Lesson:

Set and Map

PW SKILLS

Topics

- **Introduction to Set in JavaScript**
- **Set Properties with examples**
- **Set methods with examples**
- **Create set from an array and array from Set**
- **Introduction to Map in JavaScript**
- **Map Properties with examples**
- **Map methods with examples**
- **Create a map from the object and object from Map.**
- **Interview Point**

Introduction to Set in JavaScript

In Javascript, a Set is an unordered collection of unique values. It is a collection of objects that cannot contain duplicate values. The set object lets us store unique values of any type, whether primitive values or object references.

It can be created using the new Set() constructor and we can use the add method to add elements.

Here is an example of how to create and add new value

```
// new set() constructor.
const setDemo = new Set();
// add method.
setDemo.add(1);
setDemo.add(2);
console.log(setDemo);
// output - Set(2) { 1, 2 }
```

Some of the uses of sets in Javascript include -

- Unique values - Sets can only contain unique values, so they can be used to store a collection of data without worrying about duplicates.
- Fast lookups - Sets can be searched very quickly, so they're a good choice for finding a particular value in a large data collection.
- Intersection and unions - Sets can be used to find the intersection or union of two arrays, which can be useful for things like finding all of the users who have liked both of two different posts on a social media site.
- Deduplicating Data - Sets can be used to deduplicate data, which is the process of removing duplicate values from a collection.
- Set operations - Sets support various set operations, such as union, intersection, difference, and symmetric difference.

Set Properties with examples

Set Property in JavaScript includes size, which returns the number of unique elements in the Set

Example -

```
// syntax - set.size
const setDemo = new Set();
console.log(setDemo.size); // output - 0
setDemo.add(3);
setDemo.add(4);
setDemo.add(5);
console.log(setDemo.size); // output - 3
setDemo.add(5); // duplicate object will be added only
once
setDemo.add(3); //duplicate object will be added only
once

console.log(setDemo); // Set(3) { 3, 4, 5 }
console.log(setDemo.size); // same output size i.e 3
```

Set methods with examples

In JavaScript, Set methods can be used to perform a variety of operations on sets, such as adding, removing elements, iterating through the set, checking if a value is in the set, and much more.

Some of the Essential or commonly used Set methods are as follows -

- **add()** - Add the specified value to the set.

Example -

```
// set add method
const setDemo = new Set();
setDemo.add(4);
setDemo.add(5);
setDemo.add({ user: "john@gmail.com" });
console.log(setDemo);
// output - Set(3) { 4, 5, { user: 'john@gmail.com' } }
```

- **clear()** - Removes all the elements from the set.

Example -

```
// set clear method
const setDemo = new Set();
setDemo.add(4);
setDemo.add(5);
console.log(setDemo); // Set(2) { 4, 5 }
setDemo.clear();
console.log(setDemo); // Set(0) {}
```

- **delete()** - Removes the specified value from the set.

Example -

```
const setDemo = new Set();
setDemo.add(4);
setDemo.add(5);
console.log(setDemo); // Set(2) { 4, 5 }
setDemo.delete(5); // remove value 5 element
console.log(setDemo); // Set(1) { 4 }
```

- **entries()** - Returns an iterable of all the key/value pairs in the set that contains an array of [value, value] for each element in the set object. In insertion order. For a set object there is no key like map objects. However, to keep API similar to the Map object, each entry has the same value for its key and value here, so that an array [value, value] is returned.

Example -

```
// // entries
const data = new Set();
data.add("mangesh");
data.add({
  like: "movies",
});
console.log(data);
/*
Output - Set(2) { 'mangesh', { like: 'movies' } }
*/
console.log(data.entries());
/**
 *output -
 [Set Entries] {
  [ 'mangesh', 'mangesh' ],
  [ { like: 'movies' }, { like: 'movies' } ]
}
 */
```

- **forEach()** - it executes a provided function for each value in the set object, in insertion order.
- Callbackfunc - it represents the function to be executed
- value1 - it represents the key element.
- value2 - it represents the value.

Example -

```
// // forEach method in Set
//forEach syntax - Set.forEach(callbackfunc)
const data = new Set();
data.add(4);
data.add(5);
data.add(6);
console.log(data); // Set(3) { 4, 5, 6 }
function multiply(value1, value2) {
  console.log(`data[${value1}] : ${value2 * 2}`);
}
data.forEach(multiply);
/**
 * output ---
 Set(3) { 4, 5, 6 }
data[4] : 8
data[5] : 10
data[6] : 12
 */
```

- **has()** - Returns true if the set contains the specified value, false otherwise.

