**Navachethan – 1NT18IS099**

**Web Assignment (Map, set and promise)**

Map:

Navachethan\_map.js:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script src="navachethan\_map.js"></script>

</body>

</html>

Navachethan\_map.js:

let names = [{

    id: 1,

    name: 'Navachethan'

    },

    {

    id: 2,

    name: 'map\_method'

    },

    {

    id: 3,

    name: 'dark\_knight'

    },

    {

    id: 4,

    name: 'Slovakia'

    },

    {

    id: 5,

    name: 'cloud\_computing'

    }

    ]

    let userNames = names.map(function(names) {

    console.log(names.name)

    });

    let mapNms = [];

    names.forEach(function (names) {

    mapNms.push(names.name);

    });

    console.log(mapNms);

    const user = new Map();

    user.set('id', 1);

    user.set('name', 'Navachethan');

    console.log(user.get('id'));

    let iterator = user.entries();

    console.log(iterator.next().value);

    console.log(iterator.next().value);

    //Example of iteration over a map collection

    let userIDs = new Map();

    let user2 = {name: 'Jack'}, user3 = {name: 'Ron'}, user4 = {name:

    'Emma'};

    userIDs.set(user2, 1) .set(user3, 2) .set(user4, 3);

    // Method 1

    for (let [name, id] of userIDs) {

    console.log(name);

    console.log(id);

    }

    // Method 2

    userIDs.forEach((id, name) => {

    console.log(name);

    console.log(id);

    });

    //To convert an object to a map, we can use the Object.entries()

    //method.

    const obj = {

    'name': 'Aaron',

    'id': 1,

    }

    const map = new Map(Object.entries(obj));

    console.log(map.get('name'))

    //Compare an object to a map and see how they do in terms of

    performance

    let obj2 = {}, map2 = new Map();

    let users4 = ["Apple", "Mango", "Banana", "Grapes", "Carrot", "Turnip",

    "Apple", "Banana", "Turnip", "Mango"];

    for (let i = 0; i < users4.length; i++) {

    obj2[i] = users4[i];

    map2.set(i, users4[i]);

    }

    let result;

    console.time('Object');

    result = obj2.hasOwnProperty("Mango");

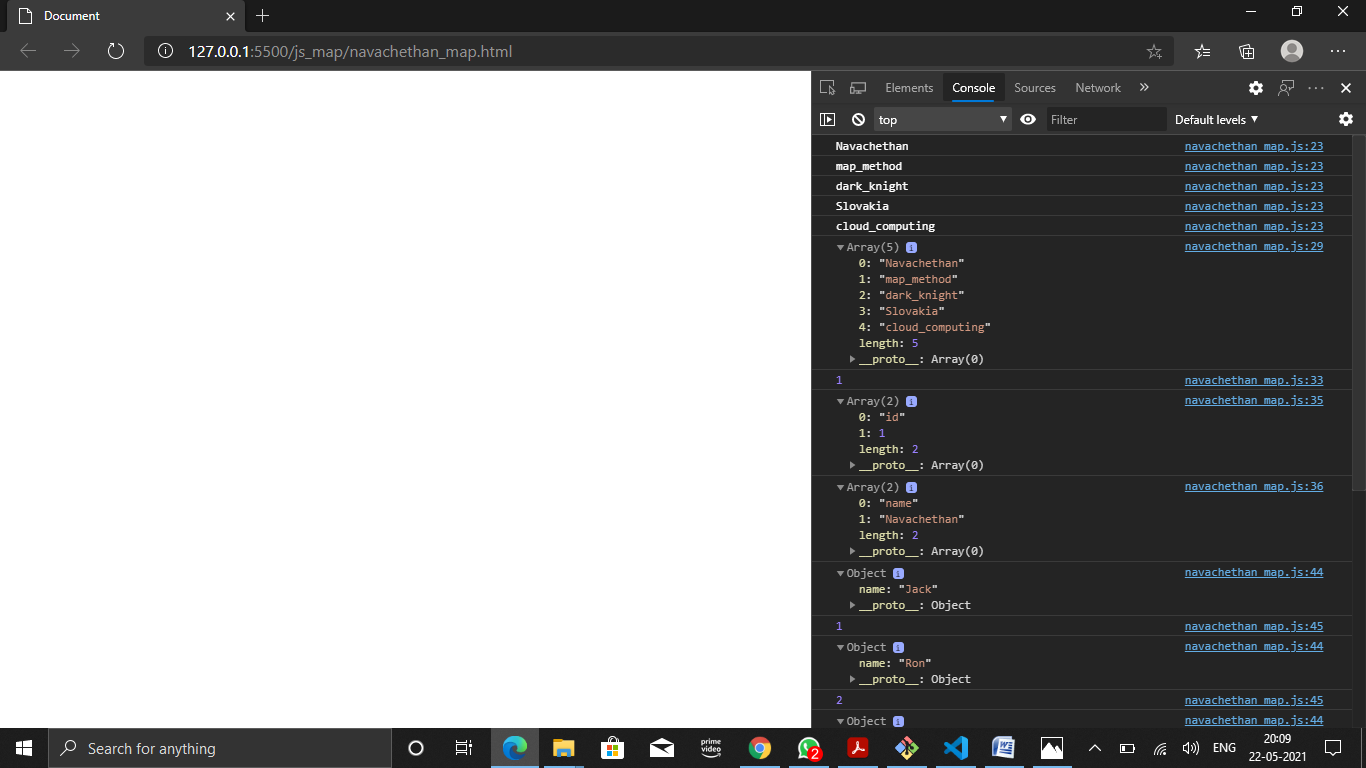
    console.timeEnd('Object');

