11th Sep 2019 | Wednesday

Javascript Scope Chaining-

|  |
| --- |
| if (true){  let fromif=1;  //Values of variable is 1.  {  //Since we re-declare variable from '{' till the declaration line the  //variable is undefine or not accessible before initialization.  console.log(fromif);  let fromif=2;    //after redeclaration value of varible till the '}' is 2.  }  //Values of variable is 1.    }  //This is known as scope chaining. |

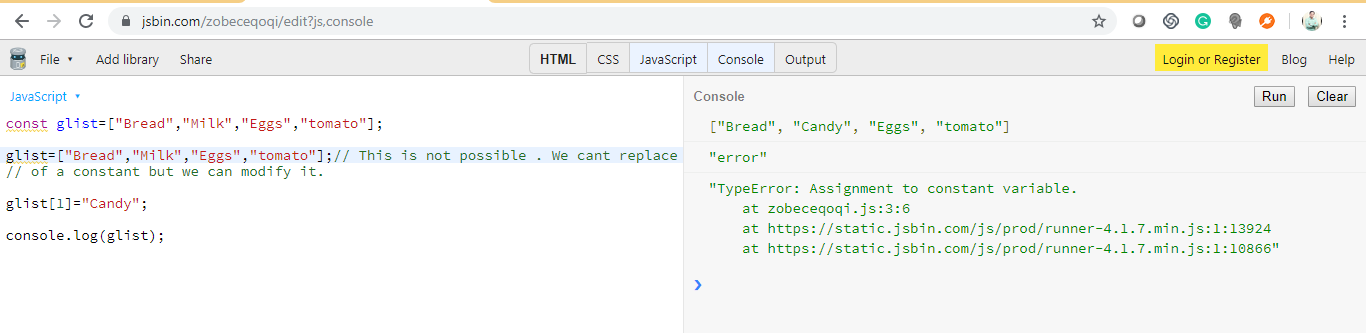
12th Sep 2019 | Thursday



Array in Javascript-

|  |
| --- |
| let glist=["Bread","Milk","Milk","Milk","Eggs","tomato"];  //console.log(glist.constructor===Array); //true  console.log(Array.isArray(glist)); //true    console.log(glist[3]); // "tomato"  console.log(glist[5]); // undefined  console.log(glist[-1]); // undefined  console.log(glist.indexOf("Eggs")); //4  console.log(glist.indexOf("Candy")); //-1  console.log(glist.indexOf("Milk")); // 1 By Default checks first occurance of Milk  console.log(glist.indexOf("Milk", 2)); //2 Second Occurance of Milk |

We should declare an array as a constant and most of the cases it should work.



|  |
| --- |
| const glist=["Bread","Milk","Eggs","tomato"];  glist=["Bread","Milk","Eggs","tomato"];// This is not possible . We cant replace value  // of a constant but we can modify it.  glist[1]="Candy";  console.log(glist); |

