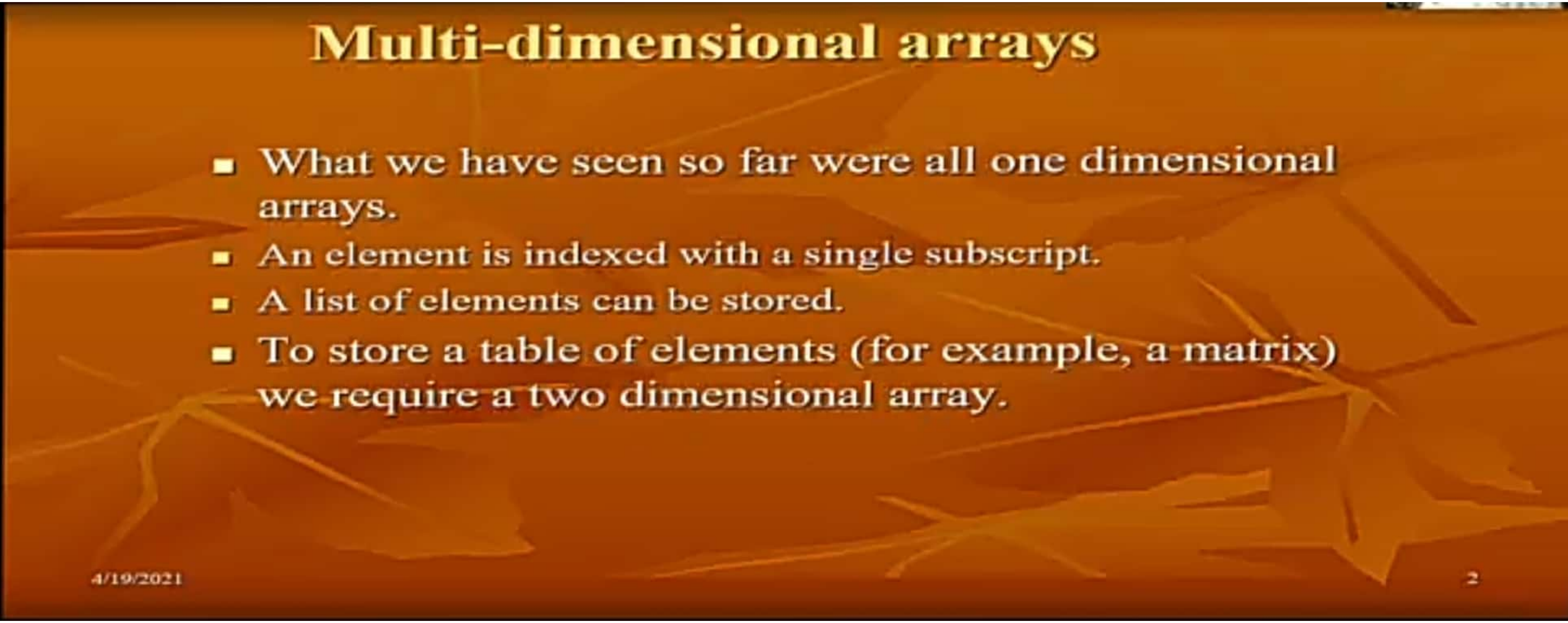
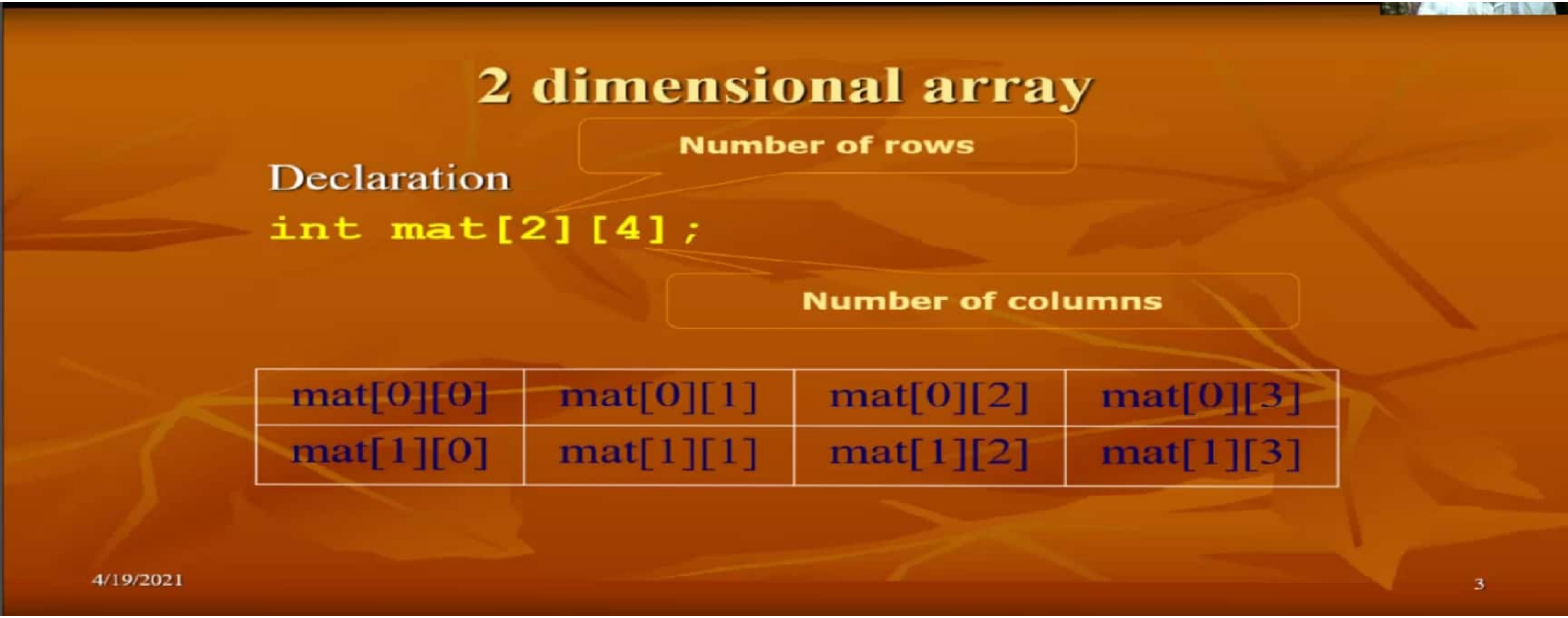
# Arrays...



