JavaScript cheatsheet – v5.8.0 – https://github.com/Serrin/Celestra/

```
Web Storage api and JSON
                                          element.dataset & data-* attributes
                                                                                                 TypedArray
                                                                               IE10+11 compatible
IE8+
                                          - IE11 compatible
                                          - element data-* attributes
                                                                               new <TypedArray>(); ES2017
localStorage:
                                          - no methods and events
                                                                               new <TypedArray>(length);
localStorage.length;
                                                                               new <TypedArray>(typedArray);
localStorage.kev(index);
                                          camelcase:
                                                                               new <TypedArray>(object);
localStorage.getItem(key);
                                          element.data-name
                                                                               new <TypedArray>(buffer[,byteOffset[,len]]);
localStorage.setItem(key, data);
                                            -> element.dataset.name
localStorage.removeItem(key);
                                          element.data-first-second
                                                                               Int8Arrav();
localStorage.clear();
                                            -> element.dataset.firstSecond
                                                                               -128 to 127, 1 byte, int8 t, Shortint
                                                                               Uint8Array();
                                                                               0 to 255, 1 byte, uint8 t, Byte
sessionStorage:
                                          set:
sessionStorage.length;
                                          element.dataset.name = "value";
                                                                               Uint8ClampedArray();
sessionStorage.kev(index);
                                          element.dataset["name"] = "value";
                                                                               0 to 255, 1 byte, uint8 t, Byte
sessionStorage.getItem(key);
                                          element.setAttribute("data-name",
                                                                               Int16Array();
sessionStorage.setItem(key, data);
                                          "value"):
                                                                               -32768 to 32767, 2 byte, int16 t, Smallint
sessionStorage.removeItem(key);
                                          element["data-name"] = "value";
                                                                               Uint16Arrav();
sessionStorage.clear();
                                                                               0 to 65535, 2 byte, uint16 t, Word
                                                                               Int32Array();
                                          element.dataset.name;
                                                                                -2147483648 to 2147483647, 4 byte, int32 t
hasItem:
localStorage.getItem(key) !== null
                                          element.dataset["name"];
                                                                               Uint32Array();
sessionStorage.getItem(key) !== null
                                          element.getAttribute("data-name");
                                                                               0 to 4294967295, 4 byte, uint32 t, Longword
                                          element["data-name"];
                                                                               BigInt64Array(); - not in IE10-11
setJSON:
                                                                               -2**63 to 2**63-1, 8 byte, int64 t, Int64
localStorage.setItem(key,
                                                                               BigUint64Array(); - not in IE10-11
                                          remove:
                                                                               0 to 2**64-1, 8 byte, uint64 t, Qword
JSON.stringify(object));
                                          element.removeAttribute("data-
sessionStorage.setItem(key,
                                          name");
                                                                               Float16Array(); - not in IE10-11
                                                                               -65504 to 65504, 2 byte
JSON.stringify(object));
                                          check:
                                                                               Float32Array();
getJSON:
                                          element.hasAttribute("data-name");
                                                                               1.2x10-38 to 3.4x1038, 4 byte, float, Real
JSON.parse(localStorage.getItem(key));
                                                                               Float64Array();
JSON.parse(sessionStorage.getItem(key));
                                                                               5.0x10-324 to 1.8x10308, 8 byte, Double
                                                         DOM events
target.addEventListener(<type>, <listener>[, useCapture]); or target.addEventListener(<type>, <listener>[, options]);
target.removeEventListener(<type>,<listener>[,useCapture]); or target.removeEventListener(<type>,<listener>[,options]);
target.dispatchEvent(<event>);
target.type(); or target["type"]();
```

```
element.classList
                                                                                           JSON
IE10+IE11 don't have support for classList on SVG or MathML | IE8+
elements.
                                                                Valid Data Types
element.classList.add(String[,String]);
                                                                - string
IE10+11: ves (except the multiple arguments)
                                                                - number
                                                                - object (containing valid JSON values)
element.classList.remove(String[,String]);
                                                                - arrav
IE10+11: yes (except the multiple arguments)
                                                                - boolean
- Removing a class that does not exist, does NOT throw an - date
error.