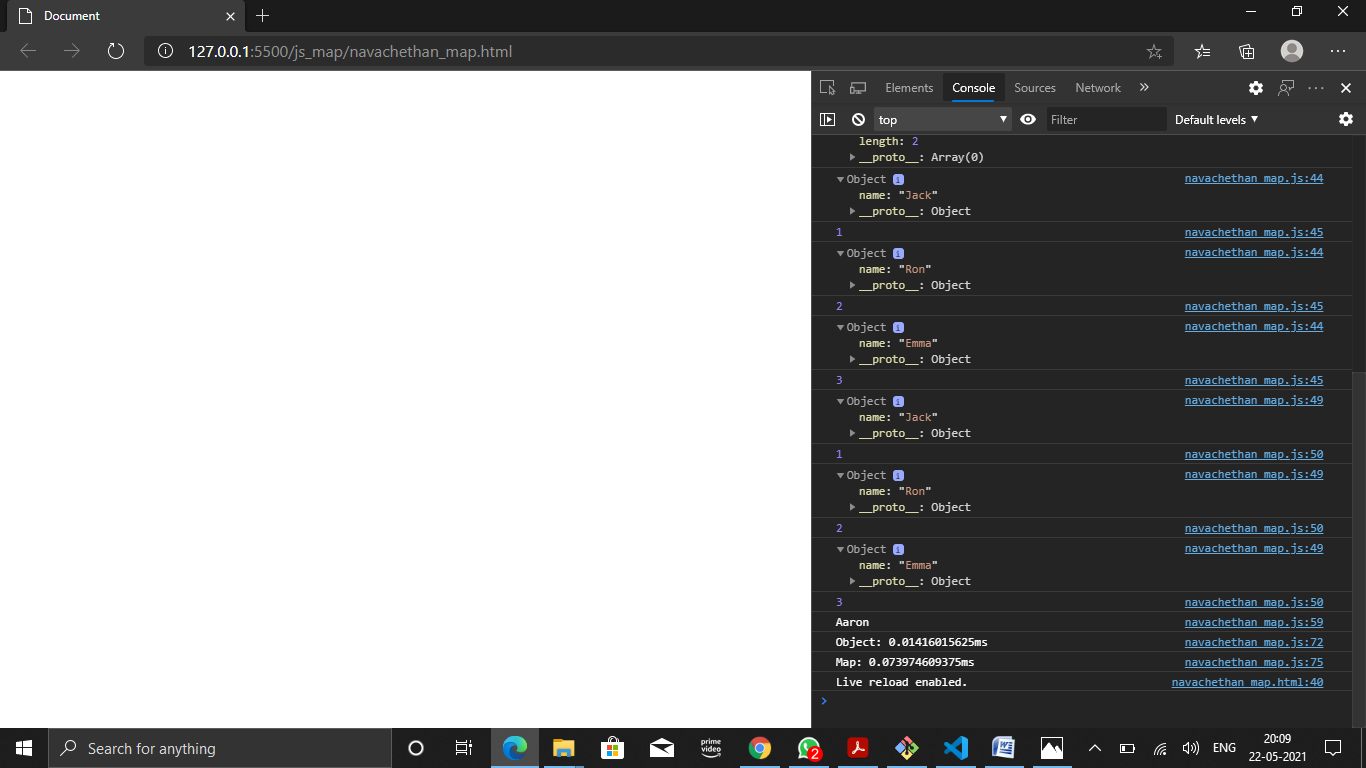
    console.time('Map');