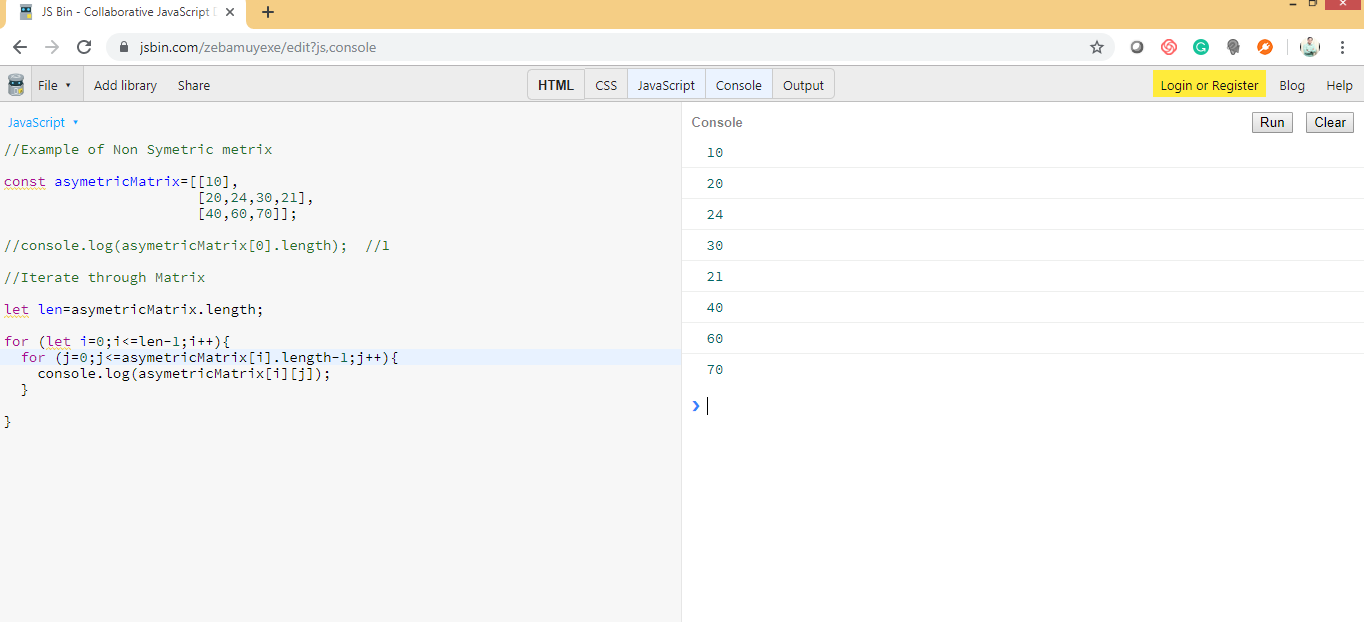
16th Sep 2019 | Monday

Mutating Array-

|  |
| --- |
| const glist=["Bread","Milk","Eggs","Tomato"];  glist.push("Candy"); // ["Bread", "Milk", "Eggs", "Tomato", "Candy"]  glist.pop(); // ["Bread", "Milk", "Eggs", "Tomato"]  glist.unshift("Candy"); //["Candy", "Bread", "Milk", "Eggs", "Tomato"]  glist.shift(); //["Bread", "Milk", "Eggs", "Tomato"]  //If you want to add/remove element from middle of an array then use splice()  // The first argument specifies the inndex where you want to add, second argument specifies  // if how many indexes you want to remove and third aregument is the value you want to add.  //glist.splice(1,1,"Spinach");  // If you dont want to remove any item just provide second argument as 0.  //glist.splice(1,0,"Spinach"); //["Bread", "Spinach", "Milk", "Eggs", "Tomato"]  //glist.splice(1,3,"Spinach"); // ["Bread", "Spinach"]  //In the next example, we are removing 3 indexes (by specifying number of  //idexes to remove in second Argument) from the array and adding threee items to it.  // We have added items (to be inserted) as arguments to the splice method.  //glist.splice(1,3,"Spinach","Paneer", "Cheese", "Onions"); //["Bread", "Spinach", "Paneer", "Cheese", "Onions"]  console.log(glist); |