                                                                - null
element.classList.contains(String);
                                                                Invalid Data Types
                                                                - function
IE10+11: yes
                                                                - Symbol
element.classList.toggle(String[,force]);
                                                                - NaN, Infinity, undefined - will be "null"
                                                                - an object with method(s) (functions)
IE10+11: yes (except the second argument)
- When only one argument is present: Toggle class value; if - Map, Set, WeakMap, WeakSet - fix: convert to array
class exists then remove it and return false, if not, then add - BigInt - fixed in Celestra - BigInt.prototype.toJSON();
it and return true.
- When a second argument is present: If the second argument | JSON.stringify(value[,replacer[,space]]);
evaluates to true, add specified class value, and if it Convert a JavaScript object to a JSON string.
evaluates to false, remove it.
                                                                JSON.stringify( { a: 1, b: "2", c: true } );
                                                                // -> "{\"a\":1,\"b\":\"2\",\"c\":true}"
element.classList.item(Number);
IE10+11: yes
                                                                JSON.stringify([1, 2, 3, 4, 5]);
element.classList.length;
                                                                // -> "[1,2,3,4,5]"
IE10+11: yes
                                                                JSON.parse(text[,reviver]);
element.classList.replace(oldClass, newClass);
                                                                Parses a JSON string and returns a JavaScript object.
IE10+11: No and the method isn't compatible with the Safari and
mobile browsers too.
                                                                JSON.parse(JSON.stringify( {a: 1, b: "2", c: true} ));
                                                                // -> Object { a: 1, b: "2", c: true }
Remove all classes:
element.className = "";
                                                                JSON.parse(JSON.stringify([1, 2, 3, 4, 5]));
                                                                // -> Array(5) [ 1, 2, 3, 4, 5 ]
```

```
BigInt (Int64)
                           DOMParser
IE9: XML support
                                                               let theBiggestInt = 9007199254740991n;
IE10+IE11: XML, SVG and HTML support
                                                               let alsoHuge = BigInt(9007199254740991);
                                                               let hugeString = BigInt("9007199254740991");
var parser = new DOMParser();
                                                               let hugeHex = BigInt("0x1ffffffffffffff");
                                                               let hugeBin =
var doc = parser.parseFromString("sourceStr", "application/xml");
Returns a Document, but not a SVGDocument nor a HTMLDocument.
                                                               1111111");
var parser = new DOMParser();
                                                                // all -> 9007199254740991n
var doc = parser.parseFromString(sourceStr, "image/svg+xml");
Returns a SVGDocument, which also is a Document.
                                                                typeof 1n;
                                                                             // -> "bigint"
                                                                typeof BigInt("1"); // -> "bigint"
var parser = new DOMParser();
var doc = parser.parseFromString(sourceStr, "text/html");
                                                               Operators: +, *, -, **, %, Bitwise operators (e.g.: >>>)
Returns a HTML document.
                                                                ! On
                                                                        // -> t.rue
                                                                        // -> false
                                                                l1n
                   DOMParser sample function
                                                                4n / 2n / / -> 2n
                                                                5n / 2n // -> 2n, not 2.5n -> rounded
function parseHTML (str) {
 return Array.from(
                                                               ln < 2 // -> true
    (new DOMParser())
                                                               2n > 1 // -> true
     .parseFromString(str, "text/html")
                                                               2n > 2 // -> false
     .childNodes[0]
                                                               2n >= 2 // -> true
     .childNodes[1]
     .childNodes
                                                               let mixed = [4n, 6, -12n, 10, 4, 0, 0n];
 );
                                                               mixed.sort();// \rightarrow [-12n, 0, 0n, 4n, 4, 6, 10]
                                                               Methods
parseHTML(
 "<div>123<"
                                                               BigInt.asIntN();
 + "<div>456</div>"
                                                                 - BigInt value to a signed integer
 + "<div>7</div>"
                                                               BigInt.asUintN();
                                                                 - BigInt value to an unsigned integer
 + "8"
                                                               BigInt.prototype.toLocaleString();
// -> Array(4) [ div, div, div, p ]
                                                               BigInt.prototype.toString();
// Tested in IE11, Edge, Firefox and Chrome.