Example -

```
// // has method
const data = new Set([1, 4, 2, 8, 6]);
console.log(data); // Set(5) { 1, 4, 2, 8, 6 }

console.log(data.has(2)); // true
console.log(data.has(8)); // true
console.log(data.has(5)); // false
```

- **values()** - Returns an iterable of all the values in the set.

Example -

```
//// keys method
const data = new Set([1, 4, 2, 8, 6]);
console.log(data); // Set(5) { 1, 4, 2, 8, 6 }
const value = data.values();
console.log(value); // [Set Iterator] { 1, 4, 2, 8, 6 }
```

- **keys()** - Returns an iterable of all the keys in the set. The keys() method is exactly the same as the values() method

Example -

```
//// keys method
const data = new Set([1, 4, 2, 8, 6]);
console.log(data); // Set(5) { 1, 4, 2, 8, 6 }

const key = data.keys();
console.log(key); // [Set Iterator] { 1, 4, 2, 8, 6 }
```

Create a Set from an array and an Array from a Set

Creating Set from an array -

To create a set from an array in javascript, a Set constructor can be used and pass the array as an argument.

Example -

```
// create set from an array
const arr = ["apple", "Mango", "Banana", "orange"];
const setDemo = new Set(arr);
console.log(setDemo);
//output - Set(4){ 'apple', 'Mango', 'Banana', 'orange'}

// remove duplicate element from an array
const numb = [1, 3, 4, 5, 2, 6, 3, 3];
const uniqNum = new Set(numb);
console.log(uniqNum);
// output - Set(6) { 1, 3, 4, 5, 2, 6 }
```

Creating an Array from set -

To create an array from set in javascript, array.from() method can be used and pass the array as an argument.

Example -

```
// create array from set
const setData = new Set();
setData.add(1);
setData.add(2);
setData.add(3);
console.log(setData); // output - Set(3) { 1, 2, 3 }
const arrSet = Array.from(setData);
console.log(arrSet); // output [ 1, 2, 3 ]
```

Introduction to Map in JavaScript

A Map in JavaScript is a collection of key-value pairs. It is similar to an object, but it has some key differences and it remembers the original insertion order of the keys. Any value non-primitive or primitive value can be used as either a key or a value.

It can be created using the new Map() constructor and we can use the set method to add elements.

Here is an example of how to create and add new value

```
/***
****** map in javascript ******
*/
const mapDemo = new Map();
console.log(mapDemo); // Map(0) {}

mapDemo.set("key1", "value1");
console.log(mapDemo); // Map(1) { 'key1' ⇒ 'value1' }
```

Some of the importance of using Map in JavaScript are as follows -

- Storing data that is frequently accessed - Maps are a good choice for storing data that is frequently accessed, such as user preferences or search results
- Implementing a hash table - A hash table is a data structure that uses a hash function to map keys to values. Maps can be used to implement hash tables, which are often used in algorithms such as searching and sorting.
- Creating a dictionary - A dictionary is a collection of words and their definitions. Maps can be used to create dictionaries, which can be used to look up the definition of a word quickly.
- Creating a lookup table - A lookup table is a table that maps keys to values. Maps can be used to create lookup tables, which can be used to quickly find the value associated with a key.

Map Properties with example

Map Property in JavaScript includes size, which returns the number of elements in the Map

Example -

```
// Math size
const days = new Map();
days.set("mon", "monday");
days.set("tue", "Tuesday");
days.set("wed", "Wednesday");

console.log(days);
/**
 output - Map(3) { 'mon' ⇒ 'monday', 'tue' ⇒
'Tuesday', 'wed' ⇒ 'Wednesday' }
*/
console.log(days.size); // 3
```

Map methods with examples

In JavaScript, Mapmethods can be used to perform a variety of operations on Map, such as adding, removing elements, iterating through the Map, checking if a value is in the Map and much more.

Some of the Essential or commonly used Map methods are as follows -

- set - adds a new key-value pair to the Map

Example -

```
// // add keys and values
const days = new Map();
days.set("mon", "monday");
console.log(days);
// output - Map(1) { 'mon' ⇒ 'monday' }
```

- clear - removes all key-value pairs from the Map.