# Multi-dimensional arrays

- What we have seen so far were all one dimensional arrays.
- An element is indexed with a single subscript.
- A list of elements can be stored.
- To store a table of elements (for example, a matrix) we require a two dimensional array.



# 2 dimensional array

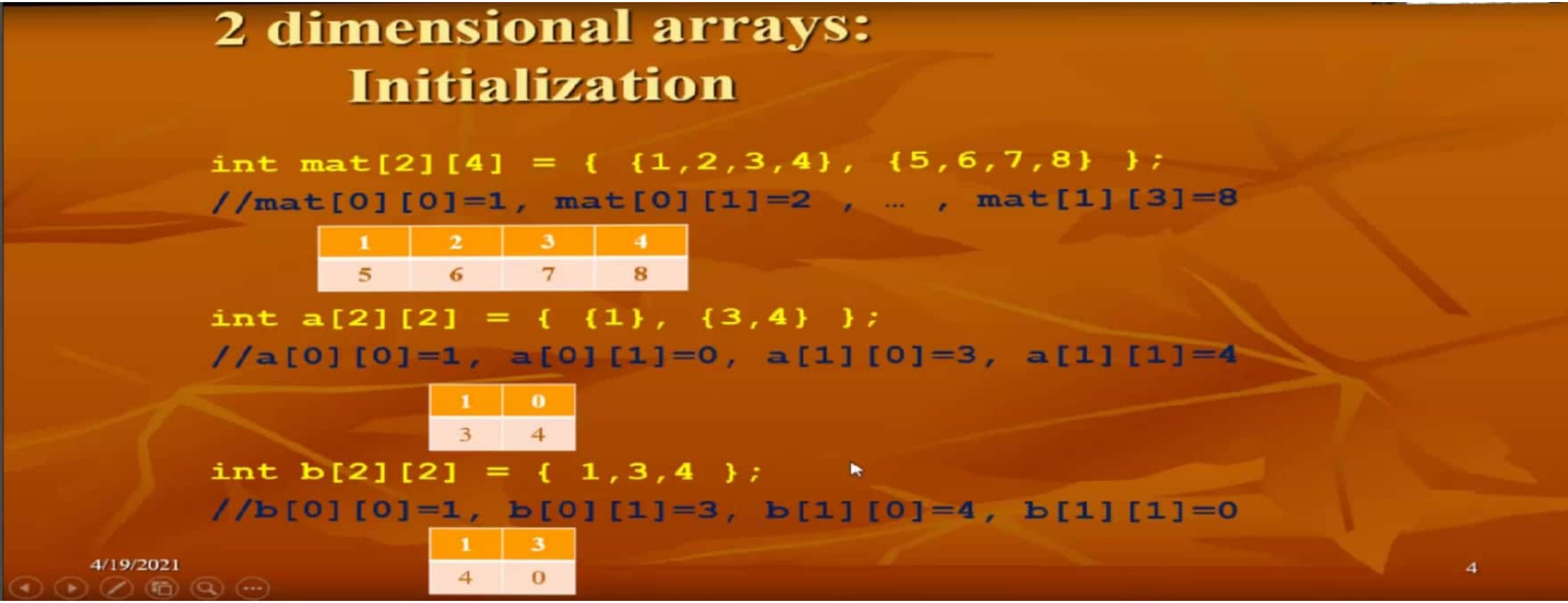
Number of rows

Declaration

```
int mat[2][4];
```

Number of columns

| | | | |
|---|---|---|---|
| mat[0][0] | mat[0][1] | mat[0][2] | mat[0][3] |
| mat[1][0] | mat[1][1] | mat[1][2] | mat[1][3] |



# 2 dimensional arrays: Initialization

```
int mat[2][4] = { {1,2,3,4}, {5,6,7,8} };
//mat[0][0]=1, mat[0][1]=2 , ... , mat[1][3]=8
```