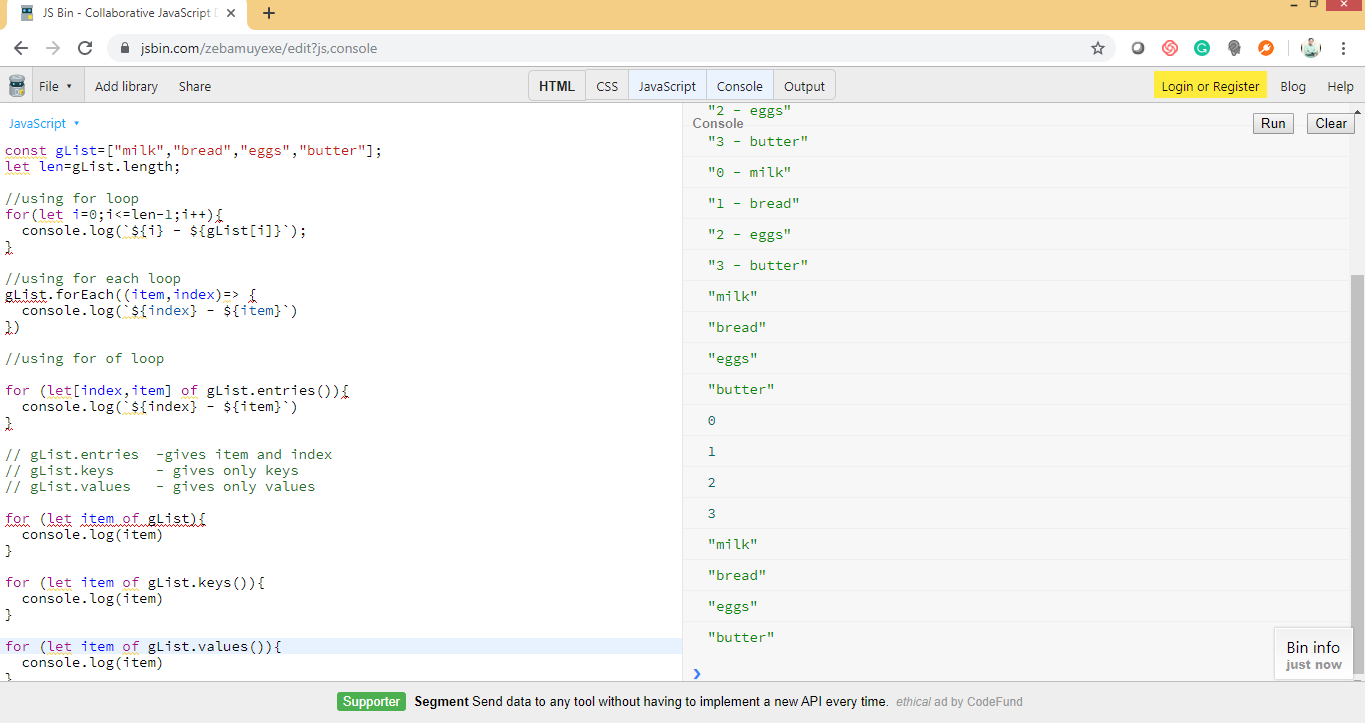
Multidimentional Array-

|  |
| --- |
| //10 15 17  //20 24 30  //40 60 70  //const row1=[10,15,17];  //const row2=[20,24,30];  //const row3=[40,60,70];  //const matrix=[row1,row2,row3];  //This is Symentric metrix  const matrix=[[10,15,17],  [20,24,30],  [40,60,70]];  //console.log(matrix);  //console.log(matrix[0].length);  //console.log(matrix[2][0]); //40  //console.log(matrix[1][1]); //24  //console.log(matrix[1][2]); //30  //console.log(matrix[2][2]); //70  //Example of Non Symetric metrix  const asymetricMatrix=[[10],  [20,24,30,21],  [40,60,70]];  console.log(asymetricMatrix[0].length); //1 |



|  |
| --- |
| //Example of Non Symetric metrix  const asymetricMatrix=[[10],  [20,24,30,21],  [40,60,70]];  //console.log(asymetricMatrix[0].length); //1  //Iterate through Matrix  let len=asymetricMatrix.length;  for (let i=0;i<=len-1;i++){  for (j=0;j<=asymetricMatrix[i].length-1;j++){  console.log(asymetricMatrix[i][j]);  }    } |

Iterating array using “for each” & “for of”



|  |
| --- |
| const gList=["milk","bread","eggs","butter"];  let len=gList.length;  //using for loop  for(let i=0;i<=len-1;i++){  console.log(`${i} - ${gList[i]}`);  }  //using for each loop  gList.forEach((item,index)=> {  console.log(`${index} - ${item}`)  })  //using for of loop  for (let[index,item] of gList.entries()){  console.log(`${index} - ${item}`)  }  // gList.entries -gives item and index  // gList.keys - gives only keys  // gList.values - gives only values  for (let item of gList){  console.log(item)  }  for (let item of gList.keys()){  console.log(item)  }  for (let item of gList.values()){  console.log(item)  } |

