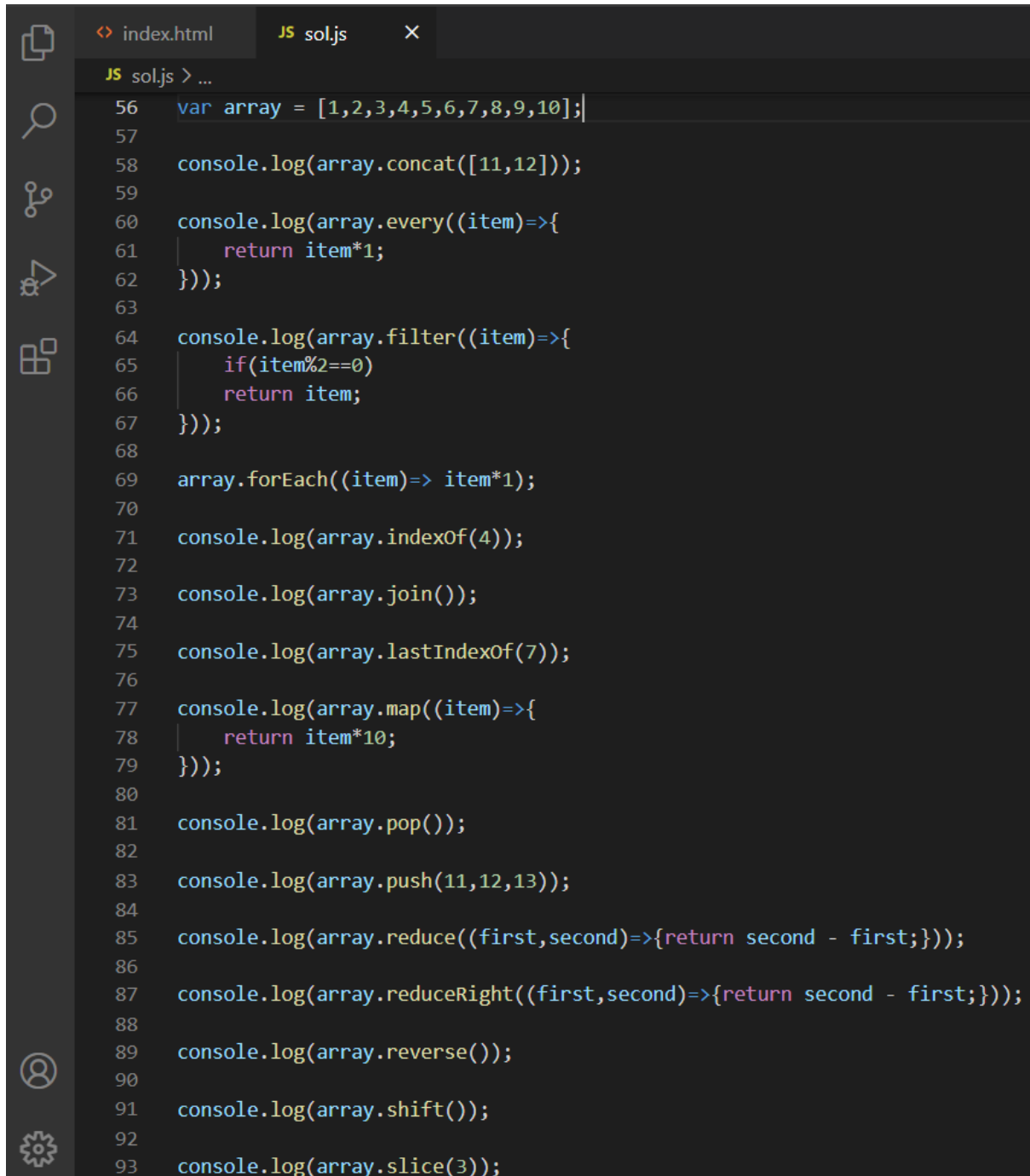


JavaScript Assignment

Submitted By – Ayushi Garg

Question 1 –



```
index.html JS sol.js X
JS sol.js > ...
56 var array = [1,2,3,4,5,6,7,8,9,10];
57
58 console.log(array.concat([11,12]));
59
60 console.log(array.every((item)=>{
61     return item*1;
62 }));
63
64 console.log(array.filter((item)=>{
65     if(item%2==0)
66     return item;
67 }));
68
69 array.forEach((item)=> item*1);
70
71 console.log(array.indexOf(4));
72
73 console.log(array.join());
74
75 console.log(array.lastIndexOf(7));
76
77 console.log(array.map((item)=>{
78     return item*10;
79 }));
80
81 console.log(array.pop());
82
83 console.log(array.push(11,12,13));
84
85 console.log(array.reduce((first,second)=>{return second - first;}));
86
87 console.log(array.reduceRight((first,second)=>{return second - first;}));
88
89 console.log(array.reverse());
90
91 console.log(array.shift());
92
93 console.log(array.slice(3));
```

```

94
95 console.log(array.some((item)=>{return item>6}));
