# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

**Argomenti**

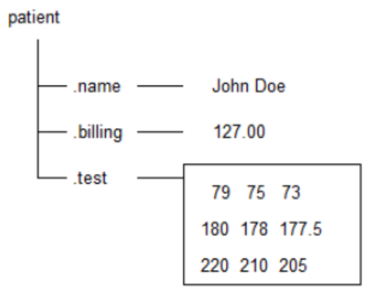
## Indicizzazione di array

### Structure Arrays

When you have data that you want to organize by name, you can use structures to store it. Structures store data in containers called *fields*, which you can then access by the names you specify. Use dot notation to create, assign, and access data in structure fields. If the value stored in a field is an array, then you can use array indexing to access elements of the array. When you store multiple structures as a structure array, you can use array indexing and dot notation to access individual structures and their fields.

#### Create Scalar Structure

First, create a structure named `patient` that has fields storing data about a patient. The diagram shows how the structure stores data. A structure like `patient` is also referred to as a *scalar structure* because the variable stores one structure.



Use dot notation to add the fields `name`, `billing`, and `test`, assigning data to each field. In this example, the syntax `patient.name` creates both the structure and its first field. The commands that follow add more fields.

```
patient.name = 'John Doe';
patient.billing = 127;
patient.test = [79 75 73; 180 178 177.5; 220 210 205]
```

```
patient = struct with fields:
       name: 'John Doe'
    billing: 127
       test: [3x3 double]
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

## Argomenti

## Indicizzazione di array

### Structure Arrays

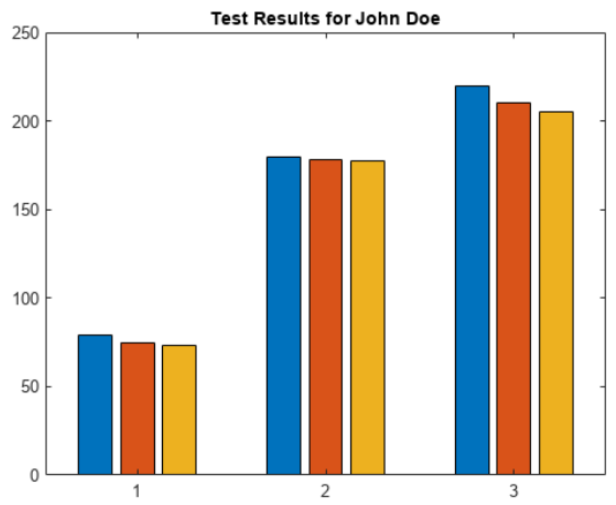
#### Access Values in Fields

After you create a field, you can keep using dot notation to access and change the value it stores.

For example, change the value of the `billing` field.

```
patient.billing = 512.00
```

```
patient = struct with fields:
       name: 'John Doe'
    billing: 512
       test: [3x3 double]
```

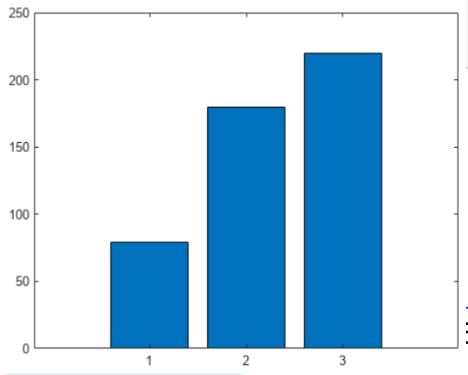


With dot notation, you also can access the value of any field. For example, make a bar chart of the values in `patient.test`. Add a title with the text in `patient.name`. If a field stores an array, then this syntax returns the whole array.

```
bar(patient.test)
title("Test Results for " + patient.name)
```

To access part of an array stored in a field, add indices that are appropriate for the size and type of the array. For example, create a bar chart of the data in one column of `patient.test`.

```
bar(patient.test(:,1))
```



# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

**Argomenti**

## Indicizzazione di array

### Structure Arrays

#### Index into Nonscalar Structure Array

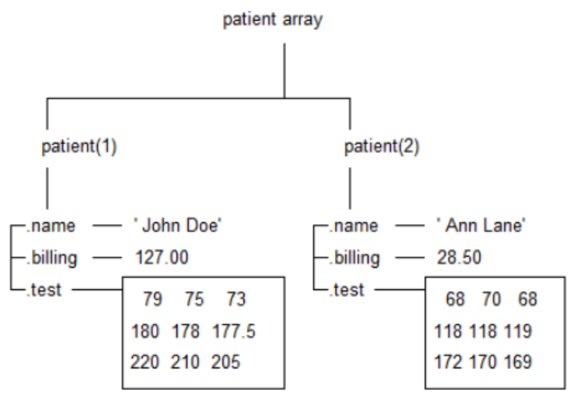
Structure arrays can be nonscalar. You can create a structure array having any size, as long as each structure in the array has the same fields.

For example, add a second structure to `patients` having data about a second patient. Also, assign the original value of `127` to the `billing` field of the first structure. Since the array now has two structures, you must access the first structure by indexing, as in `patient(1).billing = 127`.

```
patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68 70 68; 118 118 119; 172 170 169];
patient(1).billing = 127
```

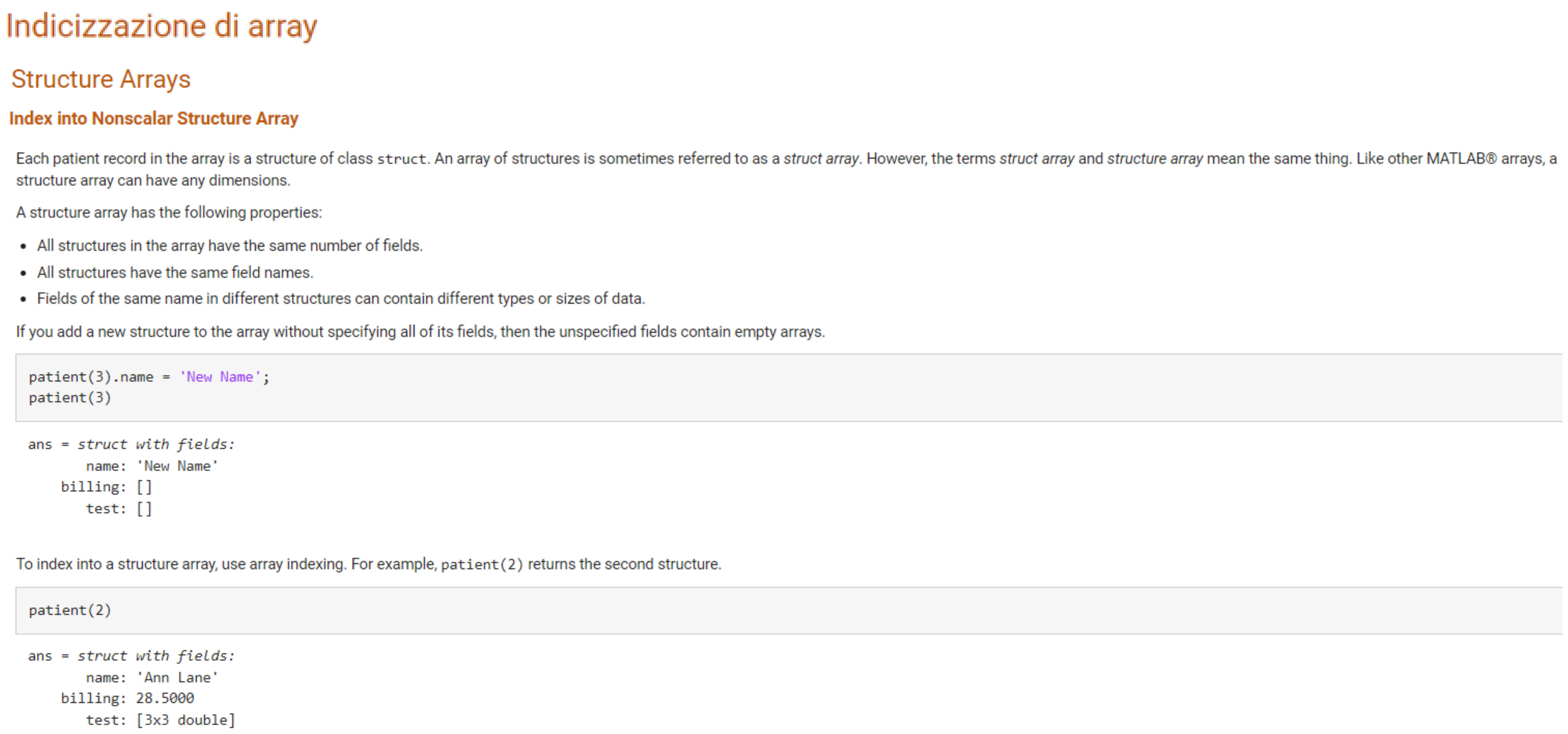
```
patient=1×2 struct array with fields:
    name
    billing
    test
```

As a result, `patient` is a 1-by-2 structure array with contents shown in the diagram.



# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Indicizzazione di array

### Structure Arrays

#### Index into Nonscalar Structure Array

Each patient record in the array is a structure of class `struct`. An array of structures is sometimes referred to as a *struct array*. However, the terms *struct array* and *structure array* mean the same thing. Like other MATLAB® arrays, a structure array can have any dimensions.

A structure array has the following properties:

- All structures in the array have the same number of fields.
- All structures have the same field names.
- Fields of the same name in different structures can contain different types or sizes of data.

If you add a new structure to the array without specifying all of its fields, then the unspecified fields contain empty arrays.

```
patient(3).name = 'New Name';
patient(3)
```

```
ans = struct with fields:
       name: 'New Name'
    billing: []
       test: []
```

To index into a structure array, use array indexing. For example, `patient(2)` returns the second structure.

