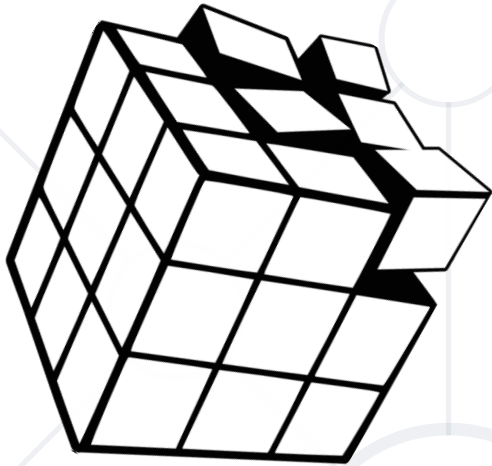


# Multidimensional Arrays

## Processing Matrices and Jagged Arrays



**SoftUni Team**  
**Technical Trainers**



**SoftUni**



**Software University**

<https://about.softuni.bg/>

sli.do

**#csharp-advanced**

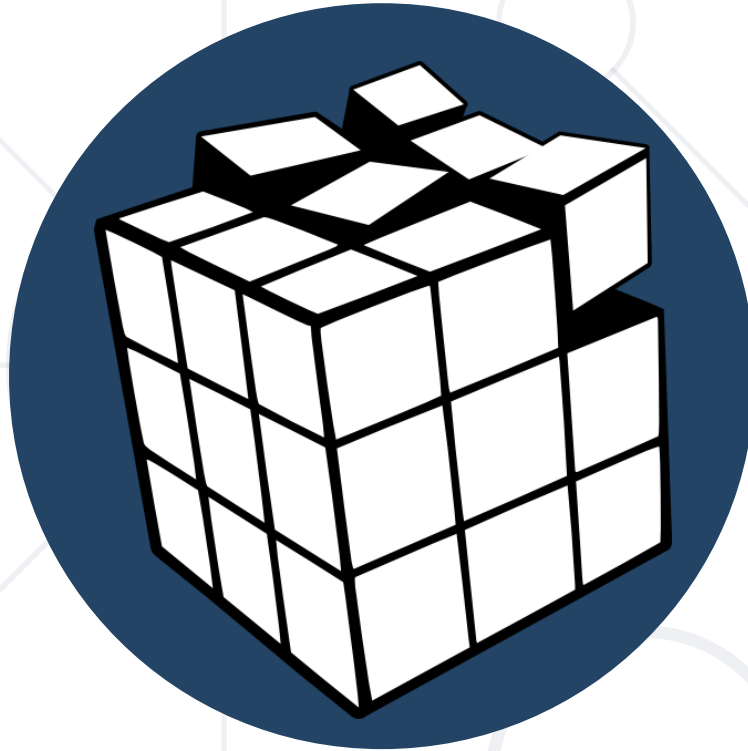
## 1. Multidimensional Arrays

- Creating Matrices and Multidimensional Arrays
- Accessing Their Elements
- Reading and Printing

## 2. Jagged Arrays (Arrays of Arrays)

- Creating a Jagger Array
- Accessing Their Elements
- Reading and Printing






# Multidimensional Arrays

Definition and Usage

# What is a Multidimensional Array?

- **Array** is a systematic arrangement of similar objects
- **Multidimensional arrays** have more than one dimension
  - The most used multidimensional arrays are the 2-dimensional, also called **matrices**



R	COLS				
O	[0, 0]	[0, 1]	[0, 2]	[0, 3]	[0, 4]
W	[1, 0]	[1, 1]	[1, 2]	[1, 3]	[1, 4]
S	[2, 0]	[2, 1]	[2, 2]	[2, 3]	[2, 4]

Col Index

Row Index

# Creating Multidimensional Arrays

- Creating a multidimensional array in C#
  - Use the **new** keyword
  - Must specify the size of each dimension

```
int[,] intMatrix = new int[3, 4];  
float[,] floatMatrix = new float[8, 2];  
string[, ,] stringCube = new string[5, 5, 5];
```



- Initializing with values:

```
int[,] matrix = {  
    {1, 2, 3, 4}, // row 0 values  
    {5, 6, 7, 8} // row 1 values  
};
```