18th Sep 2019 | Wednesday

Iterating object using for in and for of

|  |
| --- |
| const profile={  name: "John",  single : true,  "got a job" : true,  kids: [ {name: "Peter",age:3},{name:"Sara", age: 1}]  };  //If we want to get only keys, we can use prop variable directly  /\*  for(let prop in profile)  {  console.log(prop)  }  OutPut-  "name"  "single"  "got a job"  "kids"  \*/  //If we want to get values  for (let prop in profile){  console.log(`Key: ${prop} - Value: ${profile[prop]}`);  }  //Output-  "Key: name - Value: John"  "Key: single - Value: true"  "Key: got a job - Value: true"  "Key: kids - Value: [object Object],[object Object]"  \*/  /\*  //Using for of loop  for (let prop of Object.keys(profile)){  console.log(`Key: ${prop} - Value: ${profile[prop]}`);  }  //Output-  "Key: name - Value: John"  "Key: single - Value: true"  "Key: got a job - Value: true"  "Key: kids - Value: [object Object],[object Object]"  \*/ |

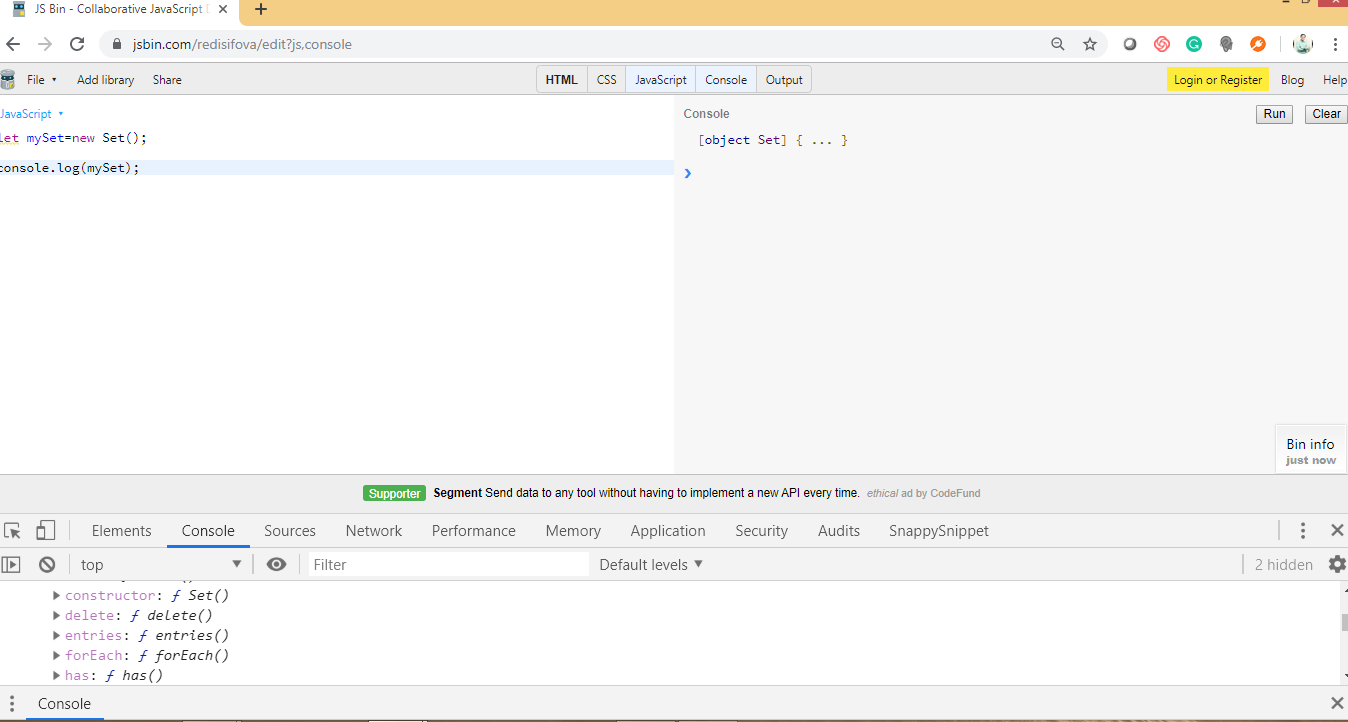
19th Sep 2019 | Thursday

Destructuring-

|  |
| --- |
| const obj={x:1,y:2};  //let x=obj.x;  //let y=obj.y;  /\*  let {x:x1,y:y1}=obj;  console.log(x1); //1  console.log(y1); //2  \*/  /\*  let {x:x,y:y}=obj;  console.log(x);  \*/  //----We can do following in latest javascript.  //If we have value name same as variable name we can simply do it as following  //let {x,y}=obj;  //console.log(x); //1  //So the destructuring allows us to get the multiple properties on the same line.  //Example of an array  /\*  const ary=[1,2];  let [x,y]=ary;  console.log(x);  \*/  //if we get only first element of an array only. Then we would have only that element available  const ary=[1,2];  let [x]=ary;  console.log(y); |