```
patient(2)
```

```
ans = struct with fields:
       name: 'Ann Lane'
    billing: 28.5000
       test: [3x3 double]
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

## Argomenti

## Indicizzazione di array

### Structure Arrays

#### Index into Nonscalar Structure Array

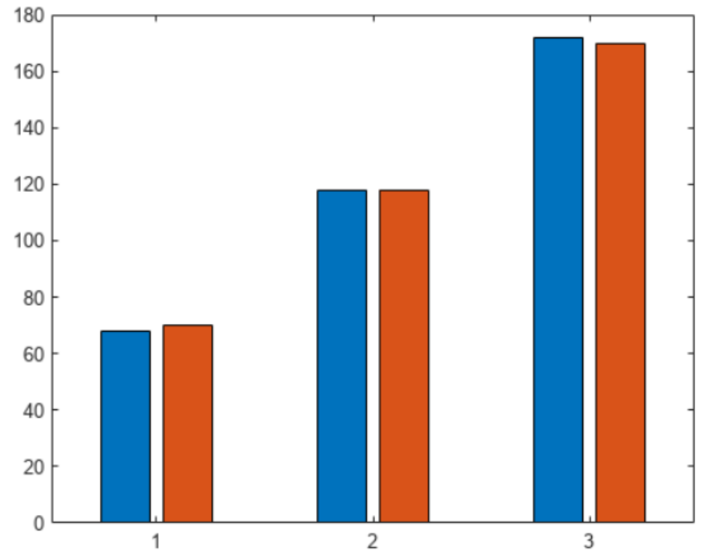
To access a field, use array indexing and dot notation. For example, return the value of the `billing` field for the second patient.

```
patient(2).billing
```

```
ans = 28.5000
```

You also can index into an array stored by a field. Create a bar chart displaying only the first two columns of `patient(2).test`.

```
bar(patient(2).test(:,[1 2]))
```



# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

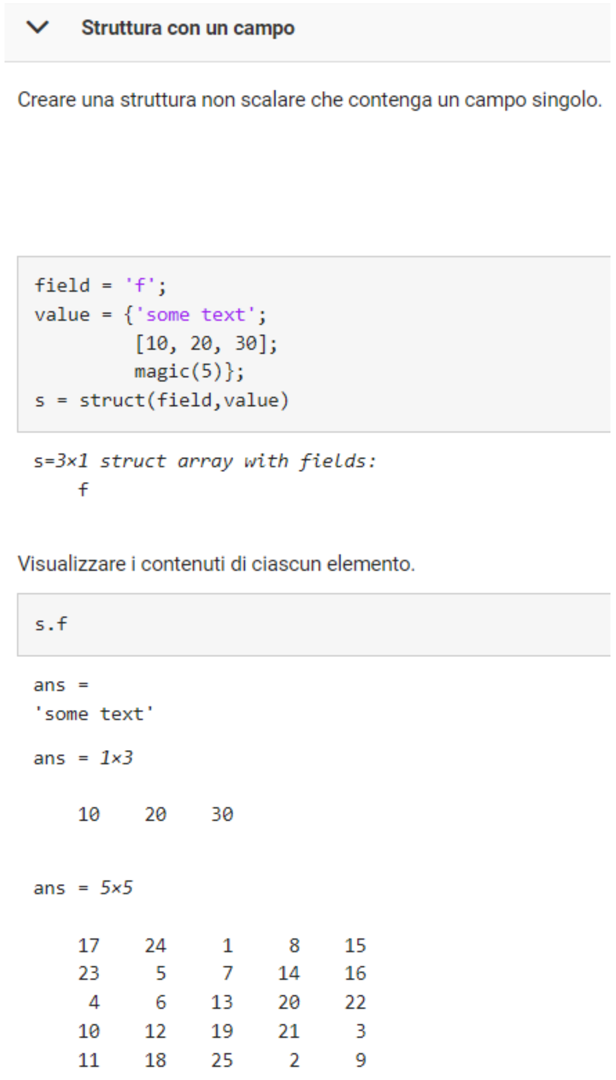
**Argomenti**

Indicizzazione di array

Structure Arrays

struct

Array di struttura



**Struttura con un campo**

Creare una struttura non scalare che contenga un campo singolo.

```
field = 'f';
value = {'some text';
         [10, 20, 30];
         magic(5)};
s = struct(field,value)
```

```
s=3×1 struct array with fields:
    f
```

Visualizzare i contenuti di ciascun elemento.

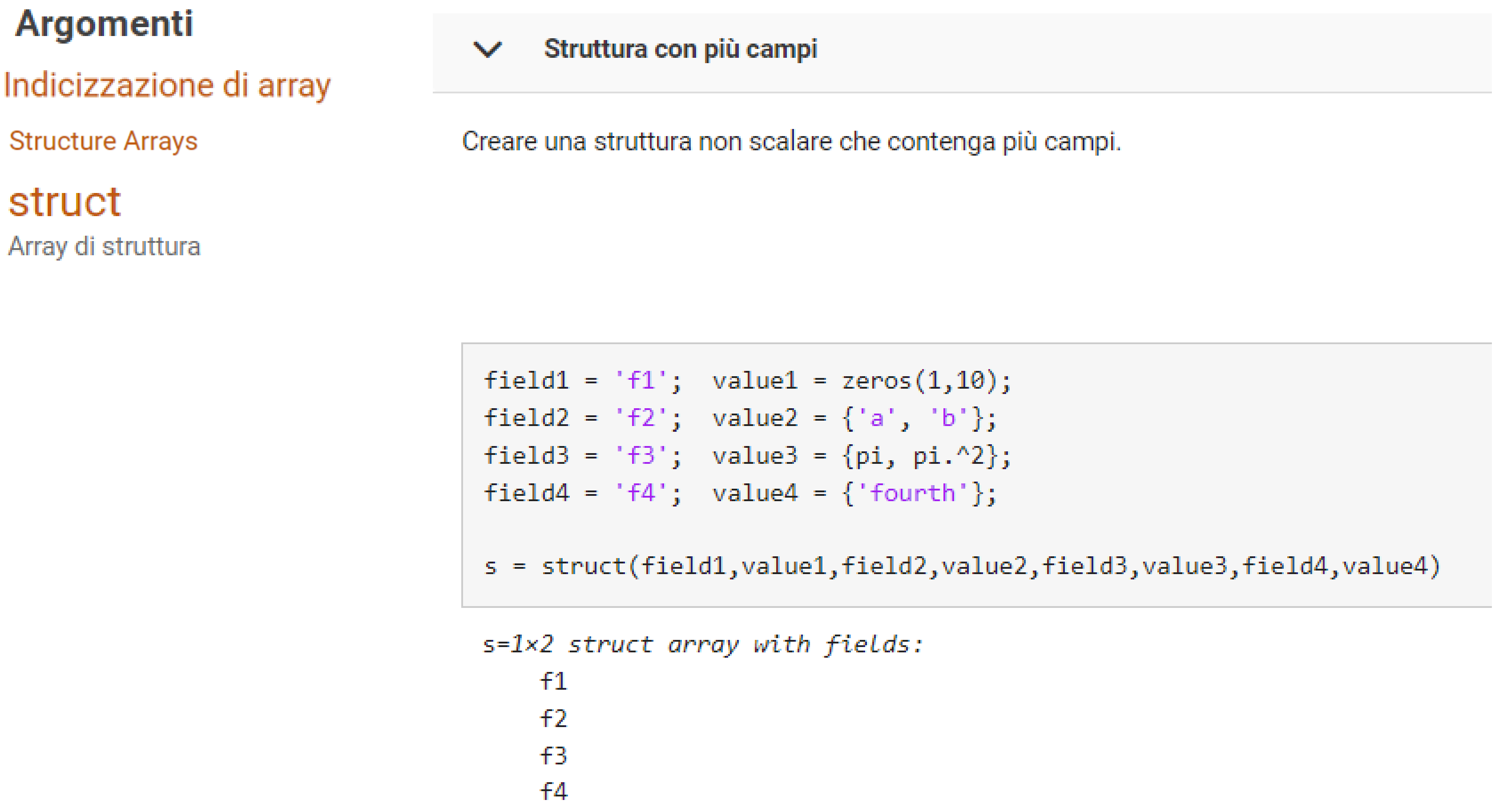
```
s.f
```

```
ans = 
'some text'

ans = 1×3

    10    20    30


ans = 5×5

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Quando si accede a un campo di una struttura non scalare, come `s.f`, MATLAB® restituisce un elenco separato da virgole. In questo caso, `s.f` equivale a `s(1).f, s(2).f, s(3).f`.

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



**Argomenti**

Indicizzazione di array

Structure Arrays

struct

Array di struttura

**Struttura con più campi**

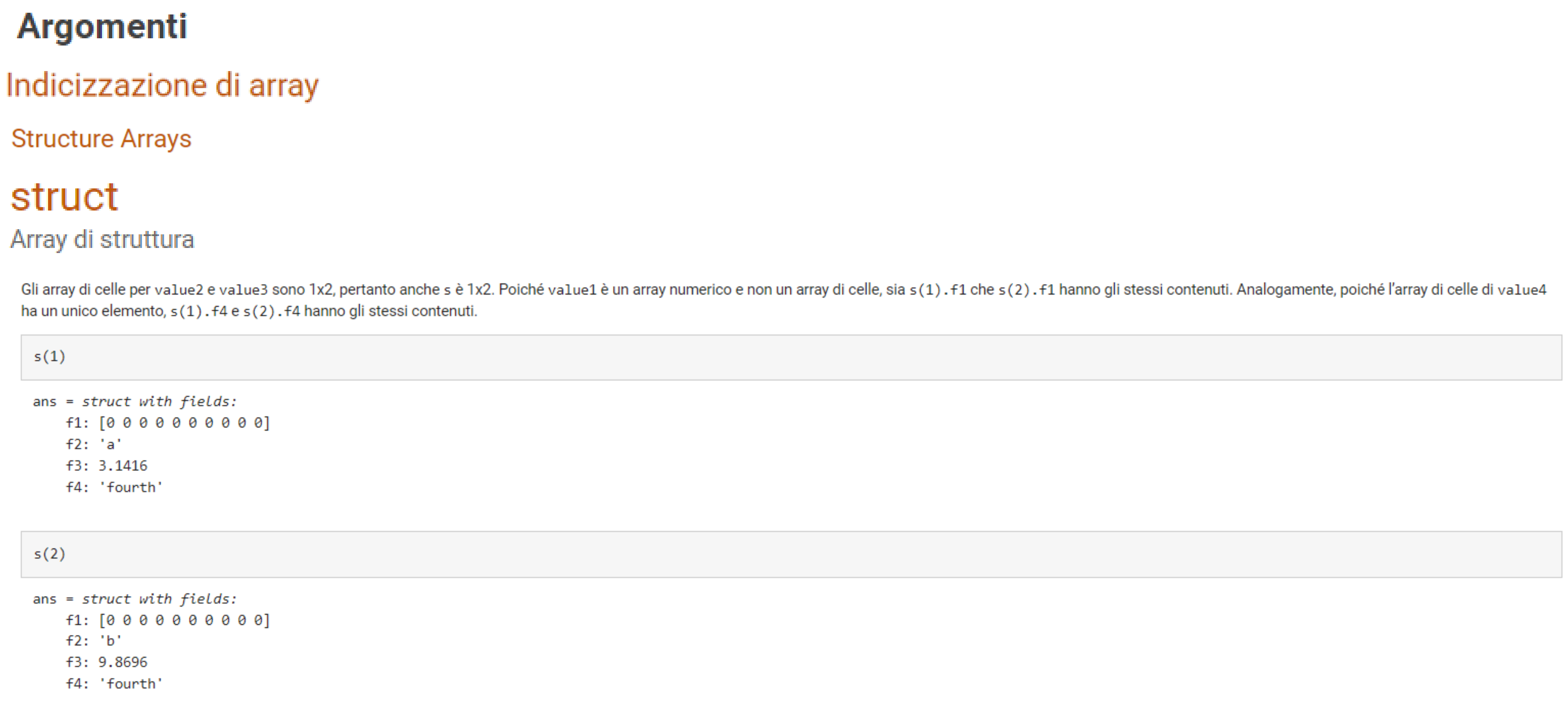
Creare una struttura non scalare che contenga più campi.

