

1. Convert the string "258.90" to: (a) integer, (b) floating number. Store in two variables.

The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
1 // JS Lab-3 Assignment //  
2 // ----- //  
3 var num = "258.90";  
4  
5 var a = parseInt(num);  
6 var b = parseFloat(num);  
7  
8 console.log(a);  
9 console.log(b);  
10
```

The browser console on the right shows the output of the code:

```
258      main.js:8  
258.9    main.js:9  
>
```

2. Format the number 7.45678 to exactly 2 decimal places (string) then convert it back to a number.

The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
1 // JS Lab-3 Assignment //  
2 // ----- //  
3  
4 var num = 7.45678;  
5  
6 num = num.toFixed(2);  
7 console.log(num);  
8  
9 num = parseFloat(num);  
10 console.log(num);  
11
```

The browser console on the right shows the output of the code:

```
7.46     main.js:7  
7.46     main.js:10  
>
```

3. Check if the value 'abc' is NaN. Also show a case where `isNaN` returns false for a non-number.

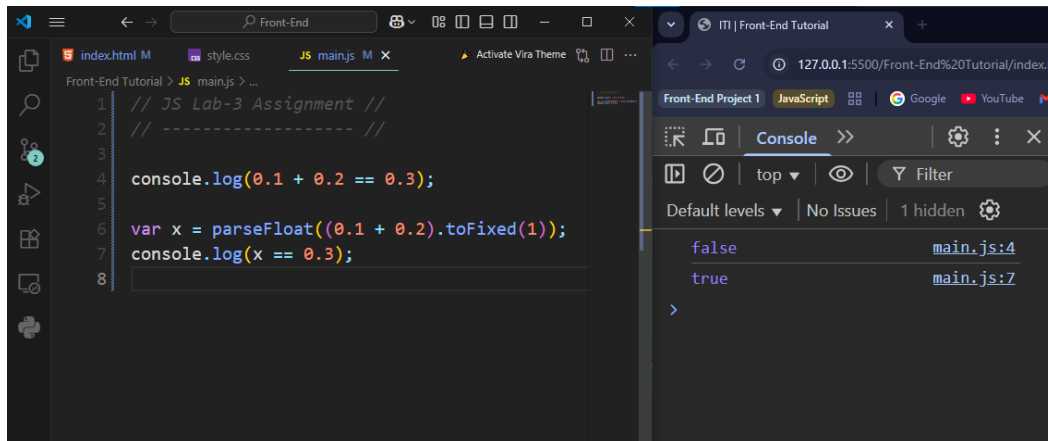
The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
1 // JS Lab-3 Assignment //  
2 // ----- //  
3  
4 console.log(isNaN("abc"));  
5 console.log(isNaN("123"));  
6  
7 console.log(isNaN(" "));  
8 console.log(isNaN([]));  
9
```

The browser console on the right shows the output of the code:

```
true      main.js:4  
false     main.js:5  
false     main.js:7  
false     main.js:8  
>
```

4. Floating point: Show that $(0.1 + 0.2) \neq 0.3$. Produce a corrected decimal string with exactly 1 decimal place using `toFixed`.



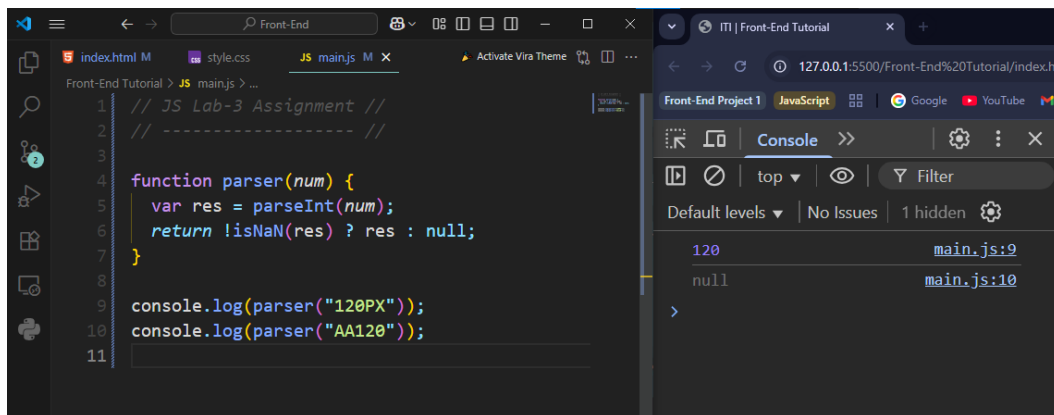
The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 console.log(0.1 + 0.2 == 0.3);
5
6 var x = parseFloat((0.1 + 0.2).toFixed(1));
7 console.log(x == 0.3);
8
```

The browser console on the right shows the output of the code:

```
false    main.js:4
true     main.js:7
>
```

5. Write a function to safely parse a string that may contain trailing text (e.g. "120px") returning the integer part or null if it starts with non-digit.



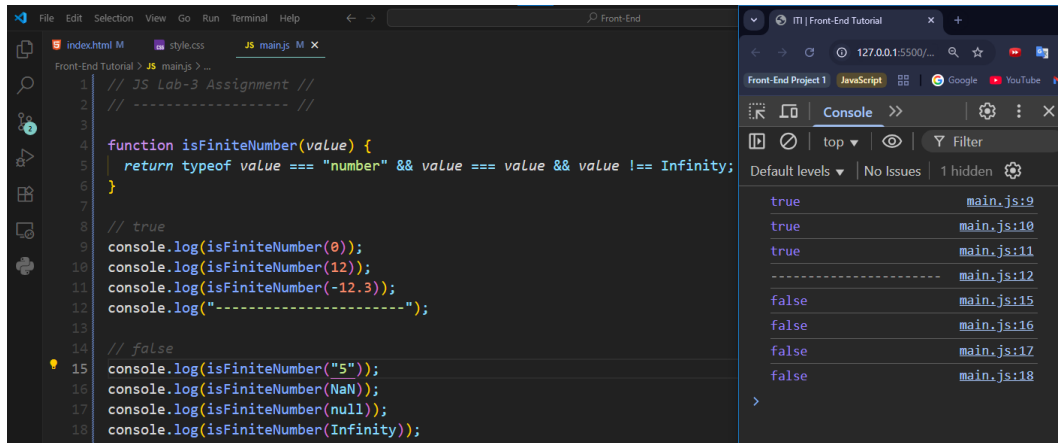
The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function parser(num) {
5   var res = parseInt(num);
6   return !isNaN(res) ? res : null;
7 }
8
9 console.log(parser("120PX"));
10 console.log(parser("AA120"));
11
```

The browser console on the right shows the output of the code:

```
120     main.js:9
null    main.js:10
>
```

6. Implement `isFiniteNumber(value)` that returns true only for finite numeric values (reject numeric strings, Infinity, NaN, null, etc.) WITHOUT using `Number.isFinite`. Provide 4 passing and 4 failing test examples (comments).



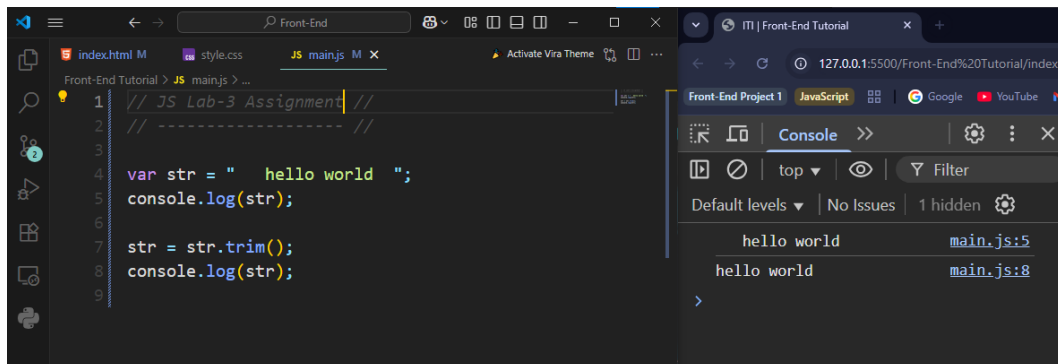
The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function isFiniteNumber(value) {
5   return typeof value === "number" && value === value && value !== Infinity;
6 }
7
8 // true
9 console.log(isFiniteNumber(0));
10 console.log(isFiniteNumber(12));
11 console.log(isFiniteNumber(-12.3));
12 console.log("-----");
13
14 // false
15 console.log(isFiniteNumber("5"));
16 console.log(isFiniteNumber(NaN));
17 console.log(isFiniteNumber(null));
18 console.log(isFiniteNumber(Infinity));
```

The browser console on the right shows the output of the code:

```
true main.js:9
true main.js:10
true main.js:11
----- main.js:12
false main.js:15
false main.js:16
false main.js:17
false main.js:18
```

7. Remove leading and trailing spaces from the string " hello world ".



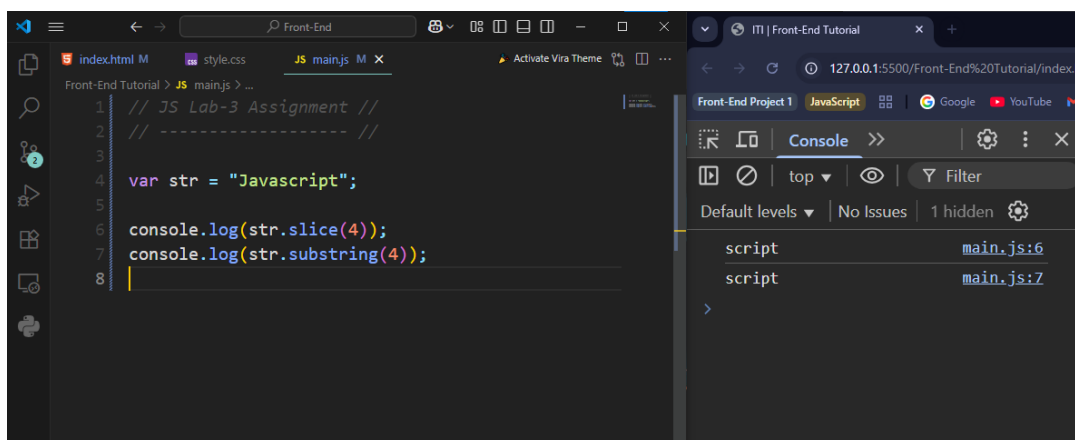
The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var str = " hello world ";
5 console.log(str);
6
7 str = str.trim();
8 console.log(str);
9
```

The browser console on the right shows the output of the code:

```
hello world main.js:5
hello world main.js:8
```

8. Get the substring "script" from "javascript" using two different methods (slice + substring).



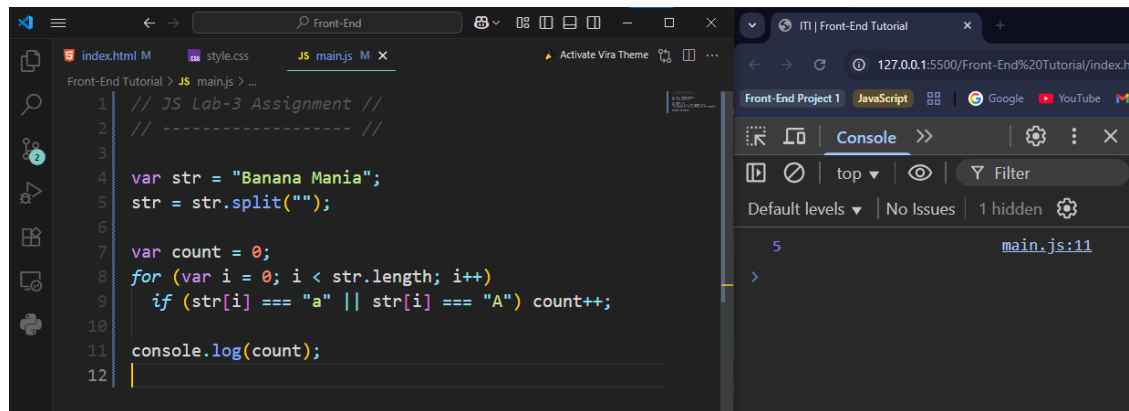
The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var str = "JavaScript";
5
6 console.log(str.slice(4));
7 console.log(str.substring(4));
8
```

The browser console on the right shows the output of the code:

```
script main.js:6
script main.js:7
```

9. Count how many times the letter 'a' appears in "Banana Mania" (case-insensitive).

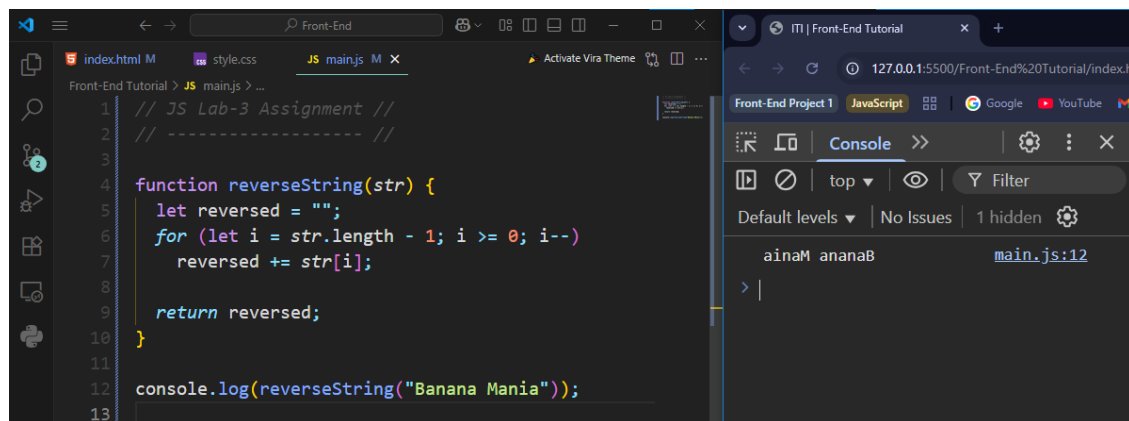


The screenshot shows a VS Code editor with a JavaScript file named `main.js`. The code defines a string `str = "Banana Mania"`, splits it into an array, and iterates through each character to count the occurrences of 'a' (case-insensitive). The console output shows the count is 5.

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var str = "Banana Mania";
5 str = str.split("");
6
7 var count = 0;
8 for (var i = 0; i < str.length; i++)
9   if (str[i] === "a" || str[i] === "A") count++;
10
11 console.log(count);
12
```

Browser console output: 5 (main.js:11)

10. Write a function `reverseString(s)` without using array reverse (iterate backwards).

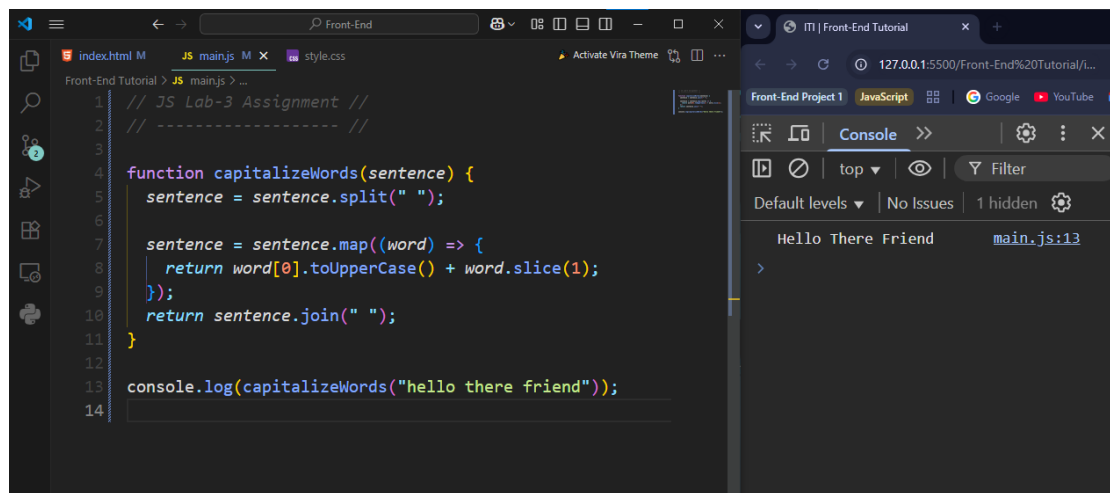


The screenshot shows a VS Code editor with a JavaScript file named `main.js`. The code defines a function `reverseString(str)` that iterates backwards through the string to build a reversed string. The console output shows the reversed string "ainam ananaB".

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function reverseString(str) {
5   let reversed = "";
6   for (let i = str.length - 1; i >= 0; i--)
7     reversed += str[i];
8
9   return reversed;
10 }
11
12 console.log(reverseString("Banana Mania"));
13
```

Browser console output: ainaM ananaB (main.js:12)

11. Write a function `capitalizeWords(sentence)` that turns "hello there friend" into "Hello There Friend".

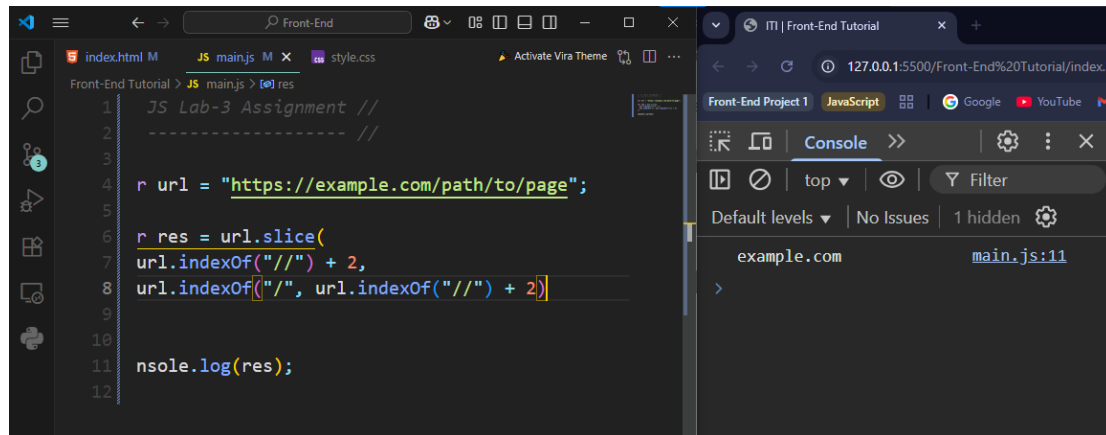


The screenshot shows a VS Code editor with a JavaScript file named `main.js`. The code defines a function `capitalizeWords(sentence)` that splits the sentence into words, capitalizes the first letter of each word, and joins them back together. The console output shows the capitalized sentence "Hello There Friend".