                                                               BigInt.prototype.valueOf();
```

```
Fetch
                                                                                            Fetch POST
Firefox 39, Chrome 42, Edge14, Opera29, Safari 10.1, Samsung I. 4.0
                                                                       // Example POST method implementation:
                                                                       // Default options are marked with *
// Example GET method implementation with TEXT:
                                                                       async function postData(url = "url", data = {}) {
fetch("https://api.coindesk.com/v1/bpi/currentprice.json")
                                                                         const response = await fetch(url, {
  .then( response => response.text() )
                                                                           method: "POST",
  .then( data => console.log(data) )
                                                                           // *GET, POST, PUT, DELETE, etc.
  .catch( error => console.log(error) );
                                                                           mode: "cors",
                                                                           // no-cors, *cors, same-origin
// Example GET method implementation with JSON:
                                                                           cache: "no-cache",
fetch("https://api.coindesk.com/v1/bpi/currentprice.json")
                                                                           // *default, no-cache, reload, force-cache,
  .then( response => response.json() )
                                                                       onlv-if-cached
  .then( data => console.log(data.bpi.USD.rate) )
                                                                           credentials: "same-origin",
  .catch( error => console.log(error) );
                                                                           // include, *same-origin, omit
                                                                           headers: {"Content-Type": "application/json"},
// Example GET method implementation with TEXT and JSON:
                                                                           // "Content-Type": "application/x-www-form-
fetch("https://api.coindesk.com/v1/bpi/currentprice.json")
                                                                       urlencoded"
  .then( response => response.text() )
                                                                           redirect: "follow",
  .then(text => console.log(JSON.parse(text).bpi.USD.rate+"\n"+text))
                                                                           // manual, *follow, error
  .catch( error => console.log(error) );
                                                                           referrerPolicy: "no-referrer",
                                                                           // no-referrer, *no-referrer-when-downgrade,
// Example POST method implementation with upload JSON data:
                                                                       origin, origin-when-cross-origin, same-origin,
const data = { username: "example" };
                                                                       strict-origin, strict-origin-when-cross-origin,
fetch("https://example.com/profile", {
                                                                       unsafe-url
   method: "POST", // or "PUT"
                                                                           body: JSON.stringify(data)
   headers: { "Content-Type": "application/json", },
                                                                           // body data type must match "Content-Type"
   body: JSON.stringify(data),
                                                                       header
                                                                         });
  .then(response => response.json())
                                                                         return response.json();
  .then(data => { console.log("Success:", data); })
                                                                        // parses JSON response into native JavaScript
  .catch((error) => { console.error("Error:", error); });
                                                                       objects
                                                                       postData("https://example.com/answer", { answer:
                                                                       42 })
                                                                         .then(data => { console.log(data); });
                                                                         // JSON data parsed by `data.json()` call
```