```
field1 = 'f1';  value1 = zeros(1,10);
field2 = 'f2';  value2 = {'a', 'b'};
field3 = 'f3';  value3 = {pi, pi.^2};
field4 = 'f4';  value4 = {'fourth'};

s = struct(field1,value1,field2,value2,field3,value3,field4,value4)
```

```
s=1×2 struct array with fields:
    f1
    f2
    f3
    f4
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



**Argomenti**

Indicizzazione di array

Structure Arrays

## struct

Array di struttura

Gli array di celle per `value2` e `value3` sono 1x2, pertanto anche `s` è 1x2. Poiché `value1` è un array numerico e non un array di celle, sia `s(1).f1` che `s(2).f1` hanno gli stessi contenuti. Analogamente, poiché l'array di celle di `value4` ha un unico elemento, `s(1).f4` e `s(2).f4` hanno gli stessi contenuti.

```
s(1)
```

```
ans = struct with fields:
    f1: [0 0 0 0 0 0 0 0 0 0]
    f2: 'a'
    f3: 3.1416
    f4: 'fourth'
```

```
s(2)
```

```
ans = struct with fields:
    f1: [0 0 0 0 0 0 0 0 0 0]
    f2: 'b'
    f3: 9.8696
    f4: 'fourth'
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

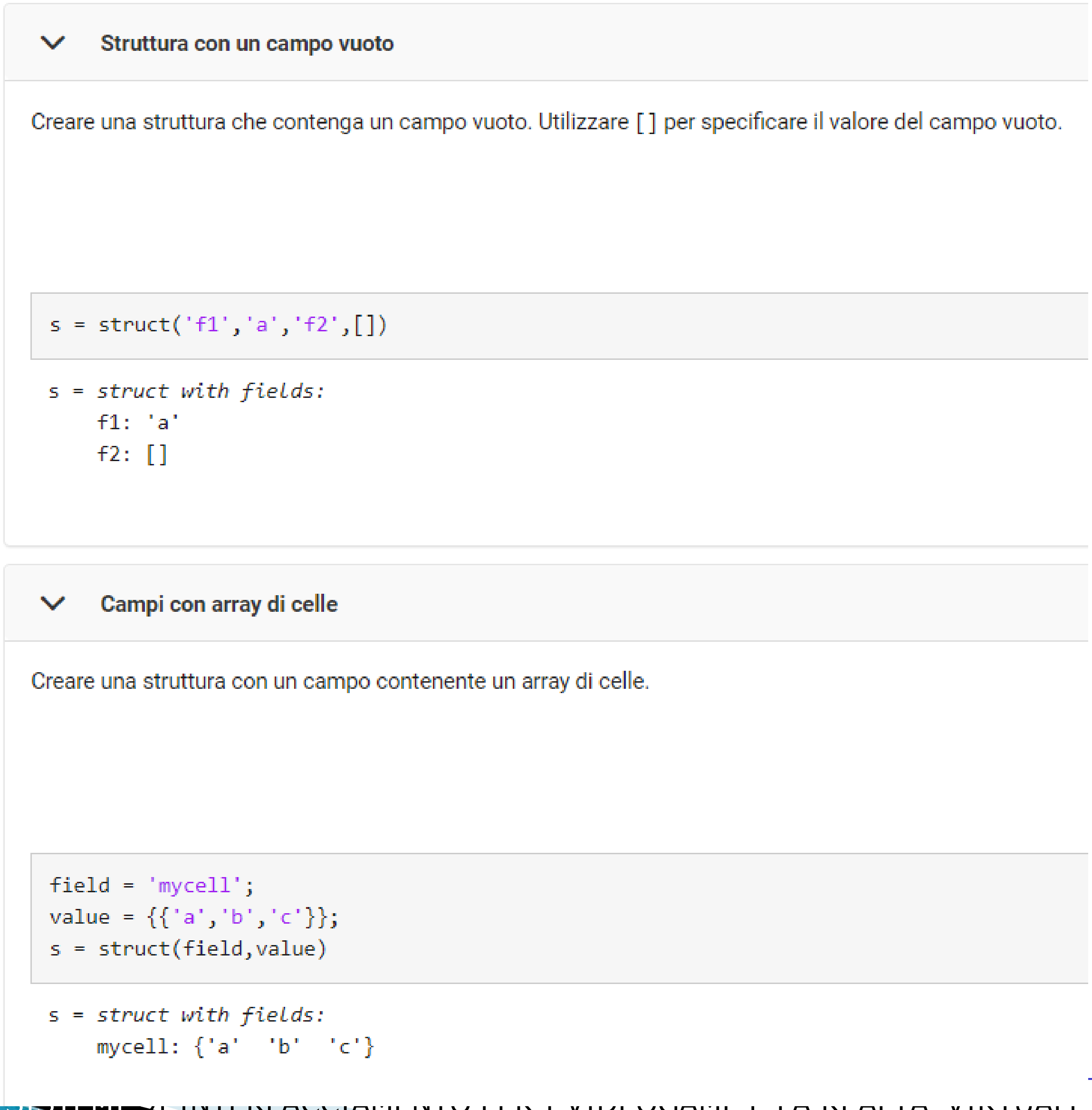
**Argomenti**

Indicizzazione di array

Structure Arrays

struct

Array di struttura



**Struttura con un campo vuoto**

Creare una struttura che contenga un campo vuoto. Utilizzare [] per specificare il valore del campo vuoto.

```
s = struct('f1','a','f2',[])
```

```
s = struct with fields:
    f1: 'a'
    f2: []
```

**Campi con array di celle**

Creare una struttura con un campo contenente un array di celle.

```
field = 'mycell';
value = {{'a','b','c'}};
s = struct(field,value)
```

```
s = struct with fields:
    mycell: {'a'  'b'  'c'}
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

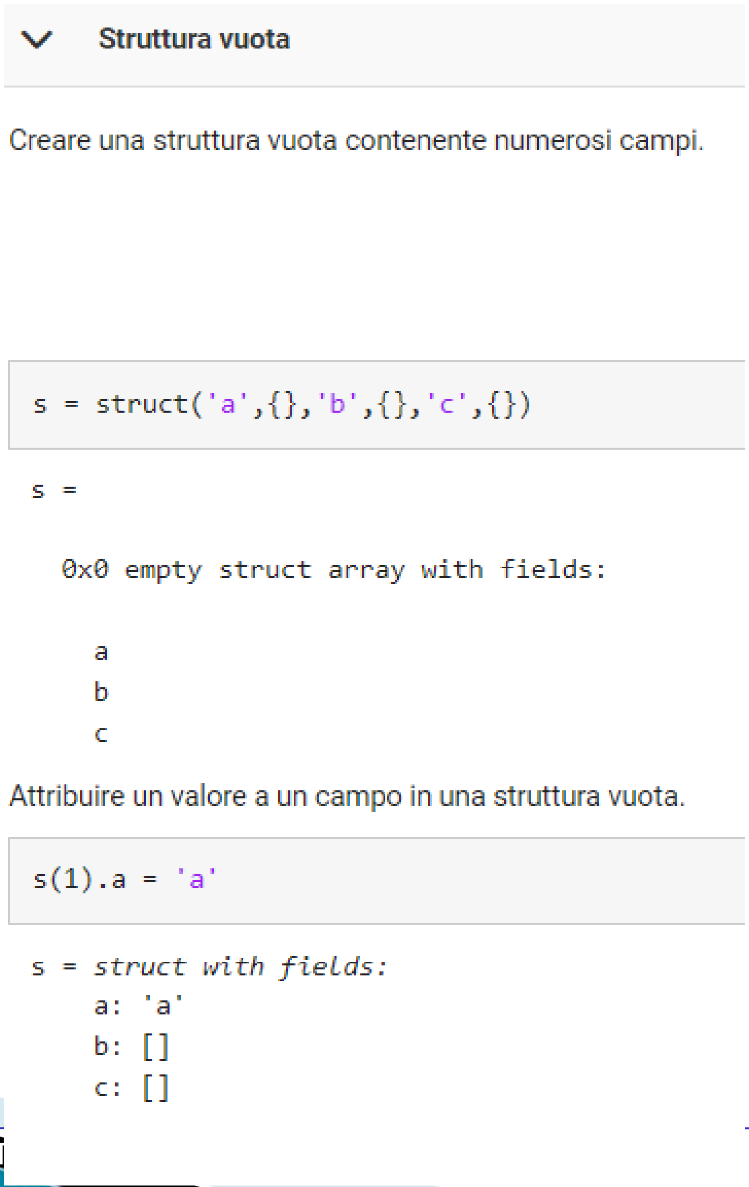
**Argomenti**

Indicizzazione di array

Structure Arrays

struct

Array di struttura



**Struttura vuota**

Creare una struttura vuota contenente numerosi campi.

```
s = struct('a',{},'b',{},'c',{})
```

```
s = 

  0x0 empty struct array with fields:

    a
    b
    c
```