- Two-dimensional arrays represent **rows with values**
- The **rows** represent the first dimension and the **columns** – the second (**the one inside the first**)

- Accessing N-dimensional array element:

```
nDimensionalArray[index1, ... , indexn]
```

- Getting element value:

```
int[,] array = {{10, 20, 30}, {40, 50, 60}};  
int element11 = array[1, 0]; // element10 = 40
```

0	1	2	
10	20	30	row 0
40	50	60	row 1

- Setting element value:

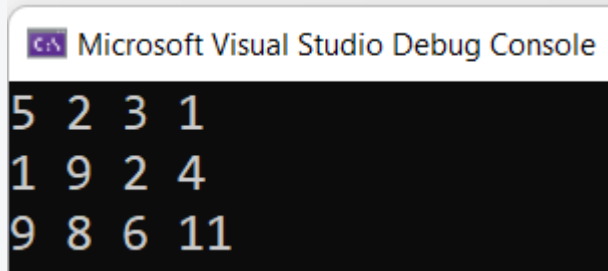
```
int[,] array = new int[3, 4];  
for (int row = 0; row < array.GetLength(0); row++)  
    for (int col = 0; col < array.GetLength(1); col++)  
        array[row, col] = row + col;
```

Returns the size  
of the dimension



# Printing a Matrix – Example

```
int[,] matrix =  
    { { 5, 2, 3, 1 },  
      { 1, 9, 2, 4 },  
      { 9, 8, 6, 11 } };  
for (int row = 0; row < matrix.GetLength(0); row++)  
{  
    for (int col = 0; col < matrix.GetLength(1); col++)  
    {  
        Console.Write("{0} ", matrix[row, col]);  
    }  
  
    Console.WriteLine();  
}
```




Microsoft Visual Studio Debug Console

```
5 2 3 1  
1 9 2 4  
9 8 6 11
```

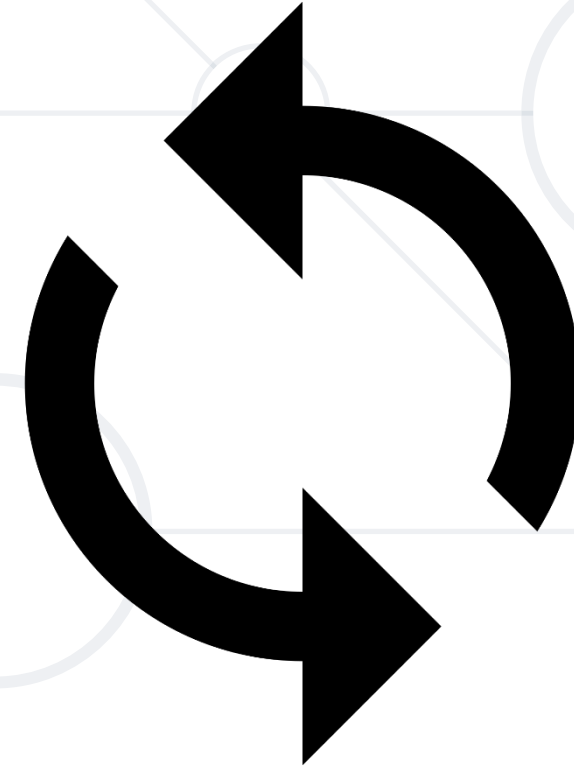
# Printing Matrix – Example

- Foreach iterates through all the elements in the matrix

```
int[,] matrix = {  
    { 5, 2, 3, 1 },  
    { 1, 9, 2, 4 },  
    { 9, 8, 6, 9 }  
};  
  
foreach (int element in matrix)  
{  
    Console.WriteLine(element + " ");  
}
```

 Microsoft Visual Studio Debug Console

5 2 3 1 1 9 2 4 9 8 6 9



# Problem: Sum Matrix Elements

- Read a matrix from the console
- Print the number of rows
- Print the number of columns
- Print the **sum of all numbers** in the matrix