Nullish coalescing operator x ?? y	Logical nullish assignment x ??= y	Logical AND assignment x &&= y	Logical OR assignment x = y	
FF 72, Chrome and Edge 80, Safari 13.1, Safari on iOS 13.4, Samsung Internet 13				
The nullish coalescing operator (??) is a logical operator that returns its right-hand side operand when its left-hand side operand is null or undefined, and otherwise returns its left-hand side operand.	The logical nullish assignment operator only assigns if x is nullish (null or undefined).	The logical AND assignment operator only assigns if x is truthy.	The logical OR assignment operator only assigns if x is falsy. (false, 0, -0, 0n, "", '', null, undefined, NaN)	
<pre>const nullValue = null; const emptyText = ""; // falsy const someNumber = 42; const valA = nullValue ?? "defaultA"; // "defaultA" const valB = emptyText ?? "default B"; // "" (empty string is not null or undefined) const valC = someNumber ?? 0; // 42</pre>	<pre>function config (options) { options.duration ??= 100; options.speed ??= 25; return options; } config({duration: 125}); // {duration: 125, speed: 25} config({}); // {duration: 100, speed: 25}</pre>	<pre>let x = 0; let y = 1; x &&= 0; // 0 x &&= 1; // 0 y &&= 1; // 1 y &&= 0; // 0</pre>	<pre>const a = { duration: 50, title: "" }; a.duration = 10; // 5 a.title = "title is empty."; // "title is empty"</pre>	
<pre>let count = 0; let text = ""; let qty = count 42; // 42 let message = text "hi!"; // "hi!"</pre>	<pre>const a = { duration: 50 }; a.duration ??= 10; // 50 a.speed ??= 25; // 25</pre>	let a = 1; let b = 0; a &&= 2; // 2 b &&= 2; // 0		
Nullish coalescing operator (??)	х ?? у	<pre>equivalent (x != null) ? x : y</pre>	not equivalent	
Logical nullish assignment (??=)	x ??= y	(x := null) : x : y x := (x = y);	x = x ?? y;	
Logical AND assignment (&&=)	x ::- y x &&= y	x & & (x = y);	x = x & & y;	
Logical OR assignment (=)	x = y	$x \mid (x = y);$	x = x y;	
Dogical on abolymient (-)	A 11- Y	Δ (Δ	Δ Δ <u>γ</u> ,	

Reflect object

ES6, no IE11 support - FF 42, Chrome 49, Edge 12, Safari 10, Safari on iOS 10, Samsung Internet 5.0

ES6, no IE11 support - FF 42, Chrome 49, Edge 12, Safari 10, Safari on iOS 10, Samsung Internet 5.0							
Function	Description	equivalent					
<pre>Reflect.apply(target, thisArgument, argumentsList);</pre>	Calls a target function with arguments as specified by the argumentsList parameter.	<pre>Function.prototype.apply.call(target, thisArgument, argumentsList);</pre>					
<pre>Reflect.construct(target,argumentsList [,newTarget]);</pre>	The new operator as a function.	<pre>new target(argumentsList);</pre>					
<pre>Reflect.defineProperty(target,propertyKey, attributes);</pre>	Similar to Object.defineProperty(). Returns a boolean that is true if the property was successfully defined.	<pre>Object.defineProperty(target, propertyKey,attributes);</pre>					
<pre>Reflect.deleteProperty(target,propertyKey);</pre>	The delete operator as a function.	<pre>delete target[propertyKey];</pre>					
<pre>Reflect.get(target,propertyKey[,receiver]);</pre>	Returns the value of the property of the object.	<pre>target[propertyKey];</pre>					
<pre>Reflect.getOwnPropertyDescriptor(target, propertyKey);</pre>	Returns a property descriptor of the given property if it exists on the object, undefined otherwise.	<pre>Object.getOwnPropertyDescriptor(target, propertyKey);</pre>					
Reflect.getPrototypeOf(target);	Object.getPrototypeOf(target);	Object.getPrototypeOf(target);					
Reflect.has(target,propertyKey);	Returns a boolean whether the target has the property.	<pre>propertyKey in target;</pre>					
Reflect.isExtensible(target);	Returns a boolean that is true if the target is extensible.	Object.isExtensible(target);					
Reflect.ownKeys(target);	Returns an array of the target object's own (not inherited) property keys.	<pre>Object.getOwnPropertyNames(target).concat(Object.getOwnPropertySymbols(target));</pre>					
<pre>Reflect.preventExtensions(target);</pre>	Prevents new properties from ever being added to an object. Similar to Object.preventExtensions().	Object.preventExtensions(target);					
<pre>Reflect.set(target,propertyKey,value [,receiver]);</pre>	Assigns values to properties. Returns a boolean that is true if the update was successful.	<pre>target[propertyKey] = value;</pre>					
<pre>Reflect.setPrototypeOf(target,prototype);</pre>	Sets the prototype of an object. Returns a boolean that is true if the update was successful.	Object.setPrototypeOf(target,prototype);					

```
Map Object
                                                                                 Set Object helper functions
                                                                 function isSuperset(set, subset) {
var myMap = new Map([iterable]);
                                                                   for (const elem of subset) {
                                                                     if (!set.has(elem)) { return false; }
// The Map objects are iterable.