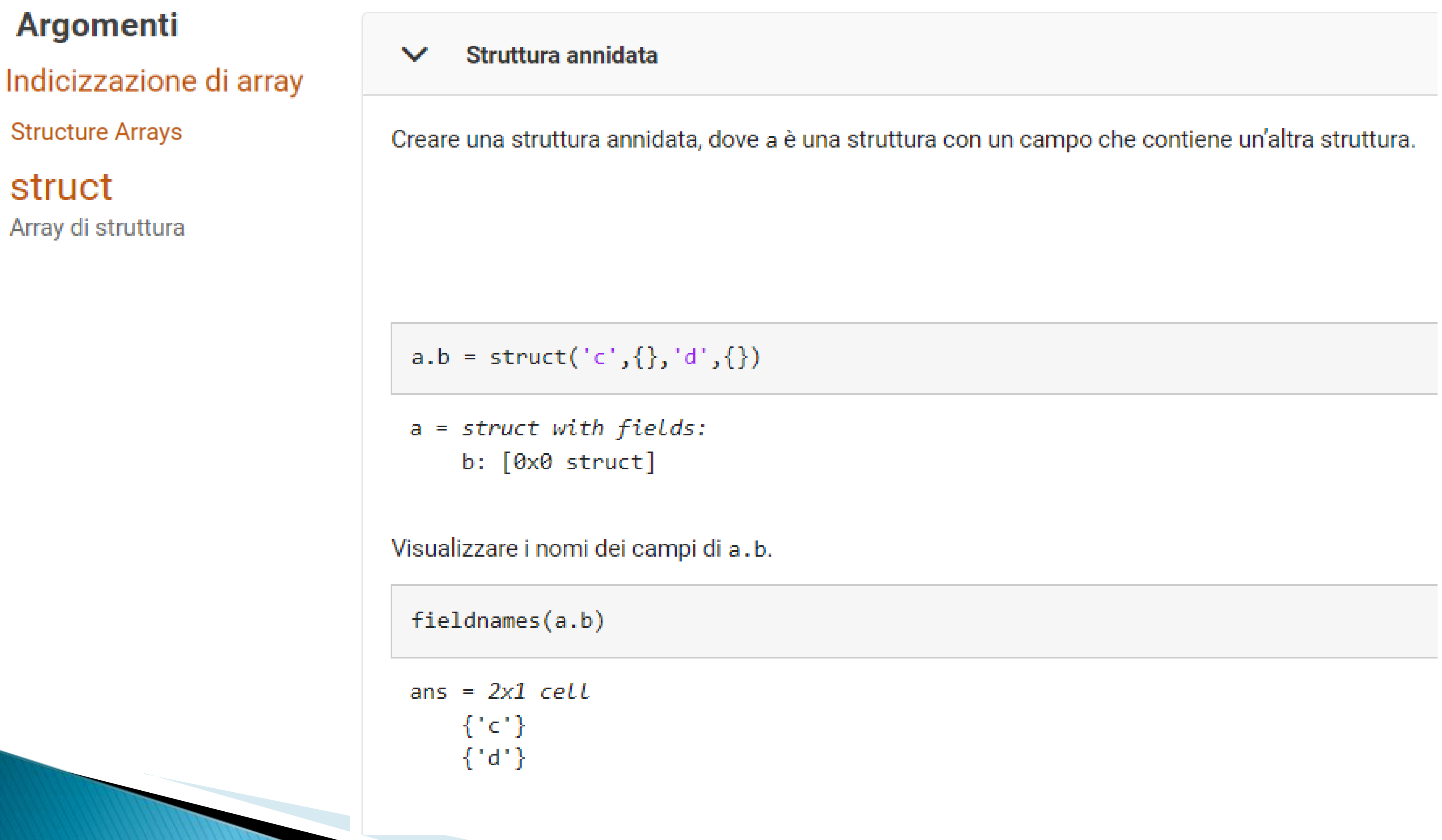
Attribuire un valore a un campo in una struttura vuota.

```
s(1).a = 'a'
```

```
s = struct with fields:
    a: 'a'
    b: []
    c: []
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



**Argomenti**

Indicizzazione di array

Structure Arrays

struct

Array di struttura

**Struttura annidata**

Creare una struttura annidata, dove a è una struttura con un campo che contiene un'altra struttura.

```
a.b = struct('c',{},'d',{})
```

```
a = struct with fields:
    b: [0x0 struct]
```

Visualizzare i nomi dei campi di `a.b`.

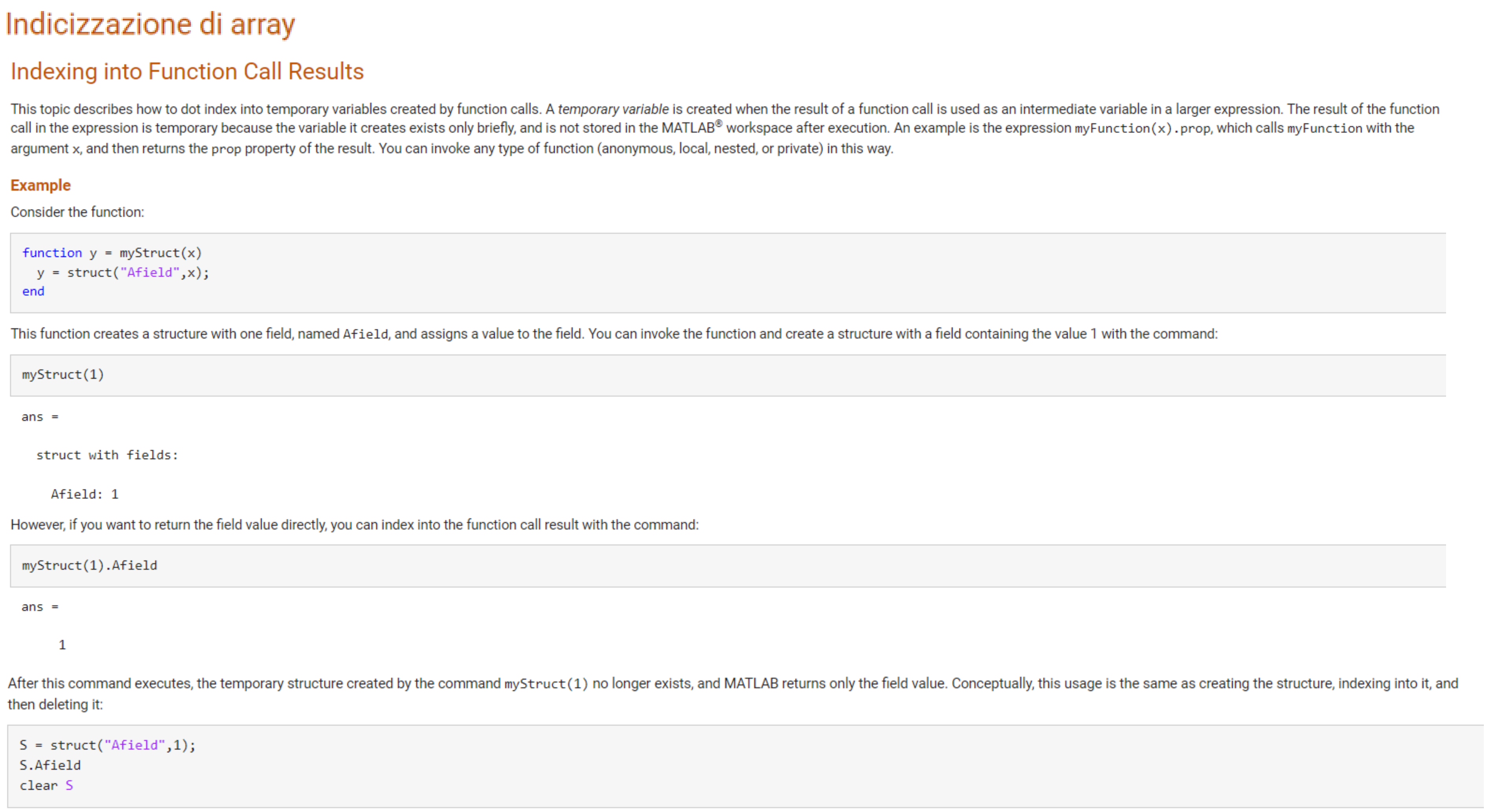
```
fieldnames(a.b)
```

```
ans = 2x1 cell
    {'c'}
    {'d'}
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

**Argomenti**



## Indicizzazione di array

### Indexing into Function Call Results

This topic describes how to dot index into temporary variables created by function calls. A *temporary variable* is created when the result of a function call is used as an intermediate variable in a larger expression. The result of the function call in the expression is temporary because the variable it creates exists only briefly, and is not stored in the MATLAB® workspace after execution. An example is the expression `myFunction(x).prop`, which calls `myFunction` with the argument `x`, and then returns the `prop` property of the result. You can invoke any type of function (anonymous, local, nested, or private) in this way.