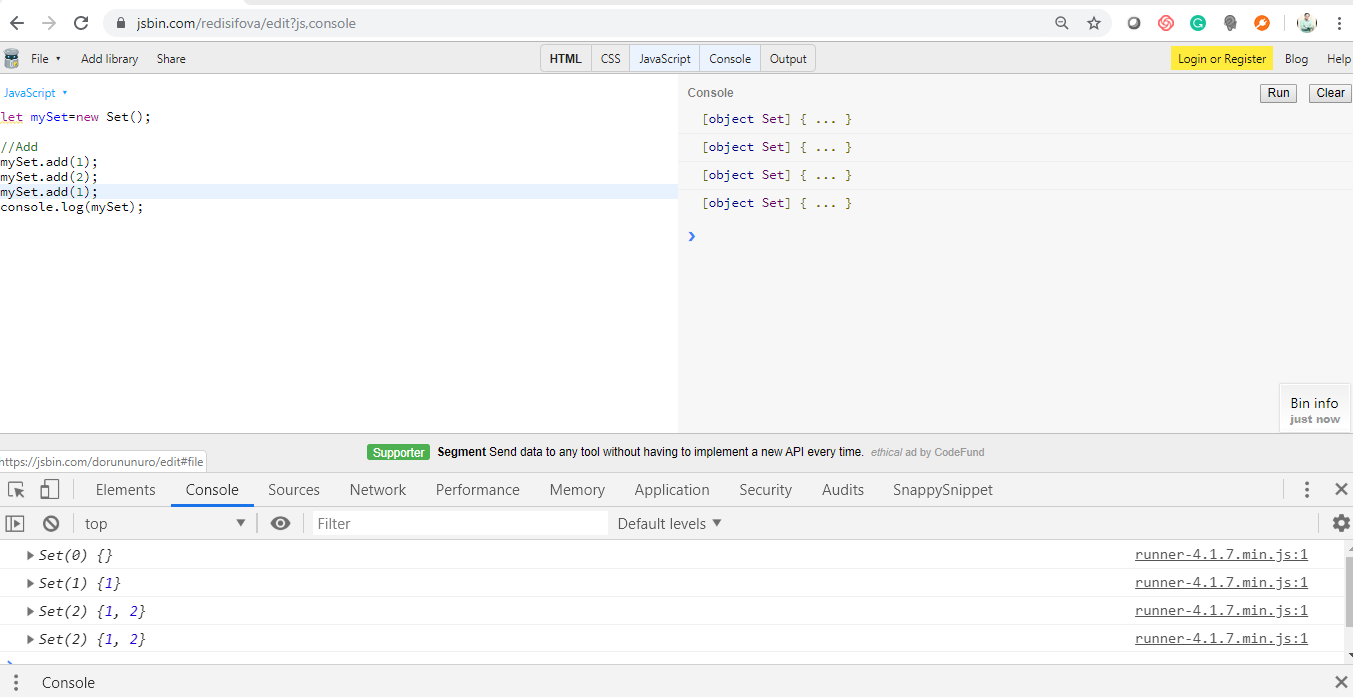
Sets-

Sets is a new data type. Sets essentially is an object that lets you store values of any data type. It could be primitive data type or object data type.So we can say that it’s simply a unique list.