for (let [key, value] of myMap)
                                                                   return true;
  console.log(`${key} = ${value}`);
                                                                 function union(setA, setB) {
var cloneMap = new Map(myMap);
                                                                   const union = new Set(setA);
Map.prototype.size;
                                                                   for (const elem of setB) { union.add(elem); }
Map.prototype.get(<key>); -> value/undefines
                                                                   return union;
Map.prototype.set(<key>,<value>); -> Map object
Map.prototype.has(<key>); -> boolean
                                                                 function intersection(setA, setB) {
Map.prototype.delete(<key>); -> boolean
                                                                   const intersection = new Set();
Map.prototype.clear(); -> undefined
                                                                   for (const elem of setB) {
                                                                     if (setA.has(elem)) { intersection.add(elem); }
Map.prototype.forEach(function (value, key, map)); -> undefined
Map.prototype.keys(); -> iterator of keys
                                                                   return intersection;
Map.prototype.values(); -> iterator of values
Map.prototype.entries(); -> iterator of [key, value]
                                                                 function difference(setA, setB) {
                                                                   const difference = new Set(setA);
                           Set Object
                                                                   for (const elem of setB) { difference.delete(elem); }
                                                                   return difference;
var mySet = new Set([iterable]);
                                                                 function symmetricDifference(setA, setB) {
                                                                   const d = new Set(setA);
// The Set objects are iterable.
for (const item of mySet) { console.log(item); }
                                                                   for (const e of setB) {
                                                                     if (d.has(e)) { d.delete(e); } else { d.add(e); }
var cloneSet = new Set(mySet);
Set.prototype.size;
                                                                   return d;
Set.prototype.add(<value>); -> Set object
Set.prototype.has(<value>); -> boolean
Set.prototype.delete(<value>); -> boolean
                                                                 const setA = new Set([1,2,3,4]), setB = new Set([2, 3]),
Set.prototype.clear(); -> undefined
                                                                 setC = new Set([3, 4, 5, 6])
                                                                 isSuperset(setA, setB);
                                                                                                  // true
Set.prototype.forEach(function (value, value, set)); -> undefined |union(setA, setC);
                                                                                                 // Set {1, 2, 3, 4, 5, 6}
                                                                 intersection(setA, setC);
                                                                                                // Set {3, 4}
Set.prototype.keys(); -> iterator of values
Set.prototype.values(); -> iterator of values
                                                                 difference(setA, setC);
                                                                                                // Set {1, 2}
Set.prototype.entries(); -> iterator of [value, value]
                                                                 symmetricDifference(setA, setC); // Set {1, 2, 5, 6}
```

```
Array.fromAsync();
                                                                                        Set methods
                                                               Chrome, Chrome Android, Edge, WebView Android v122
Array.fromAsync(<object>[,mapFn[,thisArg]])
  .then((resultArray) => /* todo with resultArray */);
                                                               Firefox, Firefox for Android v127
                                                               Safari, Safari on iOS, WebView on iOS v17
                                                               Opera v108, Opera Android v81
Object types: async iterable, iterable (Array, Map, Set,
NodeList, etc.), array-like
                                                               Samsung Internet v26.0