**Example**

Consider the function:

```
function y = myStruct(x)
  y = struct("Afield",x);
end
```

This function creates a structure with one field, named `Afield`, and assigns a value to the field. You can invoke the function and create a structure with a field containing the value 1 with the command:

```
myStruct(1)
```

```
ans = 

  struct with fields:

    Afield: 1
```

However, if you want to return the field value directly, you can index into the function call result with the command:

```
myStruct(1).Afield
```

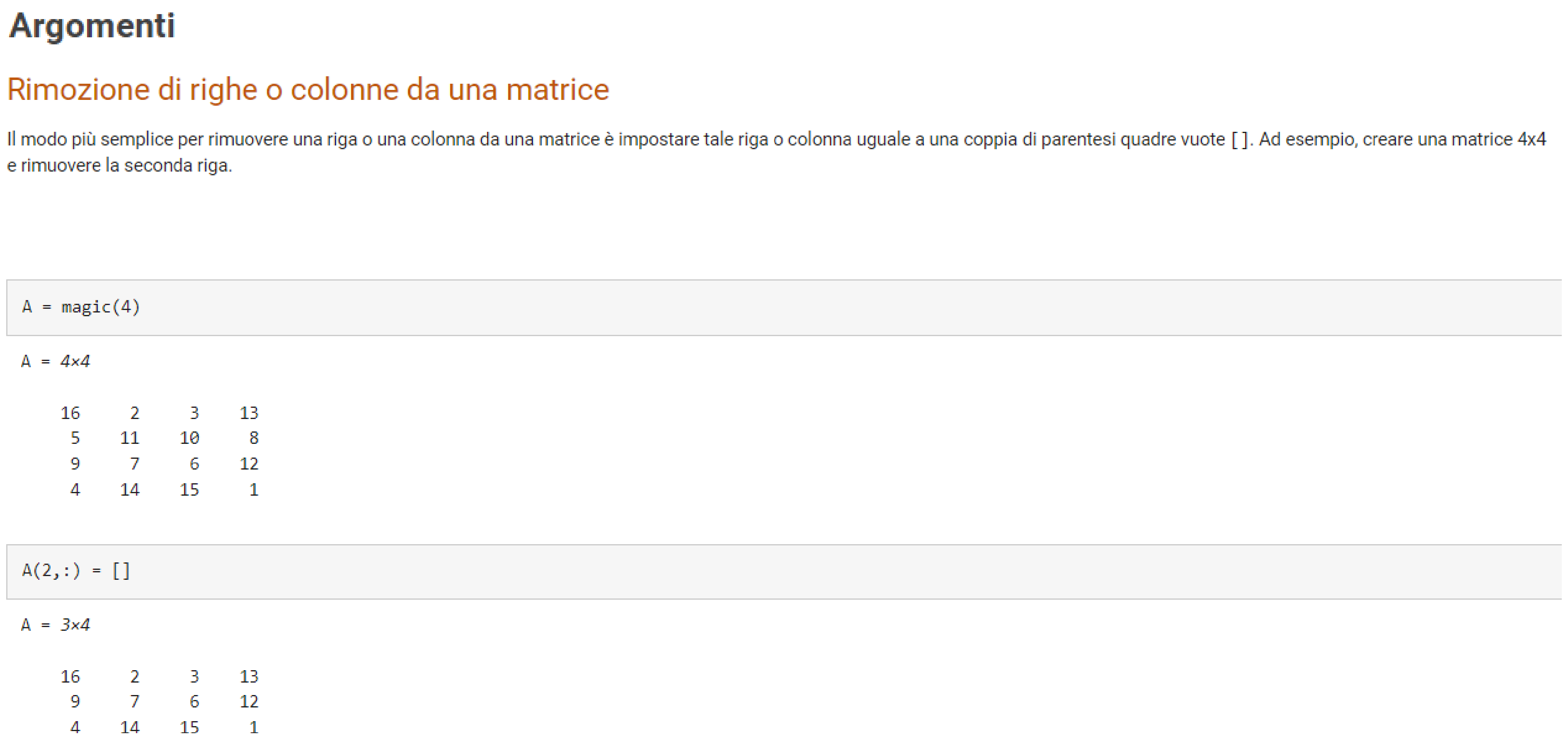
```
ans =

     1
```

After this command executes, the temporary structure created by the command `myStruct(1)` no longer exists, and MATLAB returns only the field value. Conceptually, this usage is the same as creating the structure, indexing into it, and then deleting it:

```
S = struct("Afield",1);
S.Afield
clear S
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Rimozione di righe o colonne da una matrice

Il modo più semplice per rimuovere una riga o una colonna da una matrice è impostare tale riga o colonna uguale a una coppia di parentesi quadre vuote [ ]. Ad esempio, creare una matrice 4x4 e rimuovere la seconda riga.

```
A = magic(4)
```

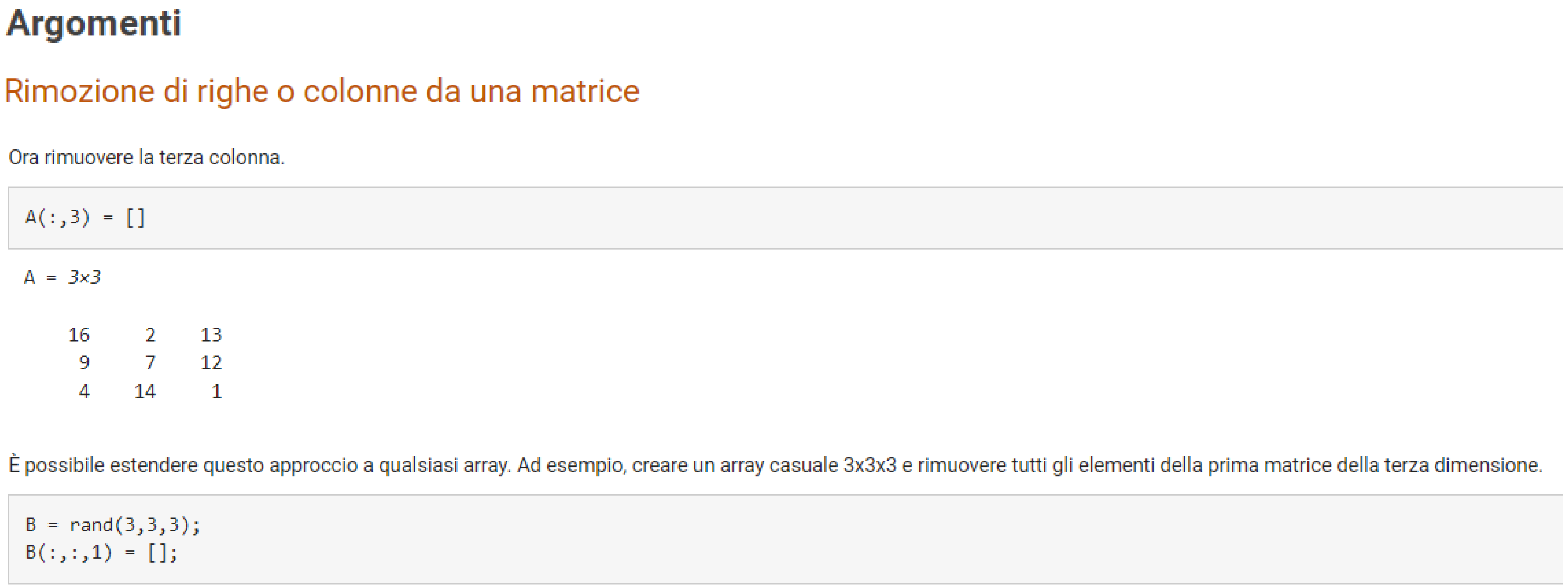
```
A = 4×4

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
A(2,:) = []
```

```
A = 3×4

    16     2     3    13
     9     7     6    12
     4    14    15     1
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



**Argomenti**

## Rimozione di righe o colonne da una matrice

Ora rimuovere la terza colonna.

```
A(:,3) = []
```

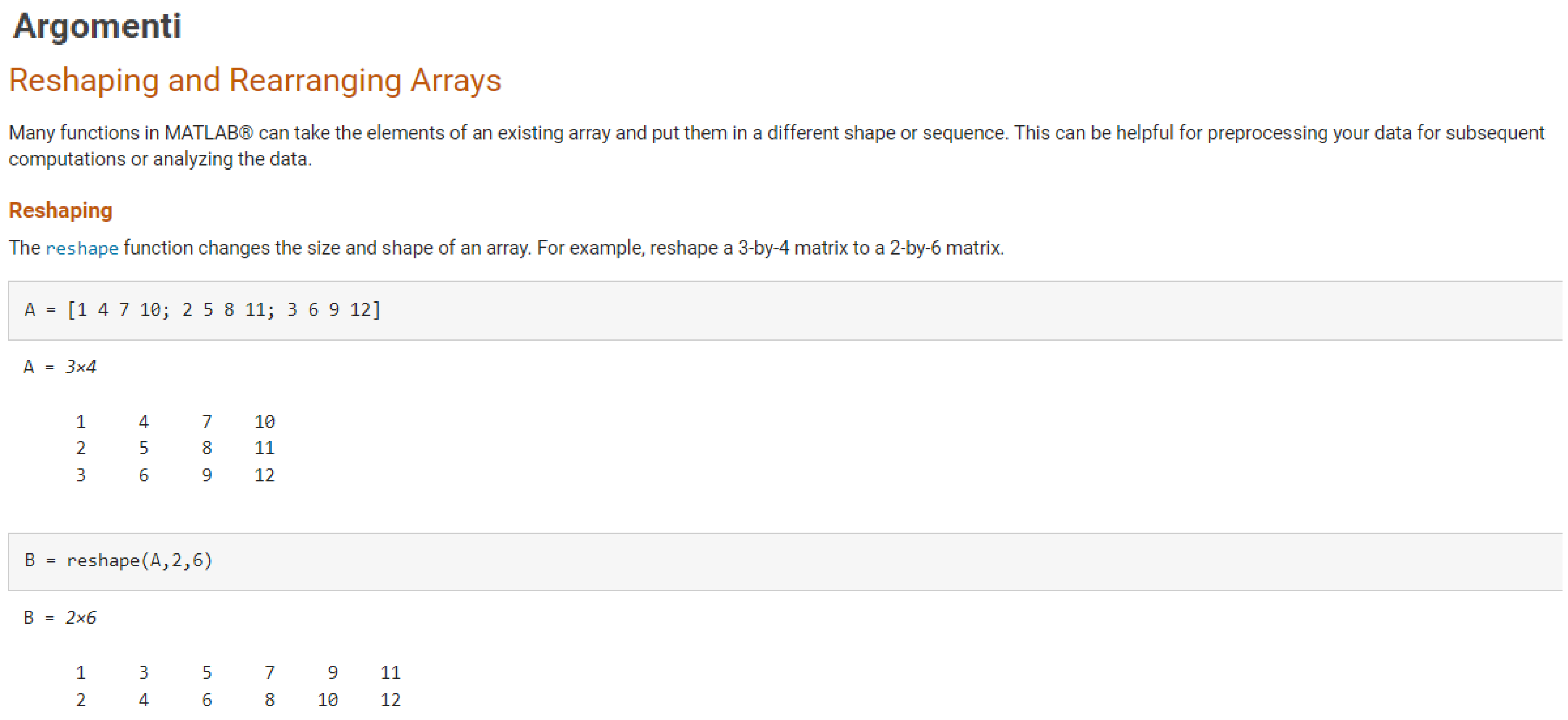
```
A = 3×3

    16     2    13
     9     7    12
     4    14     1
```

È possibile estendere questo approccio a qualsiasi array. Ad esempio, creare un array casuale 3x3x3 e rimuovere tutti gli elementi della prima matrice della terza dimensione.

```
B = rand(3,3,3);
B(:,:,1) = [];
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



## Argomenti

### Reshaping and Rearranging Arrays

Many functions in MATLAB® can take the elements of an existing array and put them in a different shape or sequence. This can be helpful for preprocessing your data for subsequent computations or analyzing the data.

#### Reshaping

The `reshape` function changes the size and shape of an array. For example, reshape a 3-by-4 matrix to a 2-by-6 matrix.