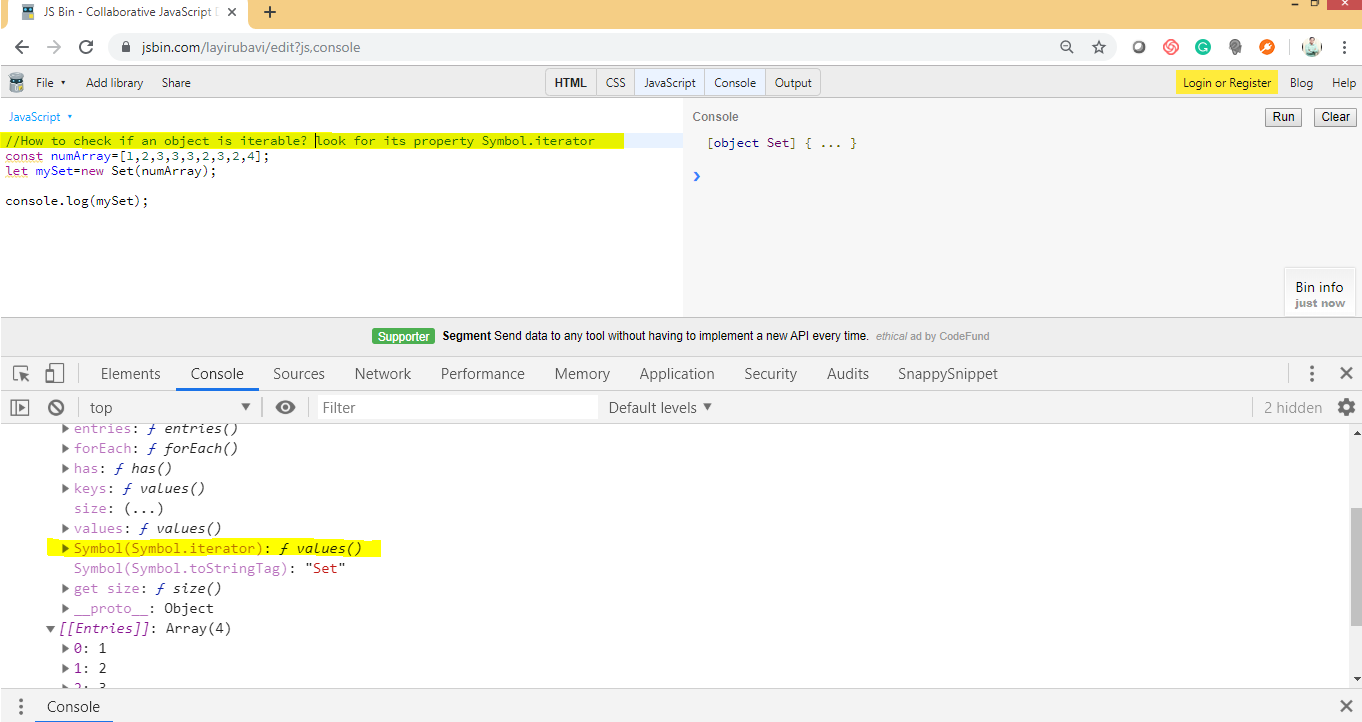


We have different set of method unlike arrays.

Since Set doesnot allow duplicate values, when we added 1 multiple times it ignores it.



|  |
| --- |
| //let mySet=new Set();  /\*  //Add  mySet.add(1);  mySet.add(2);  mySet.add(1);  console.log(mySet);  \*/  //To check if set has a value or not. Use has()  //console.log(mySet.has(1)); //true  //console.log(mySet.has(5)); //false  /\*  //To Delete  mySet.add(1).add(2).add(5).add(3).delete(1); //\* did method chaining  console.log(mySet); //It would have Set(3) {2, 5, 3} \*check in console  //To empty the set, use clear()  mySet.clear();  console.log(mySet);  \*/  /\*//To convert an array into a set  const numArray=[1,2,3];  let mySet=new Set(numArray);  console.log(mySet); \*/  //So let say we have an array with duplicate values and we want to remove those duplicates  //so instead of looping through that arrray to remove duplicate we can simply convert it to  // a set and then again convert back to an array.  const numArray=[1,2,3,3,3,2,3,2,4];  let mySet=new Set(numArray);  console.log(mySet); // Set(4) {1, 2, 3, 4} \*check through console  //To convert it back to array we have spread operator [...mySet]  console.log([...mySet]); // [1, 2, 3, 4] |



