JavaScript cheatsheet – v5.6.0 – https://github.com/Serrin/Celestra/

Web Storage api and JSON

IE8+

localStorage:
```
localStorage.length;
localStorage.key(index);
localStorage.getItem(key);
localStorage.setItem(key, data);
localStorage.removeItem(key);
localStorage.clear();
```

sessionStorage:
```
sessionStorage.length;
sessionStorage.key(index);
sessionStorage.getItem(key);
sessionStorage.setItem(key, data);
sessionStorage.removeItem(key);
sessionStorage.clear();
```

hasItem:
```
localStorage.getItem(key) !== null
sessionStorage.getItem(key) !== null
```

setJSON:
```
localStorage.setItem(key,
JSON.stringify(object));
sessionStorage.setItem(key,
JSON.stringify(object));
```

getJSON:
```
JSON.parse(localStorage.getItem(key));
JSON.parse(sessionStorage.getItem(key));
```

element.dataset & data-* attributes

- IE11 compatible
- element data-* attributes
- no methods and events

camelcase:
```
element.data-name
  -> element.dataset.name
element.data-first-second
  -> element.dataset.firstSecond
```

set:
```
element.dataset.name = "value";
element.dataset["name"] = "value";
element.setAttribute("data-name",
"value");
element["data-name"] = "value";
```

get:
```
element.dataset.name;
element.dataset["name"];
element.getAttribute("data-name");
element["data-name"];
```

remove:
```
element.removeAttribute("data-
name");
```

check:
```
element.hasAttribute("data-name");
```

TypedArray

IE10+11 compatible

```
new <TypedArray>(); ES2017
new <TypedArray>(length);
new <TypedArray>(typedArray);
new <TypedArray>(object);
new <TypedArray>(buffer[,byteOffset[,len]]);
```

Int8Array();
-128 to 127, 1 byte, int8_t, Shortint
Uint8Array();
0 to 255, 1 byte, uint8_t, Byte
Uint8ClampedArray(); - not in IE10-11
0 to 255, 1 byte, uint8_t, Byte
Int16Array();
-32768 to 32767, 2 byte, int16_t, Smallint
Uint16Array();
0 to 65535, 2 byte, uint16_t, Word
Int32Array();
-2147483648 to 2147483647, 4 byte, int32_t
Uint32Array();
0 to 4294967295, 4 byte, uint32_t, Longword
BigInt64Array(); - not in IE10-11
-2**63 to 2**63-1, 8 byte, int64_t, Int64
BigUint64Array(); - not in IE10-11
0 to 2**64-1, 8 byte, uint64_t, QWord
Float32Array();
1.2x10-38 to 3.4x1038, 4 byte, float, Real
Float64Array();
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

IE10+IE11 don't have support for classList on SVG or MathML elements.

element.classList.add(String[,String]);
IE10+11: yes (except the multiple arguments)

element.classList.remove(String[,String]);
IE10+11: yes (except the multiple arguments)
- Removing a class that does not exist, does NOT throw an error.

element.classList.contains(String);
IE10+11: yes

element.classList.toggle(String[,force]);
IE10+11: yes (except the second argument)
- When only one argument is present: Toggle class value; if class exists then remove it and return false, if not, then add it and return true.
- When a second argument is present: If the second argument evaluates to true, add specified class value, and if it evaluates to false, remove it.

element.classList.item(Number);
IE10+11: yes

element.classList.length;
IE10+11: yes

element.classList.replace(oldClass, newClass);
IE10+11: No and the method isn't compatible with the Safari and mobile browsers too.

Remove all classes:
element.className = "";

JSON

IE8+

Valid Data Types
- string
- number
- object (containing valid JSON values)
- array
- boolean
- date
- null

Invalid Data Types
- function
- Symbol
- NaN, Infinity, undefined – *will be "null"*
- an object with method(s) (functions)
- Map, Set, WeakMap, WeakSet – *fix: convert to array*
- BigInt - *fixed in Celestra - BigInt.prototype.toJSON();*

JSON.stringify(value[,replacer[,space]]);
Convert a JavaScript object to a JSON string.

```
JSON.stringify( { a: 1, b: "2", c: true } );
// -> "{\"a\":1,\"b\":\"2\",\"c\":true}"

JSON.stringify( [1, 2, 3, 4, 5] );
// -> "[1,2,3,4,5]"
```