```
A = [1 4 7 10; 2 5 8 11; 3 6 9 12]
```

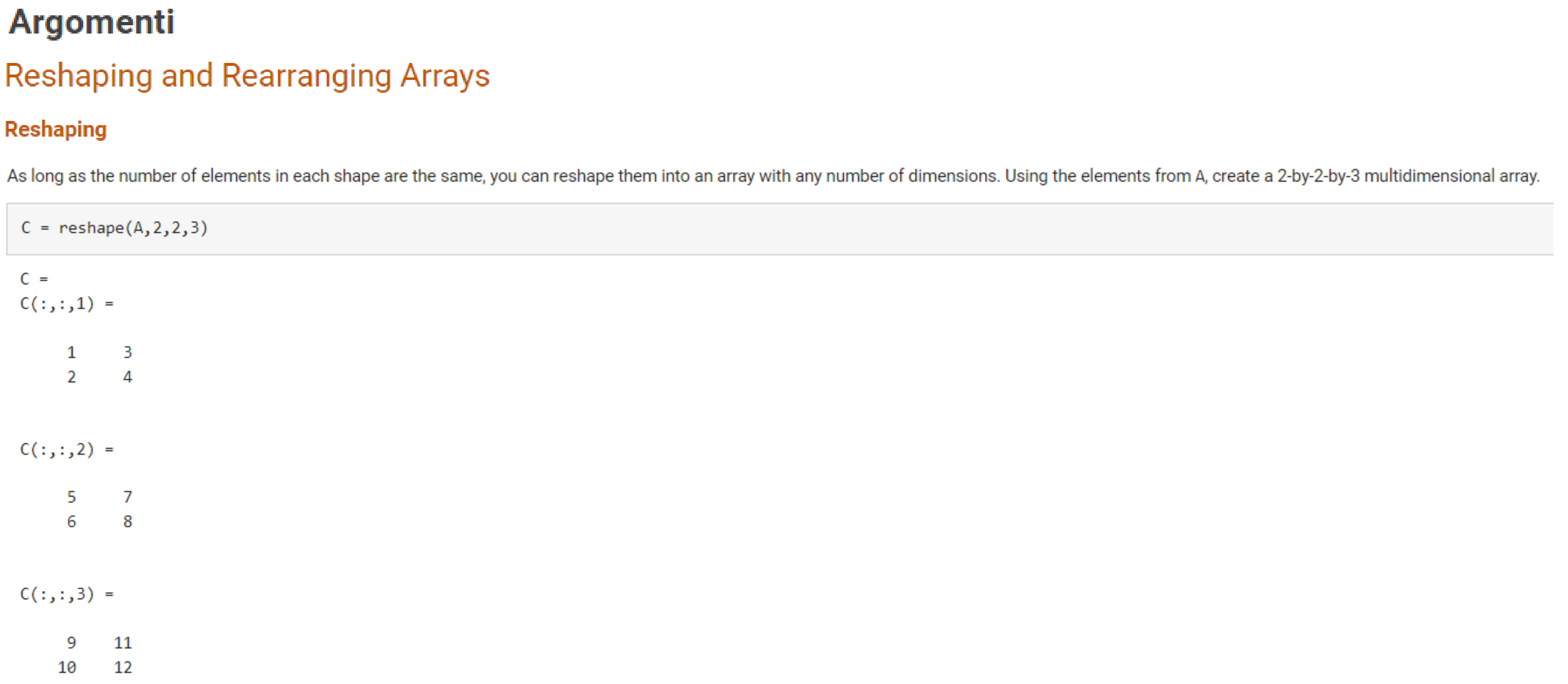
```
A = 3×4

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

```
B = reshape(A,2,6)
```

```
B = 2×6

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

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



## Argomenti

## Reshaping and Rearranging Arrays

### Reshaping

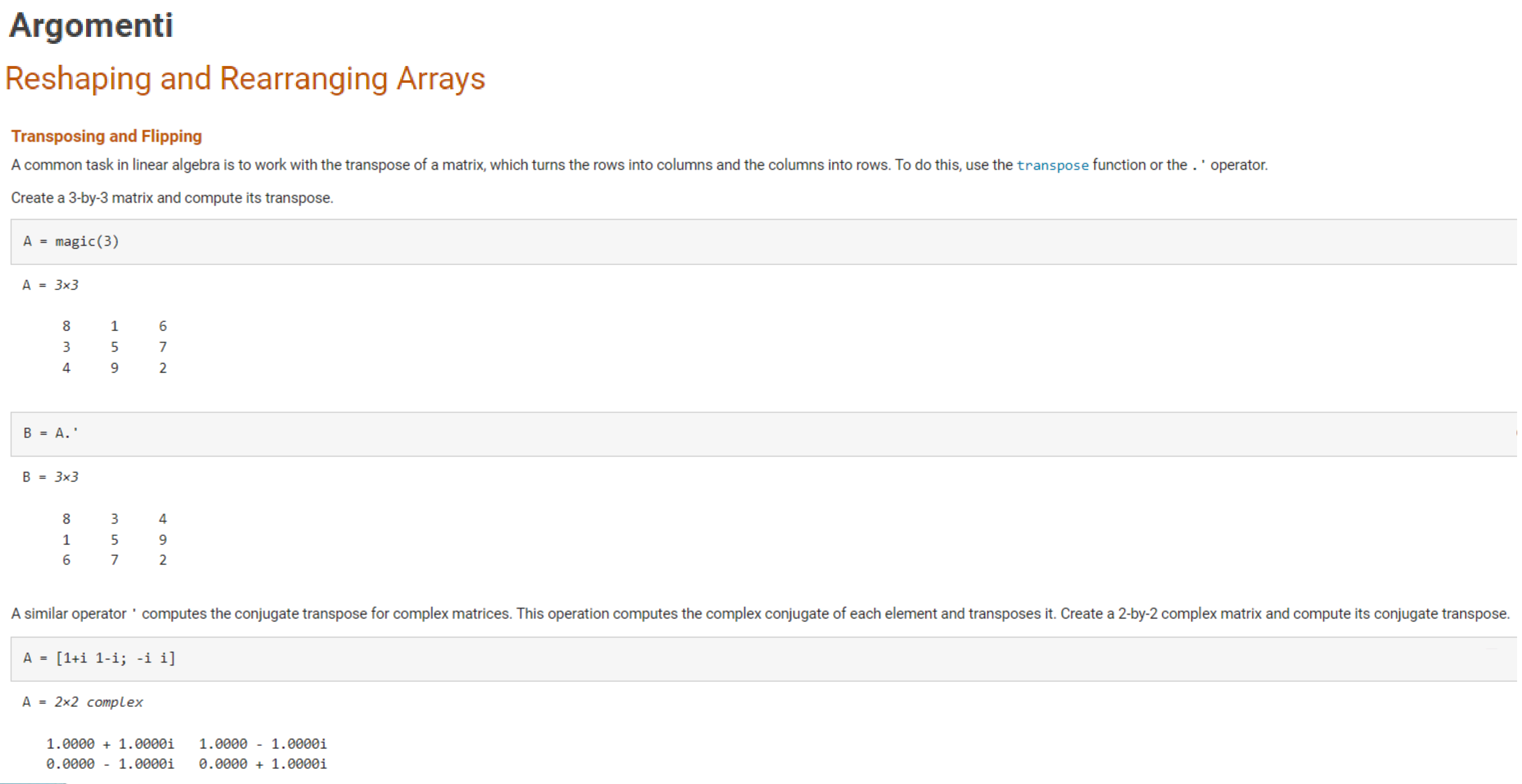
As long as the number of elements in each shape are the same, you can reshape them into an array with any number of dimensions. Using the elements from A, create a 2-by-2-by-3 multidimensional array.

```
C = reshape(A,2,2,3)
```

```
C = 
C(:,:,1) =

     1     3
     2     4


C(:,:,2) =

     5     7
     6     8


C(:,:,3) =

     9    11
    10    12
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Reshaping and Rearranging Arrays

#### Transposing and Flipping

A common task in linear algebra is to work with the transpose of a matrix, which turns the rows into columns and the columns into rows. To do this, use the `transpose` function or the `.'` operator.

Create a 3-by-3 matrix and compute its transpose.

```
A = magic(3)
```

```
A = 3x3

     8     1     6
     3     5     7
     4     9     2
```

```
B = A.'
```

```
B = 3x3

     8     3     4
     1     5     9
     6     7     2
```

A similar operator `'` computes the conjugate transpose for complex matrices. This operation computes the complex conjugate of each element and transposes it. Create a 2-by-2 complex matrix and compute its conjugate transpose.