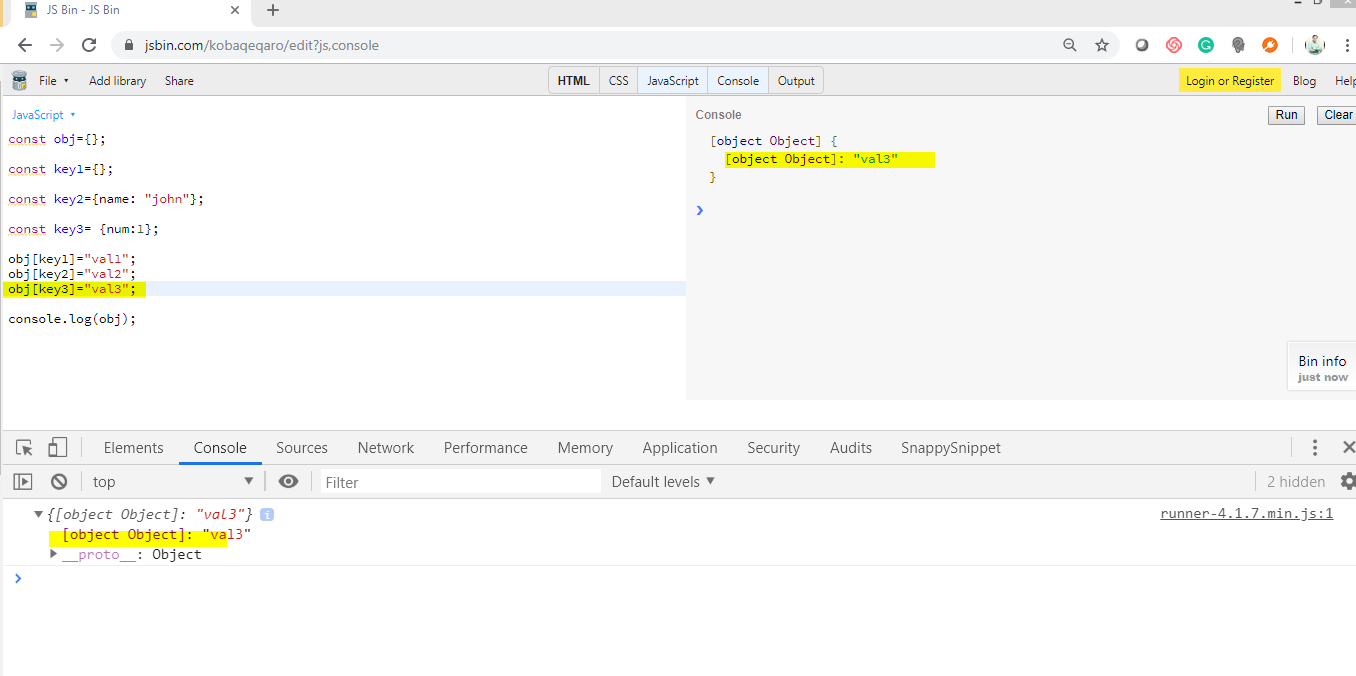
|  |
| --- |
| //How to check if an object is iterable? look for its property Symbol.iterator  const numArray=[1,2,3,3,3,2,3,2,4];  let mySet=new Set(numArray);  console.log(mySet);  //looping through Sets  for (let val of mySet){  console.log(val);  }  /\*Output-  1  2  3  4  \*/ |

One important point about set is It does not have ordering and it also doesn’t have methods such as sort.