JSON.parse(text[,reviver]);
Parses a JSON string and returns a JavaScript object.

```
JSON.parse(JSON.stringify( {a: 1, b: "2", c: true} ));
// -> Object { a: 1, b: "2", c: true }

JSON.parse(JSON.stringify( [1, 2, 3, 4, 5] ));
// -> Array(5) [ 1, 2, 3, 4, 5 ]
```

DOMParser

```
IE9: XML support
IE10+IE11: XML, SVG and HTML support

var parser = new DOMParser();
var doc = parser.parseFromString("sourceStr", "application/xml");
Returns a Document, but not a SVGDocument nor a HTMLDocument.

var parser = new DOMParser();
var doc = parser.parseFromString(sourceStr, "image/svg+xml");
Returns a SVGDocument, which also is a Document.

var parser = new DOMParser();
var doc = parser.parseFromString(sourceStr, "text/html");
Returns a HTML document.
```

DOMParser sample function

```
function parseHTML (str) {
  return Array.from(
    (new DOMParser())
      .parseFromString(str, "text/html")
      .childNodes[0]
      .childNodes[1]
      .childNodes
  );
}

parseHTML(
  "<div>1<p>2</p><p>3</p></div>"
  + "<div>4<p>5</p><p>6</p></div>"
  + "<div>7</div>"
  + "<p>8</p>"
);
// -> Array(4) [ div, div, div, p ]
// Tested in IE11, Edge, Firefox and Chrome.
```

BigInt (Int64)

```
let theBiggestInt = 9007199254740991n;
let alsoHuge = BigInt(9007199254740991);
let hugeString = BigInt("9007199254740991");
let hugeHex = BigInt("0x1fffffffffffff");
let hugeBin =
BigInt("0b11111111111111111111111111111111111111111111111
1111111");
// all -> 9007199254740991n

typeof 1n;          // -> "bigint"
typeof BigInt("1"); // -> "bigint"

Operators: +, *, -, **, %, Bitwise operators (e.g.: >>>)

!0n      // -> true
!1n      // -> false

4n / 2n  // -> 2n
5n / 2n  // -> 2n, not 2.5n -> rounded

1n < 2   // -> true
2n > 1   // -> true
2n > 2   // -> false
2n >= 2  // -> true

let mixed = [4n, 6, -12n, 10, 4, 0, 0n];
mixed.sort();// -> [-12n, 0, 0n, 4n, 4, 6, 10]

Methods
BigInt.asIntN();
  - BigInt value to a signed integer
BigInt.asUintN();
  - BigInt value to an unsigned integer
BigInt.prototype.toLocaleString();
BigInt.prototype.toString();
BigInt.prototype.valueOf();
```