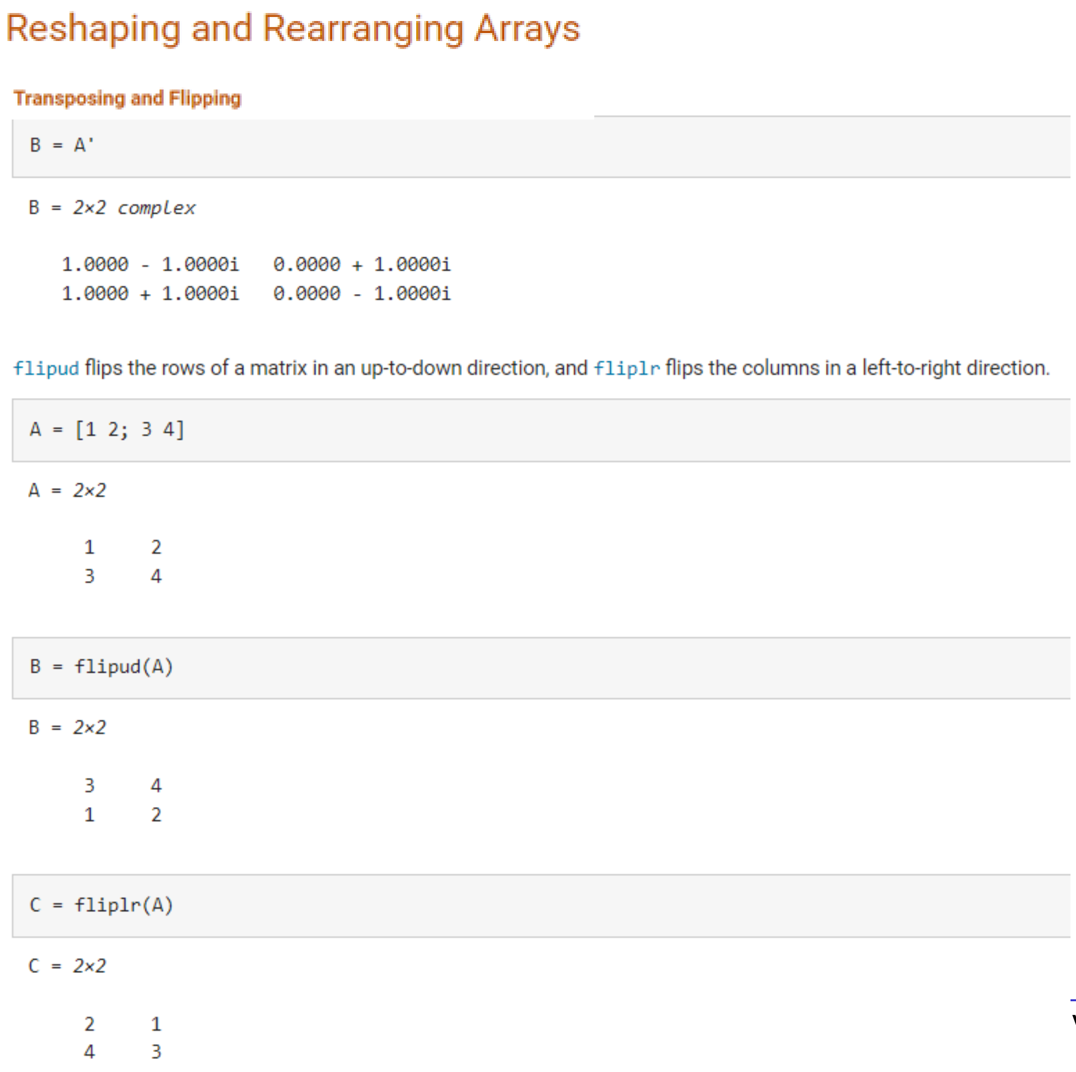
```
A = [1+i 1-i; -i i]
```

```
A = 2x2 complex

   1.0000 + 1.0000i   1.0000 - 1.0000i
   0.0000 - 1.0000i   0.0000 + 1.0000i
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

## Argomenti



### Reshaping and Rearranging Arrays

#### Transposing and Flipping

```
B = A'
```

```
B = 2×2 complex

   1.0000 - 1.0000i   0.0000 + 1.0000i
   1.0000 + 1.0000i   0.0000 - 1.0000i
```

`flipud` flips the rows of a matrix in an up-to-down direction, and `fliplr` flips the columns in a left-to-right direction.

```
A = [1 2; 3 4]
```

```
A = 2×2

     1     2
     3     4
```

```
B = flipud(A)
```

```
B = 2×2

     3     4
     1     2
```

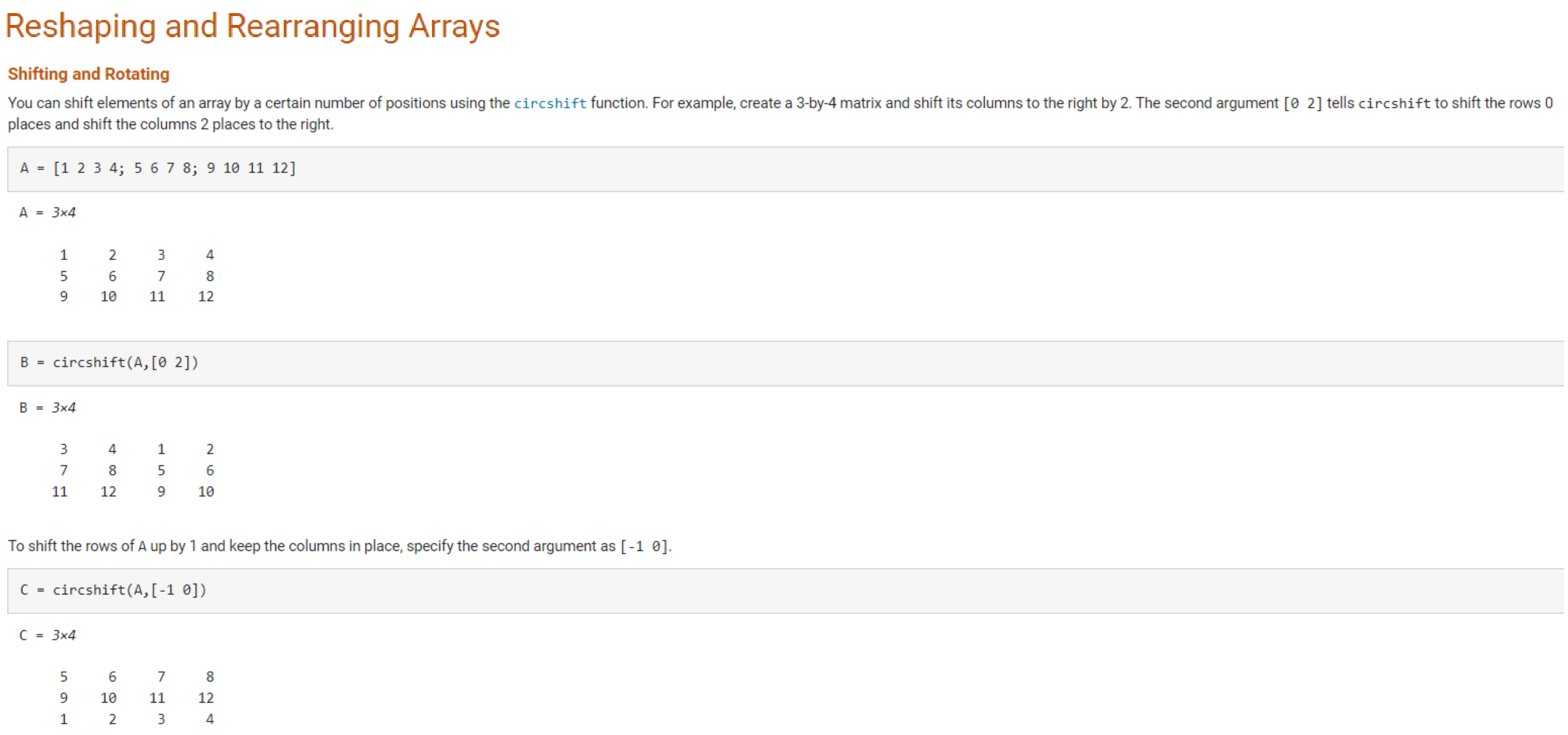
```
C = fliplr(A)
```

```
C = 2×2

     2     1
     4     3
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

## Argomenti



## Reshaping and Rearranging Arrays

### Shifting and Rotating

You can shift elements of an array by a certain number of positions using the `circshift` function. For example, create a 3-by-4 matrix and shift its columns to the right by 2. The second argument `[0 2]` tells `circshift` to shift the rows 0 places and shift the columns 2 places to the right.

```
A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
```

```
A = 3×4

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

```
B = circshift(A,[0 2])
```

```
B = 3×4

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

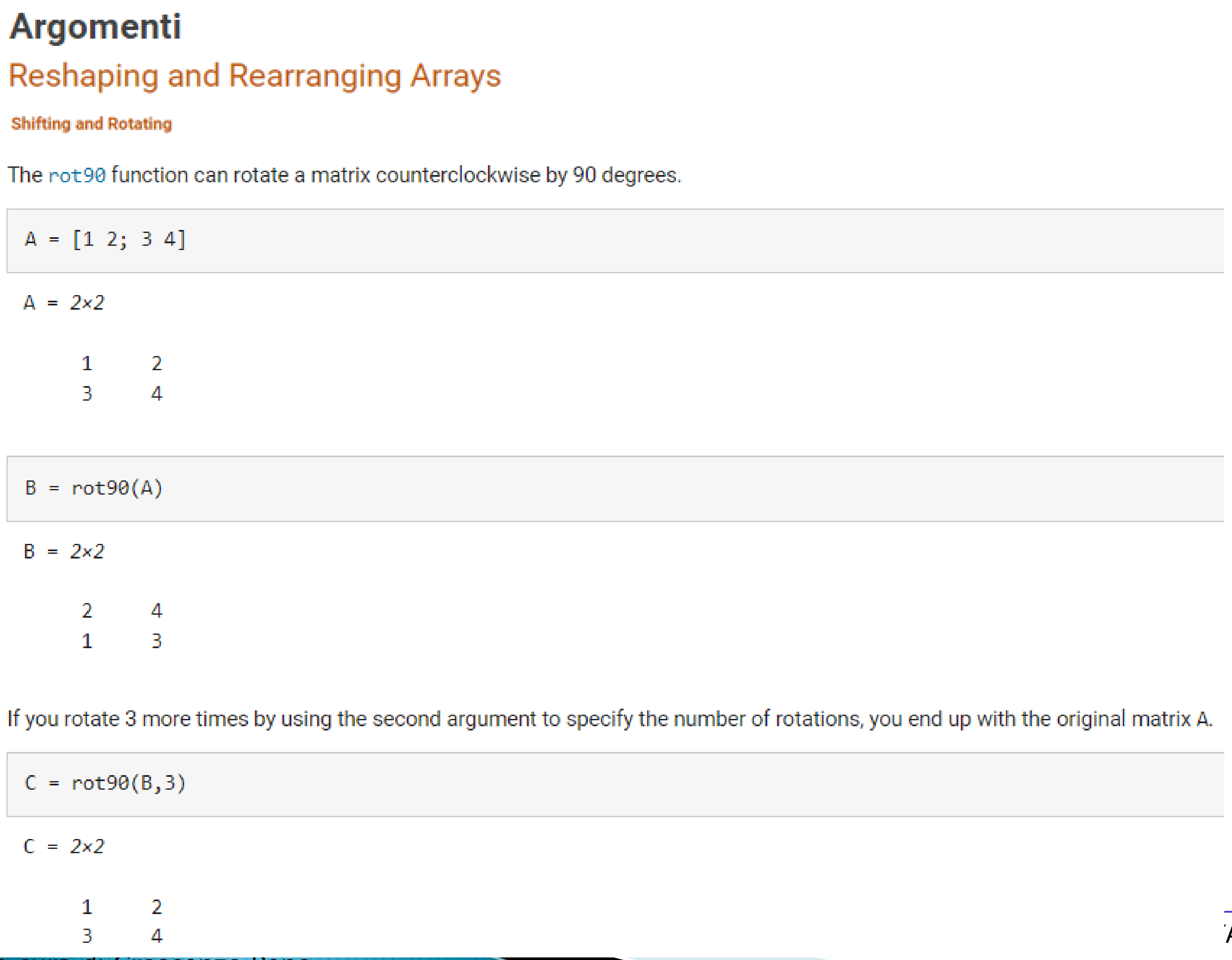
To shift the rows of A up by 1 and keep the columns in place, specify the second argument as `[-1 0]`.

```
C = circshift(A,[-1 0])
```

```
C = 3×4

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

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



## Argomenti

### Reshaping and Rearranging Arrays

#### Shifting and Rotating

The `rot90` function can rotate a matrix counterclockwise by 90 degrees.