96
97 console.log(array.sort());
98
99 console.log(array.splice(3,4));
100
101 console.log(array.toString());
102
103 console.log(array.unshift(-2,-1,0));

```

Sources	Elements	Console	Network	Performance	Memory	Application	»	⚙	⋮	✕
▶	⊞	top	▼	👁	Filter	Default levels ▼		⚙		
▶ (12)	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]									sol.js:58
	true									sol.js:60
▶ (5)	[2, 4, 6, 8, 10]									sol.js:64
	3									sol.js:71
	1,2,3,4,5,6,7,8,9,10									sol.js:73
	6									sol.js:75
▶ (10)	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]									sol.js:77
	10									sol.js:81
	12									sol.js:83
	7									sol.js:85
	-7									sol.js:87
▶ (12)	[13, 12, 11, 9, 8, 7, 6, 5, 4, 3, 2, 1]									sol.js:89
	13									sol.js:91
▶ (8)	[8, 7, 6, 5, 4, 3, 2, 1]									sol.js:93
	true									sol.js:95
▶ (11)	[1, 11, 12, 2, 3, 4, 5, 6, 7, 8, 9]									sol.js:97
▶ (4)	[2, 3, 4, 5]									sol.js:99
	1,11,12,6,7,8,9									sol.js:101
	10									sol.js:103
	>									