    result = map2.has("Mango");

    console.timeEnd('Map');

output:





Js\_map.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script src="js\_map.js"></script>

</body>

</html>

Js\_map.js:

let map1 = Object.prototype.toString.call(new Map());

console.log(map1);

let map2 = new Map();

map2.set(1,'one');  // set() method

map2.set(2,'two');

console.log(map2.size);

const iterator1 = map2[Symbol.iterator]();

for (const item of iterator1) {

  console.log(item);

}

console.log(map2.size);

map2.clear();          // clear(); to completely delete the data

console.log(map2.size); // size

map2.set(1,'three');

map2.set(2,'four');

const iterator2 = map2[Symbol.iterator]();

for (const item of iterator2) {

  console.log(item);

}

console.log(map2.delete(1));   // delete(); method

console.log(map2.has(1));

console.log(map2.size);

map2.set(3,'five');

let ittr = map2.entries();  // entries(); for itteration

console.log(ittr.next().value);

console.log(ittr.next().value);

let ittr1 = map2.keys();  // keys(); for itteration of only keys

console.log(ittr1.next().value);

console.log(ittr1.next().value);

let ittr2 = map2.values();  // entries(); for itteration of values only

console.log(ittr2.next().value);

console.log(ittr2.next().value);

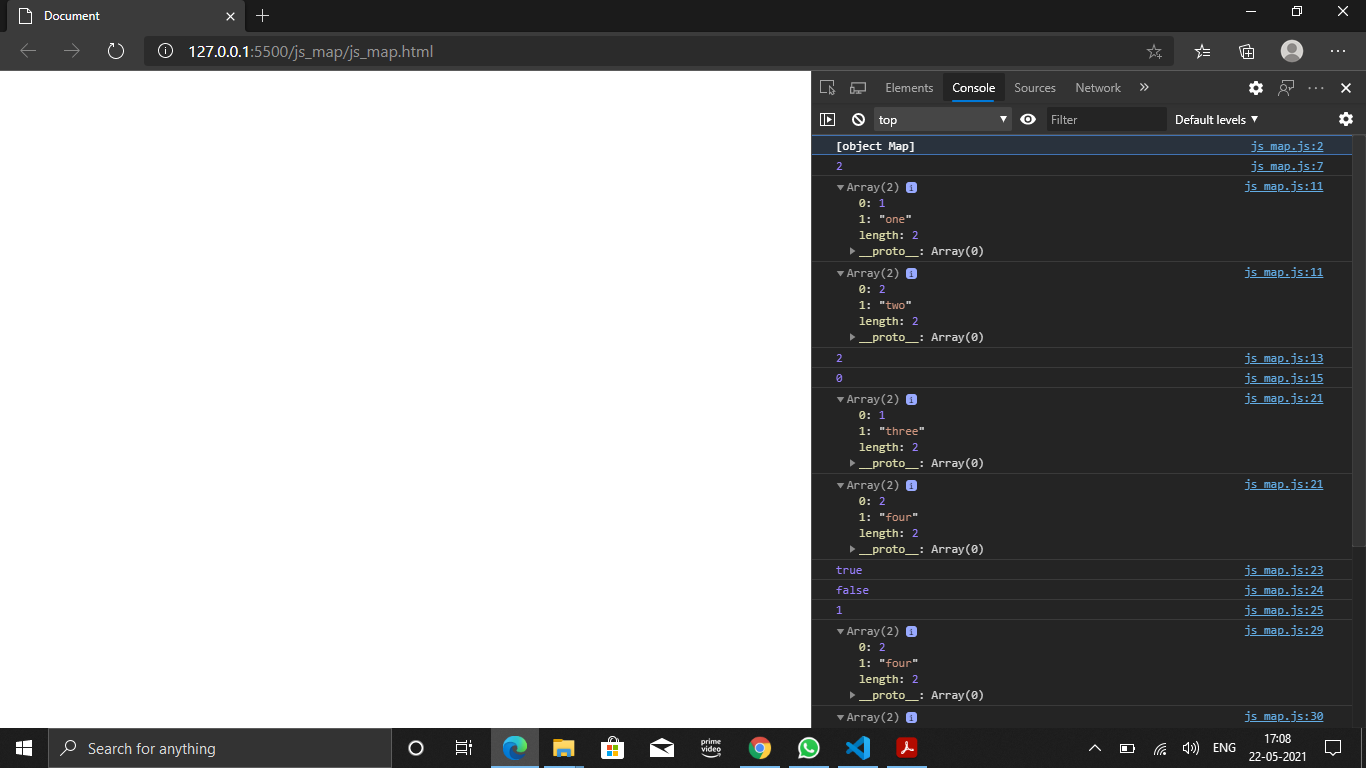
function logMapElements(value, key, map) {