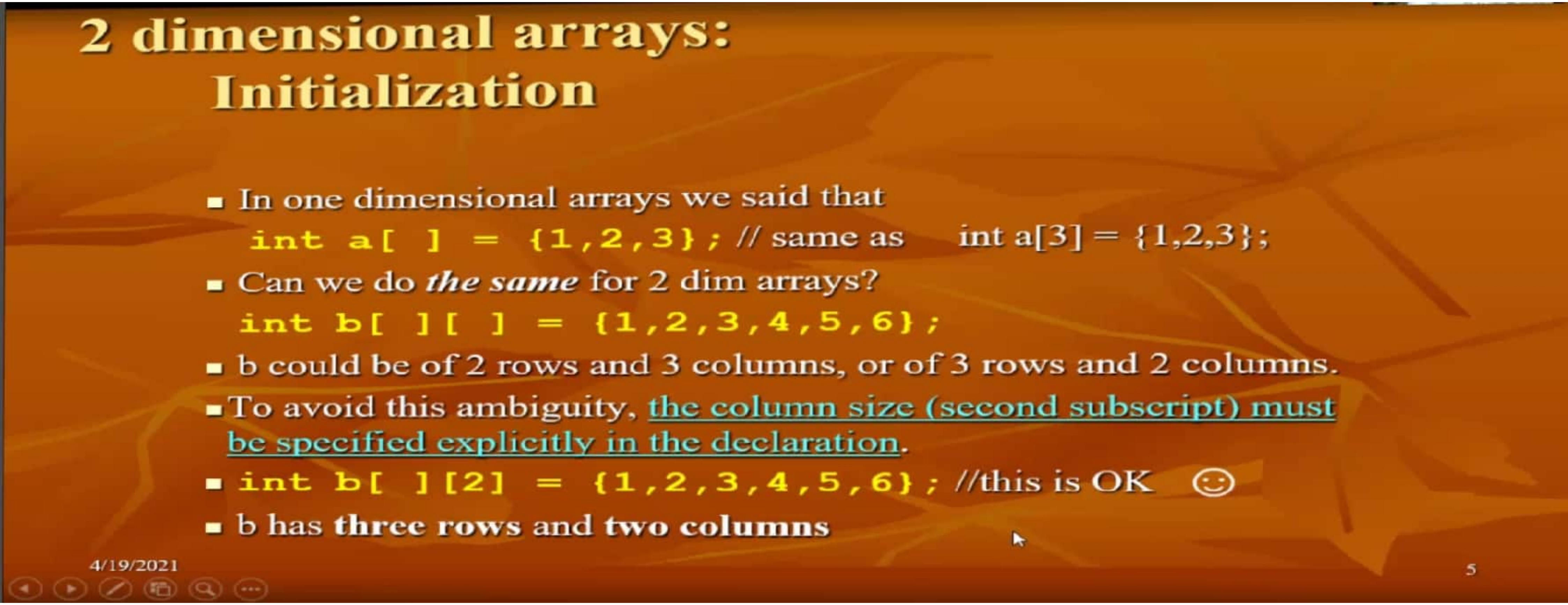
| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |

```
int a[2][2] = { {1}, {3,4} };
//a[0][0]=1, a[0][1]=0, a[1][0]=3, a[1][1]=4
```

| | |
|---|---|
| 1 | 0 |
| 3 | 4 |

```
int b[2][2] = { 1,3,4 };
//b[0][0]=1, b[0][1]=3, b[1][0]=4, b[1][1]=0
```

| | |
|---|---|
| 1 | 3 |
| 4 | 0 |



# 2 dimensional arrays: Initialization

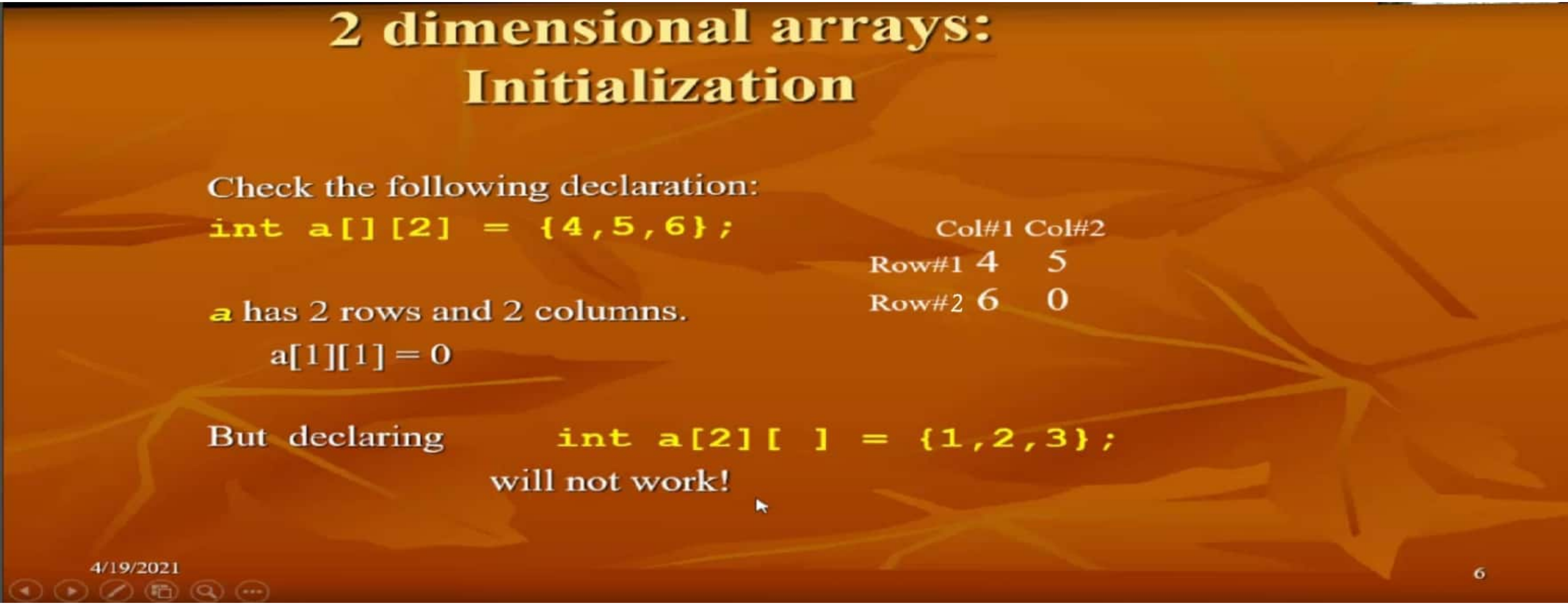
- In one dimensional arrays we said that