3,	6				
7,	1,	3,	3,	2,	1
1,	3,	9,	8,	5,	6
4,	6,	7,	9,	1,	0



3
6
76

3,	4		
1,	2,	3,	1
1,	2,	2,	4
2,	2,	2,	2



3
4
24

Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1452#0>

# Solution: Sum Matrix Elements

```
int[] sizes = Console.ReadLine().Split(", ")
    .Select(int.Parse).ToArray();
int[,] matrix = new int[sizes[0], sizes[1]];
for (int row = 0; row < matrix.GetLength(0); row++) {
    int[] colElements = Console.ReadLine().Split(", ")
        .Select(int.Parse).ToArray();
    for (int col = 0; col < matrix.GetLength(1); col++)
        matrix[row, col] = colElements[col];
}
```

Gets length of 0th dimension (rows)

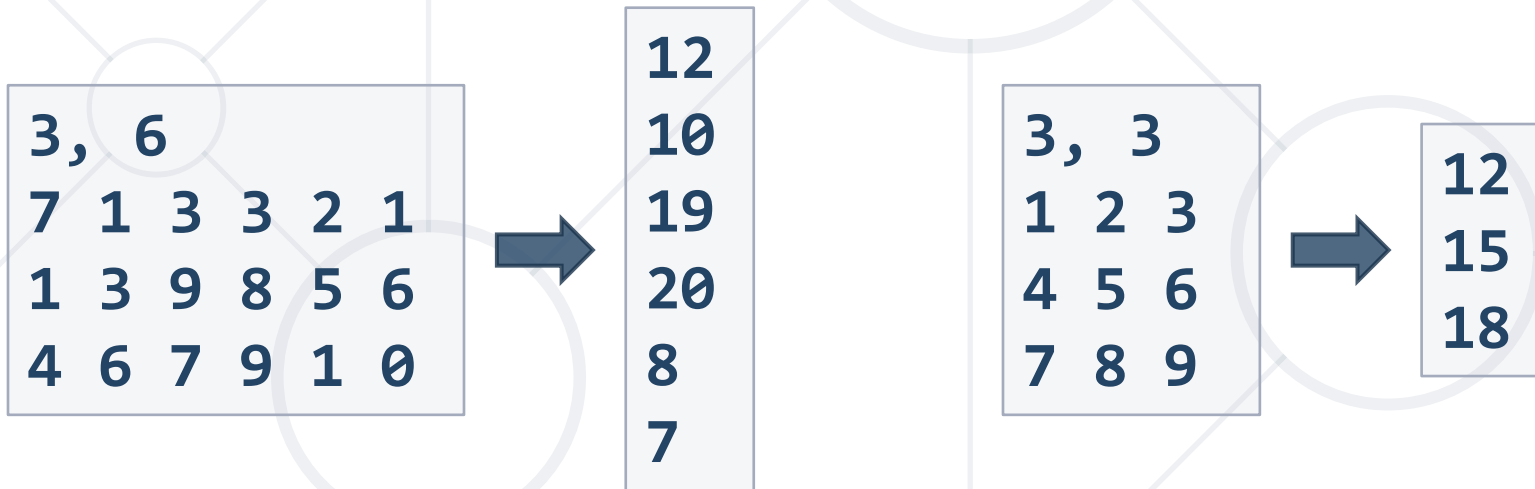
Gets length of 1st dimension (cols)

# Solution: Sum Matrix Elements

```
int sum = 0;
for (int row = 0; row < matrix.GetLength(0); row++)
{
    for (int col = 0; col < matrix.GetLength(1); col++)
        sum += matrix[row, col];
}
Console.WriteLine(matrix.GetLength(0));
Console.WriteLine(matrix.GetLength(1));
Console.WriteLine(sum);
```

# Problem: Sum Matrix Columns

- Read matrix sizes
- Read a matrix from the console
- Print the **sum of all numbers** in matrix columns



Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1452#1>

# Solution: Sum Matrix Columns

```
var sizes = Console.ReadLine()
    .Split(", ").Select(int.Parse).ToArray();
int[,] matrix = new int[sizes[0], sizes[1]];
for (int r = 0; r < matrix.GetLength(0); r++) {
    var col = Console.ReadLine().Split().Select(int.Parse).ToArray();
    for (int c = 0; c < matrix.GetLength(1); c++) {
        matrix[r, c] = col[c];
    }
}
```

# Solution: Sum Matrix Columns

```
for (int c = 0; c < matrix.GetLength(1); c++) {  
    int sum = 0;  
    for (int r = 0; r < matrix.GetLength(0); r++) {  
        sum += matrix[r, c];  
    }  
    Console.WriteLine(sum);  
}
```