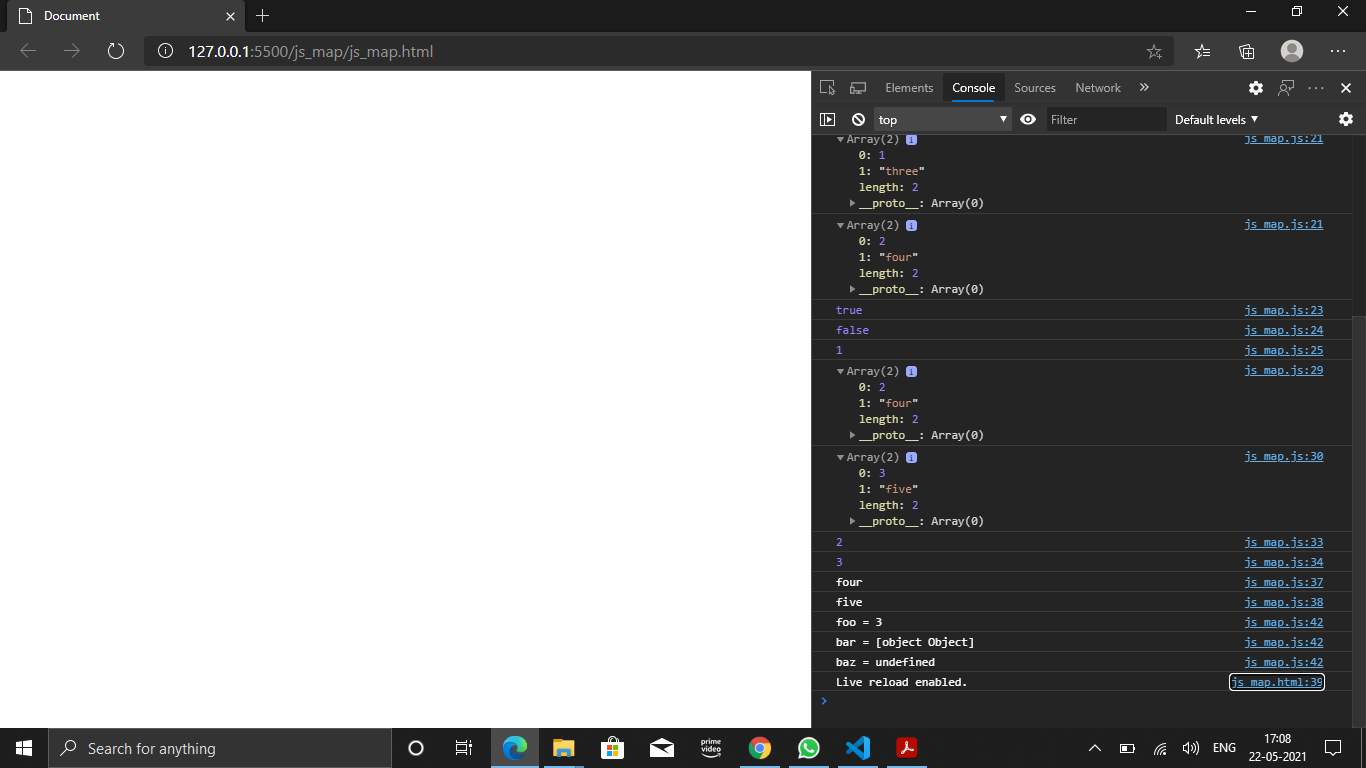
    console.log(`${key} = ${value}`);

