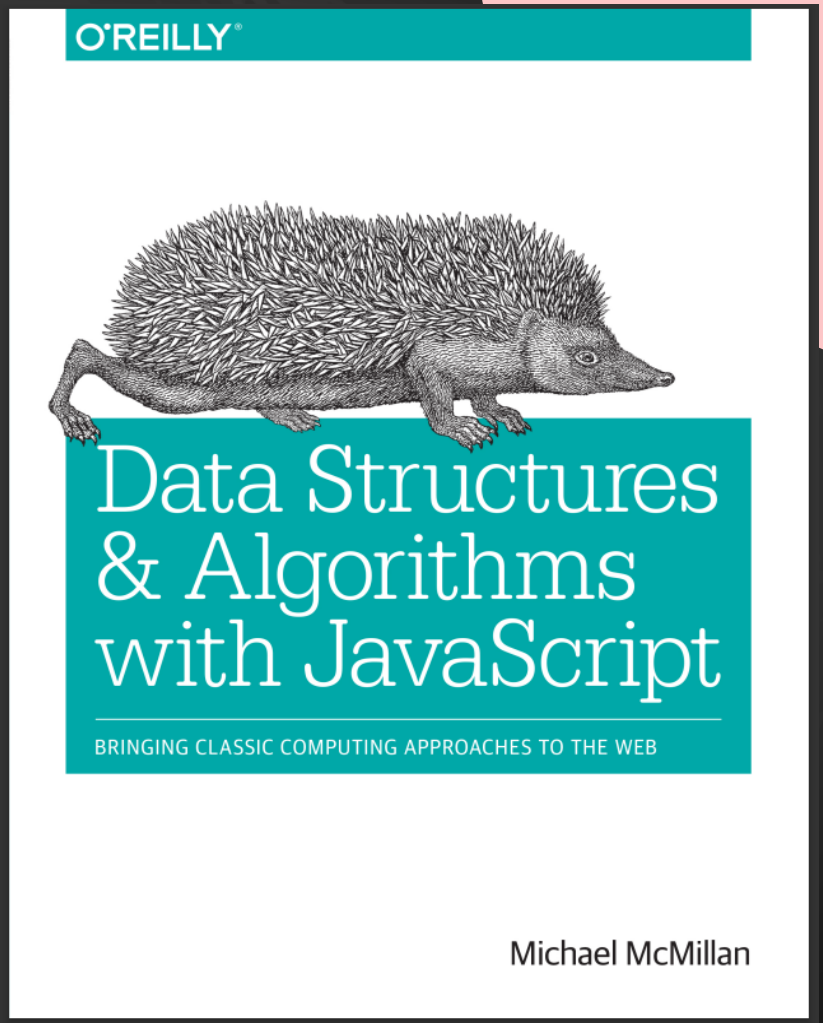
Chapter 2 (Arrays-Part 2)
Data Structures & Algorithms With javascript



Two-Dimensional and Multidimensional Arrays :-

- JavaScript arrays are only one-dimensional, but you can create multidimensional arrays by creating arrays of arrays.
- A two-dimensional array, also known as a 2D array, is a collection of data elements arranged in a grid-like structure with rows and columns. Each element in the array is referred to as a cell and can be accessed by its row and column indices/indexes.
- In this section we'll describe how to create two-dimensional arrays in JavaScript.

Creating Two-Dimensional Arrays :-

There are two options for creating a multi-dimensional array.

01

- you can create the array manually with the array literal notation, which uses square brackets [ ] to wrap a list of elements separated by commas.

```
1  var grades = [[89, 77, 78],[76, 82, 81],[91, 94, 89]];
2  console.log(grades[2][2]);
```

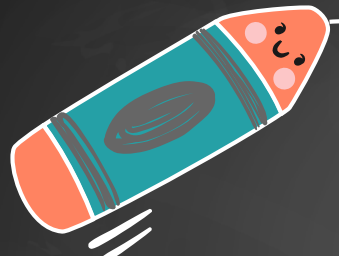
Console ×

89

02

You can also use a nested loop.