# Problem: Square with Maximum Sum

- Find **2x2 square** with max sum in given matrix
  - Read matrix from the console
  - Find **biggest sum** of 2x2 submatrix
  - Print the result as a **new matrix**, followed by **the sum**



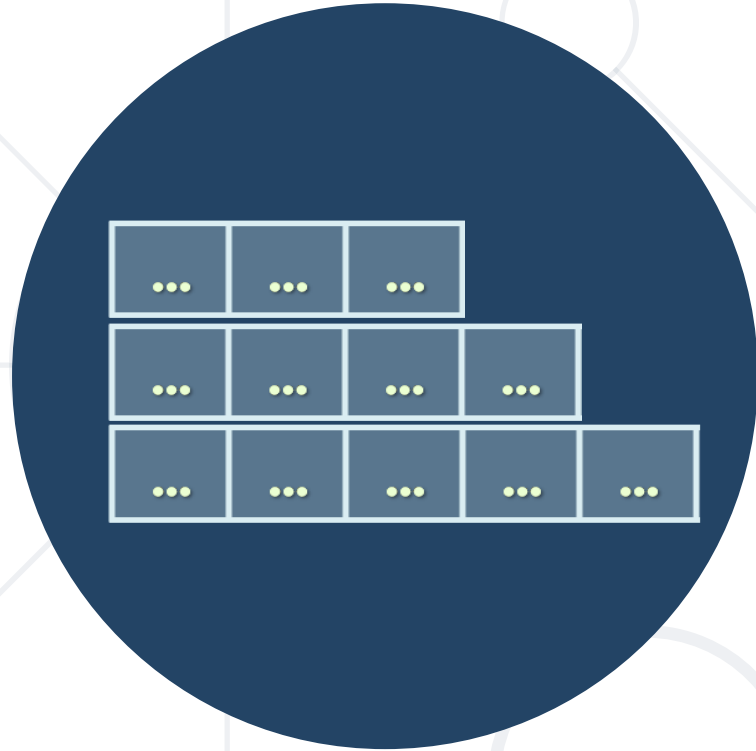
Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1452#4>

# Solution: Square with Maximum Sum

```
// TODO: Read the input from the console
for (int row = 0; row < matrix.GetLength(0) - 1; row++) {
    for (int col = 0; col < matrix.GetLength(1) - 1; col++) {
        var newSquareSum = matrix[row, col] +
            matrix[row + 1, col] +
            matrix[row, col + 1] +
            matrix[row + 1, col + 1];

        // TODO: Check if the sum is bigger
        // → remember the best sum, row and col
    }
}

// TODO: Print the square with the max sum
```



# Jagged Arrays

## Definition and Usage

# What is Jagged Array

- Jagged arrays are multidimensional arrays
  - But each dimension may have a different size
  - A jagged array is an **array of arrays**
  - Each of the arrays has **different length**

```
int[][] jagged = new int[2][];
jagged[0] = new int[3];
jagged[1] = new int[2];
```

	0	1	2
row 0	...	...	...
row 1	...	...	

- **Accessing elements**

```
int element = jagged[0][1];
```

Col Index

Row Index



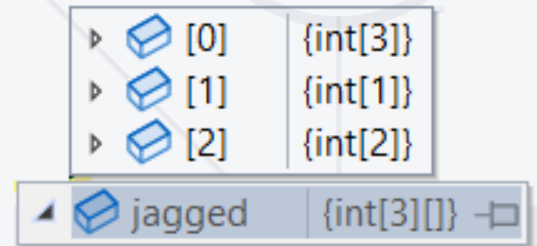
# Reading a Jagged Array

```
int rowCount = int.Parse(Console.ReadLine());
int[][] jagged = new int[rowCount][];

for (int row = 0; row < jagged.Length; row++)
{
    string[] nums = Console.ReadLine().Split(' ');
    jagged[row] = new int[nums.Length];

    for (int col = 0; col < jagged[row].Length; col++)
    {
        jagged[row][col] = int.Parse(nums[col]);
    }
}
```