Question 2 –

```

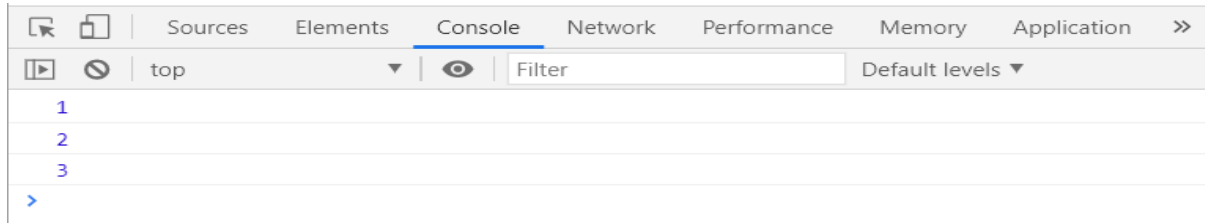
var add = (function () {
    var counter = 0;
    return function () {return counter += 1;}
})();

add();
add();
add();

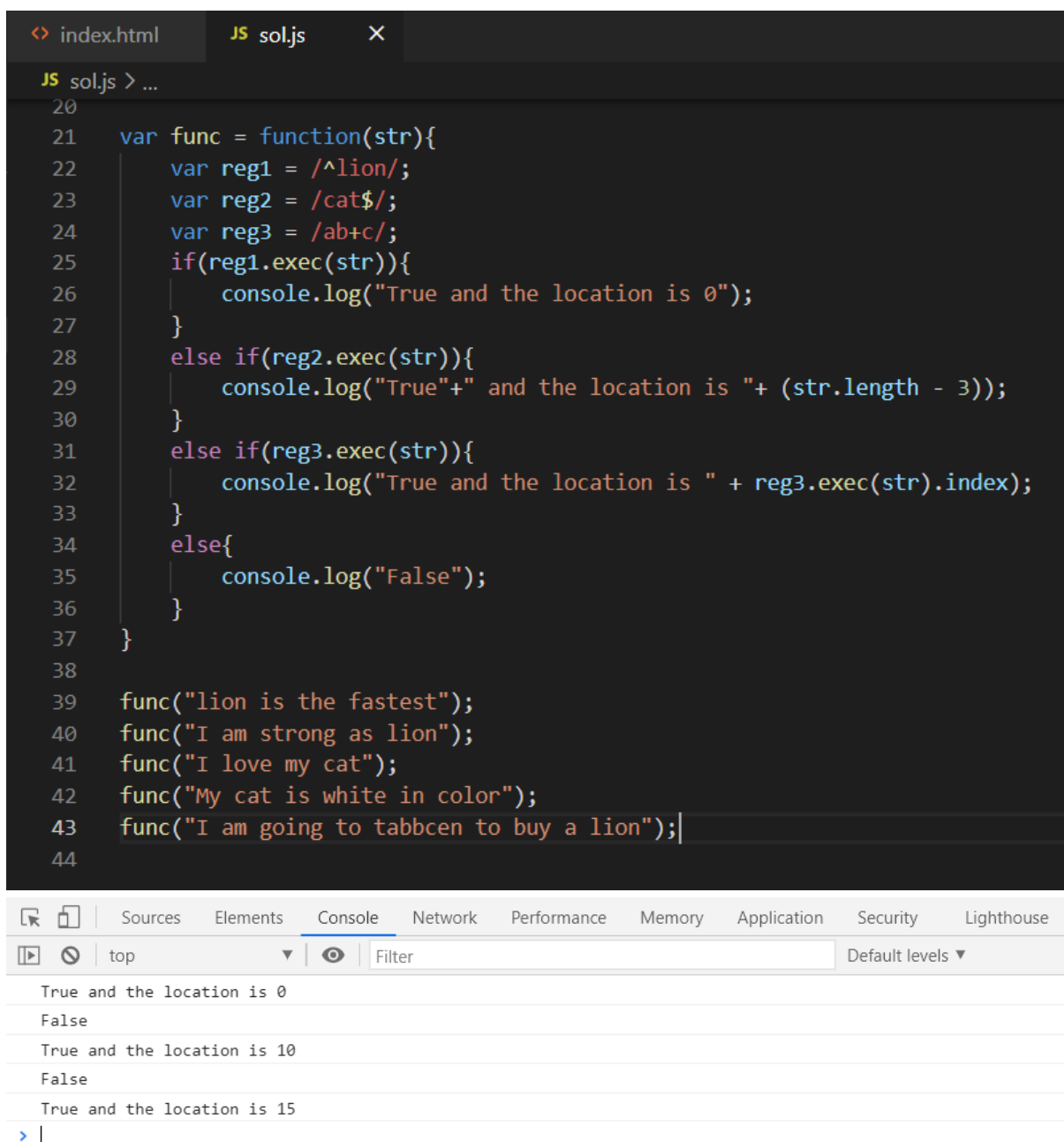
```

In the above piece of code, it can be seen that add is a function that returns a self-invoking function. It has a variable named counter, assigned 0 as an initial value. The self-invoking function increments the value of counter by 1 and return the same to the calling statement.