  }

  new Map([['foo', 3], ['bar', {}], ['baz', undefined]])

    .forEach(logMapElements);

Output:



**Set:**

Navachethan\_set.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script src="navachethan\_set.js"></script>

</body>

</html>

Navachethan\_set.js:

let glist = new Set();

let g1 = { type: "sugar", day: 1 };

let g2 = { type: "corn\_flour", day: 0.5};

let g3 = { type: "chillies", day: 0.25};

// Let's add each event to the set

glist.add(g1);

glist.add(g2);

glist.add(g3);

for (let item of glist.values()) {

console.log(item);

}

console.log(glist.has(g2));

glist.delete(g3);

for (let value of glist) {

console.log(value);

}

console.log(glist.size);

glist.clear();

console.log(glist.has(g1));

/\*

console.log(events.size);

events.clear();

console.log(events.has(event4));\*/

let cities = new Set();

cities.add("Banglore");

cities.add("Mumbai");

cities.add("chennai");

cities.add("Hyderabad");

cities.forEach((cities) => {

console.log(cities);

});

//Converting Set into an arrays

let set = new Set([19, "string", {"objKey": "objVal"}]);

set.add(true);

let arr = [...set]; // destructuring

console.log(arr);

//Converting Arrays into an sets

let arr2 = [15, 15, "string", {"objKey": "objVal"}];

let arr2converted = [...new Set(arr2)];

console.log(arr2converted);

//we have duplicate items in an array and we want to filter them out.

//We can do this in two ways

let users = ["M", "K", "M", "R", "K"];

/\*function unique(users) {

return Array.from(new Set(users));

}

console.log(unique(users));\*/

let set3 = new Set(users);

let arrFromSet = [...set3];

console.log(arrFromSet);

/\*let’s say we want to add all the users from above to a new set and a

new array.

Let’s see which collection performs the operation faster\*/

let arr1 = [], set4 = new Set();

let users2 = ["English", "German", "Latin", "French", "English",

"German", "Latin",  "French"];

for (let i = 0; i < users2.length; i++) {

arr1.push(users2[i]);

set4.add(users2[i]);

}

let result;

console.time('Array');

result = arr1.indexOf("German") !== -1;