Creating Two-Dimensional Arrays :-



1- To create a two-dimensional array in JavaScript by use a nested loop , we have to create an array and then make each element of the array an array as well. At the very least, we need to know the number of rows we want the array to contain. With that information, we can create a two*dimensional array with n rows and one column.

```
Array.matrix = function (numrows, numcols, initial) {
  var arr = [];
  for (var i = 0; i < numrows; ++i) {
    var columns = [];
    for (var j = 0; j < numcols; ++j) {
      columns[j] = initial;
    }
    arr[i] = columns;
    //  console.log('coulmns ',columns) ;
  }
  return arr;
}
var nums = Array.matrix(5, 5, 0);
console.log(nums[1][1]);
var names = Array.matrix(3, 3, "");
names[1][2] = "Joe";
console.log(names[1][2]);
```

Console ×

```
0
Joe
```

Processing Two-Dimensional Array Elements :-

```
var grades = [[89, 77, 78], [76, 82, 81], [91, 94, 89]];
var total = 0;
var average = 0.0;
for (var row = 0; row < grades.length; ++row) {
  for (var col = 0; col < grades[row].length; ++col) {
    total += grades[row][col];
  }
  average = total / grades[row].length;
  console.log("Student " + parseInt(row + 1) + " average: " +
    average.toFixed(2));
  total = 0;
  average = 0.0;
}
```

Console ×

```
Student 1 average: 81.33
Student 2 average: 79.67
Student 3 average: 91.33
```

Processing Two-Dimensional Array Elements :-

Notice :-

To perform a row-wise computation, we simply have to flip the for loops so that the outer loop controls the columns and the inner loop controls the rows. Here is the calculation for each test:

```
1   var grades = [[89, 77, 78], [76, 82, 81], [91, 94, 89]];
2   var total = 0;
3   var average = 0.0;
4   for (var col = 0; col < grades.length; ++col) {
5     for (var row = 0; row < grades[col].length; ++row) {
6       total += grades[row][col];
7     }
8     average = total / grades[col].length;
9     console.log("Test " + parseInt(col + 1) + " average: " +
10      average.toFixed(2));
11    total = 0;
12    average = 0.0;
13  }
```

Console ×

```
Test 1 average: 85.33
Test 2 average: 84.33
Test 3 average: 82.67
```

Jagged Arrays

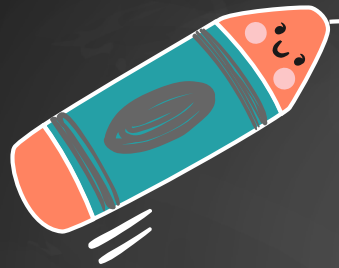
Jagged Arrays

is an array where the rows in the array may have a different number of elements. One row may have three elements, while another row may have five elements, while yet another row may have just one element. imagine the grades array where students have an unequal number of grades recorded. We can still compute the correct average for each student without changing the program at all:

```
1   var grades = [[89, 77],[76, 82, 81],[91, 94, 89, 99]];
2   var total = 0;
3   var average = 0.0;
4   for (var row = 0; row < grades.length; ++row) {
5    for (var col = 0; col < grades[row].length; ++col) {
6    total += grades[row][col];
7    }
8    average = total / grades[row].length;
9    console.log("Student " + parseInt(row+1) + " average: " +
10   average.toFixed(2));
11   total = 0;
12   average = 0.0;
13  }
```

Console ×

```
Student 1 average: 83.00
Student 2 average: 79.67
Student 3 average: 93.25
```



Arrays of Objects

All of the examples in this chapter have consisted of arrays whose elements have been primitive data types, such as numbers and strings. Arrays can also consist of objects, and all the functions and properties of arrays work with objects.

Arrays in Objects

All of the examples in this chapter have consisted of arrays whose elements have been primitive data types, such as numbers and strings. Arrays can also consist of objects, and all the functions and properties of arrays work with objects.