                                                               Deno 1.42, Node. is 22.0.0
mapfn parameters: element, index
                                                               Set.prototype.intersection(other): Set
async function* asyncIterable () {
                                                               Set.prototype.union(other): Set
  for (let i = 0; i < 5; i++) { await new
                                                               Set.prototype.difference(other): Set
Promise((resolve) => setTimeout(resolve, 50*i)); yield i; }
                                                               Set.prototype.symmetricDifference(other): Set
                                                               Set.prototype.isSubsetOf(other): Boolean
                                                               Set.prototype.isSupersetOf(other): Boolean
Arrav.fromAsvnc(asvncIterable())
                                                               Set.prototype.isDisjointFrom(other): Boolean
  .then((res) => console.log("asyncIterable1: "+res));
// asyncIterable1: 0,1,2,3,4
                                                               var setA = new Set([1]);
Array.fromAsync(asyncIterable(), (x) \Rightarrow x*2)
                                                               var setB = new Set([1,2]);
  .then((res) => console.log("asyncIterable2: "+res));
                                                               var setC = new Set([2,3]);
// asyncIterable2: 0,2,4,6,8
                                                               console.log( setB.intersection(setC) );
Array.fromAsync([4,5,6,7,8])
                                                               // Set [ 2 ]
  .then((res) => console.log("[4,5,6,7,8]: "+res));
                                                               console.log( setB.union(setC) );
// [4,5,6,7,8]: 4,5,6,7,8
                                                               // Set(3) [ 1, 2, 3 ]
Array.fromAsync([4,5,6,7,8], (x) => x*2)
                                                               console.log( setB.difference(setC) );
  .then((res) => console.log("[4,5,6,7,8] + fn: "+res));
                                                               // Set [ 1 ]
// [4,5,6,7,8] + fn: 8,10,12,14,16
                                                               console.log( setB.symmetricDifference(setC) );
                                                               // Set [ 1, 3 ]
Arrav.fromAsvnc(new Set([4,5,6,6,10]))
  .then((res) => console.log("Set: "+res));
                                                               console.log( setB.isSupersetOf(setA) );
// Set: 4,5,6,10
                                                               // true
                                                               console.log( setA.isSupersetOf(setC) );
Array.fromAsync(new Set([4,5,6,6,10]), (x) \Rightarrow x*2)
                                                               // false
  .then((res) => console.log("Set + fn: "+res));
                                                               console.log( setA.isSubsetOf(setB) );
// Set + fn: 8,10,12,20
                                                               // true
                                                               console.log( setA.isSubsetOf(setC) );
Array.fromAsync({"0": 3, "1": 4, "2": 5, length: 3})
                                                               // false
  .then((res) => console.log("arraylike: "+res));
// arravlike: 3,4,5
                                                               console.log( setA.isDisjointFrom(setC) );
                                                               // true - there are no common elements
Array.fromAsync({"0": 3, "1": 4, "2": 5, length: 3}, (x) =>
                                                               console.log( setA.isDisjointFrom(setB) );
x*2).then((res) => console.log("arraylike + fn: "+res));
                                                               // false - there are common elements
// arraylike + fn: 6,8,10
```

```
Iterator methods
                                                                                        Iterator methods samples
                                                                    var A1 = [1, 2, 3, 4, 5, 6];
Firefox, Firefox for Android 131
Chrome, Edge, Webview Android 122
                                                                    console.log( Iterator.from(A1) );
Opera 108
                                                                    // Iterator { 1, 2, 3, 4, 5, 6 }
Safari, Safari in iOS, Webview on iOS 18.4 (?)
Samsung Internet 26.0
                                                                   console.log(A1.values().drop(3));
                                                                    // Iterator { 4, 5, 6 }
Deno 1.42, Node. is (?)
