecase(string);
	
	strHTMLEscape(string);
	strHTMLRemoveTags(string);
	strHTMLUnEscape(string);
		sum(value1[,valueN]);
		avg(value1[,valueN]);
		product(value1[,valN]);
		
		clamp(value,min,max);
		minmax(value,min,max);
		inRange(value,min,max);
		
		signbit(value);
		
		randomInt([max]);
		randomInt(min,max);
		
		randomFloat([max]);
		randomFloat(min,max);
		
		add(value1,value2);
		sub(value1,value2);
		mul(value1,value2);
		div(value1,value2);
		
		divMod(value1,value2);
		mod(value1,value2);
		
		pow(base, power);
			isEven(value);
			isOdd(value);
			isInt8(value);
			isInt16(value);
			isInt32(value);
			isUInt32(value);
			isUInt8(value);
			isUInt16(value);
			isBigInt64(value);
			isBigUInt64(value);
			isFloat16(value);
			isFloat(value);
			
			toInteger(value);
			toIntegerOrInfinity(val);
			toInt8(value);
			toInt16(value);
			toInt32(value);
			toUInt8(value);
			toUInt16(value);
			toUInt32(value;
			toBigInt64(value);
			toBigUInt64(value);
			toFloat16(value);
			toFloat32(value);

Cookie API

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);

Collections API		Type API

castArray(value);
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]]]);
arrayCycle(iterator[,n = 100]);
arrayRepeat(value[,n = 100]);
 	
iterRange([start=0[,step=1[,end=Infinity]]]);
iterCycle(iterator[,n = Infinity]);
iterRepeat(value[,n = Infinity]);
 	
count(iterator,callback);
 	
compact(collection);
 	
unique(iterator[,resolver]);
 	
slice(iterator[,begin=0[,end = Infinity]]);
 	
withOut(iterator,filterIterator);
 	
reduce(iterator,callback[,initialvalue]);
 	
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);
	forEach(iterator,callback);
	map(iterator,callback);
	enumerate(iterator[,offset = 0]);
	size(collection);
	
	every(iterator,callback);
	some(iterator,callback);
	none(iterator,callback);
	
	includes(collection,val[,comparator]);
	find(iterator,callback);
	findLast(iterator,callback);
	filter(iterator,callback);
	reject(iterator,callback);
	partition(iterator,callback);
	
	zip(iterator1[,iteratorN]);
	unzip(iterator);
	zipObj(iterator1,iterator2);
	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 = ","]);
		typeOf(value);
		is(val[,expectedType[,Throw=false]]);
		isTypedCollection(iter,expectedType,
		Throw=false);
		isSameType(value1,value2);
		isSameInstance(val1,val2,Contructor);
		isDeepStrictEqual(value1,value2);
		isCoercedObject(object);
		isEmpty(value);
		isNull(value);
		isUndefined(value);
		isNullish(value);
		isNonNullable(value);
		isNonNullablePrimitive(value);
		isPlainObject(value);
		isFunction(value);
		isGeneratorFunction(value);
		isAsyncFunction(value);
		isAsyncGeneratorFunction(value);
		isArrowFunction(value);
		isProxy(value);
		isElement(value);
		isRegexp(value);
		isArraylike(value);
		isTypedArray(value);
		isIterator(value);
		isIterable(value);
		isAsyncIterator(value);
		isAsyncIterable(value);
		isPropertyKey(value);
		toPropertyKey(value);
		isPrimitive(value);
		toPrimitive(value);
		isObject(value);
		toObject(value);
		isIndex(value); *and* toIndex(value);
		isLength(value); *and* toLength(value);
		toSafeString(value);

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, map } from "./celestra.browser.js";
globalThis.first = first;
globalThis.map = map;
</script>

<script type="module">
// dynamic import
const celestra = await import("./celestra.browser.js");
globalThis.celestra = celestra;
globalThis.CEL = celestra;
</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, 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;
```

Removed APIs in the *celestra.node.mjs*

DOM 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();
		v6.5.0
		Array.fromAsync(); Array.prototype.toReversed(); Array.prototype.toSorted(); Array.prototype.toSpliced(); Array.prototype.with(); Map.groupBy(); *and* Object.groupBy(); Object.hasOwn(); TypedArray.prototype.toReversed(); TypedArray.prototype.toSorted(); TypedArray.prototype.with();