Fetch	Fetch POST
Firefox 39, Chrome 42, Edge14, Opera29 , Safari 10.1, Samsung I. 4.0 *// Example GET method implementation with TEXT:* fetch("https://api.coindesk.com/v1/bpi/currentprice.json") .then(response => response.text()) .then(data => console.log(data)) .catch(error => console.log(error)); *// Example GET method implementation with JSON:* fetch("https://api.coindesk.com/v1/bpi/currentprice.json") .then(response => response.json()) .then(data => console.log(data.bpi.USD.rate)) .catch(error => console.log(error)); *// Example GET method implementation with TEXT and JSON:* fetch("https://api.coindesk.com/v1/bpi/currentprice.json") .then(response => response.text()) .then(text => console.log(JSON.parse(text).bpi.USD.rate+"\n"+text)) .catch(error => console.log(error)); *// Example POST method implementation with upload JSON data:* const data = { username: "example" }; fetch("https://example.com/profile", { method: "POST", // or "PUT" headers: { "Content-Type": "application/json", }, body: JSON.stringify(data), }) .then(response => response.json()) .then(data => { console.log("Success:", data); }) .catch((error) => { console.error("Error:", error); });	*// Example POST method implementation:* *// Default options are marked with ** async function postData(url = "url", data = {}) { const response = await fetch(url, { method: "POST", *// *GET, POST, PUT, DELETE, etc.* mode: "cors", *// no-cors, *cors, same-origin* cache: "no-cache", *// *default, no-cache, reload, force-cache, only-if-cached* credentials: "same-origin", *// include, *same-origin, omit* headers: {"Content-Type": "application/json"}, *// "Content-Type": "application/x-www-form-urlencoded"* redirect: "follow", *// manual, *follow, error* referrerPolicy: "no-referrer", *// no-referrer, *no-referrer-when-downgrade, origin, origin-when-cross-origin, same-origin, strict-origin, strict-origin-when-cross-origin, unsafe-url* body: JSON.stringify(data) *// body data type must match "Content-Type" header* }); return response.json(); *// parses JSON response into native JavaScript 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	FF 79, Chrome and Edge 85, Safari 14, Samsung Internet 14		
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)
`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`	`function config (options) {` `  options.duration ??= 100;` `  options.speed ??= 25;` `  return options;` `}` `config({duration: 125});` `// {duration: 125, speed: 25}` `config({});` `// {duration: 100, speed: 25}`	`let x = 0;` `let y = 1;` `x &&= 0; // 0` `x &&= 1; // 0` `y &&= 1; // 1` `y &&= 0; // 0`	`const a = {` `  duration: 50,` `  title: ""` `};` `a.duration ||= 10;` `// 5` `a.title ||= "title is empty.";` `// "title is empty"`
`let count = 0;` `let text = "";` `let qty = count || 42; // 42` `let message = text || "hi!"; // "hi!"`	`const a = { duration: 50 };` `a.duration ??= 10; // 50` `a.speed ??= 25;  // 25`	`let a = 1;` `let b = 0;` `a &&= 2; // 2` `b &&= 2; // 0`	

		equivalent	**not equivalent**
Nullish coalescing operator (??)	**x ?? y**	`(x != null) ? x : y`	
Logical nullish assignment (??=)	**x ??= y**	`x ?? (x = y);`	`x = x ?? y;`
Logical AND assignment (&&=)	**x &&= y**	`x && (x = y);`	`x = x && y;`
Logical OR assignment (||=)	**x ||= y**	`x || (x = y);`	`x = x || y;`

Reflect object		
ES6, no IE11 support - FF 42, Chrome 49, Edge 12, Safari 10, Safari on iOS 10, Samsung Internet 5.0		
Function	**Description**	**equivalent**
`Reflect.apply(target, thisArgument, argumentsList);`	Calls a target function with arguments as specified by the argumentsList parameter.	`Function.prototype.apply();`
`Reflect.construct(target,argumentsList [,newTarget]);`	The new operator as a function.	`new target(...argumentsList);`
`Reflect.defineProperty(target,propertyKey, attributes);`	Similar to Object.defineProperty(). Returns a boolean that is true if the property was successfully defined.	`Object.defineProperty(target, propertyKey,attributes);`
`Reflect.deleteProperty(target,propertyKey);`	The delete operator as a function.	`delete target[propertyKey];`
`Reflect.get(target,propertyKey[,receiver]);`	Returns the value of the property of the object.	`target[propertyKey];`
`Reflect.getOwnPropertyDescriptor(target, propertyKey);`	Returns a property descriptor of the given property if it exists on the object, undefined otherwise.	`Object.getOwnPropertyDescriptor(target, propertyKey);`
`Reflect.getPrototypeOf(target);`	`Object.getPrototypeOf(target);`	`Object.getPrototypeOf(target);`
`Reflect.has(target,propertyKey);`	Returns a boolean whether the target has the property.	`propertyKey in target;`
`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.	`Object.getOwnPropertyNames(target).concat( Object.getOwnPropertySymbols(target));`
`Reflect.preventExtensions(target);`	Prevents new properties from ever being added to an object. Similar to Object.preventExtensions().	`Object.preventExtensions(target);`
`Reflect.set(target,propertyKey,value [,receiver]);`	Assigns values to properties. Returns a boolean that is true if the update was successful.	`target[propertyKey] = value;`
`Reflect.setPrototypeOf(target,prototype);`	Sets the prototype of an object. Returns a boolean that is true if the update was successful.	`Object.setPrototypeOf(target,prototype);`