  `int a[ ] = {1,2,3};` // same as int a[3] = {1,2,3};

- Can we do ***the same*** for 2 dim arrays?

  `int b[ ][ ] = {1,2,3,4,5,6};`

- b could be of 2 rows and 3 columns, or of 3 rows and 2 columns.
- To avoid this ambiguity, the column size (second subscript) must be specified explicitly in the declaration.
- `int b[ ][2] = {1,2,3,4,5,6};` //this is OK ☺
- b has **three rows and two columns**



# 2 dimensional arrays: Initialization

Check the following declaration:

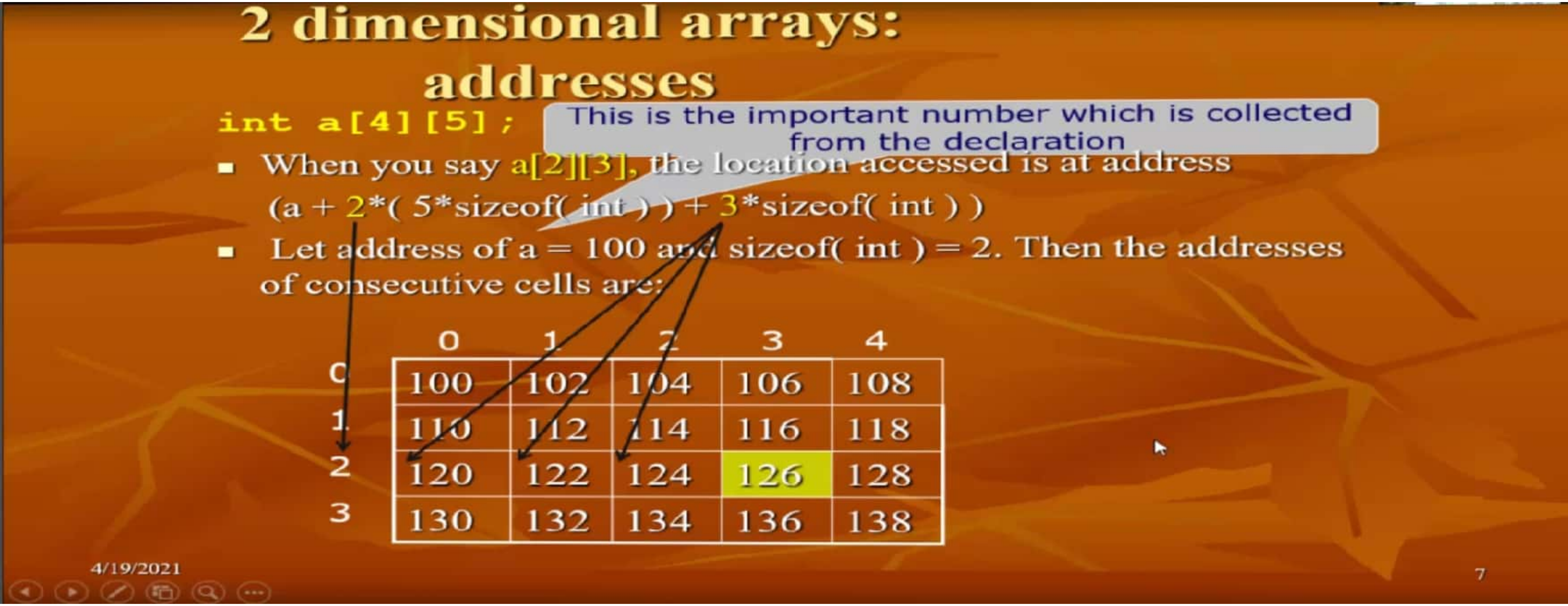
```
int a[][2] = {4,5,6};
```

|  | Col#1 | Col#2 |
|---|---|---|
| Row#1 | 4 | 5 |
| Row#2 | 6 | 0 |

a has 2 rows and 2 columns.

a[1][1] = 0

But declaring `int a[2][ ] = {1,2,3};` will not work!



