Celestra cheatsheet – v5.6.4 – https://github.com/Serrin/Celestra/

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

Core API

```
javaHash(<data>[,hexa=false]);
b64Encode(<string>); b64Decode(<string>);
extend([deep,]<target>,<source1>[,srcN]);
sizeIn(<object>);
popIn(<obj>,<prop>);, forIn(<obj>,<cb>);
filterIn(<object>,<callback>);
delay + sleep(<ms>).then(<callback>);
inherit(<subclass>,<superclass>);
randomBoolean();
randomUUIDv7();
timestampID([size=21[,alphabet=BASE58]]);
nanoid([size=21[,alphabet="A-Za-z0-9-
_"]]);
BASE16; BASE32; BASE36; BASE58; BASE62;
WORDSAFEALPHABET;
getUrlVars([str=location.search]);
obj2string(<object>);
classof(<variable>[,type[,throw=false]]);
bind(<fn>,<context>); and unBind(<fn>);
constant(<value>); and identity(<value>);
noop(); and T(); and F();
assertEq(<msg>,<v1>,<v2>[,strict=true]);
assertNotEq(<m>,<v1>,<v2>[,strict=true]);
assertTrue(<message>,<value>);
assertFalse(<message>,<value>);
noConflict(); and VERSION;
```

String API

```
strPropercase(<str>); strTitlecase(<s>);
strCapitalize(<string>);
strUpFirst(<str>); + strDownFirst(<str>);
strReverse(<s>); + strCodePoints(<s>);
strFromCodePoints(<iterator>);
strAt(<string>,<index>[,newChar]);
strSplice(<str>,<index>,<count>[,add]);
strHTMLRemoveTags(<string>);
strHTMLEscape(<s>); strHTMLUnEscape(<s>);
```

DOM API

```
qsa(<selector>[,context]).forEach(<cb>);
qs(<selector>[,context]);
domReady(<callback>);
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();
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> or <element>);
setFullscreenOff();
getFullscreen();
form2array(<form>);
form2string(<form>);
getDoNotTrack();
getLocation(<success>[,error]);
createFile(<filename>,<content>[,dType]);
```

Type checking API

```
isMap(<value>); and isWeakMap(<v>);
isSet(<value>); and isWeakSet(<v>);
isNumber(<v>); and isNumeric(<v>);
isFloat(<val>); and isBigInt(<v>);
isString(<v>); and isChar(<val>);
isDate(<val>); and isError(<val>);
isRegexp(<v>); and isSymbol(<v>);
isElement(<v>); and isObject(<v>);
isDataView(<value>);
isBoolean(<value>);
isNull(<value>);
isUndefined(<value>);
isNullOrUndefined(<v>); isNil(<v>);
isPlainObject(<value>);
isTruthy(<value>); + isFalsy(<v>);
isFunction(<v>); + isCallable(<v>);
isConstructorFn(<value>);
isGeneratorFn(<value>);
isAsyncGeneratorFn(<value>);
isAsyncFn(<value>);
isArraylike(<value>);
isTypedArray(<value>);
isArrayBuffer(<value>);
isPrimitive(<value>);
isPromise(<value>);
isIterator(<v>);
isIterable(<v>);
isEmptyObject(<value>);
isEmptyArray(<value>);
isEmptyMap(<v>); + isEmptySet(<v>);
isEmptyIterator(<value>);
isSameObject(<object1>,<object2>);
isSameArray(<array1>,<array2>);
isSameMap(<map1>,<map2>);
isSameSet(<set1>,<set2>);
isSameIterator(<iter1>,<iter2>);
```

Collections API		Polyfills

arrayCreate([length=0]);
 	
arrayDeepClone(<array>);
 	
arrayMerge(<target>,<source1>[,sourceN]);
 	
arrayUnique(<iterator>);
 	
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(<iter>[,n=Infinity]);
 	
arrayRepeat(<value>[,n=100]);
iterRepeat(<value>[,n=Infinity]);
 	
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>);
	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>,<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=","]);
		Array.fromAsync();
		
		Array.prototype.toReversed();
		
		Array.prototype.toSorted();
		
		Array.prototype.toSpliced();
		
		Array.prototype.with();
		
		crypto.randomUUID();
		
		Error.isError();
		
		globalThis;
		
		Map.groupBy();
		
		Object.groupBy();
		
		Object.hasOwn();
		
		TypedArray.prototype.toReversed();
		
		TypedArray.prototype.toSorted();
		
		TypedArray.prototype.with();
		
		Non-standard polyfills
		
		BigInt.prototype.toJSON();
		
		window.AsyncFunction();
		
		window.GeneratorFunction();

Math API		Abstract API

sum(<value1>[,valueN]);
avg(<value1>[,valueN]);
product(<value1>[,valueN]);
 	
clamp(<value>,<min>,<max>);
minmax(<value>,<min>,<max>);
 	
isEven(<value>);
isOdd(<value>);
 	
randomInt([max]);
randomInt(<min>,<max>);
 	
randomFloat([max]);
randomFloat(<min>,<max>);
 	
inRange(<value>,<min>,<max>);
signbit(<value>);
	toInt8(<value>);
	toInt16(<value>);
	toInt32(<value>);
	toUInt8(<value>);
	toUInt16(<value>);
	toUInt32(<value>;
	toBigInt64(<value>);
	toBigUInt64(<value>);
	toFloat16(<value>);
	toFloat32(<value>);
	
	isInt8(<value>);
	isInt16(<value>);
	isInt32(<value>);
	isUInt8(<value>);
	isUInt16(<value>);
	isUInt32(<value>);
	isBigInt64(<value>);
	isBigUInt64(<value>);
	isFloat16(<value>);
		getIn(<object>,<property>);
		getInV(<object>,<property>);
		hasIn(<object>,<property>);
		setIn(<object>,<property>,<value>);
		
		toIndex(<value>);
		toPropertyKey(<value>);
		toInteger(<value>);
		toArray(value);
		toObject(<value>);
		
		isIndex(<value>);
		isPropertyKey(<value>);
		isSameValue(<value1>,<value2>);
		isSameValueZero(<value1>,<value2>);
		isSameValueNonNumber(<value1>,<value2>);
		
		type(<value>);
		createDataProperty(<object>,<property>,<value>);
		createMethodProperty(<object>,<property>,<value>);

Cookie API

getCookie([name]);
hasCookie(<name>);
setCookie(<Options object>);
setCookie(<name>,<value>[,hours=8760[,path="/"[,domain[,secure[,SameSite="Lax"[,HttpOnly]]]]]]);
removeCookie(<Options object>);
removeCookie(<name>[,path="/"[,domain[,secure[,SameSite="Lax"[,HttpOnly]]]]]);
clearCookies(<Options object>);
clearCookies([path="/"[,domain[,sec[,SameSite="Lax"[,HttpOnly]]]]]);

AJAX and CORS API

getText(<url>,<success>);
getJson(<url>,<success>);

ajax(<Options object>);
Options object properties (* = default value): url: *string*, data: *string*, queryType: **"ajax"/"cors"*, type: **"get"/"post"*, success: *function*, error: *function*, format: **"text"/"json"/"xml"*, user: *string*, password: *string*

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