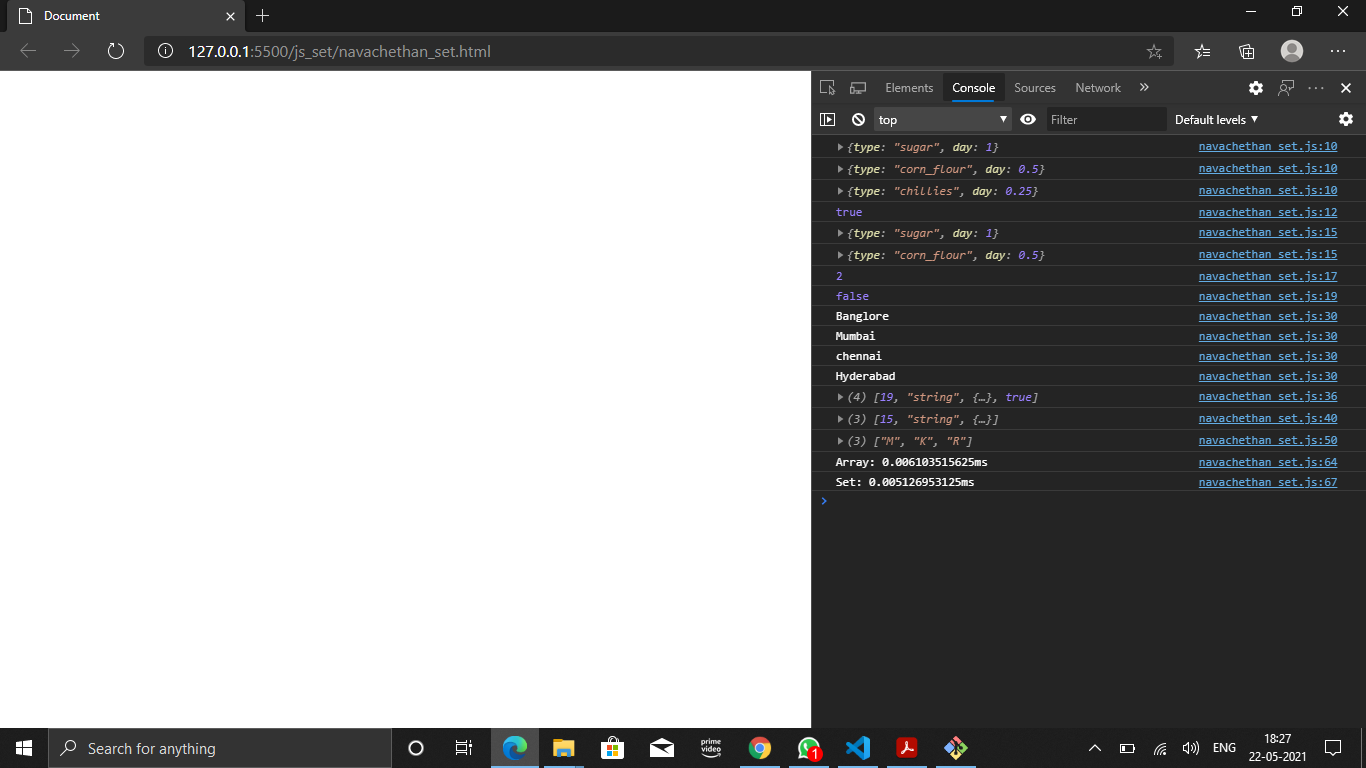
console.timeEnd('Array');

console.time('Set');

result = set4.has("German");

console.timeEnd('Set');

output:



Js\_set.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script src="js\_set.js"></script>

</body>

</html>

Js\_set.js:

const set1 = new Set();

set1.add(65);   // add(); method

set1.add('sixty five');

console.log(set1.size);  // size method;

set1.clear();   // clear(); method

console.log(set1.size);

set1.add({x:10}).add({x:20}).add({x:30}).add({x:5});

console.log(set1.size);

set1.forEach((point) => {   // forEach(); for iteration

    if (point.x > 10) {

      set1.delete(point);  // delete(); method

    }

  });

  console.log(set1.size);

const iterator1 = set1.entries();  // entries(); to print the data- "Note:observe data duplication"

for (const entry of iterator1) {

  console.log(entry);

}

console.log(set1.has(1));  // has(); method to check the element existence.