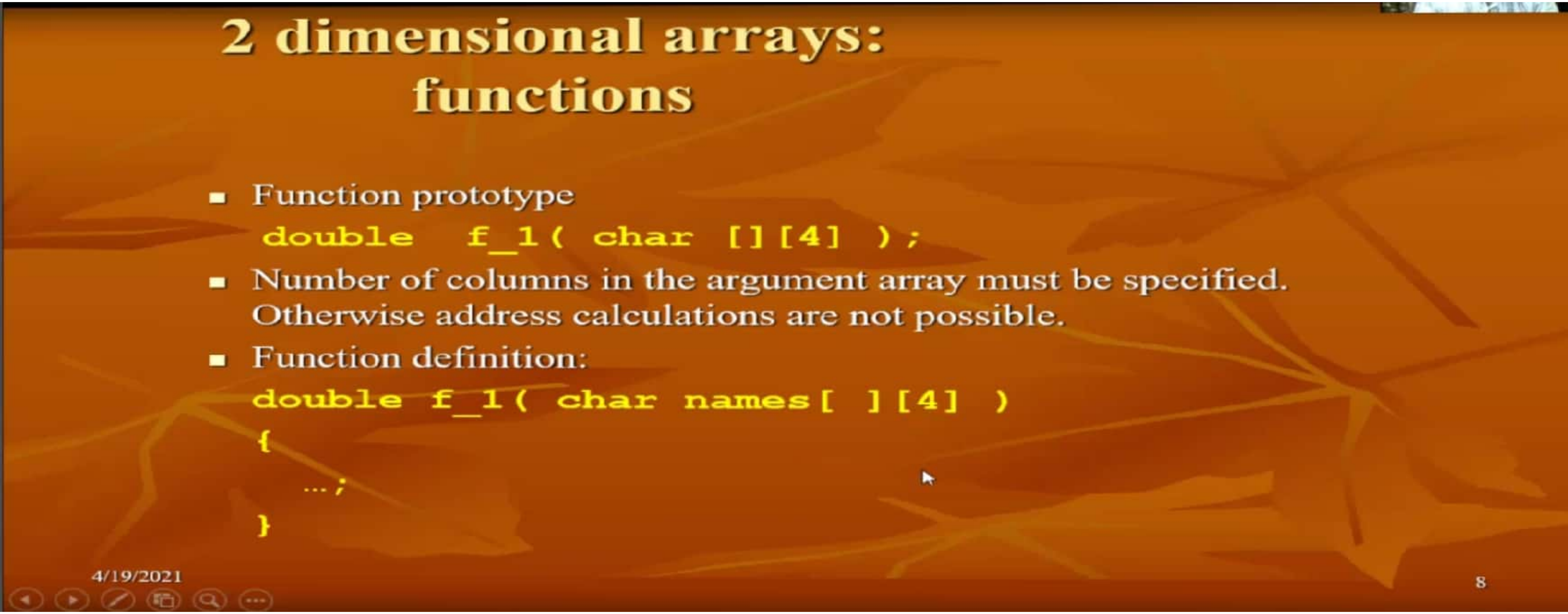
# 2 dimensional arrays: addresses

```
int a[4][5];
```

This is the important number which is collected from the declaration

- When you say a[2][3], the location accessed is at address (a + 2*( 5*sizeof( int ) ) + 3*sizeof( int ) )
- Let address of a = 100 and sizeof( int ) = 2. Then the addresses of consecutive cells are:

|   | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 100 | 102 | 104 | 106 | 108 |
| 1 | 110 | 112 | 114 | 116 | 118 |
| 2 | 120 | 122 | 124 | 126 | 128 |
| 3 | 130 | 132 | 134 | 136 | 138 |



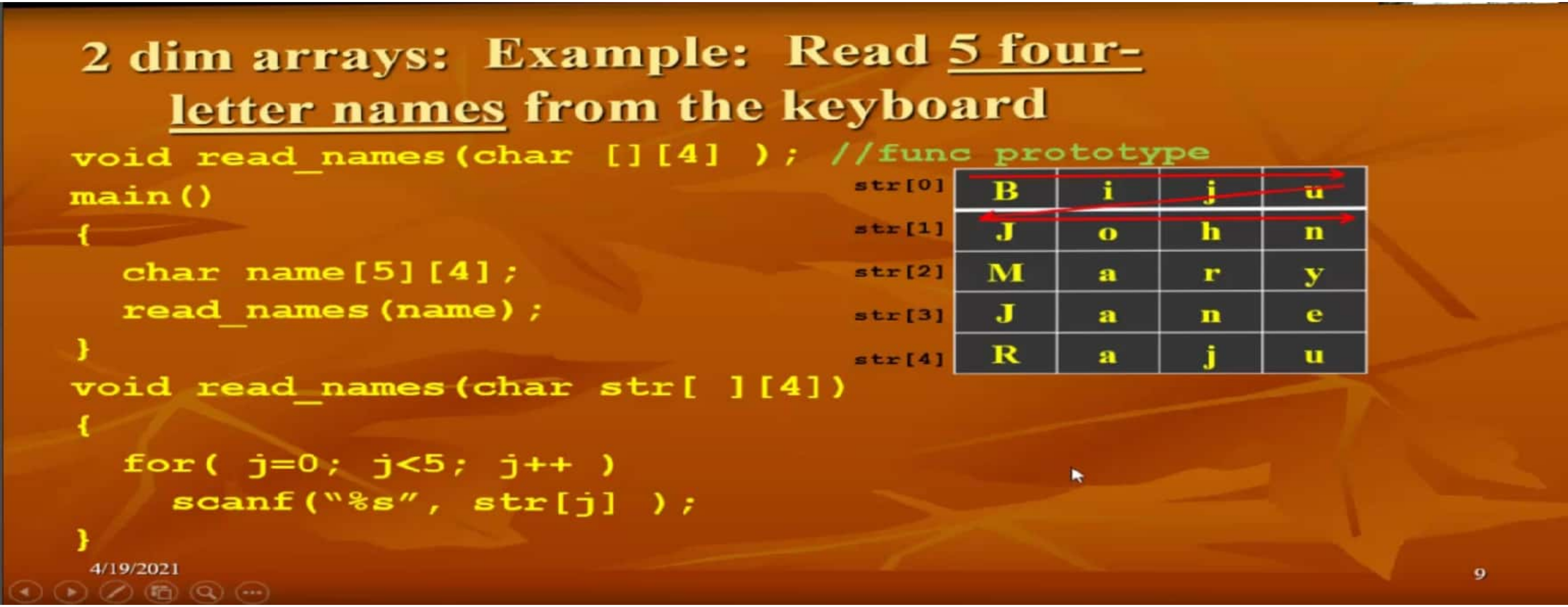
# 2 dimensional arrays: functions

- Function prototype

```
double  f_1( char [][4] );
```

- Number of columns in the argument array must be specified. Otherwise address calculations are not possible.
- Function definition:

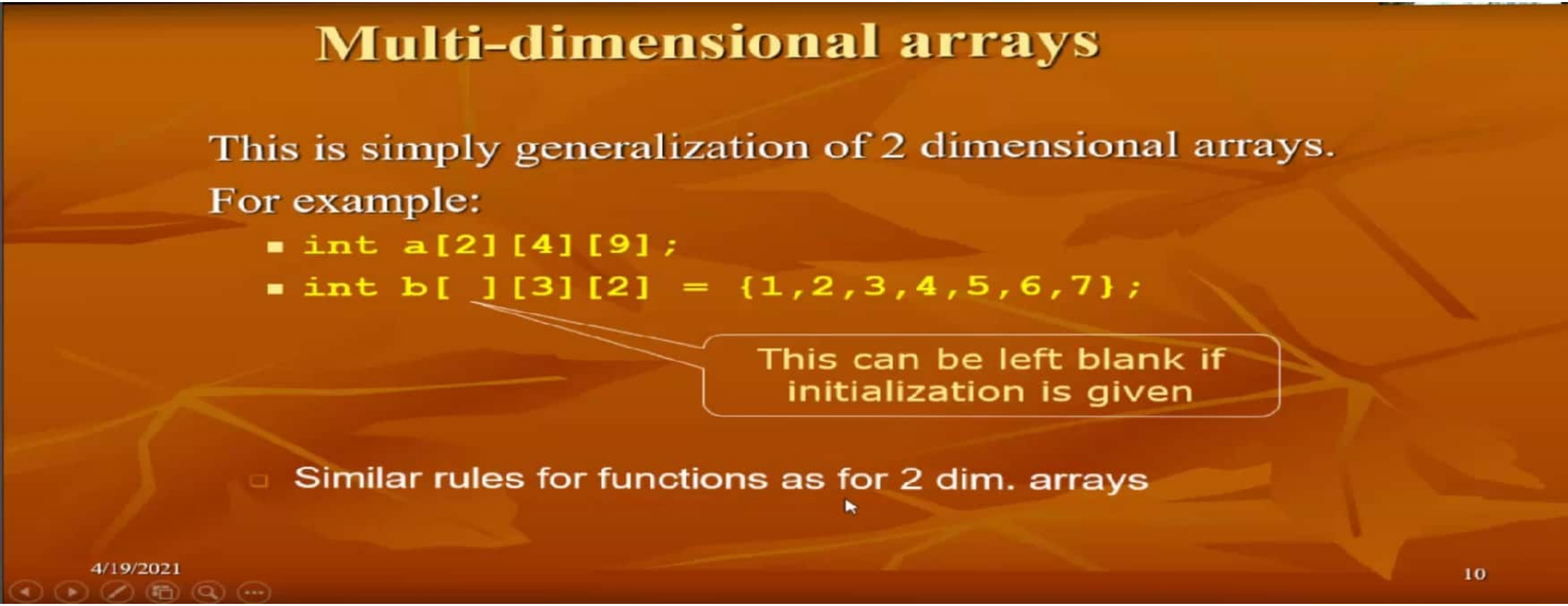
```
double f_1( char names[ ][4] )
{
   ...;
}
```



## 2 dim arrays: Example: Read 5 four-letter names from the keyboard

```
void read_names(char [][4] ); //func prototype
main()
{
   char name[5][4];
   read_names(name);
}
void read_names(char str[ ][4])
{
   for( j=0; j<5; j++ )
     scanf(“%s”, str[j] );
}
```

| | | | | |
|---|---|---|---|---|
| str[0] | B | i | j | u |
| str[1] | J | o | h | n |
| str[2] | M | a | r | y |
| str[3] | J | a | n | e |
| str[4] | R | a | j | u |