3  
10 20 30  
40  
50 60



# Printing a Jagged Array – Example

- Using a **for** loop

Implement your custom method

```
int[][] matrix = ReadJaggedArray();  
for (int row = 0; row < matrix.Length; row++)  
{  
    for (int col = 0; col < matrix[row].Length; col++)  
        Console.Write("{0} ", matrix[row][col]);  
    Console.WriteLine();  
}
```

- Using a **foreach** loop

```
int[][] matrix = ReadJaggedArray();  
foreach (int[] row in matrix)  
    Console.WriteLine(string.Join(" ", row));
```

# Read and Print a Jagged Array (Short Version)

*// Allocate the array rows*

```
int rows = int.Parse(Console.ReadLine());  
int[][] jagged = new int[rows][];
```

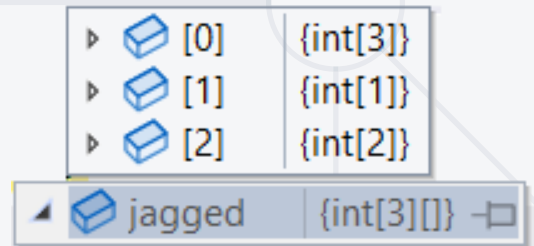
3
10 20 30
40
50 60

*// Read the jagged array*

```
for (int row = 0; row < jagged.Length; row++)  
    jagged[row] = Console.ReadLine().Split(' ')  
        .Select(int.Parse).ToArray();
```

*// Print the jagged array*

```
foreach (int[] row in jagged)  
    Console.WriteLine(string.Join(" ", row));
```



# Problem: Jagged-Array Modification

- On the first line you will get the number **rows**
- On the next lines you will get the **elements for each row**
- Until you receive "**END**", read commands
  - **Add {row} {col} {value}**
  - **Subtract {row} {col} {value}**
- If the coordinates are invalid, print "**Invalid coordinates**"
- When you receive "**END**", print the jagged array

Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1452#5>



# Jagged-Array Modification – Example

```
3
1 2 3
4 5 6 7
8 9 10
Add 0 0 5
Subtract 1 2 2
Subtract 1 4 7
END
```



**Invalid coordinates**

```
6 2 3
4 5 4 7
8 9 10
```

	0	1	2	
row 0	1	2	3	
row 1	4	5	6	7
row 2	8	9	10	

# Solution: Jagged-Array Modification

```
int rowSize = int.Parse(Console.ReadLine());
int[][] matrix = new int[rowSize][];

for (int row = 0; row < rowSize; row++)
{
    int[] columns = Console.ReadLine()
        .Split()
        .Select(int.Parse)
        .ToArray();
    matrix[row] = columns;
}
// continues on the next slide...
```

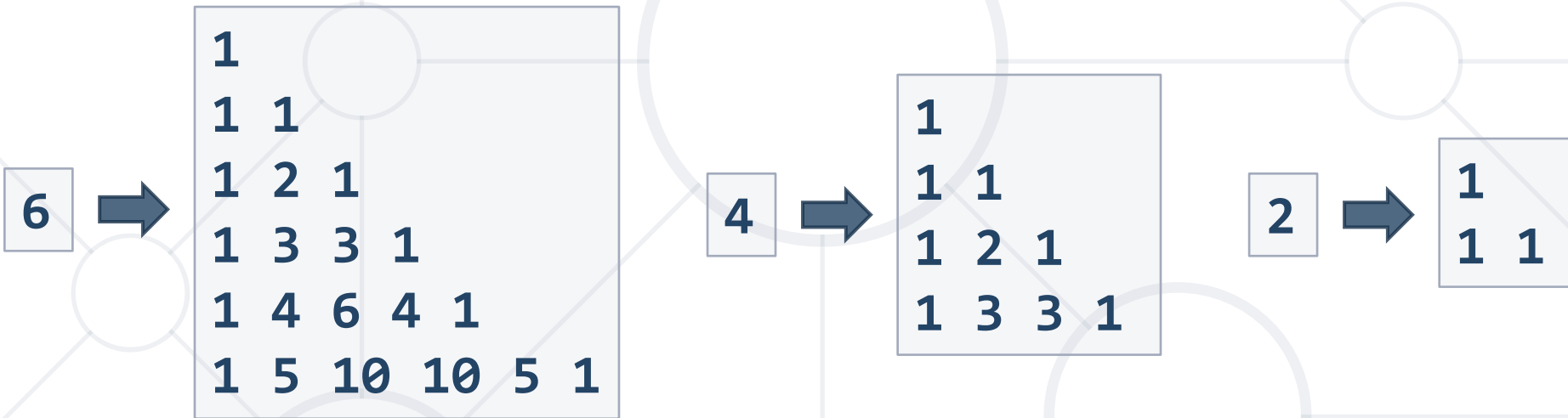
# Solution: Jagged-Array Modification

```
string line;
while ((line = Console.ReadLine()) != "END") {
    string[] tokens = line.Split();
    string command = tokens[0];
    int row = int.Parse(tokens[1]);
    int col = int.Parse(tokens[2]);
    int value = int.Parse(tokens[3]);
    if (row < 0 || row >= matrix.Length || ... )
        Console.WriteLine("Invalid coordinates");
    else
        { // TODO: Execute the command }
}
// TODO: Print the matrix
```