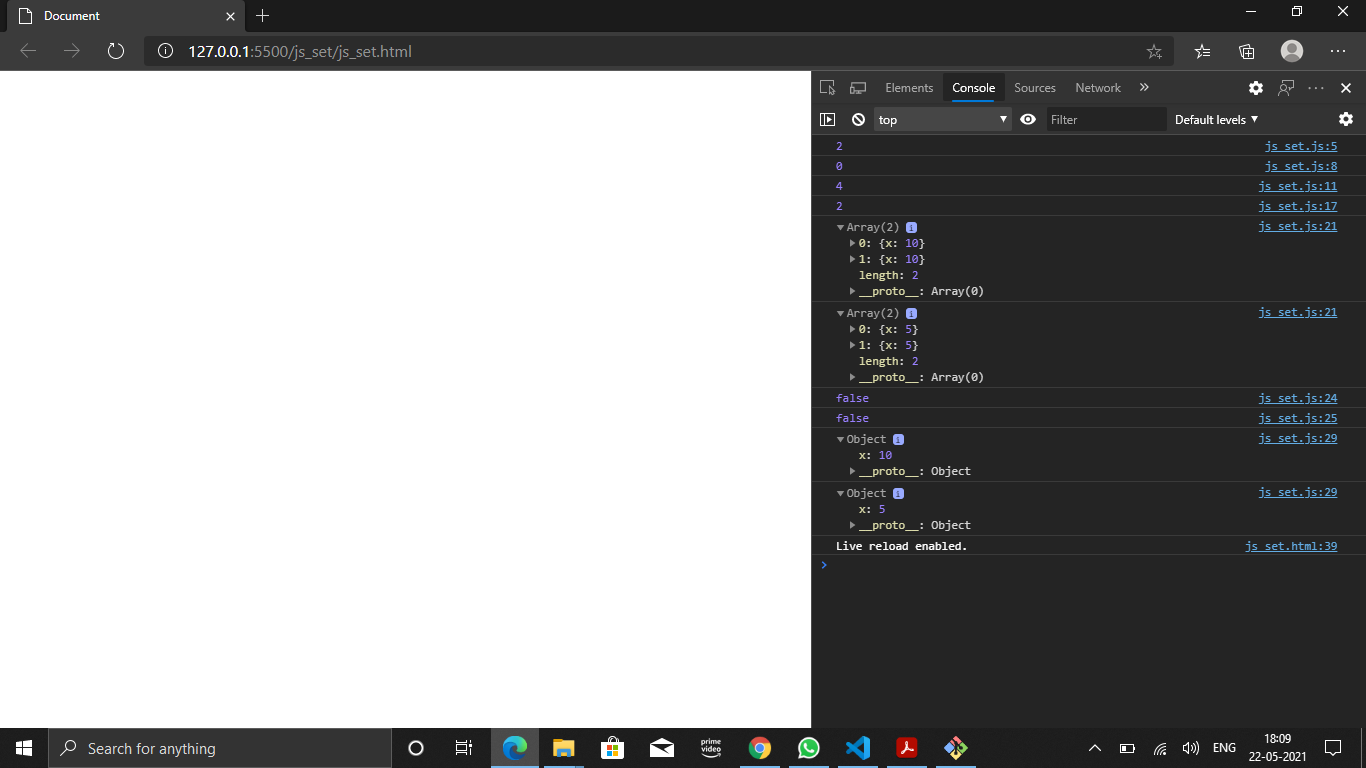
console.log(set1.has(5));

const iterator2 = set1.values();  // values(); to print the data- "Note:no data duplication"

for (const entry1 of iterator2) {

  console.log(entry1);

}

Output:

Navachethan\_promise.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <script src="navachethan\_promise.js"></script>

</body>

</html>

Navachethan\_promise.js:

/\*let count = true;

let countValue = new Promise(function (resolve, reject) {

if (count) {

resolve("promise resolved");

} else {

reject("promise rejected");

}

});

console.log(countValue); \*/

// returns a promise

/\*let countValue = new Promise(function (resolve, reject) {

resolve("Promise has been resolved");

});

// executes when promise is resolved successfully

countValue

.then(function successValue(result) {

console.log(result);

})

.then(function successValue1() {

console.log("Multiple functions can be called.");

}); \*/

// returns a promise

/\*let countValue = new Promise(function (resolve, reject) {

resolve('Promise accepted');

});

// executes when promise is resolved successfully

countValue.then(

function successValue(result) {

console.log(result);

},

)

// executes if there is an error

.catch(

function errorValue(result) {

console.log(result);

}

); \*/

// returns a promise

/\*let countValue = new Promise(function (resolve, reject) {

    // could be resolved or rejected

    resolve('Promise resolved');

    });

    // add other blocks of code

    countValue.finally(

    function greet() {

    console.log('This code is executed.');