# Multi-dimensional arrays

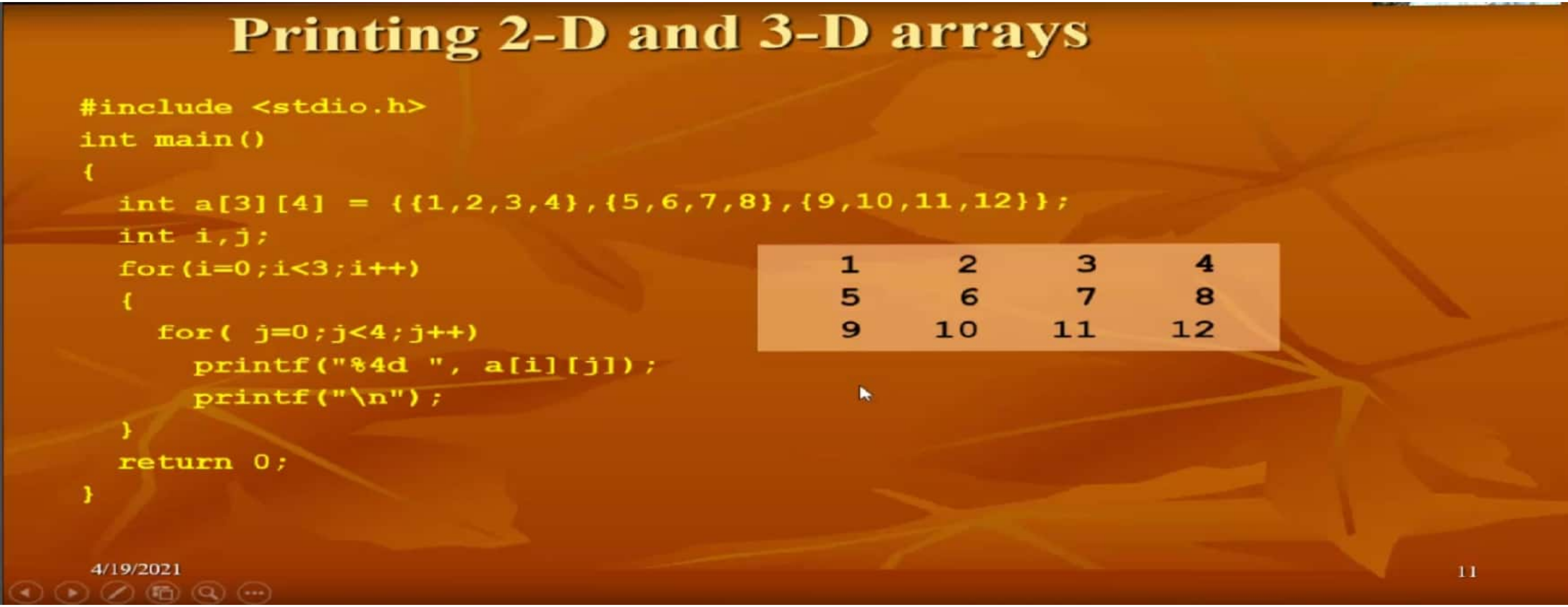
This is simply generalization of 2 dimensional arrays.

For example:

- `int a[2][4][9];`
- `int b[ ][3][2] = {1,2,3,4,5,6,7};`

This can be left blank if initialization is given

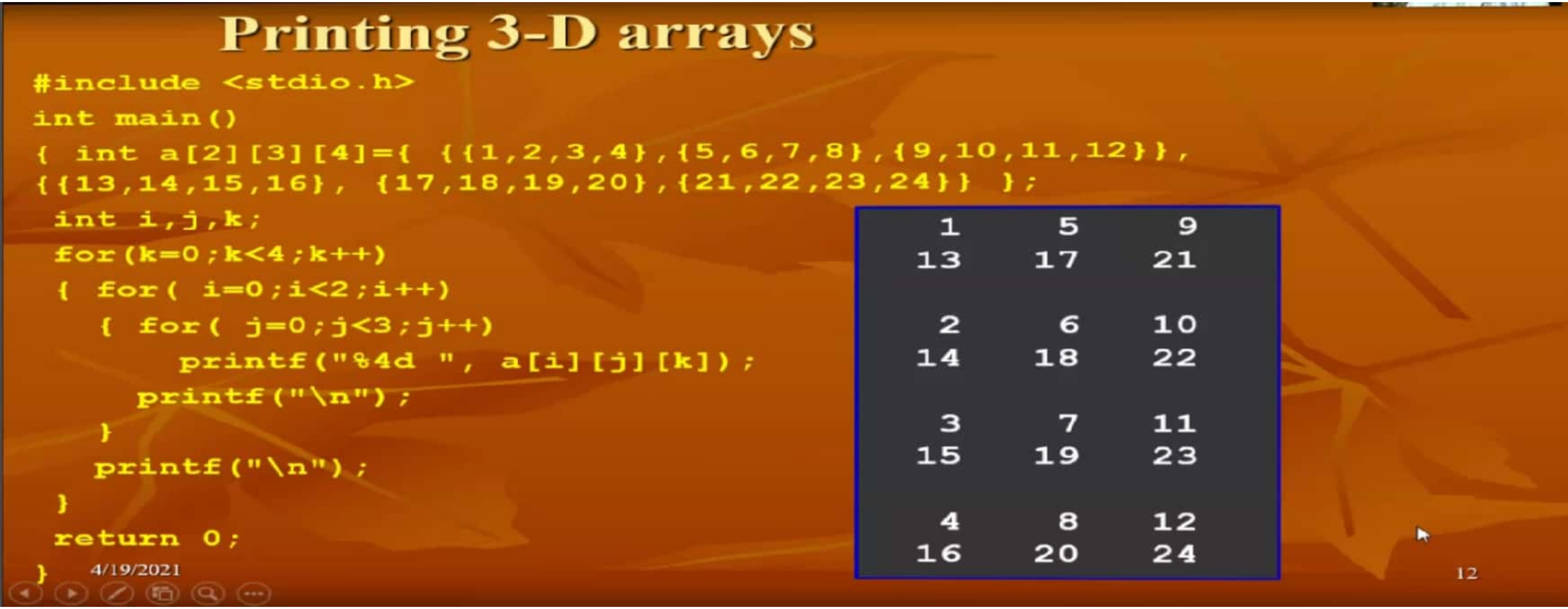
- Similar rules for functions as for 2 dim. arrays