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function capitalizeWords(sentence) {
5   sentence = sentence.split(" ");
6
7   sentence = sentence.map((word) => {
8     return word[0].toUpperCase() + word.slice(1);
9   });
10  return sentence.join(" ");
11 }
12
13 console.log(capitalizeWords("hello there friend"));
14
```

Browser console output: Hello There Friend (main.js:13)

12. Extract the domain (without protocol and path) from a URL string like "https://example.com/path/to/page" (result: example.com) using indexOf + slice.

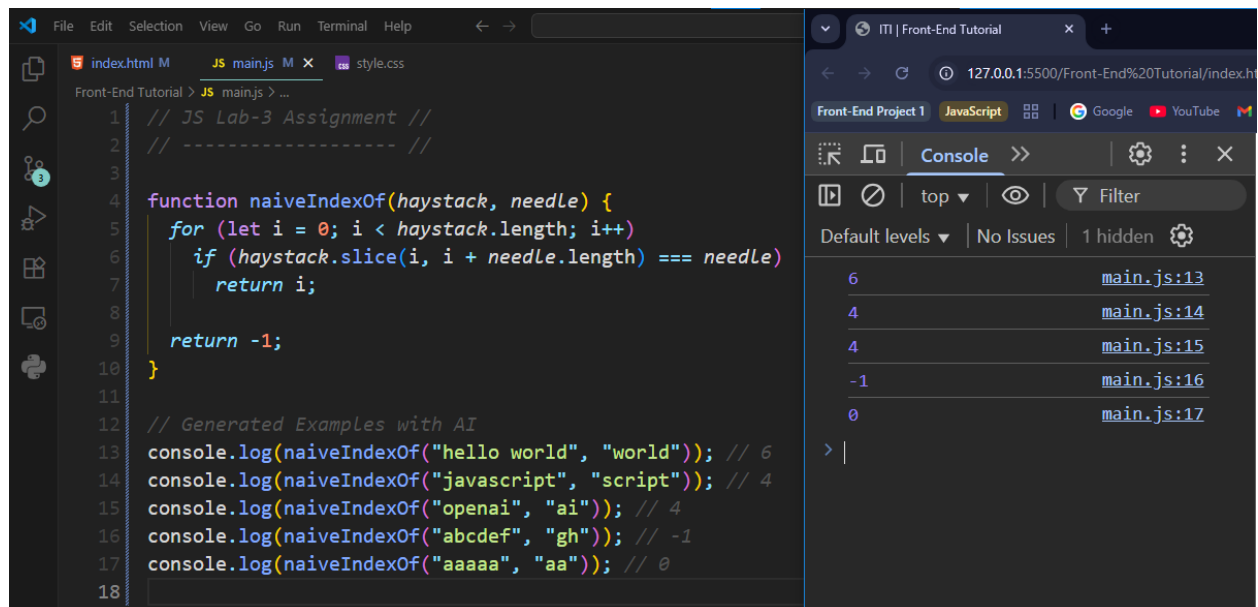


The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 r url = "https://example.com/path/to/page";
5
6 r res = url.slice(
7   url.indexOf("/") + 2,
8   url.indexOf("/", url.indexOf("/") + 2)
9
10
11 console.log(res);
12
```

The browser console on the right shows the result of the code execution: `example.com` at `main.js:11`.

13. Implement a simple substring search function `naiveIndexOf(haystack, needle)` that returns first index or -1 (do not call built-in `indexOf` inside the loop).



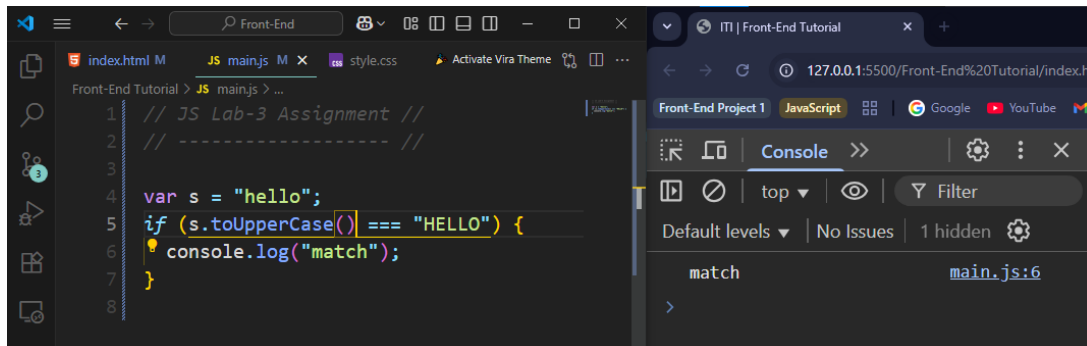
The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function naiveIndexOf(haystack, needle) {
5   for (let i = 0; i < haystack.length; i++)
6     if (haystack.slice(i, i + needle.length) === needle)
7       return i;
8   return -1;
9 }
10
11 // Generated Examples with AI
12 console.log(naiveIndexOf("hello world", "world")); // 6
13 console.log(naiveIndexOf("javascript", "script")); // 4
14 console.log(naiveIndexOf("openai", "ai")); // 4
15 console.log(naiveIndexOf("abcdef", "gh")); // -1
16 console.log(naiveIndexOf("aaaaa", "aa")); // 0
17
18
```

The browser console on the right shows the results of the function calls:

Index	Location
6	main.js:13
4	main.js:14
4	main.js:15
-1	main.js:16
0	main.js:17

14. Buggy code: `var s = 'hello'; if (s.toUpperCase = 'HELLO') { console.log('match'); } // Fix and explain issue.`

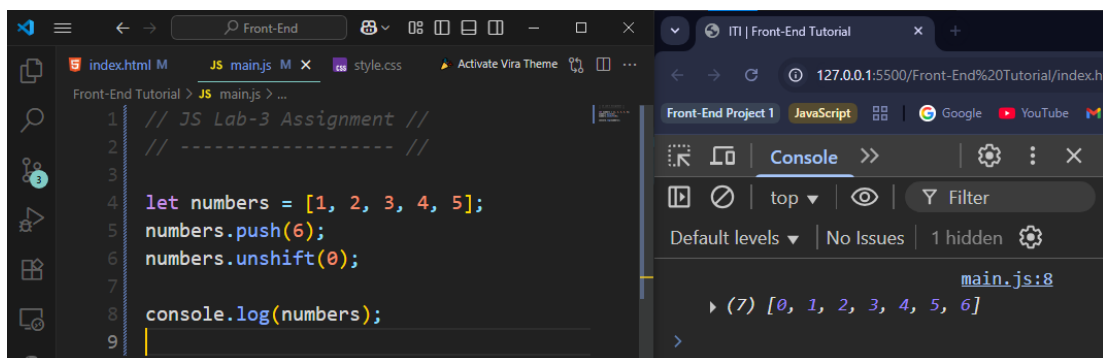


The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var s = "hello";
5 if (s.toUpperCase() === "HELLO") {
6   console.log("match");
7 }
8
```

The browser console on the right shows the output: `match` from `main.js:6`. The code is correct, and the console output matches the expected result.

15. Create an array of the numbers 1..5, then push 6 and unshift 0.

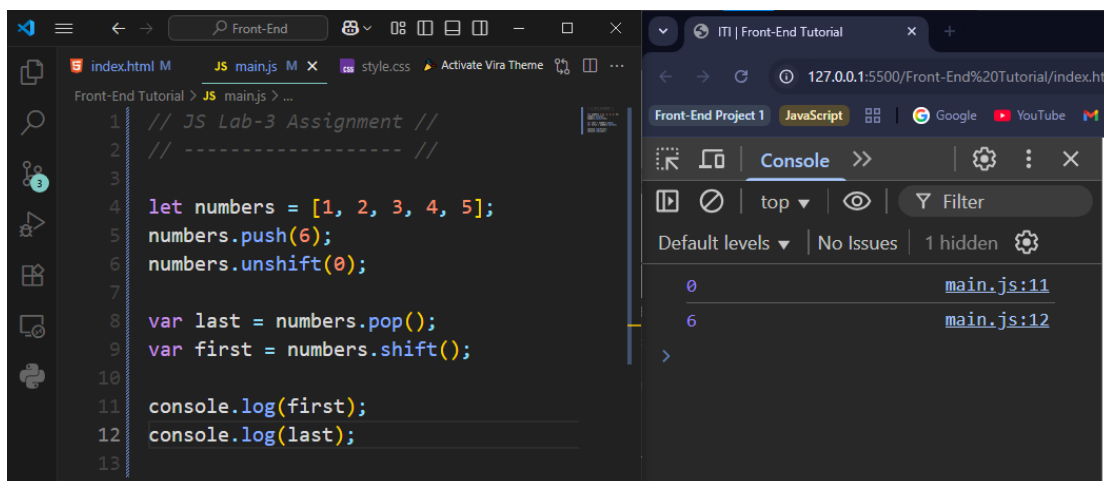


The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 let numbers = [1, 2, 3, 4, 5];
5 numbers.push(6);
6 numbers.unshift(0);
7
8 console.log(numbers);
9
```

The browser console on the right shows the output: `(7) [0, 1, 2, 3, 4, 5, 6]` from `main.js:8`. The code is correct, and the console output matches the expected result.

16. Remove the last element and store it. Remove the first element and store it.

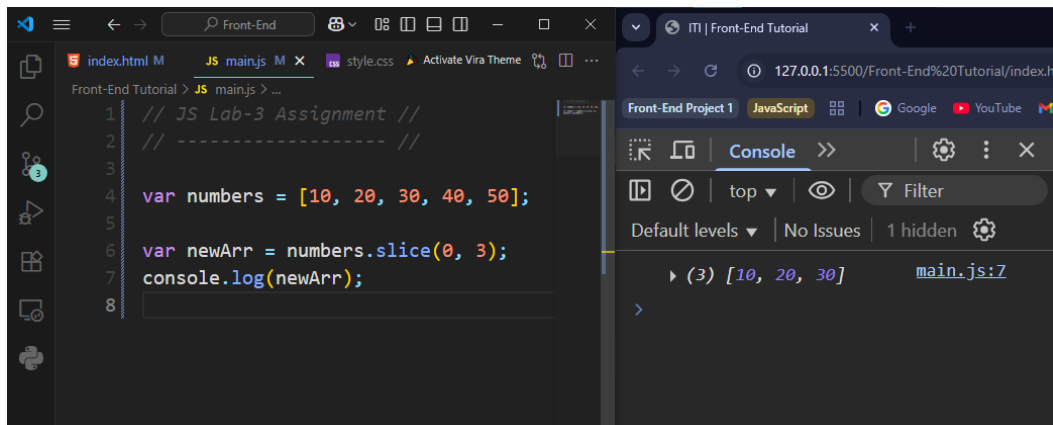


The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 let numbers = [1, 2, 3, 4, 5];
5 numbers.push(6);
6 numbers.unshift(0);
7
8 var last = numbers.pop();
9 var first = numbers.shift();
10
11 console.log(first);
12 console.log(last);
13
```

The browser console on the right shows the output: `0` from `main.js:11` and `6` from `main.js:12`. The code is correct, and the console output matches the expected result.