Example -

```
// // clear method
const days = new Map();
days.set("mon", "monday");
days.set("tue", "tuesday");
days.set("wed", "wednesday");
console.log(days); // Map(3) { 'mon' ⇒ 'monday', 'tue'
// ⇒ 'tuesday', 'wed' ⇒ 'wednesday' }
days.clear();
console.log(days); // Map(0) {}
```

- delete - removes a key-value pair from the Map.

Example -

```
// // delete method
const days = new Map();
days.set("mon", "monday");
days.set("tue", "tuesday");
days.set("wed", "wednesday");
console.log(days); // Map(3) { 'mon' ⇒ 'monday', 'tue'
⇒ 'tuesday', 'wed' ⇒ 'wednesday' }
days.delete("wed");
console.log(days); Map(2) { 'mon' ⇒ 'monday', 'tue' ⇒
// 'tuesday' }
```

- entries - returns an iterator object that iterates over the key-value pairs in the Map.

Example -

```
// entries method
const days = new Map();
days.set("mon", "monday");
days.set("tue", "tuesday");
days.set("wed", "wednesday");
```

```
console.log(days.entries());
/**
 output -
 [Map Entries] {
  [ 'mon', 'monday' ],
  [ 'tue', 'tuesday' ],
  [ 'wed', 'wednesday' ]
}
 */
```

- forEach - executes a callback function for each key-value pair in the Map

Example -

```
// // forEach -
const days = new Map();
days.set("mon", "monday");
days.set("tue", "tuesday");
days.set("wed", "wednesday");

days.forEach(function (value, key) {
  console.log(`${key} = ${value}`);
});
/**
 output -
 mon = monday
tue = tuesday
wed = wednesday
 */
```

- get - returns the value associated with a key.

Example -

```
// // get method
const days = new Map();
days.set("mon", "monday");
days.set("tue", "tuesday");
days.set("wed", "wednesday");
console.log(days.get("mon")); // output - monday
```

- has - returns a boolean indicating whether an element with the specified key exist or not.

Example -

```
// has method
const days = new Map();
days.set("mon", "monday");
days.set("tue", "tuesday");
days.set("wed", "wednesday");
console.log(days.has("tue")); // output - true
```

- key - returns an iterator object that iterates over the keys in the Map.

Example -

```
// keys method
const days = new Map();
days.set("mon", "monday");
days.set("tue", "tuesday");
days.set("wed", "wednesday");

console.log(days.keys());
// output - [Map Iterator] { 'mon', 'tue', 'wed' }
```

- values - returns an iterator object that iterates over the values in the Map.

Example -

```
// values
const days = new Map();
days.set("mon", "monday");
days.set("tue", "tuesday");
days.set("wed", "wednesday");

console.log(days.values());
// output - [Map Iterator] { 'monday', 'tuesday',
'wednesday' }
```

Create a map from the object and an object from Map.

Creating Map from an object-

To create a map from an object in javascript, Map constructor can be used and pass the object which is converted into an iterable array as an argument.

Example -

```
// map constructor --
const user = {
  name: "mangesh",
  email: "mangesh@gmail.com",
};
const userFinal = new Map(Object.entries(user));
console.log(userFinal);
//output - Map(2) { 'name' ⇒ 'mangesh', 'email' ⇒
//'mangesh@gmail.com' }


// looping of object and adding to map
const newData = new Map();
console.log(newData); // Map(0){}

for (let key in user) {
  if (user.hasOwnProperty(key)) {
    newData.set(key, user[key]);
  }
}
console.log(newData);
/*
Map(2) { 'name' ⇒ 'mangesh', 'email' ⇒
'mangesh@gmail.com' }
/*
```

Creating an object from map -

To create an object from a map in javascript, Object.fromEntries() method or loops can be used.

