Celestra cheatsheet – v5.7.0 – https://github.com/Serrin/Celestra/

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

Core API

```
noConflict();
VERSION;
```

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

```
javaHash(<data>[, hexadecimal = false]);
b64Encode(<string>);
b64Decode(<string>);

extend([deep,]<target>,<source1>[,sourceN]);
sizeIn(<object>);
popIn(<object>,<property>);
forIn(<object>,<callback>);
filterIn(<object>,<callback>);

delay(<milisec>).then(<callback>);
sleep(<milisec>).then(<callback>);

randomBoolean();
randomUUIDv7();
timestampID([size=21[,alphabet="ABCDEFGHIJKLMNOPQRSTU
VWXYZabcdefghijklmnopqrstuvwxyz0123456789_-"]]);
nanoid([size=21[,alphabet="123456789ABCDEFGHJKLMNPQRS
TUVWXYZabcdefghijkmnopqrstuvwxyz"]]);
getUrlVars([str=location.search]);
obj2string(<object>);
classof(<variable>[,type[,throw=false]]);
getType(<variable>[,type[,throw=false]]);
bind(<function>,<context>);
unBind(<function>);
constant(<value>);
identity(<value>);
noop();
T();
F();
```

Type checking API

```
isBoolean(<value>);
isTruthy(<value>);
isFalsy(<value>);

isNumber(<value>);
isNumeric(<value>);
isFloat(<value>);
isBigInt(<value>);

isString(<value>);
isChar(<value>);

isFunction(<value>);
isCallable(<value>);

isConstructorFn(<value>);
isClass(<value>);

isGeneratorFn(<value>);

isAsyncFn(<value>);
isAsyncGeneratorFn(<value>);

isNull(<value>);
isUndefined(<value>);
isNullOrUndefined(<value>);
isNil(<value>);

isRegexp(<value>);

isSymbol(<value>);

isPrimitive(<value>);
```

```
isObject(<value>);
isPlainObject(<value>);

isDataView(<value>);
isDate(<value>);
isPromise(<value>);

isElement(<value>);

isMap(<value>);
isWeakMap(<value>);

isSet(<value>);
isWeakSet(<value>);

isArraylike(<value>);
isTypedArray(<value>);
isArrayBuffer(<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>);
```

String API

```
strPropercase(<string>);
strTitlecase(<string>);

strCapitalize(<string>);

strTruncate(<string>);

strUpFirst(<string>);
strDownFirst(<string>);

strReverse(<string>);

strCodePoints(<string>);
strFromCodePoints(<iterator>);

strAt(<string>,<index>[, newChar]);
strSplice(<str>,<index>,<count>[, add]);

strHTMLRemoveTags(<string>);
strHTMLEscape(<string>);
strHTMLUnEscape(<string>);
```

Assertion API

```
assert(<value>[, message = "value"]);
assertTrue(<value>[, message = "value"]);

assertFalse(<value>[, message = "value"]);

assertEqual(<value1>,<value2>[, message = "values"]);
assertNotEqual<value1>,<value2>[, message = "values"]);

assertStrictEqual<value1>,<value2>[, message = "value"]);
assertNotStrictEqual<value1>,<value2>[, message = "values"]);
```

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(<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]);
```

Collections API

```
arrayCreate([length = 0]);
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(<iter>[, 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();
arrayDifference();
arrayIntersection();
arraySymmetricDifference();
arrayUnion();
setDifference();
setIntersection();
setSymmetricDifference();
setUnion();
```

```
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 = ","]);
```

Polyfills

```
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();
```

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>);
```

Cookie API

```
getCookie([name]); and 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]]]]]);
```

Abstract API

```
createDataProperty(<object>,<property>,<value>);
createDataPropertyOrThrow(<object>,<property>,<value>);
createMethodProperty(<object>,<property>,<value>);
createMethodPropertyOrThrow(<object>,<prop>,<value>);
createPolyfillMethod(<object>,<property>,<value>);
createPolyfillProperty(<object>,<property>,<value>);
deleteOwnProperty(<object>,<property>[, Throw = false]);
deletePropertyOrThrow(<object>,<property>);
getIn(<object>,<property>);
getInV(<object>,<property>);
hasIn(<object>,<prop>);

isIndex(<value>);
isLength(<value>);
isLessThan(<value1>,<value2>[, leftFirst = true]);
isPropertyKey(<value>);
isSameType(<value1>,<value2>);
isSameValue(<value1>,<value2>);
isSameValueNonNumber(<value1>,<value2>);
isSameValueZero(<value1>,<value2>);

requireObjectCoercible(<object>);
setIn(<object>,<property>,<value>[, Throw = false]);
toArray(value);
toIndex(<value>);
toInteger(<value>);
toIntegerOrInfinity(<value>);
toLength(<value>);
toObject(<value>);
toPrimitiveValue(<value>);
toPropertyKey(<value>);
type(<value>);
```

AJAX and CORS API

```
getText(<url>,<success>); and 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();	Number.MIN_SAFE_INTEGER;
String.prototype.endsWith();		Number.MAX_SAFE_INTEGER;
String.prototype.startsWith();	Element.prototype.closest();	
Math.acosh();	Element.prototype.getAttributeNames();	Object.fromEntries();
Math.asinh();	Element.prototype.matches();	
Math.atanh();	Element.prototype.toggleAttribute();	Object.is();
Math.cbrt();		
Math.clz32();	ParentNode.append();	String.prototype.at();
Math.cosh();		
Math.expm1();	ParentNode.prepend();	String.prototype.matchAll();
Math.fround();		
Math.hypot();	String.prototype[Symbol.iterator]();	String.prototype.padStart();
Math.imul();	String.prototype.includes();	String.prototype.padEnd();
Math.log1p();	String.prototype.repeat();	
Math.log10();		String.prototype.replaceAll();
Math.log2();	NodeList.prototype.forEach();	
Math.sign();		String.prototype.trimStart();
Math.sinh();	Object.assign();	String.prototype.trimLeft();
Math.tanh();	Object.entries();	
Math.trunc();		String.prototype.trimEnd();
Number.EPSILON;	Object.getOwnPropertyDescriptors();	String.prototype.trimRight();
Number.isNaN(); *and* isNaN();	Object.values();	
Number.isInteger();		Typedarray.prototype.at();
Number.isFinite();	RegExp.prototype.flags;	
Number.isSafeInteger();		TypedArray.prototype.findLast();
Number.parseInt();	window.screenLeft;	TypedArray.prototype.findLastIndex();
Number.parseFloat();	window.screenTop;	