Check the row  
and col ranges

# Problem: Pascal Triangle

- Write a program to prints on the console the Pascal's Triangle



Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1452#6>

# Solution: Pascal Triangle

```
int height = int.Parse(Console.ReadLine());
long[][] triangle = new long[height][];
int currentWidth = 1;
for (long row = 0; row < height; row++)
{
    triangle[row] = new long[currentWidth];
    long[] currentRow = triangle[row];
    currentRow[0] = 1;
    currentRow[currentRow.Length - 1] = 1;
    currentWidth++;
    // TODO: Fill elements for each row (next slide)
}
```

# Solution: Pascal Triangle

```
if (currentRow.Length > 2)
{
    for (int i = 1; i < currentRow.Length - 1; i++)
    {
        long[] previousRow = triangle[row - 1];
        long prevoiousRowSum = previousRow[i] + previousRow[i - 1];
        currentRow[i] = prevoiousRowSum;
    }
}
// TODO: Print triangle
foreach (long[] row in triangle)
    Console.WriteLine(string.Join(" ", row));
```

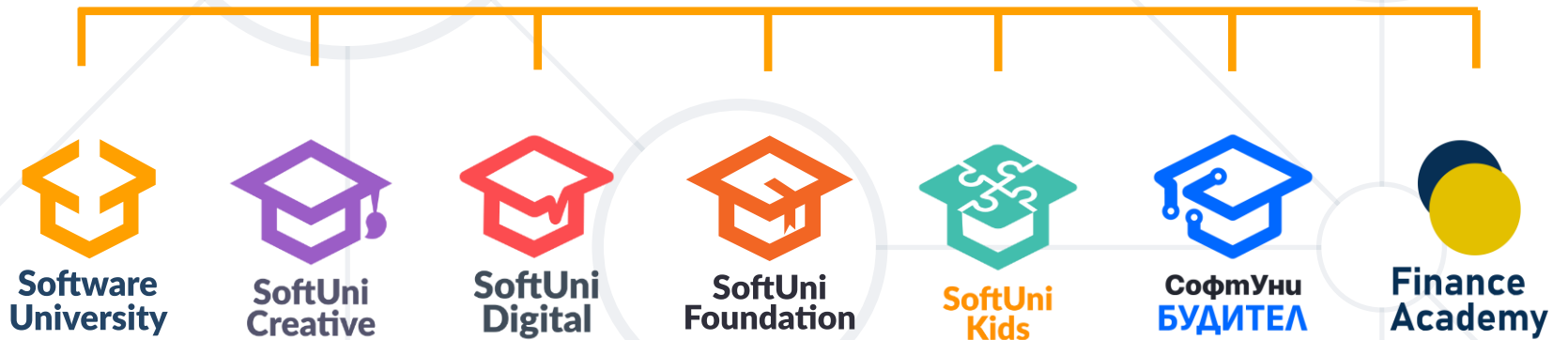
- Multidimensional arrays
  - Have **more than one** dimension
  - Two-dimensional arrays are like tables with **rows** and **columns**
- Jagged arrays
  - Arrays of arrays
  - Each **element** is an array **itself**



# Questions?



SoftUni





# SoftUni Diamond Partners



THE CROWN IS YOURS



- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [about.softuni.bg](http://about.softuni.bg)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