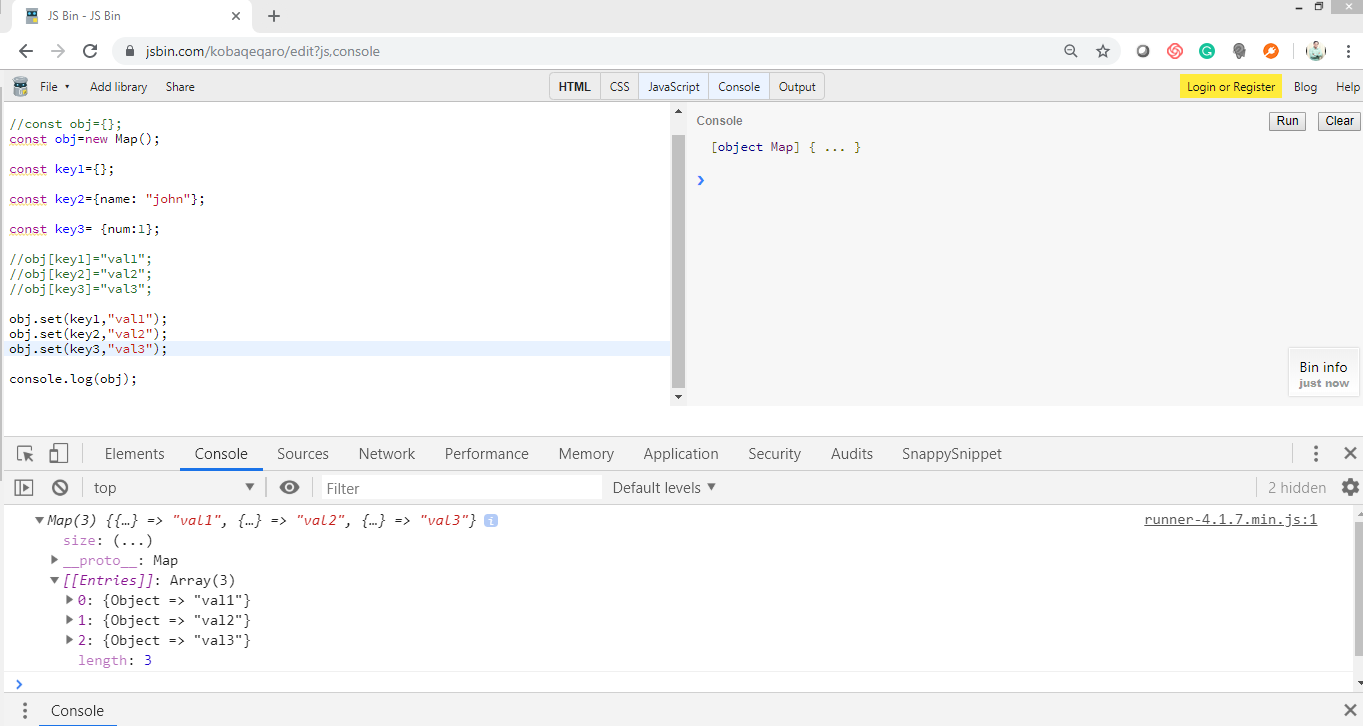
22nd Sep 2019 | Sunday

Maps and WeakMaps (ES6 feature)

If we try to use object as a key. In below example we are doing that



In above example, we have only one object and that is the last one. That is because in typical JavaScript object, it can hold only one object as a key. So in recent version they introduced Maps and Weakmaps.



Example of Map datastructure-

|  |
| --- |
| //const obj={};  const obj=new Map();  let key1={};  let key2={name: "john"};  let key3= {num:1};  //obj[key1]="val1";  //obj[key2]="val2";  //obj[key3]="val3";  /\*  obj.set(key1,"val1");  obj.set(key2,"val2");  obj.set(key3,"val3");  \*/  //Or use method chaining syntax like below  obj.set(key1,"val1").set(key2,"val2").set(key3,"val3");  console.log(obj);  /\*  Map(3) {{…} => "val1", {…} => "val2", {…} => "val3"}size: (...)\_\_proto\_\_: Map[[Entries]]: Array(3)0: {Object => "val1"}1: {Object => "val2"}2: {Object => "val3"}length: 3  \*/  console.log(obj.get(key2)); // Output- "val2"  //If we have empty object then we will get undefined as an output.Since object is a reference type.  //To interate - we can see Symbol iterator property in proto hence we can use for of  /\*  for(let entry of obj){  console.log(entry);  }  \*/  / |

Example of WeakMap datastructure-

|  |
| --- |
| const obj=new WeakMap();  let key1={};  let key2={name: "john"};  let key3= {num:1};  obj.set(key1,"val1").set(key2,"val2").set(key3,"val3");  key1=null;  //If we set a key as null still we can see the key1 is there. for that they introduce  //new datastructure called WeakMap  console.log(obj.get(key1)); //Output- undefined  //Note- There is rare scenario where we need WeakMap but Maps are very useful.  //For knowledge purpose knowing Weakmap is good. |