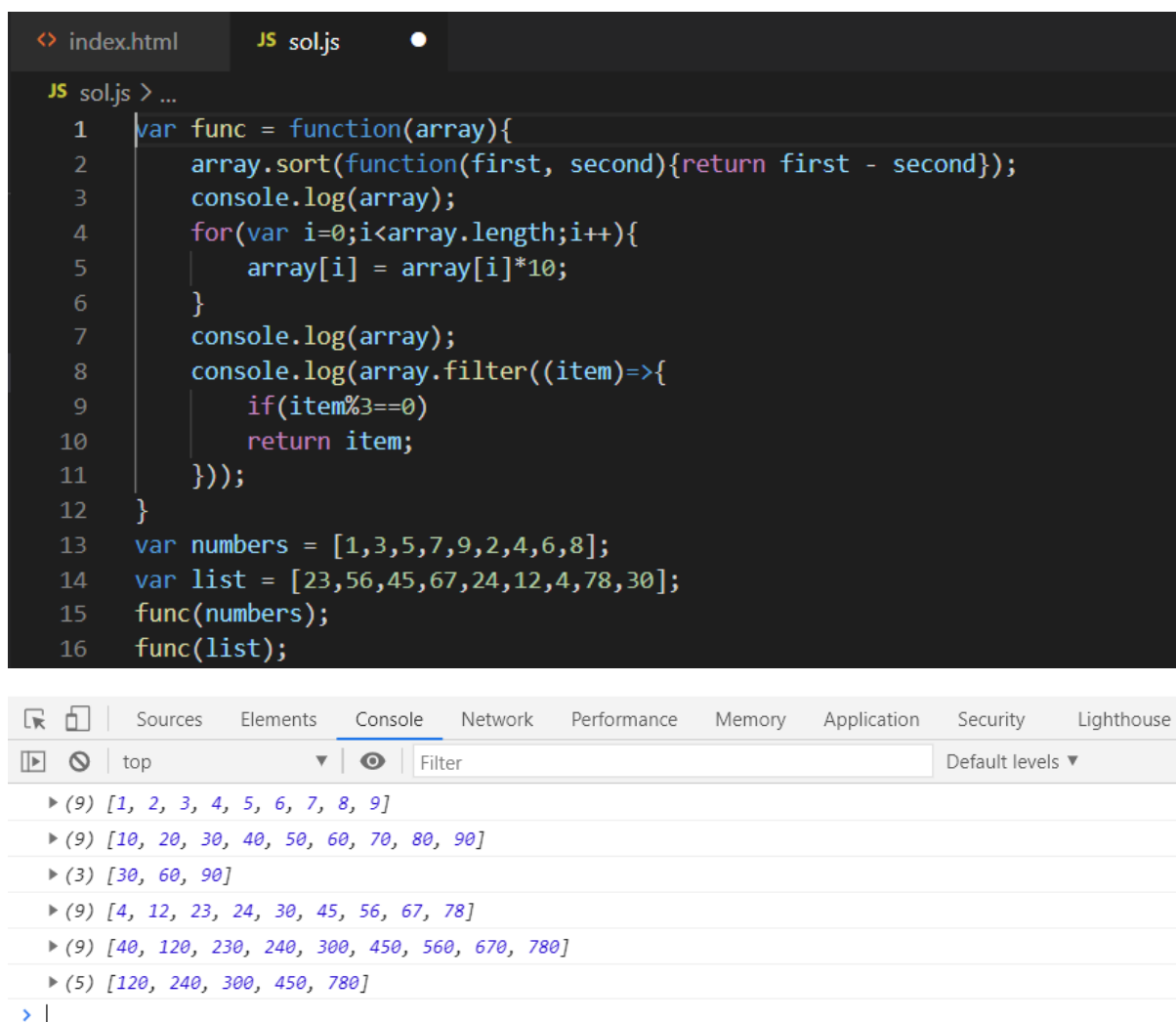
As soon as the control reaches first add (), the add function is called and self-invoking function increments counter value to 1 and return it to the add function which results in returning of 1 to the called add (). As the counter belongs to local scope of add function, the value of incremented counter stays for the next call. Hence the output of the above code is as follows -



Question 3 –



Question 4 –



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the output of a JavaScript script. The script defines a function 'func' that sorts an array, logs it, multiplies each element by 10, logs it again, and then filters the array to only include elements divisible by 3. The script then calls 'func' on two arrays: 'numbers' and 'list'.

```
JS sol.js > ...
1  var func = function(array){
2      array.sort(function(first, second){return first - second});
3      console.log(array);
4      for(var i=0;i<array.length;i++){
5          array[i] = array[i]*10;
6      }
7      console.log(array);
8      console.log(array.filter((item)=>{
9          if(item%3==0)
10             return item;
11         }));
12 }
13 var numbers = [1,3,5,7,9,2,4,6,8];
14 var list = [23,56,45,67,24,12,4,78,30];
15 func(numbers);
16 func(list);
```

The console output shows the following log messages:

- (9) [1, 2, 3, 4, 5, 6, 7, 8, 9]
- (9) [10, 20, 30, 40, 50, 60, 70, 80, 90]
- (3) [30, 60, 90]
- (9) [4, 12, 23, 24, 30, 45, 56, 67, 78]
- (9) [40, 120, 230, 240, 300, 450, 560, 670, 780]
- (5) [120, 240, 300, 450, 780]

Question 5 –

== : Let's say we take a string and a number variable. When we compare the two variable using == operator, the operator automatically converts one type to other and return true if the value of the variables is same else it returns false.

=== : The operator will check the type of both variables as well the values. If both type and values of the variables are same, then it will return true. Otherwise, it returns false.

For example –

2 == 2 // True

2 == '2' // True

2 === 2 // True

2 === '2' // False