17. Use slice to copy the first 3 elements of [10,20,30,40,50] into a new array.



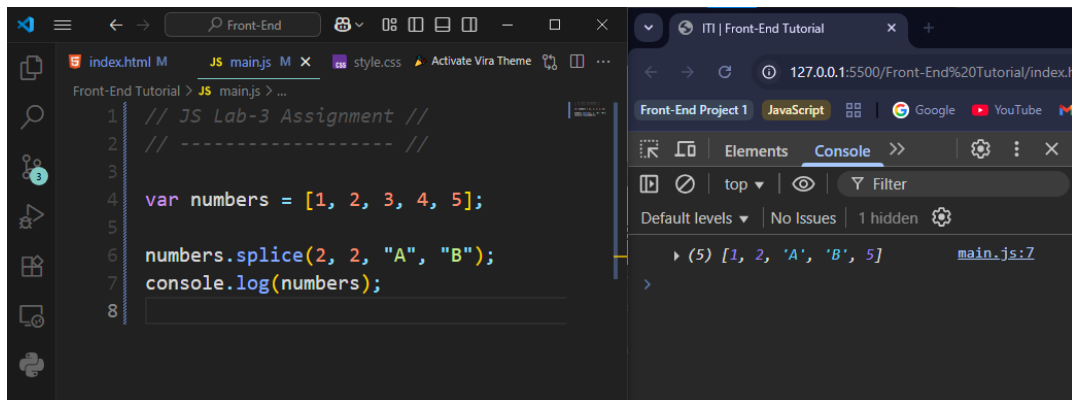
The screenshot shows a code editor with the following JavaScript code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var numbers = [10, 20, 30, 40, 50];
5
6 var newArr = numbers.slice(0, 3);
7 console.log(newArr);
8
```

The browser console on the right shows the output:

```
(3) [10, 20, 30] main.js:7
```

18. Use splice on [1,2,3,4,5] to remove 3 and 4 and insert 'a','b'. Result should be [1,2,'a','b',5].



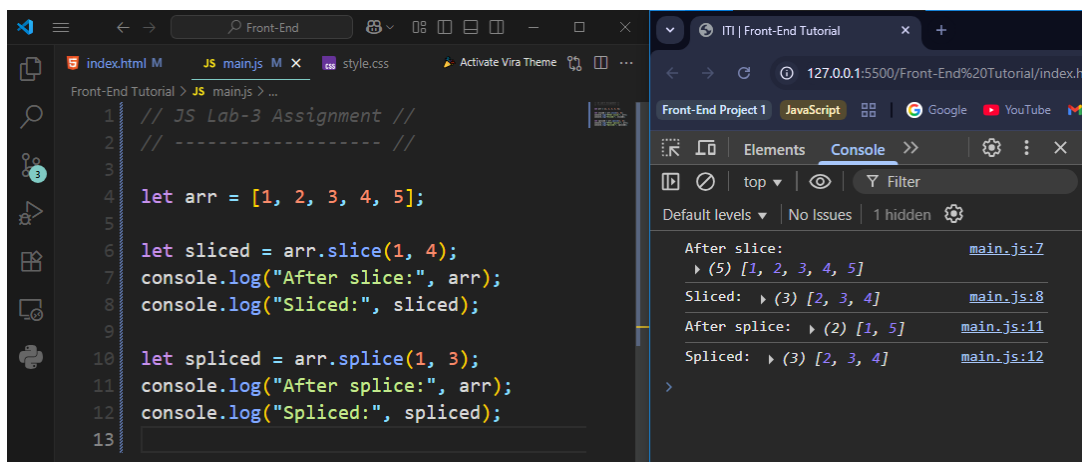
The screenshot shows a code editor with the following JavaScript code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var numbers = [1, 2, 3, 4, 5];
5
6 numbers.splice(2, 2, "A", "B");
7 console.log(numbers);
8
```

The browser console on the right shows the output:

```
(5) [1, 2, 'A', 'B', 5] main.js:7
```

19. Demonstrate the difference between slice and splice on the same starting array (show original after each).



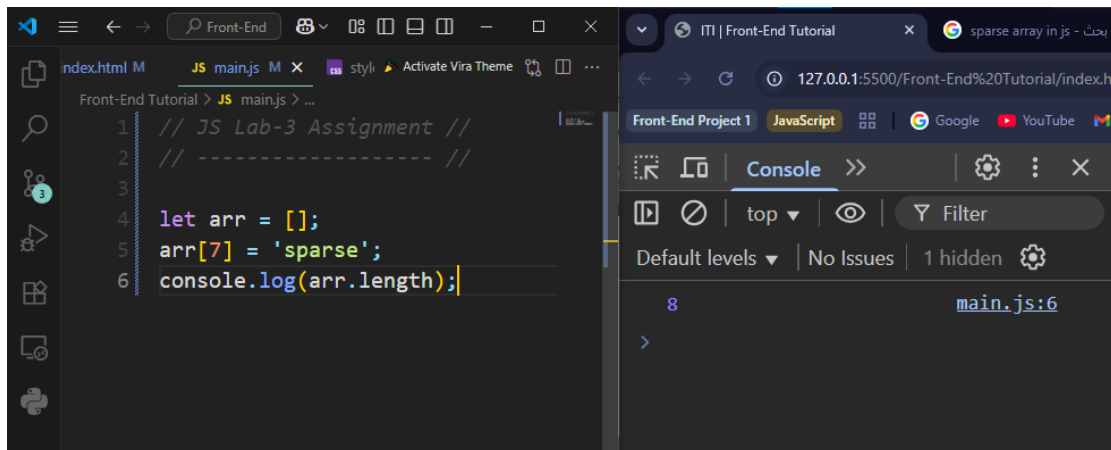
The screenshot shows a code editor with the following JavaScript code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 let arr = [1, 2, 3, 4, 5];
5
6 let sliced = arr.slice(1, 4);
7 console.log("After slice:", arr);
8 console.log("Sliced:", sliced);
9
10 let spliced = arr.splice(1, 3);
11 console.log("After splice:", arr);
12 console.log("Spliced:", spliced);
13
```

The browser console on the right shows the output:

```
After slice: (5) [1, 2, 3, 4, 5] main.js:7
Sliced: (3) [2, 3, 4] main.js:8
After splice: (2) [1, 5] main.js:11
Spliced: (3) [2, 3, 4] main.js:12
```

20. Create a sparse array by assigning index 7 on a fresh [] then log length.



The screenshot shows a code editor with the following JavaScript code in `main.js`:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 let arr = [];
5 arr[7] = 'sparse';
6 console.log(arr.length);
```

The browser console on the right shows the output of `console.log(arr.length)` as `8`, indicating that the array has a length of 8 because it has a value at index 7.

21. Write a function `compact(array)` that returns a new array without falsy values (manual loop, no filter).



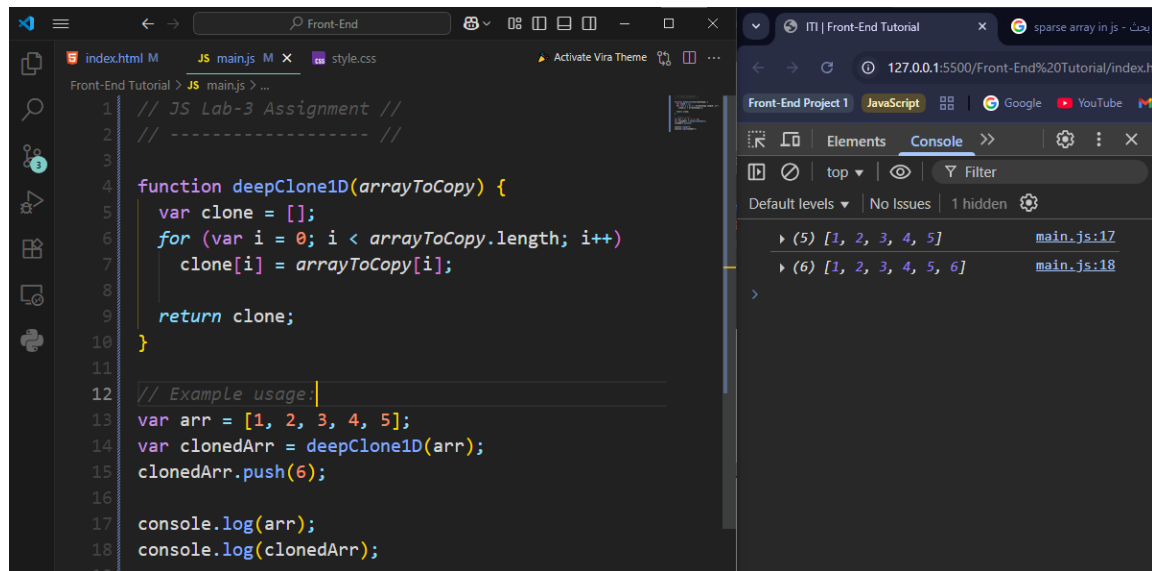
The screenshot shows a code editor with the following JavaScript code in `main.js`:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function compact(array) {
5   for (let i = 0; i < array.length; i++)
6     if (Boolean(array[i]) == false) {
7       array.splice(i, 1);
8       i--;
9     }
10  return array;
11 }
12
13 // AI Generated Examples for the compact function
14 console.log(compact([0, 1, false, 2, '', 3, null, undefined, NaN])); // [1, 2, 3]
15 console.log(compact(['hello', '', 'world', false, 42, 0])); // ['hello', 'world', 42]
16 console.log(compact([true, false, true, false])); // [true, true]
17 console.log(compact([null, undefined, NaN, '', 0])); // []
18 console.log(compact([1, 2, 3])); // [1, 2, 3]
```

The browser console on the right shows the output of the `compact` function for various inputs:

- `(3) [1, 2, 3]` (main.js:15)
- `(3) ['hello', 'world', 42]` (main.js:16)
- `(2) [true, true]` (main.js:17)
- `[]` (main.js:18)
- `(3) [1, 2, 3]` (main.js:19)

22. Implement a manual array clone `deepClone1D(a)` for a 1D array without using `slice/concat`.



The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
// JS Lab-3 Assignment //
// ----- //

function deepClone1D(arrayToCopy) {
  var clone = [];
  for (var i = 0; i < arrayToCopy.length; i++)
    clone[i] = arrayToCopy[i];
  return clone;
}

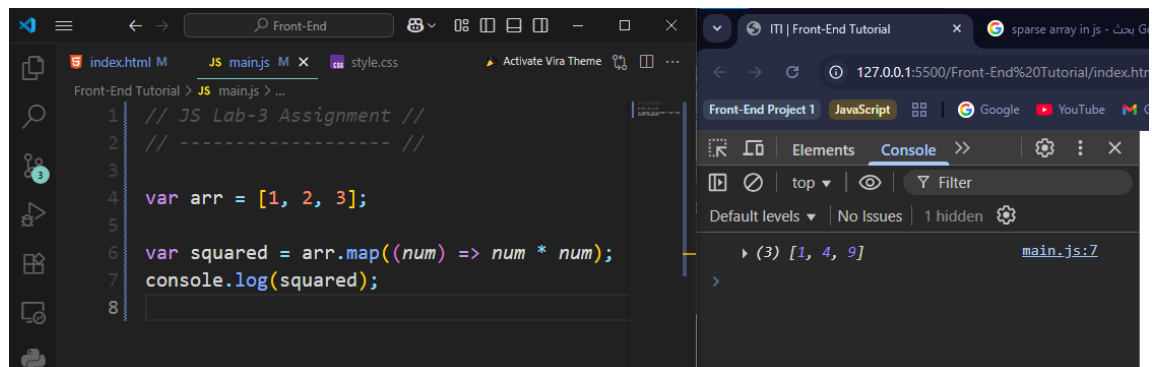
// Example usage:
var arr = [1, 2, 3, 4, 5];
var clonedArr = deepClone1D(arr);
clonedArr.push(6);

console.log(arr);
console.log(clonedArr);
```

The browser console on the right shows the output of the code:

```
(5) [1, 2, 3, 4, 5]    main.js:17
(6) [1, 2, 3, 4, 5, 6]  main.js:18
```

23. Map `[1,2,3]` to their squares using `map`.



The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
// JS Lab-3 Assignment //
// ----- //

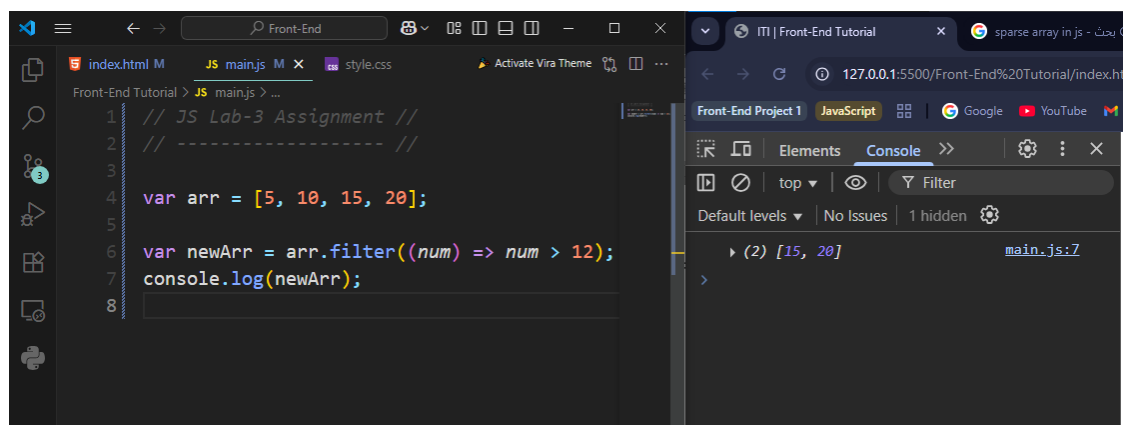
var arr = [1, 2, 3];

var squared = arr.map((num) => num * num);
console.log(squared);
```

The browser console on the right shows the output of the code:

```
(3) [1, 4, 9]    main.js:7
```

24. Filter `[5,10,15,20]` to keep values `>= 12`.



The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
// JS Lab-3 Assignment //
// ----- //

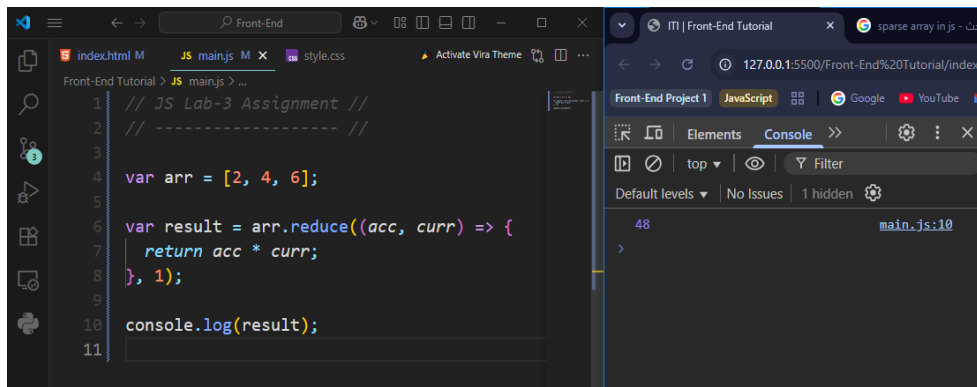
var arr = [5, 10, 15, 20];

var newArr = arr.filter((num) => num > 12);
console.log(newArr);
```

The browser console on the right shows the output of the code:

```
(2) [15, 20]    main.js:7
```

25. Reduce [2,4,6] to product.

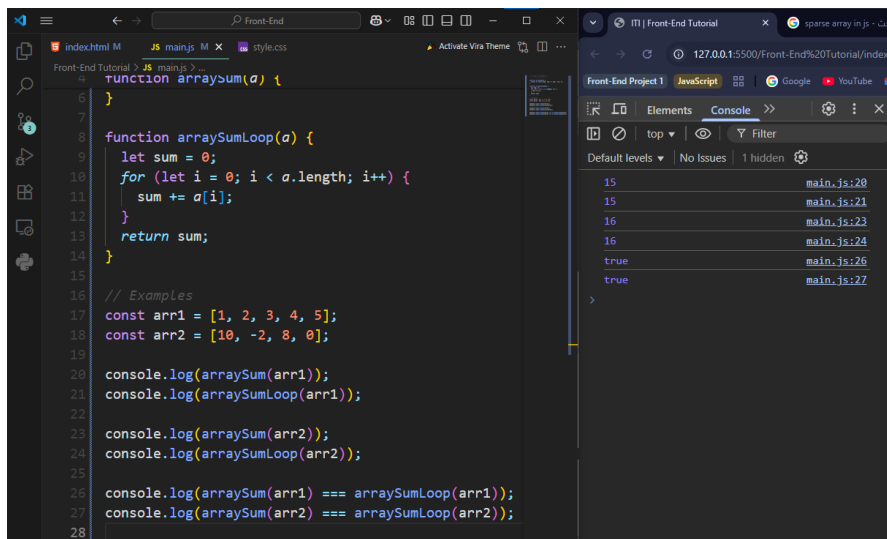


The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var arr = [2, 4, 6];
5
6 var result = arr.reduce((acc, curr) => {
7   return acc * curr;
8 }, 1);
9
10 console.log(result);
11
```

The browser console on the right shows the output of the code, which is the number 48, indicating the product of the array [2, 4, 6].

26. Implement arraySum(a) using reduce; then implement arraySumLoop(a) using a for loop. Confirm outputs equal.



The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 function arraySum(a) {
2   // Implement using reduce
3 }
4
5 function arraySumLoop(a) {
6   let sum = 0;
7   for (let i = 0; i < a.length; i++) {
8     sum += a[i];
9   }
10  return sum;
11 }
12
13 // Examples
14 const arr1 = [1, 2, 3, 4, 5];
15 const arr2 = [10, -2, 8, 0];
16
17 console.log(arraySum(arr1));
18 console.log(arraySumLoop(arr1));
19
20 console.log(arraySum(arr2));
21 console.log(arraySumLoop(arr2));
22
23 console.log(arraySum(arr1) === arraySumLoop(arr1));
24 console.log(arraySum(arr2) === arraySumLoop(arr2));
25
26 console.log(arraySum(arr1) === arraySumLoop(arr1));
27 console.log(arraySum(arr2) === arraySumLoop(arr2));
28
```

The browser console on the right shows the output of the code, which is the number 15, indicating the sum of the array [1, 2, 3, 4, 5].

27. Given ['Ali','Sara','Kareem'] produce ['A','S','K'] (first letters) without using map (use for loop).



The screenshot shows a VS Code editor with a file named `main.js` containing the following code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var arr = ["Ali", "Sara", "Kareem"];
5 var newArr = [];
6
7 for (var i = 0; i < arr.length; i++) {
8   newArr.push(arr[i][0]);
9 }
10
11 console.log(newArr);
12
```

The browser console on the right shows the output of the code, which is the array ['A', 'S', 'K'], indicating the first letters of the names in the array.

28. Implement unique(a) returning new array with duplicates removed (no ES6 Set). Complexity target: $O(n^2)$ acceptable; comment how to improve.

>> To Improve The Complexity, We Can Use Set Data Structure

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function unique(a) {
5   a = a.sort();
6   var newArr = [];
7
8   for (var i = 0; i < a.length; i++) {
9     if (a[i] !== a[i - 1]) {
10      newArr.push(a[i]);
11    }
12  }
13  return newArr;
14 }
15
16 var arr = [1, 2, 1, 2, 1, 3, 4, 5, 6];
17
18 console.log(unique(arr));
19
```

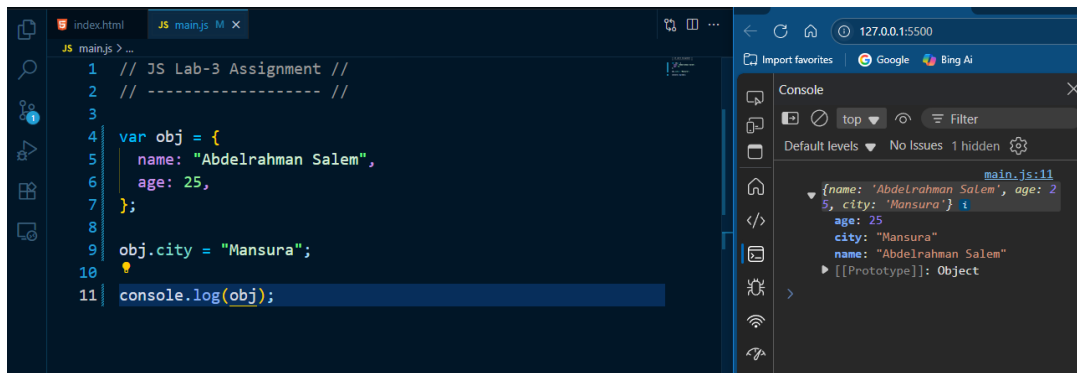
Console: (6) [1, 2, 3, 4, 5, 6] main.js:18

29. Flatten one level: flatten1([1,[2,3],[4],5]) => [1,2,3,4,5] without using concat inside a loop (manual pushing and detection of Array).

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function flatten(arr) {
5   let flattened = [];
6   for (let i = 0; i < arr.length; i++) {
7     const current = arr[i];
8     for (let j = 0; j < current.length; j++) {
9       flattened.push(current[j]);
10    }
11  }
12  return flattened;
13 }
14
15 console.log(flatten([1, 2, 3], [4, 5], [6]));
16
```

Console: (6) [1, 2, 3, 4, 5, 6] main.js:15

31. Create object person with name and age; add a new property city after creation.

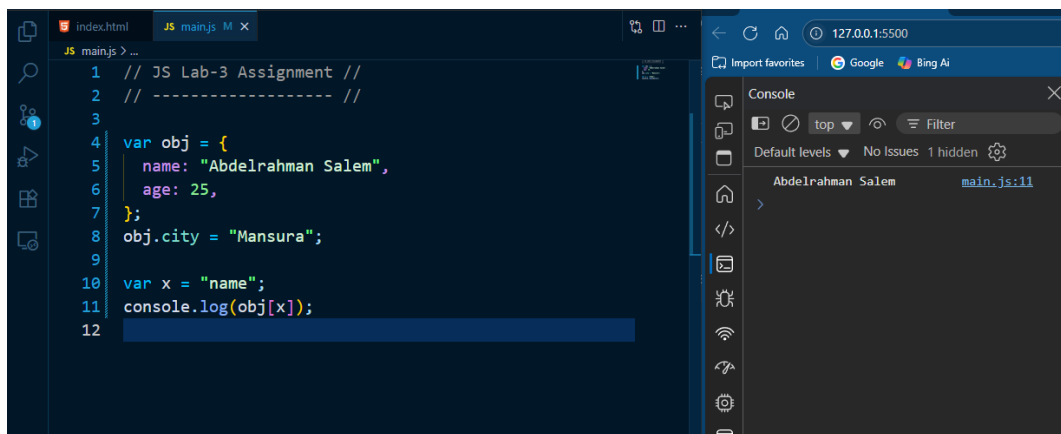


The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var obj = {
5   name: "Abdelrahman Salem",
6   age: 25,
7 };
8
9 obj.city = "Mansura";
10
11 console.log(obj);
```

The browser console on the right shows the output of `console.log(obj)` as an object: `{name: 'Abdelrahman Salem', age: 25, city: 'Mansura'}`.

32. Access a property via bracket notation with a dynamic key variable.

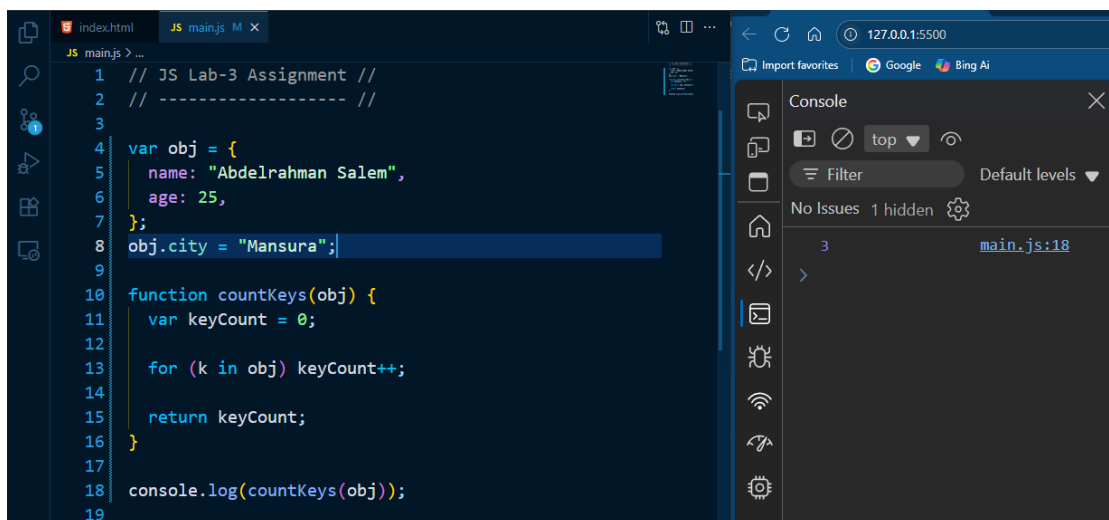


The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var obj = {
5   name: "Abdelrahman Salem",
6   age: 25,
7 };
8
9 obj.city = "Mansura";
10
11 var x = "name";
12 console.log(obj[x]);
```

The browser console on the right shows the output of `console.log(obj[x])` as the string `Abdelrahman Salem`.

33. Write function `countKeys(obj)` returning number of own enumerable properties (use `for-in`).

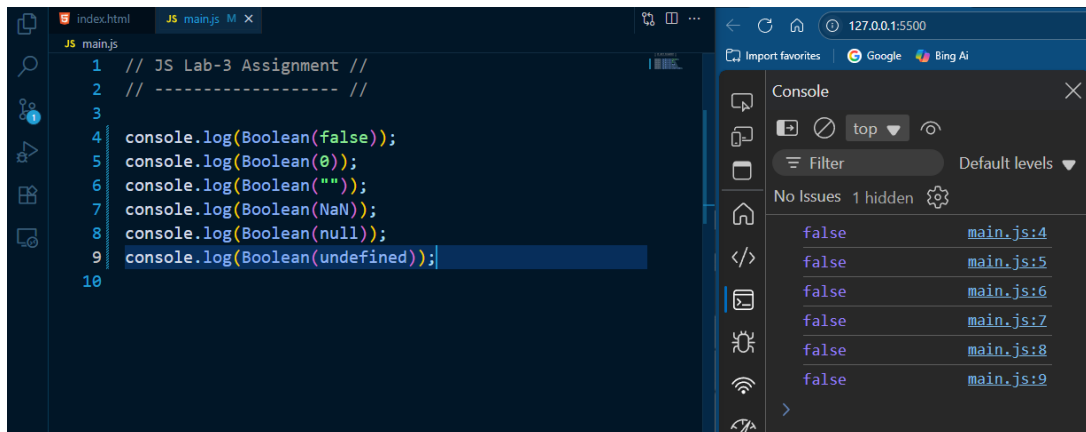


The screenshot shows a VS Code editor with a file named `main.js` containing the following JavaScript code:

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var obj = {
5   name: "Abdelrahman Salem",
6   age: 25,
7 };
8
9 obj.city = "Mansura";
10
11 function countKeys(obj) {
12   var keyCount = 0;
13   for (k in obj) keyCount++;
14   return keyCount;
15 }
16
17 console.log(countKeys(obj));
```

The browser console on the right shows the output of `console.log(countKeys(obj))` as the number `3`.

39. List (as comments) 5 different values that coerce to false in ES5.

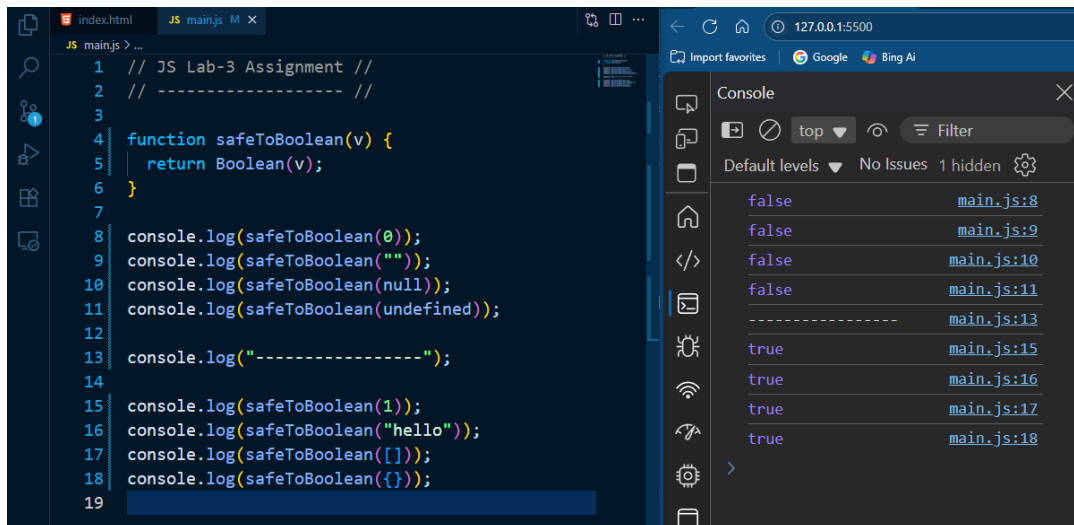


```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 console.log(Boolean(false));
5 console.log(Boolean(0));
6 console.log(Boolean(""));
7 console.log(Boolean(NaN));
8 console.log(Boolean(null));
9 console.log(Boolean(undefined));
10
```

The console shows five log entries, all displaying 'false':

- false main.js:4
- false main.js:5
- false main.js:6
- false main.js:7
- false main.js:8
- false main.js:9

40. safeToBoolean(v): return true only if v is strictly true, 'true', 1, or '1'; else false.

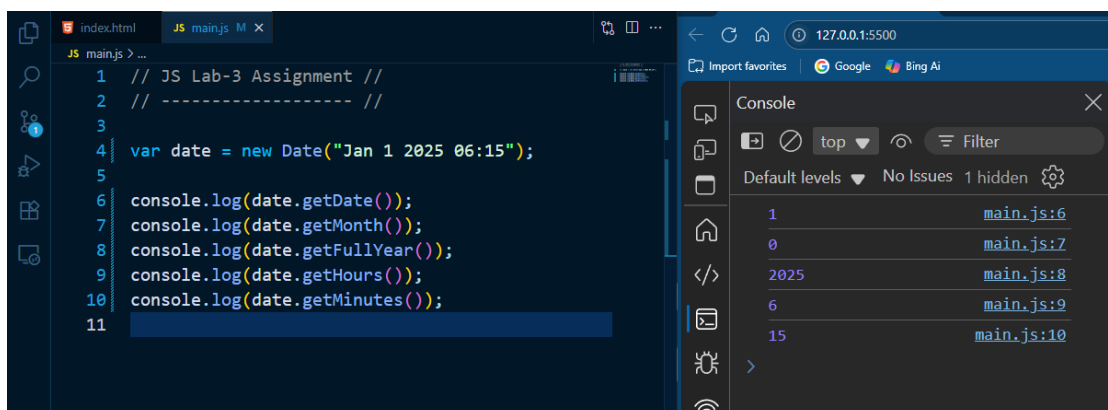


```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function safeToBoolean(v) {
5   return Boolean(v);
6 }
7
8 console.log(safeToBoolean(0));
9 console.log(safeToBoolean(""));
10 console.log(safeToBoolean(null));
11 console.log(safeToBoolean(undefined));
12
13 console.log("-----");
14
15 console.log(safeToBoolean(1));
16 console.log(safeToBoolean("hello"));
17 console.log(safeToBoolean([]));
18 console.log(safeToBoolean({}));
19
```

The console shows log entries for the safeToBoolean function:

- false main.js:8
- false main.js:9
- false main.js:10
- false main.js:11
- main.js:13
- true main.js:15
- true main.js:16
- true main.js:17
- true main.js:18

41. Create a Date for Jan 1, 2025 00:00 local.

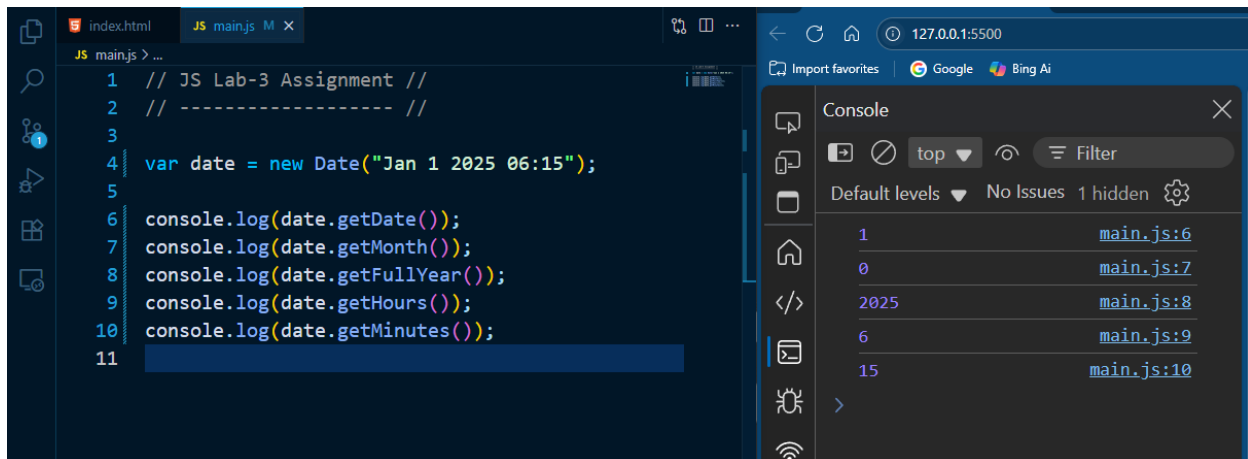


```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var date = new Date("Jan 1 2025 06:15");
5
6 console.log(date.getDate());
7 console.log(date.getMonth());
8 console.log(date.getFullYear());
9 console.log(date.getHours());
10 console.log(date.getMinutes());
11
```

The console shows log entries for the Date object:

- 1 main.js:6
- 0 main.js:7
- 2025 main.js:8
- 6 main.js:9
- 15 main.js:10

42. Get the current year from new Date().



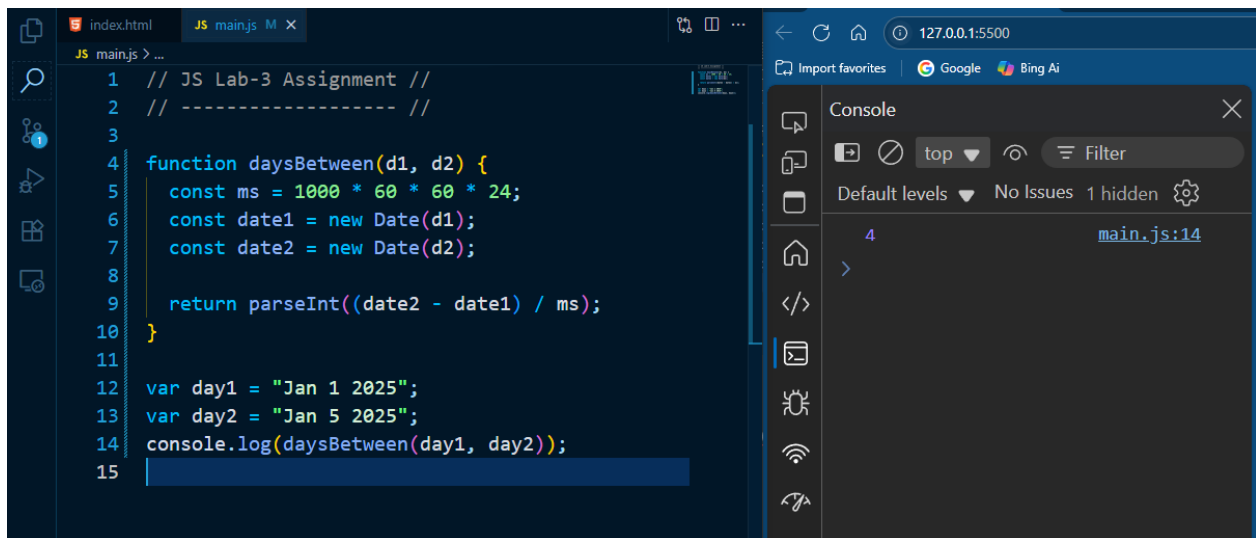
The screenshot shows a web browser with a JavaScript console. The console displays the following logs:

```
1 // JS Lab-3 Assignment //  
2 // ----- //  
3  
4 var date = new Date("Jan 1 2025 06:15");  
5  
6 console.log(date.getDate());  
7 console.log(date.getMonth());  
8 console.log(date.getFullYear());  
9 console.log(date.getHours());  
10 console.log(date.getMinutes());  
11
```

The console output shows the following values:

- 1 (main.js:6)
- 0 (main.js:7)
- 2025 (main.js:8)
- 6 (main.js:9)
- 15 (main.js:10)

43. Write function daysBetween(d1, d2) returning whole day difference (ignore DST intricacies; ms/(1000*60*60*24)).



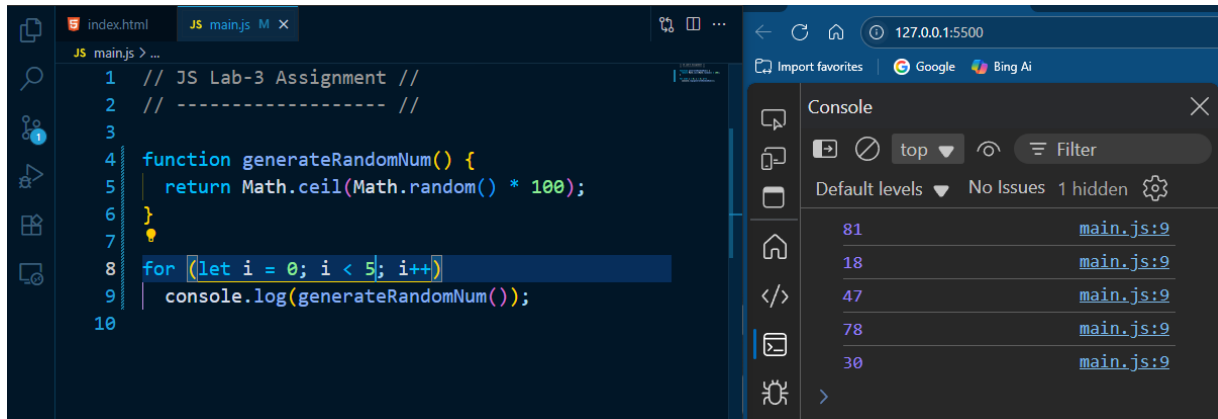
The screenshot shows a web browser with a JavaScript console. The console displays the following logs:

```
1 // JS Lab-3 Assignment //  
2 // ----- //  
3  
4 function daysBetween(d1, d2) {  
5   const ms = 1000 * 60 * 60 * 24;  
6   const date1 = new Date(d1);  
7   const date2 = new Date(d2);  
8  
9   return parseInt((date2 - date1) / ms);  
10 }  
11  
12 var day1 = "Jan 1 2025";  
13 var day2 = "Jan 5 2025";  
14 console.log(daysBetween(day1, day2));  
15
```

The console output shows the following value:

- 4 (main.js:14)

