Celestra cheatsheet – v4.1.0 – https://github.com/Serrin/Celestra/

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

Core API	DOM	Type checking
delay(<ms>).then(<callback>);	qsa(<selector>[,context]).forEach(<cb>);	isMap(<v>); *and* isWeakMap(<v>);
inherit(<subclass>,<superclass>);	qs(<selector>[,context]);	isSet(<v>); *and* isWeakSet(<v>);
randomInt([max] *or* <min>,<max>);	domReady(<callback>);	isString(<v>); *and* isChar(<v>);
randomFloat([max] *or* <min>,<max>);	domCreate(<type>[,properties[,innerHTML]]);	isNumber(<v>); *and* isNumeric(<v>);
randomString([length[,specChar]]);	domCreate(<element descriptive object>);	isFloat(<v>); *and* isBigInt(<v>);
b64Encode(<string>);	domToElement(<htmlString>);	isDate(<v>); *and* isError(<v>);
b64Decode(<str>);	domGetCSS(<element>[,property]);	isRegexp(<v>); *and* isSymbol(<v>);
javaHash(<data>[,hexa]);	domSetCSS(<element>,<property>,<value>);	isNull(<v>); *and* isUndefined(<v>);
getUrlVars([str=location.search]);	domSetCSS(<element>,<properties>);	isNullOrUndefined(<v>);isNil(<v>);
obj2string(<object>);	domFadeIn(<element>[,duration[,display]]);	isFunction(<value>);
getType(<variable>[,type]);	domFadeOut(<element>[,duration]);	isGeneratorFn(<v>);isAsyncFn(<v>);
extend([deep,]<target>,<source1>[,srcN]);	domFadeToggle(<elem.>[,duration[,display]]);	isDataView(<value>);
deepAssign(<target>,<source1>[,srcN]);	domShow(<element>[,display]);, domHide(<el>);	isBoolean(<v>);, isElement(<v>);
forIn(<object>,<callback>);	domToggle(<element>[,display]);	isObject(<value>);
strRemoveTags(<string>);	domIsHidden(<element>);	isEmptyObject(<value>);
strReverse(<string>);	domSiblings(<element>);	isArraylike(<value>);
strReplaceAll(<str>,<search>,<replace>);	domGetCSSVar(<name>);	isEmptyArray(<value>);
strCodePoints(<string>);	domSetCSSVar(<name>,<value>);	isTypedArray(<value>);
strFromCodePoints(<collection>);	importScript(<url>[,success]);	isArrayBuffer(<value>);
strAt(<string>,<pos>);	importScripts(<scripts> *or* <script1>[,scN]);	isPrimitive(<value>);
toFunction(<function>);	importStyle(<href>[,success]);	isIterator(<v>);, isIterable(<v>);
bind(<function>,<context>);	importStyles(<styles> *or* <style1>[,styleN]);	isPromise(<value>);
constant(<value>); *and* identity(<value>);	setFullscreenOn(<selector> *or* <element>);	isSameObject(<object1>,<object2>);
noop(); *and* T(); *and* F();	setFullscreenOff(); *and* getFullscreen();	isSameArray(<array1>,<array2>);
assert(<condition>[,message]);	form2array(<form>); *and* form2string(<form>);	isSameMap(<map1>,<map2>);
assertLog(<condition>[,message]);	getDoNotTrack();	isSameSet(<set1>,<set2>);
assertAlert(<condition>[,message]);	getLocation(<success>[,error]);	isSameIterator(<iter1>,<iter2>);
noConflict(); *and* VERSION;	createFile(<filename>,<content>[,dType]);	

AJAX and CORS
ajax(<Options object>);, getJson(<url>,<success>);, getText(<url>,<success>);
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*

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

Collections		**Polyfills**
arrayMerge([deep,]<target>,<source1>[,srcN]);	forEach(<collect.>,<callback>);	
arrayUnique(<collection>);	map(<collection>,<callback>);	Array.prototype.flat();
arrayAdd(<array>,<value>);	enumerate(<collection>);	Array.prototype.flatMap();
arrayClear(<array>);		
arrayRemove(<array>,<value>[,all]);	size(<collection>);	globalThis;
arrayRange([start=0[,end=100[,step=1]]]);	min(<collection>);	Object.fromEntries();
arrayCycle(<collection>[,n]);	max(<collection>);	Object.hasOwn();
arrayRepeat(<value>[,n]);		
	includes(<collection>,<value>);	String.prototype.matchAll();
iterRange([start=0[,step=1[,end]]]);	find(<collection>,<callback>);	String.prototype.padStart();
iterCycle(<iter>[,n]);	filter(<collection>,<callback>);	String.prototype.padEnd();
iterRepeat(<value>[,n]);		String.prototype.replaceAll();
	partition(<collection>,<callback>);	String.prototype.trimStart();
arrayUnion(<collection1>[,collectionN]);	groupBy(<collection>,<callback>);	String.prototype.trimLeft();
arrayIntersection(<collection1>,<collection2>);		String.prototype.trimEnd();
arrayDifference(<collection1>,<collection2>);	every(<collection>,<callback>);	String.prototype.trimRight();
arraySymmetricDifference(<collection1>,<collection2>);	some(<collection>,<callback>);	
	none(<collection>,<callback>);	**Non-standard polyfills**
setUnion(<collection1>[,collectionN]);		
setIntersection(<set1>,<set2>);	item(<collection>,<index>);	
setDifference(<set1>,<set2>);	nth(<collection>,<index>);	BigInt.prototype.toJSON();
setSymmetricDifference(<set1>,<set2>);	first(<collection>);	
	head(<collection>);	window.AsyncFunction();
isSuperset(<superset>,<subset>);	last(<collection>);	window.GeneratorFunction();
	initial(<collection>);	
withOut(<collection>,<filterCollection>);	tail(<collection>);	
	slice(<collection>[,begin[,end]]);	
reduce(<collection>,<callback>[,initialvalue]);		
	sort(<collection>[,numberSort]);	
take(<collection>[,n]);	reverse(<collection>);	
takeWhile(<collection>,<callback>);	shuffle(<collection>);	
takeRight(<collection>[,n]);		
takeRightWhile(<collection>,<callback>);	flat(<collection>);	
	concat(<collection1>[,collectionN]);	
drop(<collection>[,n]);	join(<collection>[,separator=","]);	
dropWhile(<collection>,<callback>);		
dropRight(<collection>[,n]);	zip(<collection1>[,collectionN]);	
dropRightWhile(<collection>,<callback>);	unzip(<collection>);	