                                                                    console.log(A1.values().every((x) \Rightarrow x \Rightarrow 0)); // true
                                                                    console.log(A1.values().every((x) \Rightarrow x \Rightarrow 3)); // false
Iterator.from(object);
                                                                    console.log(A1.values().filter((x) \Rightarrow x > 3));
Iterator.prototype.drop(limit);
                                                                    // Iterator { 4, 5, 6 }
Iterator.prototype.every(callbackFn(Element, index));
                                                                   console.log(A1.values().find((x) \Rightarrow x > 3));
                                                                    // 4
Iterator.prototype.filter(callbackFn(Element, index));
                                                                    var M1 = new Map([["a", 1], ["b", 2], ["c", 3]]);
Iterator.prototype.find(callbackFn(Element, index));
                                                                    var M2 = new Map([["d", 4], ["e", 5], ["f", 6]]);
                                                                    console.log(new Map([M1, M2].values().flatMap((x) \Rightarrow x)));
Iterator.prototype.flatMap(callbackFn(Element, index));
                                                                    // \text{Map}(6) \{ a \rightarrow 1, b \rightarrow 2, c \rightarrow 3, d \rightarrow 4, e \rightarrow 5, f \rightarrow 6 \}
                                                                   console.log(A1.values().take(2).forEach((el, i) =>
Iterator.prototype.forEach(callbackFn(Element, index));
                                                                    console.log(i+": "+el)));
                                                                   // "0 : 1", "1 : 2"
Iterator.prototype.map(callbackFn(Element, index));
Iterator.prototype.reduce(callbackFn(accumulator,
                                                                   console.log(A1.values().map((x) \Rightarrow x * 2));
currentValue, currentIndex), initialValue: Optional);
                                                                   // Iterator { 2, 4, 6, 8, 10, 12 }
Iterator.prototype.some(callbackFn(Element, index));
                                                                    console.log(A1.values().reduce((ac, it) \Rightarrow (ac + it), 0));
                                                                    // 21
Iterator.prototype.take(limit);
                                                                    console.log(A1.values().some((x) => x > 3)); // true
                                                                    console.log(A1.values().some((x) \Rightarrow x \Rightarrow 6)); // false
Iterator.prototype.toArray();
 Equivalent to Array.from(iterator) and [...iterator]
                                                                    console.log(A1.values().take(3));
Iterator.prototype[Symbol.iterator]();
                                                                    // Iterator { 1, 2, 3 }
                                                                    console.log(A1.values().toArray());
                                                                    // Array(6) [ 1, 2, 3, 4, 5, 6 ]
```

Javascript Equality comparisons and sameness							
х	Y	loose equality	strict equality	Same-value	Same-value-zero		
		X == Y	х === ч	Object.is(X, Y)	[X].includes(Y)		
					X === Y (X !== X && Y !== Y)		
undefined	undefined	✓ true	✓ true	✓ true	✓ true		
null	null	✓ true	✓ true	✓ true	✓ true		
true	true	✓ true	✓ true	✓ true	✓ true		
false	false	✓ true	✓ true	✓ true	✓ true		
"foo"	"foo"	✓ true	✓ true	✓ true	✓ true		
0	0	✓ true	✓ true	✓ true	✓ true		
+0	-0	✓ true	✓ true	X false	✓ true		
+0	0	✓ true	✓ true	✓ true	✓ true		
-0	0	✓ true	✓ true	X false	✓ true		
0 n	-0n	✓ true	✓ true	✓ true	✓ true		
0	false	✓ true	X false	X false	X false		
11 11	false	✓ true	X false	X false	X false		
11 11	0	✓ true	X false	X false	X false		
"0"	0	✓ true	X false	X false	X false		
"17"	17	✓ true	X false	X false	X false		
[1, 2]	"1,2"	✓ true	X false	X false	X false		
new String("foo")	"foo"	✓ true	X false	X false	X false		
null	undefined	✓ true	X false	X false	X false		
null	false	X false	X false	X false	X false		
undefined	false	X false	X false	X false	X false		
{foo: "bar"}	{foo: "bar"}	X false	X false	X false	X false		
new String("foo")	new String("foo")	X false	X false	X false	X false		
0	null	X false	X false	X false	X false		
0	NaN	X false	X false	X false	X false		
"foo"	NaN	X false	X false	X false	🗙 false		
NaN	NaN	X false	X false	✓ true	✓ true		