44. Generate a random integer 1..100.

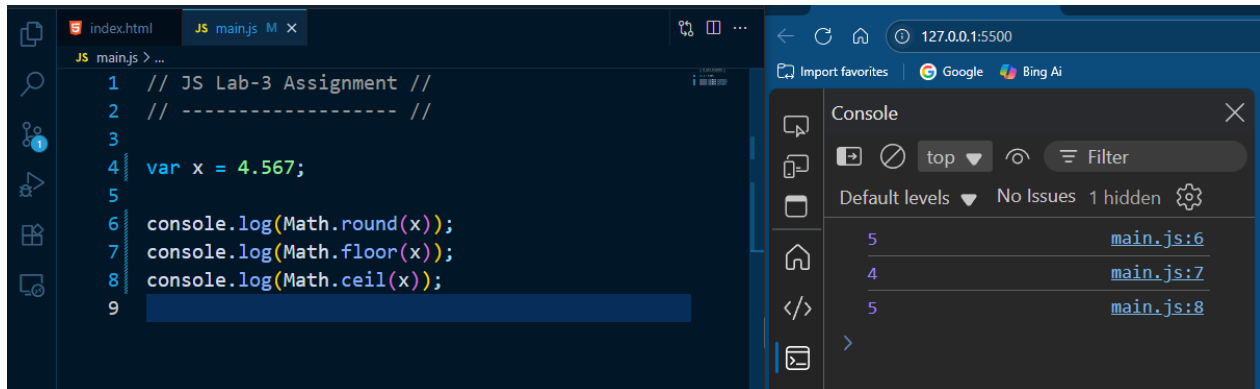


```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function generateRandomNum() {
5   return Math.ceil(Math.random() * 100);
6 }
7
8 for (let i = 0; i < 5; i++) {
9   console.log(generateRandomNum());
10 }
```

Console

Message	File
81	main.js:9
18	main.js:9
47	main.js:9
78	main.js:9
30	main.js:9

45. Round 4.567 to nearest integer, round down, and round up (three results).



```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 var x = 4.567;
5
6 console.log(Math.round(x));
7 console.log(Math.floor(x));
8 console.log(Math.ceil(x));
9
```

Console

Message	File
5	main.js:6
4	main.js:7
5	main.js:8

46. randomIntArray(n, min, max): return array of length n with random ints (loop + push).

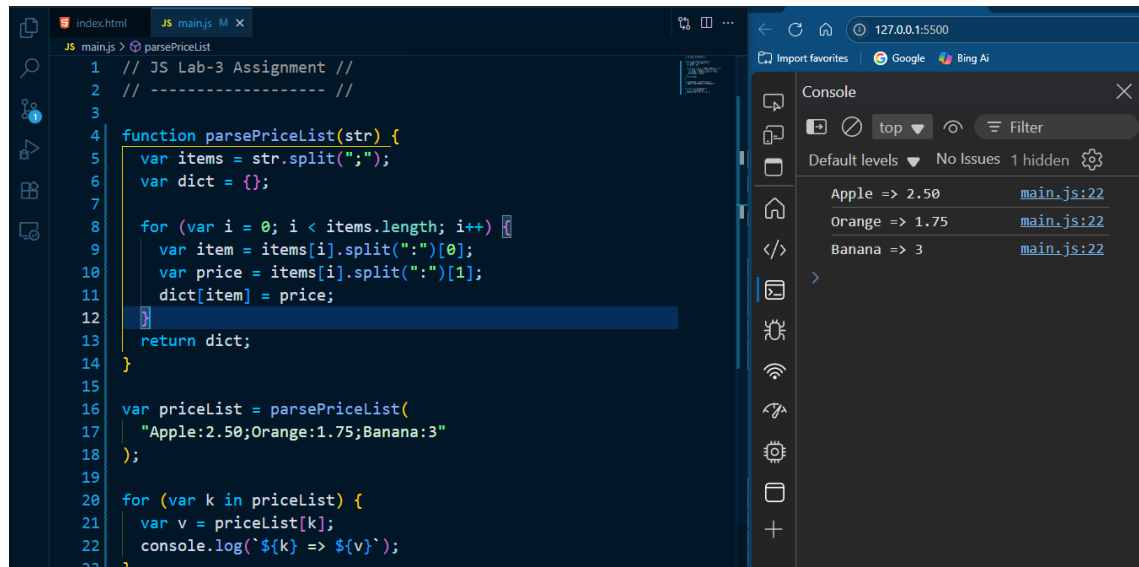


```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function randomIntArray(n, min, max) {
5   var arr = [];
6   for (let i = 0; i < n; i++) {
7     arr.push(Math.ceil(Math.random() * (max - min)) + min);
8   }
9   return arr;
10 }
11
12 console.log(randomIntArray(10, 1, 100));
13
```

Console

Message	File
(10) [92, 16, 75, 8, 71, 90, 19, 99, 89, 41]	main.js:12

46. `parsePriceList(str)`: Given "Apple:2.50;Orange:1.75;Banana:3" return object {Apple:2.5, Orange:1.75, Banana:3} (strings to numbers).



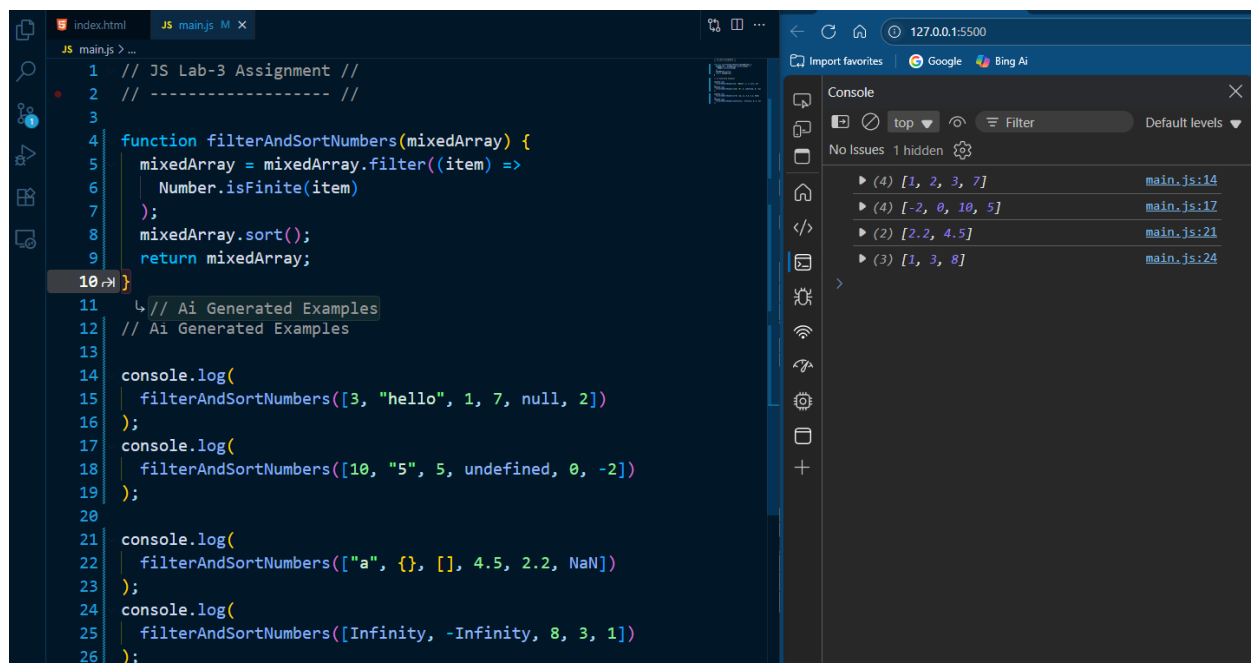
The screenshot shows a web browser with a JavaScript file named `main.js` open in the editor. The function `parsePriceList` is defined to take a string `str` and return an object. The string is split by semicolons into an array of items. Each item is then split by a colon to extract the item name and its price, which are stored in a dictionary. The function is called with the string "Apple:2.50;Orange:1.75;Banana:3", and the console shows the resulting object: {Apple => 2.50, Orange => 1.75, Banana => 3}.

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function parsePriceList(str) {
5   var items = str.split(";");
6   var dict = {};
7
8   for (var i = 0; i < items.length; i++) {
9     var item = items[i].split(":")[0];
10    var price = items[i].split(":")[1];
11    dict[item] = price;
12  }
13  return dict;
14 }
15
16 var priceList = parsePriceList(
17   "Apple:2.50;Orange:1.75;Banana:3"
18 );
19
20 for (var k in priceList) {
21   var v = priceList[k];
22   console.log(`${k} => ${v}`);
23 }
```

Console output:

```
Apple => 2.50
Orange => 1.75
Banana => 3
```

47. `filterAndSortNumbers(mixedArray)`: keep only finite numbers then sort ascending (provide sample input and output). Use a numeric compare fn.



The screenshot shows a web browser with a JavaScript file named `main.js` open in the editor. The function `filterAndSortNumbers` is defined to take a mixed array and return a new array containing only finite numbers, sorted in ascending order. The function uses `Number.isFinite` to filter the array and `sort` to sort it. The function is called with several sample inputs, and the console shows the resulting arrays: [1, 2, 3, 7], [-2, 0, 10, 5], [2.2, 4.5], and [1, 3, 8].

```
1 // JS Lab-3 Assignment //
2 // ----- //
3
4 function filterAndSortNumbers(mixedArray) {
5   mixedArray = mixedArray.filter((item) =>
6     Number.isFinite(item)
7   );
8   mixedArray.sort();
9   return mixedArray;
10 }
11
12 // Ai Generated Examples
13
14 console.log(
15   filterAndSortNumbers([3, "hello", 1, 7, null, 2])
16 );
17 console.log(
18   filterAndSortNumbers([10, "5", 5, undefined, 0, -2])
19 );
20
21 console.log(
22   filterAndSortNumbers(["a", {}, [], 4.5, 2.2, NaN])
23 );
24 console.log(
25   filterAndSortNumbers([Infinity, -Infinity, 8, 3, 1])
26 );
```

Console output:

```
[1, 2, 3, 7]
[-2, 0, 10, 5]
[2.2, 4.5]
[1, 3, 8]
```