# Printing 2-D and 3-D arrays

```
#include <stdio.h>
int main()
{
   int a[3][4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
   int i,j;
   for(i=0;i<3;i++)
   {
      for( j=0;j<4;j++)
         printf("%4d ", a[i][j]);
         printf("\n");
   }
   return 0;
}
```

```
   1      2      3      4
   5      6      7      8
   9     10     11     12
```



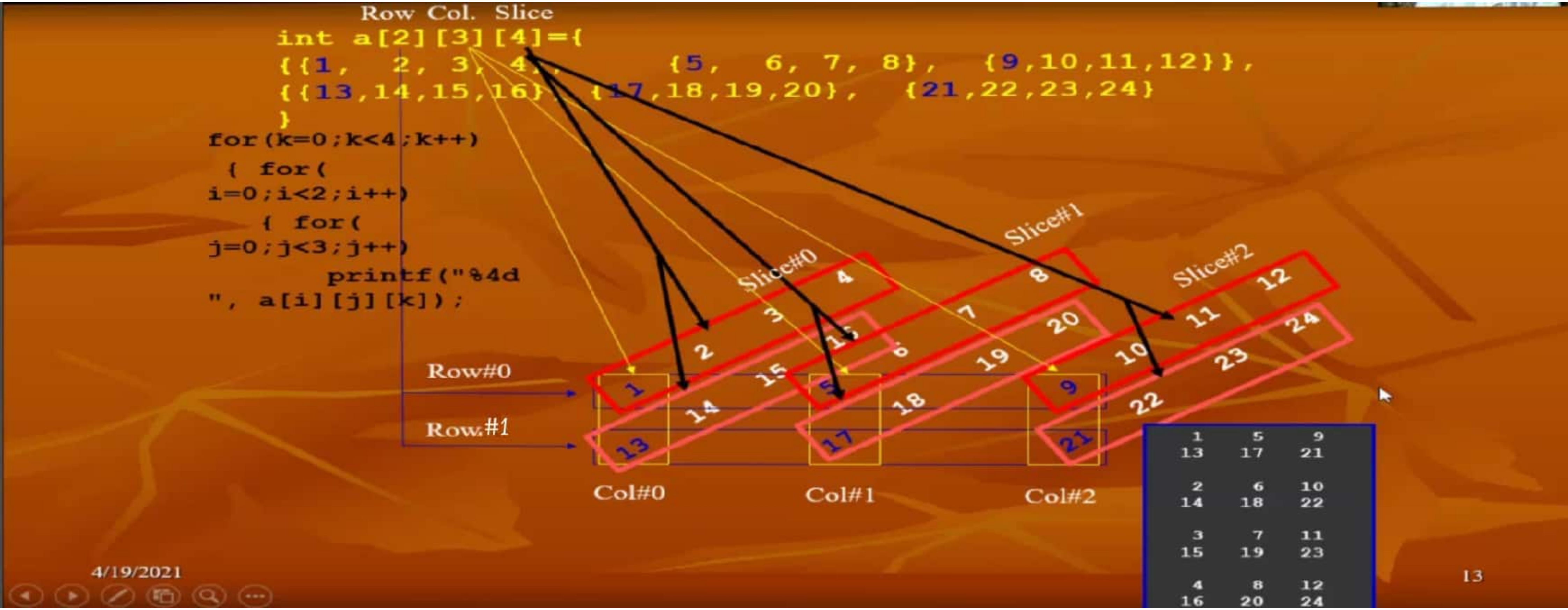
# Printing 3-D arrays

```
#include <stdio.h>
int main()
{ int a[2][3][4]={ {{1,2,3,4},{5,6,7,8},{9,10,11,12}},
{{13,14,15,16}, {17,18,19,20},{21,22,23,24}} };
 int i,j,k;
 for(k=0;k<4;k++)
 { for( i=0;i<2;i++)
   { for( j=0;j<3;j++)
       printf("%4d ", a[i][j][k]);
     printf("\n");
   }
   printf("\n");
 }
 return 0;
}
```

```
   1     5     9
  13    17    21

   2     6    10
  14    18    22

   3     7    11
  15    19    23

   4     8    12
  16    20    24
```

4/19/2021

12



Row Col. Slice

```
int a[2][3][4]={
{{1,  2, 3, 4},    {5,  6, 7, 8},   {9,10,11,12}},
{{13,14,15,16}, {17,18,19,20},  {21,22,23,24}
}
for(k=0;k<4;k++)
 { for(
i=0;i<2;i++)
   { for(
j=0;j<3;j++)
       printf("%4d
", a[i][j][k]);
```

```
   1    5    9
  13   17   21

   2    6   10
  14   18   22

   3    7   11
  15   19   23

   4    8   12
  16   20   24
```

4/19/2021