    }

    );\*/

    //NC

    // multiple promises in a array

/\*const promise1 = new Promise((resolve, reject) => {

    resolve("promise resolved successfully.");

  });

const promise2 = new Promise((resolve, reject) => {

  resolve("promise accepted.");

});

Promise.all([promise1, promise2]).then((values) => {

  console.log(values);

}); \*/

/\*const promise1 = new Promise((resolve, reject) => {

  setTimeout(resolve, 500, 'promise accepted');

});

const promise2 = new Promise((resolve, reject) => {

  setTimeout(resolve, 100, 'resolved successfully');

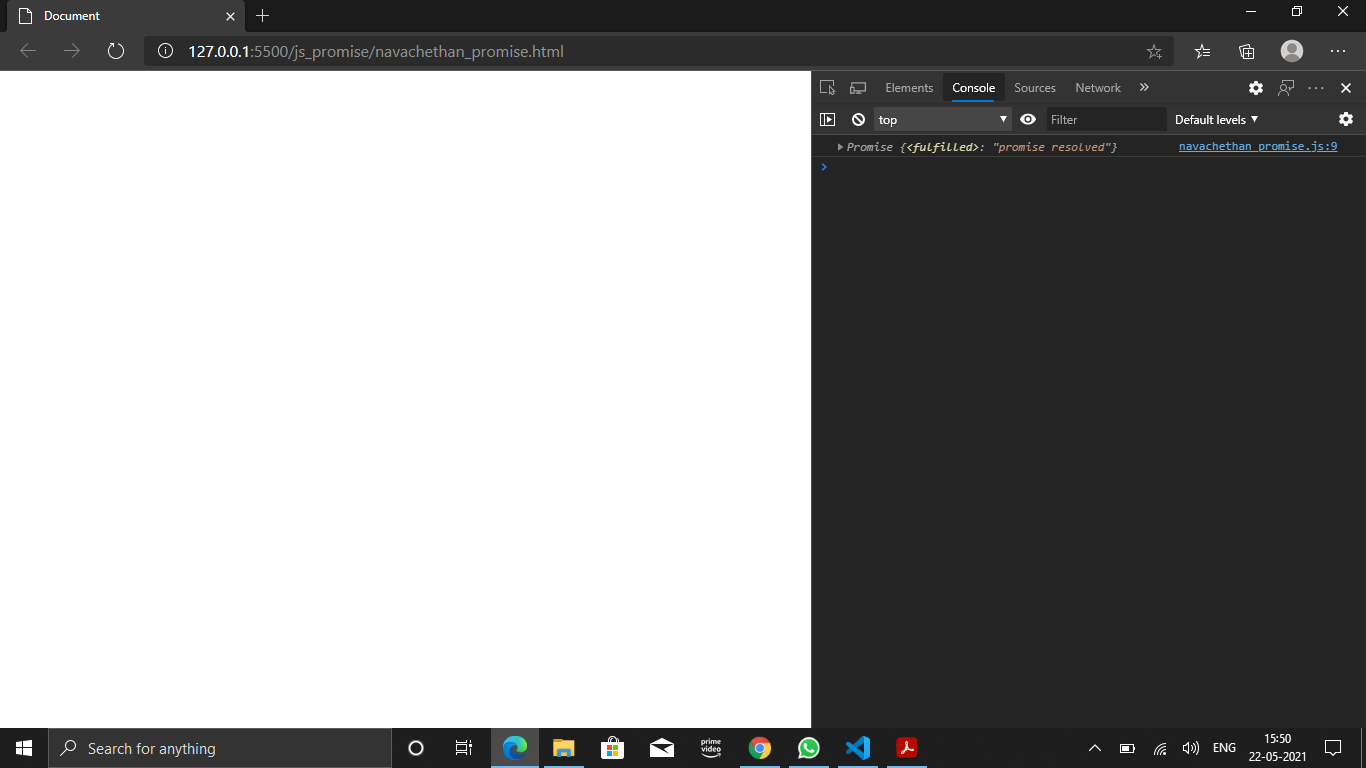
});

Promise.race([promise1, promise2]).then((value) => {

  console.log(value);

});

\*/

Output:

