Celestra cheatsheet – v5.8.1 – 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;
```

```
toObject(value);
extend([deep,]target,source1[,sourceN]);
deleteOwnProperty(obj,prop[,Throw=false]);
sizeIn(object);
createPolyfillMethod(object,prop,value);
createPolyfillProperty(object,prop,value);

randomBoolean();
randomUUIDv7();
timestampID([size=21[,alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_-"]]);
nanoid([size=21[,alphabet="123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz"]]);

getUrlVars([str=location.search]);
obj2string(object);

delay(milisec).then(callback);
bind(function,context);
unBind(function);
```

`constant(value);`	`asyncConstant(value);`
`identity(value);`	`asyncIdentity(value);`
`noop();`	`asyncNoop();`
`T();`	`asyncT();`
`F();`	`asyncF();`

Type API

```
type(value);
getType(val[,class[,throw=false]]);
classof(val[,class[,throw=false]]);
isSameType(value1,value2);
isSameClass(value1,value2);
isSameInstance(v1,v2,Contructor);
isDeepStrictEqual(value1,value2);
isNull(value);
isUndefined(value);
isNil(value);
isNumeric(value);
isChar(value);
isFunction(value);
isCallable(value);
isClass(value);
isGeneratorFn(value);
isAsyncFn(value);
isAsyncGeneratorFn(value);
isObject(value);
isPlainObject(value)
isProxy(value);
isElement(value);
isRegexp(value);
isArraylike(value);
isTypedArray(value);
isIterator(value);
isIterable(value);
isCoercedObject(object);
isEmptyValue(value);
isIndex(value);
toIndex(value);
isLength(value);
toLength(value);
isPropertyKey(value);
toPropertyKey(value);
isPrimitive(value);
toPrimitiveValue(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(element[,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]);
	
	assertTypeOf(value,type[,message|error]);
	assertNotTypeOf(value,type[,message|error]);
	
	assertInstanceOf(value,constructor[,message|error]);
	assertNotInstanceOf(value,constructor[,message|error]);
	
	assertIsNil(value[,message|error]);
	assertIsNotNil(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

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(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,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();
		
		window.AsyncFunction();
		
		window.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]]]]]);
```

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(); *and* 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();