Celestra cheatsheet – v5.6.2 – 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(<string>);
strCodePoints(<string>);
strFromCodePoints(<collection>);
strAt(<string>,<index>[,newChar]);
strSplice(<str>,<index>,<count>[,add]);
strHTMLRemoveTags(<string>);
strHTMLEscape(<string>);
strHTMLUnEscape(<string>);
```

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(<value>);
isNil(<value>);
isPlainObject(<value>);
isTruthy(<value>);
isFalsy(<value>);
isFunction(<v>); + isCallable(<v>);
isConstructorFn(<value>);
isGeneratorFn(<value>);
isAsyncGeneratorFn(<value>);
isAsyncFn(<value>);
isArraylike(<value>);
isTypedArray(<value>);
isArrayBuffer(<value>);
isPrimitive(<value>);
isPromise(<value>);
isIterator(<value>);
isIterable(<value>);
isEmptyObject(<value>);
isEmptyArray(<value>);
isEmptyMap(<value>);
isEmptySet(<value>);
isEmptyIterator(<value>);
isSameObject(<object1>,<object2>);
isSameArray(<array1>,<array2>);
isSameMap(<map1>,<map2>);
isSameSet(<set1>,<set2>);
isSameIterator(<iter1>,<iter2>);
```

Collections API

```
arrayCreate([length=0]);

arrayDeepClone(<array>);

arrayMerge(<target>,<source1>[,sourceN]);

arrayUnique(<collection>);

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(<collection>[,n=100]);
iterCycle(<iter>[,n=Infinity]);

arrayRepeat(<value>[,n=100]);
iterRepeat(<value>[,n=Infinity]);

slice(<collection>[,begin=0[,end=Infinity]]);
withOut(<collection>,<filterCollection>);

reduce(<collection>,<callback>[,initialvalue]);
count(<collection>,<callback>);

take(<collection>[,n=1]);
takeWhile(<collection>,<callback>);
takeRight(<collection>[,n=1]);
takeRightWhile(<collection>,<callback>);

drop(<collection>[,n=1]);
dropWhile(<collection>,<callback>);
dropRight(<collection>[,n=1]);
dropRightWhile(<collection>,<callback>);
```

```
forEach(<collection>,<callback>);
map(<collection>,<callback>);
enumerate(<collection>[,offset=0]);
entries(<collection>[,offset=0]);
size(<collection>);

every(<collection>,<callback>);
some(<collection>,<callback>);
none(<collection>,<callback>);

includes(<collection>,<value>);
contains(<collection>,<value>);
find(<collection>,<callback>);
findLast(<collection>,<callback>);

filter(<collection>,<callback>);
reject(<collection>,<callback>);
partition(<collection>,<callback>);

shuffle(<collection>);
min(<value1>[,valueN]);
max(<value1>[,valueN]);
sort(<collection>[,numbers=false]);
reverse(<collection>);

zip(<collection1>[,collectionN]);
unzip(<collection>);
zipObj(<collection1>,<collection2>);

item(<collection>,<index>);
nth(<collection>,<index>);
first(<collection>);
head(<collection>);
last(<collection>);
initial(<collection>);
tail(<collection>);
flat(<collection>);
concat(<collection1>[,collectionN]);
join(<collection>[,separator=","]);
```

Polyfills

```
Array.fromAsync();

Array.prototype.toReversed();

Array.prototype.toSorted();

Array.prototype.toSpliced();

Array.prototype.with();

crypto.randomUUID();

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

```
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>);
```

Abstract API

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