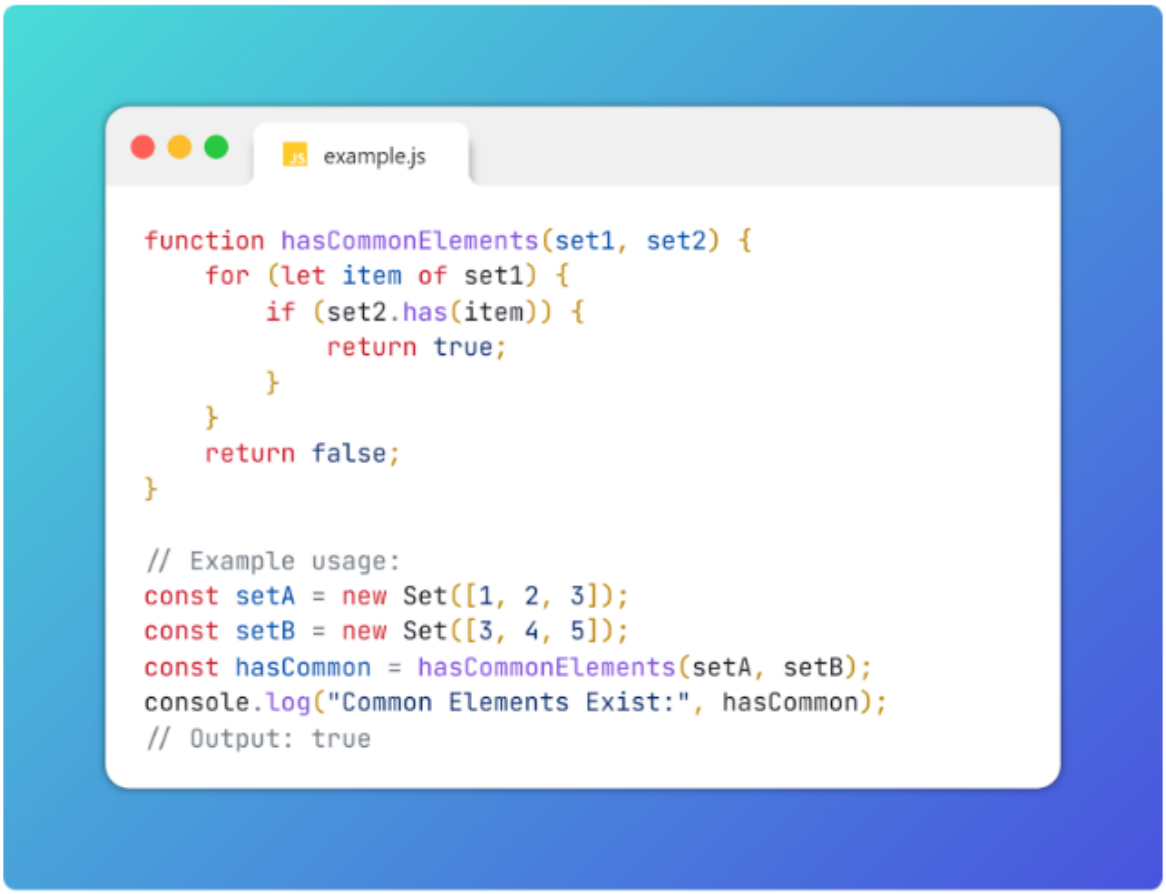
Example -

```
// convert map to object using Object.fromEntries //
method
const map = new Map([
  ["fruit", "apple"],
  ["vegetables", "cabbage"],
]);
console.log(map);
/* output - Map(2) { 'fruit' ⇒ 'apple', 'vegetables' ⇒
'cabbage' } */

const obj = Object.fromEntries(map);
console.log(obj);
// output - { fruit: 'apple', vegetables: 'cabbage' }

// looping
const obj1 = {};
map.forEach((value, key) ⇒ {
  obj1[key] = value;
});
console.log(obj1);
// output - { fruit: "apple", vegetables: "cabbage" };
```

Interview Points

Q1.Write a function that takes two Sets as input and returns true if they share at least one common element, otherwise “false”.

Answer -



Q2.Write a function that takes an array of objects and a property name, and returns a Map where the specified property of each object is the key.

Answer -

example.js

```
function arrayToMap(arr, key) {
    const resultMap = new Map();
    arr.forEach(obj => resultMap.set(obj[key], obj));
    return resultMap;
}

// Example usage:
const objects = [
    { id: 1, name: "Alice" },
    { id: 2, name: "Bob" },
    { id: 3, name: "Charlie" }
];
const mapByKey = arrayToMap(objects, 'id');
console.log("Map by ID:", mapByKey);
/*
Map by ID: Map(3) {
  1 => { id: 1, name: 'Alice' },
  2 => { id: 2, name: 'Bob' },
  3 => { id: 3, name: 'Charlie' }
}
*/
```