Map Object

```
var myMap = new Map([iterable]);

// The Map objects are iterable.
for (let [key, value] of myMap) {
  console.log(`${key} = ${value}`);
}

var cloneMap = new Map(myMap);
Map.prototype.size;
Map.prototype.get(<key>); -> value/undefines
Map.prototype.set(<key>,<value>); -> Map object
Map.prototype.has(<key>); -> boolean
Map.prototype.delete(<key>); -> boolean
Map.prototype.clear(); -> undefined

Map.prototype.forEach(function (value,key,map)); -> undefined
Map.prototype.keys(); -> iterator of keys
Map.prototype.values(); -> iterator of values
Map.prototype.entries(); -> iterator of [key, value]
```

Set Object

```
var mySet = new Set([iterable]);

// The Set objects are iterable.
for (const item of mySet) { console.log(item); }

var cloneSet = new Set(mySet);
Set.prototype.size;
Set.prototype.add(<value>); -> Set object
Set.prototype.has(<value>); -> boolean
Set.prototype.delete(<value>); -> boolean
Set.prototype.clear(); -> undefined

Set.prototype.forEach(function (value,value,set)); -> undefined
Set.prototype.keys(); -> iterator of values
Set.prototype.values(); -> iterator of values
Set.prototype.entries(); -> iterator of [value, value]
```

Set Object helper functions

```
function isSuperset(set, subset) {
  for (const elem of subset) {
    if (!set.has(elem)) { return false; }
  }
  return true;
}

function union(setA, setB) {
  const _union = new Set(setA);
  for (const elem of setB) { _union.add(elem); }
  return _union;
}
function intersection(setA, setB) {
  const _intersection = new Set();
  for (const elem of setB) {
    if (setA.has(elem)) { _intersection.add(elem); }
  }
  return _intersection;
}

function difference(setA, setB) {
  const _difference = new Set(setA);
  for (const elem of setB) { _difference.delete(elem); }
  return _difference;
}
function symmetricDifference(setA, setB) {
  const _d = new Set(setA);
  for (const e of setB) {
    if (_d.has(e)) { _d.delete(e); } else { _d.add(e); }
  }
  return _d;
}

const setA = new Set([1,2,3,4]), setB = new Set([2, 3]),
setC = new Set([3, 4, 5, 6])
isSuperset(setA, setB);          // true
union(setA, setC);               // Set {1, 2, 3, 4, 5, 6}
intersection(setA, setC);        // Set {3, 4}
difference(setA, setC);          // Set {1, 2}
symmetricDifference(setA, setC); // Set {1, 2, 5, 6}
```

Array.fromAsync();

```
Array.fromAsync(<object>[,mapFn[,thisArg]])
  .then((resultArray) => /* todo with resultArray */);
```

Object types: async iterable, iterable (Array, Map, Set, NodeList, etc.), array-like

mapfn parameters: element, index

```
async function* asyncIterable () {
  for (let i = 0; i < 5; i++) { await new Promise((resolve)=> setTimeout(resolve,50*i)); yield i; }
}

Array.fromAsync(asyncIterable())
  .then((res) => console.log("asyncIterable1: "+res));
// asyncIterable1: 0,1,2,3,4

Array.fromAsync(asyncIterable(), (x) => x*2)
  .then((res) => console.log("asyncIterable2: "+res));
// asyncIterable2: 0,2,4,6,8

Array.fromAsync([4,5,6,7,8])
  .then((res) => console.log("[4,5,6,7,8]: "+res));
// [4,5,6,7,8]: 4,5,6,7,8

Array.fromAsync([4,5,6,7,8], (x) => x*2)
  .then((res) => console.log("[4,5,6,7,8] + fn: "+res));
// [4,5,6,7,8] + fn: 8,10,12,14,16

Array.fromAsync(new Set([4,5,6,6,10]))
  .then((res) => console.log("Set: "+res));
// Set: 4,5,6,10

Array.fromAsync(new Set([4,5,6,6,10]), (x) => x*2)
  .then((res) => console.log("Set + fn: "+res));
// Set + fn: 8,10,12,20

Array.fromAsync({"0": 3, "1": 4, "2": 5, length: 3})
  .then((res) => console.log("arraylike: "+res));
// arraylike: 3,4,5

Array.fromAsync({"0": 3, "1": 4, "2": 5, length: 3}, (x) => x*2)
  .then((res) => console.log("arraylike + fn: "+res));
// arraylike + fn: 6,8,10
```