DSA in JavaScript

Pattern to be matched. Input will be an integer

Expected Output:-

For n=3

1 2 3

7 8 9

4 5 6

For n=4

1 2 3 4

9 10 11 12

13 14 15 16

5 6 7 8

For n=5

1 2 3 4 5

11 12 13 14 15

21 22 23 24 25

16 17 18 19 20

6 7 8 9 10

```
function pattern(size){
    console.log("length is " +size);
    let matrix=[];
    //creating matrix filled with zeroes.
    for(let i=0;i<size;i++){
        matrix[i]=[];
        for(let j=0;j<size;j++){
            matrix[i][j]=0;
        }
    }
    let n = size;
```

```
let num =1;// to print numbers
let top=0;
let bottom = size-1;
while(true){
    for(let j=0;j<size;j++){
        matrix[top][j] = num;
        // console.log(matrix[top][j]+ "top");
        num++;
    }
    top++;
    n--;// reducing no. of rows to be filled
    if(n == 0){// to exit when every row is filled
        break;
    }
    for(let k=0;k<size;k++){
        matrix[bottom][k] = num;
        // console.log(matrix[bottom][k]+ "bottom");
        num++;
    }
    bottom--;
    n--;
    if(n == 0){
        break;
    }
}
for(let i=0;i<size;i++){
    for(let j=0;j<size;j++){
        process.stdout.write(matrix[i][j]+" ");
        /*console.log() takes always new line. so use process.stdout.write */
```

```
        }
        console.log(); //to add new line after each row
    }

}
pattern(5); // pass size of matrix as argument
```

Arrays: -

An array in JavaScript is a type of object used to store multiple values in a single variable.

It can hold mixed data types.

Creating an Array: -

Using literal (Recommended)

```
let fruits=[“Apple”,”Banana”,”Orange”];
```

using constructor (new) keyword

```
let cars= new Array(“Toyota”,”Tesla”];
```

Also ,

```
 const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

Accessing array elements: -

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
console.log(cars[1]);
```

last element can be accessed by

```
console.log(cars[cars.length-1]);
```

Modification can be like

```
Cars[2] = “Ferrari”;
```

Converting an Array to a String

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
```

fruits.toString();

//Result : Banana,Orange,Apple,Mango

The typeof operator return object for arrays.

Note: -

You can have variables of different types in the same Array. You can have objects in an Array. You can have functions in an Array. You can have arrays in an Array.

myArray[0] = Date.now;

myArray[1] = myFunction;

myArray[2] = cars;

Array Properties and Methods: -

length:-

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

let length = fruits.length; //4
```

Looping Array Elements: -

for loop:-

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

for(let i=0;i < fruits.length;i++){
   console.log(fruits[i]);
}
```

Array.forEach() function:-

```
fruits.forEach(fruit => console.log(fruit));
```

for of loop:-

```
for(let fruit of fruits){
  console.log(fruit);
}
```

Adding and Removing Array Elements: -

push(): -

```
const fruits = ["Banana", "Orange", "Apple"];

fruits.push("Lemon");  // Adds a new element (Lemon) to fruits
```

New element can also be added to an array using the length property.

fruits[fruits.length] = "Lemon";

undefined "holes" in an array.

const fruits = ["Banana", "Orange", "Apple"];
fruits[6] = "Lemon"; // Creates undefined "holes" in fruits.

pop(): -

Fruits.pop(); //remove from end.

unshift(): -

fruits.unshift("strawberry"); //add at start

shift(): -

fruits.shift(); // remove from start

Associative Arrays: -

Arrays with named indexes are called associative arrays. JavaScript does not support arrays with named indexes.

In JavaScript, arrays use numbered indexes while objects use named indexes.

isArray(): -

Array.isArray(fruits); // returns true if it is an array

Using typeof , we get object so use this to determine whether it is an array.

at(): -

let fruit = fruits.at(2); // returns element at index 2

join(): -

It behaves just like toString(), but in addition you can specify the separator.

const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.join(" * "); // Banana * Orange * Apple * Mango

delete(): -

delete fruits[0]; // deletes first element . it can lead to undefined holes.

JavaScript Array concat(): -

const myGirls = ["Cecilie", "Lone"];

const myBoys = ["Emil", "Tobias", "Linus"];

const myChildren = myGirls.concat(myBoys);

// Cecilie,Lone,Emil,Tobias,Linus

The concat() method does not change the existing arrays. It always returns a new array. It can take any number of array arguments.

copyWithin(): -

The copyWithin() method copies array elements to another position in an array.

Copy to index 2, all elements from index 0:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.copyWithin(2, 0);
```

// Banana,Orange,Banana,Orange

Copy to index 2, the elements from index 0 to 2.

```
const fruits = ["Banana", "Orange", "Apple", "Mango", "Kiwi"];
fruits.copyWithin(2, 0, 2);
```

Banana,Orange,Banana,Orange,Kiwi,Papaya

Flattening an Array: -

Flattening an array is the process of reducing the dimensionality of an array.

flat(): -

The flat() method creates a new array with sub-array elements concatenated to a specified depth.

```
const myArr = [[1,2],[3,4],[5,6]];
const newArr = myArr.flat(); // 1,2,3,4,5,6
```

Splicing and Slicing Arrays: -

The splice() method can be used to add new items to an array.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

// Banana,Orange,Lemon,Kiwi,Apple,Mango

The first parameter (2) defines the position where new elements should be added (spliced in).

The second parameter (0) defines how many elements should be removed.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.

The splice() method returns an array with the deleted items.

fruits.splice(0, 1); // used to remove elements

toSpliced(): -

splice an array without altering the original array.

slice(): -

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1); // Orange,Lemon,Apple,Mango
```

The slice() method can also take two arguments like slice(1, 3).

The method then selects elements from the start argument, and up to (but not including) the end argument.