**Map in Javascript**

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

A Map has a property that represents the size of the map.

## **How to Create a Map**

You can create a JavaScript Map by:

* Passing an Array to new Map()
* Create a Map and use Map.set()

**Map property**

## **1. Map.size**

The size property returns the number of elements in a Map:

**Example**

fruits.size;

## **Map Methods**

## **1. new Map()**

You can create a Map by passing an Array to the new Map() constructor:

**Example**

// Create a Map  
const fruits = new Map([  
  ["apples", 500],  
  ["bananas", 300],  
  ["oranges", 200]  
]);

## **2. Map.set()**

You can add elements to a Map with the set() method:

**Example**

// Create a Map  
const fruits = new Map();

// Set Map Values  
fruits.set("apples", 500);  
fruits.set("bananas", 300);  
fruits.set("oranges", 200);

## **3. Map.get()**

The get() method gets the value of a key in a Map:

**Example**

fruits.get("apples");    // Returns 500

## **4. Map.delete()**

The delete() method removes a Map element:

**Example**

fruits.delete("apples");

## **5. Map.clear()**

The clear() method removes all the elements from a Map:

**Example**

fruits.clear();

## **6. Map.has()**

The has() method returns true if a key exists in a Map:

**Example**

fruits.has("apples");

## **7. Map.forEach()**

The forEach() method invokes a callback for each key/value pair in a Map:

**Example**

// List all entries  
let text = "";  
fruits.forEach (function(value, key) {  
  text += key + ' = ' + value;  
})

## **8. Map.keys()**

The keys() method returns an iterator object with the keys in a Map:

**Example**

// List all keys  
let veggies = "";  
for (const x of fruits.keys()) {  
  veggies += x;  
}

## **9. Map.values()**

The values method returns an iterator object with the values in a Map:

**Example**

// Sum all values  
let total = 0;  
for (const x of fruits.values()) {  
  total += x;  
}

## **10. Map.entries()**

The entries() method returns an iterator object with the [key,values] in a Map:

**Example**

let text = "";  
for (const x of fruits.entries()) {  
  text += x;  
}

# JavaScript Sets

## A JavaScript Set is a collection of unique values.

## Each value can only occur once in a Set.

## A Set can hold any value of any data type.

## **How to Create a Set**

You can create a JavaScript Set by:

* Passing an Array to new Set()
* Create a new Set and use add() to add values
* Create a new Set and use add() to add variables

## **The new Set() Method**

Pass an Array to the new Set() constructor:

**Example**

// Create a Set  
const letters = new Set(["a","b","c"]);

## **Set Methods**

## **1. add() method**

**Example**

letters.add("d");  
letters.add("e");

letters.add("c");

## **2. forEach() Method**

The forEach() method invokes a function for each Set element:

**Example**

const letters = new Set(["a","b","c"]);  
let text = "";  
letters.forEach (function(value) {  
  text += value;   
})

## **3. values() Method**

The values() method returns an Iterator object containing all the values in a Set:

Use the Iterator object to access the elements:

**Example**

// Create an Iterator  
const myIterator = letters.values();  
  
// List all Values  
let text = "";  
for (const entry of myIterator) {  
  text += entry;  
}

**4. keys() Method**

A Set has no keys.

keys() returns the same as values().

This makes Sets compatible with Maps.

**Example**

letters.keys()   // Returns [object Set Iterator]

## **5. entries() Method**

A Set has no keys.

entries() returns [value,value] pairs instead of [key,value] pairs.

**Example**

// Create an Iterator  
const myIterator = letters.entries();  
  
// List all Entries  
let text = "";  
for (const entry of myIterator) {  
  text += entry;  
}

# WeakMap Object

The JavaScript WeakMap object is a type of collection which is almost similar to Map. It stores each element as a key-value pair where keys are weakly referenced. Here, the keys are objects and the values are arbitrary values.

## **Syntax**

new WeakMap([iterable])

## **WeakMap Methods**

# 1. delete() method

The JavaScript WeakMap **delete()** method is used to remove the specified element from a **WeakMap** object.

## **Syntax**

WeakMapObj.delete()

**Example**

var wm = new WeakMap();

var obj1 = {};  var obj2 = {};  var obj3= {};

wm.set(obj1, 'jQuery');

wm.set(obj2, 'AngularJS');

wm.set(obj3,'Bootstrap');

wm.delete(obj1);

# 2. get() method

The JavaScript WeakMap **get()** method returns the value of specified key of an element from a **WeakMap** object.

## **Syntax**

WeakMapObj.get(key)

**Example**

var wm = new WeakMap();

var obj1 = {};   var obj2 = {};   var obj3= {};

wm.set(obj1, "jQuery");

wm.set(obj2, "AngularJS");

wm.set(obj3,"Bootstrap");

console.log(wm.get(obj1));

console.log(wm.get(obj2));

console.log (wm.get(obj3));

# 3. has() method

The JavaScript WeakMap **has()** method indicates whether the **WeakMap** object contains the specified key. It returns true if the specified key is present, otherwise false.

## **Syntax**

WeakMapObj.has(key)

**Example**

var wm = new WeakMap();

var obj1 = {};  var obj2 = {};  var obj3= {};

wm.set(obj1, "jQuery");

wm.set(obj2, "AngularJS");

wm.set(obj3,"Bootstrap");

console.log(wm.has(obj2));

# 4. set() method

The JavaScript WeakMap **set()** method is used to add or update an element to WeakMap object with the particular key-value pair. Each value must have a unique key.

## **Syntax**

weakMapObj.set(key,value)

**Example**

var wm = new WeakWeakMap();

var obj1 = {};  var obj2 = {};  var obj3= {};

wm.set(obj1, "jQuery");

wm.set(obj2, "AngularJS");

wm.set(obj3,"Bootstrap");

# WeakSet Object

The JavaScript WeakSet object is the type of collection that allows us to store weakly held objects. Unlike Set, the WeakSet are the collections of objects only. It doesn't contain the arbitrary values.

## **Syntax**

new WeakSet([iterable])

## **WeakSet Methods**

# 1. add() method

The JavaScript Set **add()**method is used to add a new object to the end of a **WeakSet** object.

## **Syntax**

The add() method is represented by the following syntax:

weakSetObj.add(value)

**Example**

var ws = new WeakSet();

var obj1={};

var obj2={};

ws.add(obj1);

ws.add(obj2);

# 2. has() method

The JavaScript WeakSet **has()** method indicates whether the **WeakSet** object contains the specified object. It returns true if the specified object is present, otherwise false.

## **Syntax**

The has() method is represented by the following syntax:

weakSetObj.has(value)

**Example**

var ws = new WeakSet();

var obj1={};

var obj2={};

console.log(ws.has(obj1));

console.log(ws.has(obj2));

# 3. delete() method

The JavaScript WeakSet **delete()** method is used to remove the specified object from **WeakSet** object.

## **Syntax**

The delete() method is represented by the following syntax:

WeakSetObj.delete()

**Example**

var ws = new WeakSet();

var obj={};

ws.add(obj);

document.writeln("Element present before invoking delete() method: "+ws.has(obj)+"<br>");

ws.delete(obj);