```
A = [1 2; 3 4]
```

```
A = 2×2

     1     2
     3     4
```

```
B = rot90(A)
```

```
B = 2×2

     2     4
     1     3
```

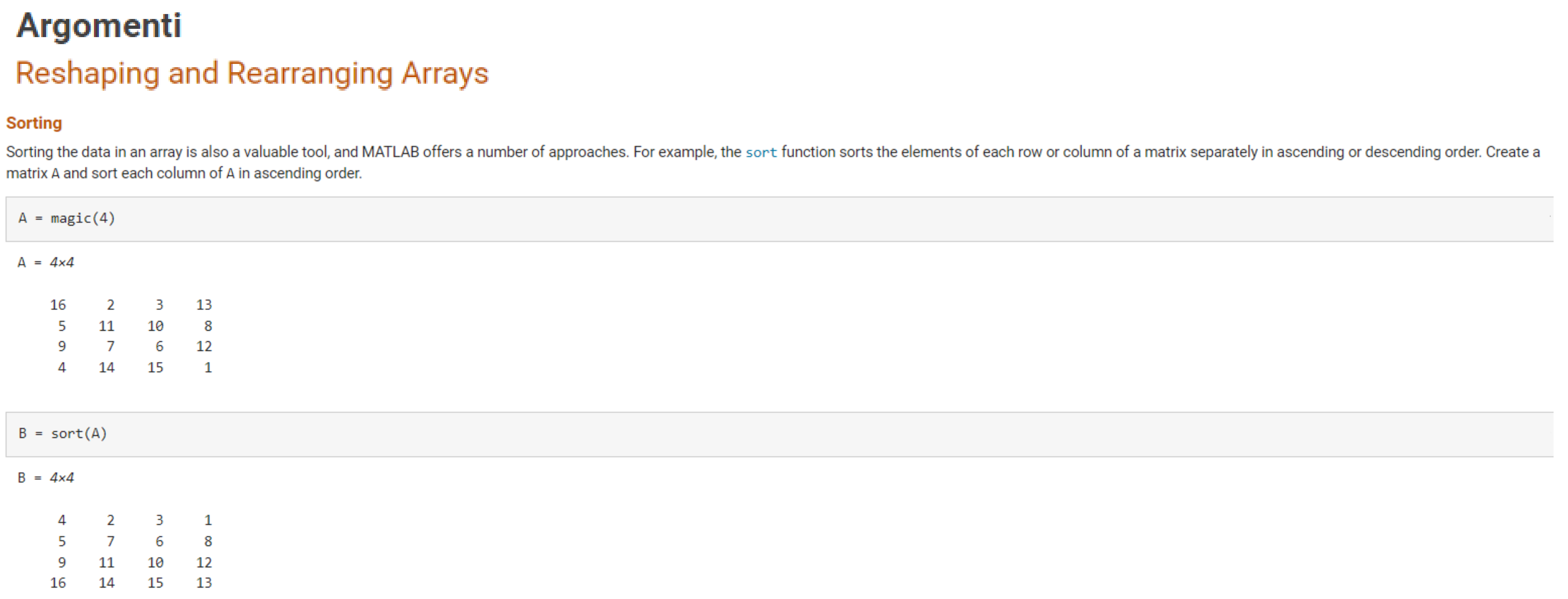
If you rotate 3 more times by using the second argument to specify the number of rotations, you end up with the original matrix `A`.

```
C = rot90(B,3)
```

```
C = 2×2

     1     2
     3     4
```

## ESERCITAZIONI
## MATLAB – Nozioni fondamentali

*Matrici e array*



**Argomenti**

### Reshaping and Rearranging Arrays

**Sorting**

Sorting the data in an array is also a valuable tool, and MATLAB offers a number of approaches. For example, the `sort` function sorts the elements of each row or column of a matrix separately in ascending or descending order. Create a matrix A and sort each column of A in ascending order.

```
A = magic(4)
```

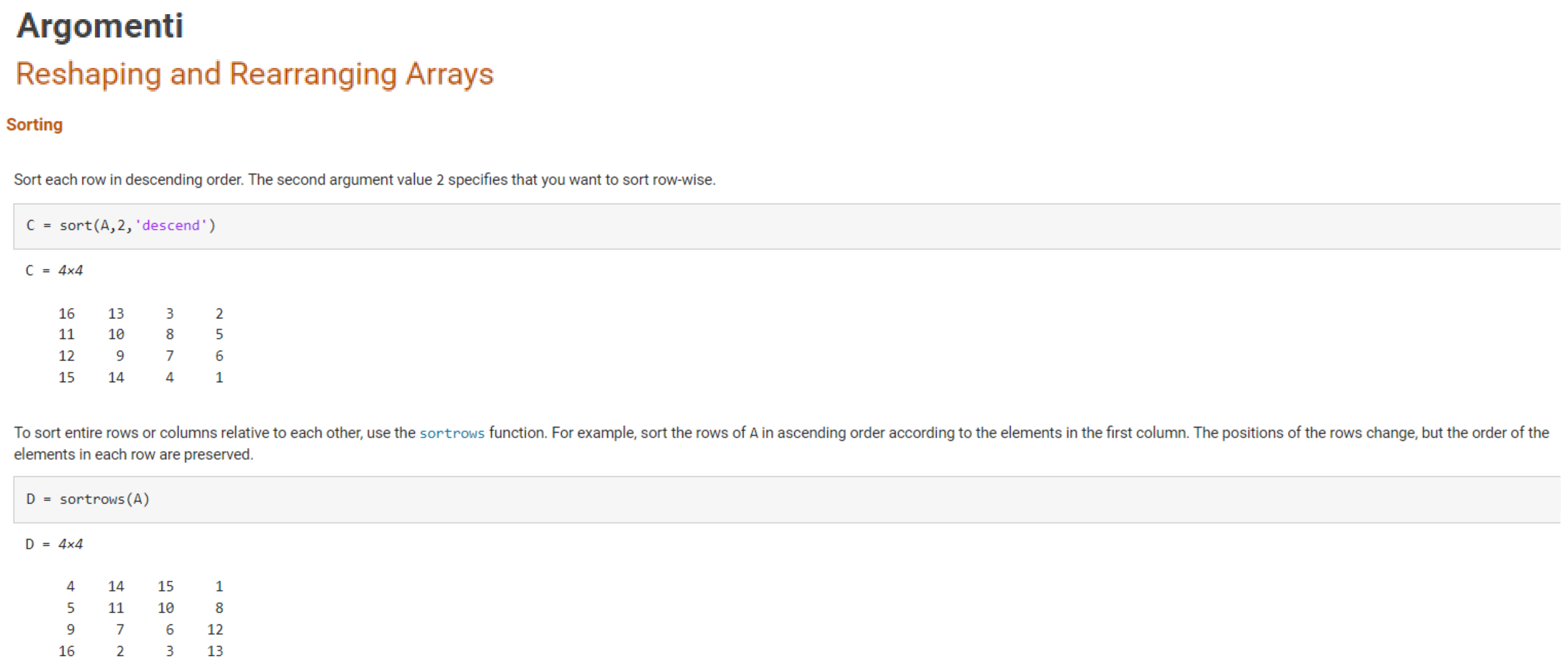
```
A = 4×4

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
B = sort(A)
```

```
B = 4×4

     4     2     3     1
     5     7     6     8
     9    11    10    12
    16    14    15    13
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Reshaping and Rearranging Arrays

#### Sorting

Sort each row in descending order. The second argument value 2 specifies that you want to sort row-wise.

```
C = sort(A,2,'descend')
```

```
C = 4×4

    16    13     3     2
    11    10     8     5
    12     9     7     6
    15    14     4     1
```

To sort entire rows or columns relative to each other, use the `sortrows` function. For example, sort the rows of A in ascending order according to the elements in the first column. The positions of the rows change, but the order of the elements in each row are preserved.

```
D = sortrows(A)
```

```
D = 4×4

     4    14    15     1
     5    11    10     8
     9     7     6    12
    16     2     3    13
```

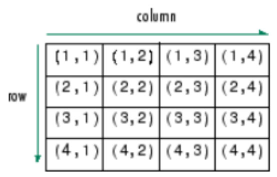
# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*

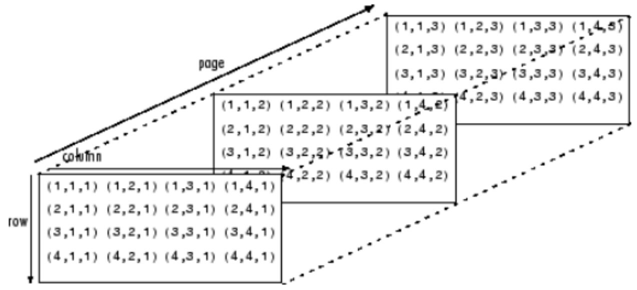
### Argomenti

### Multidimensional Arrays

A multidimensional array in MATLAB® is an array with more than two dimensions. In a matrix, the two dimensions are represented by rows and columns.

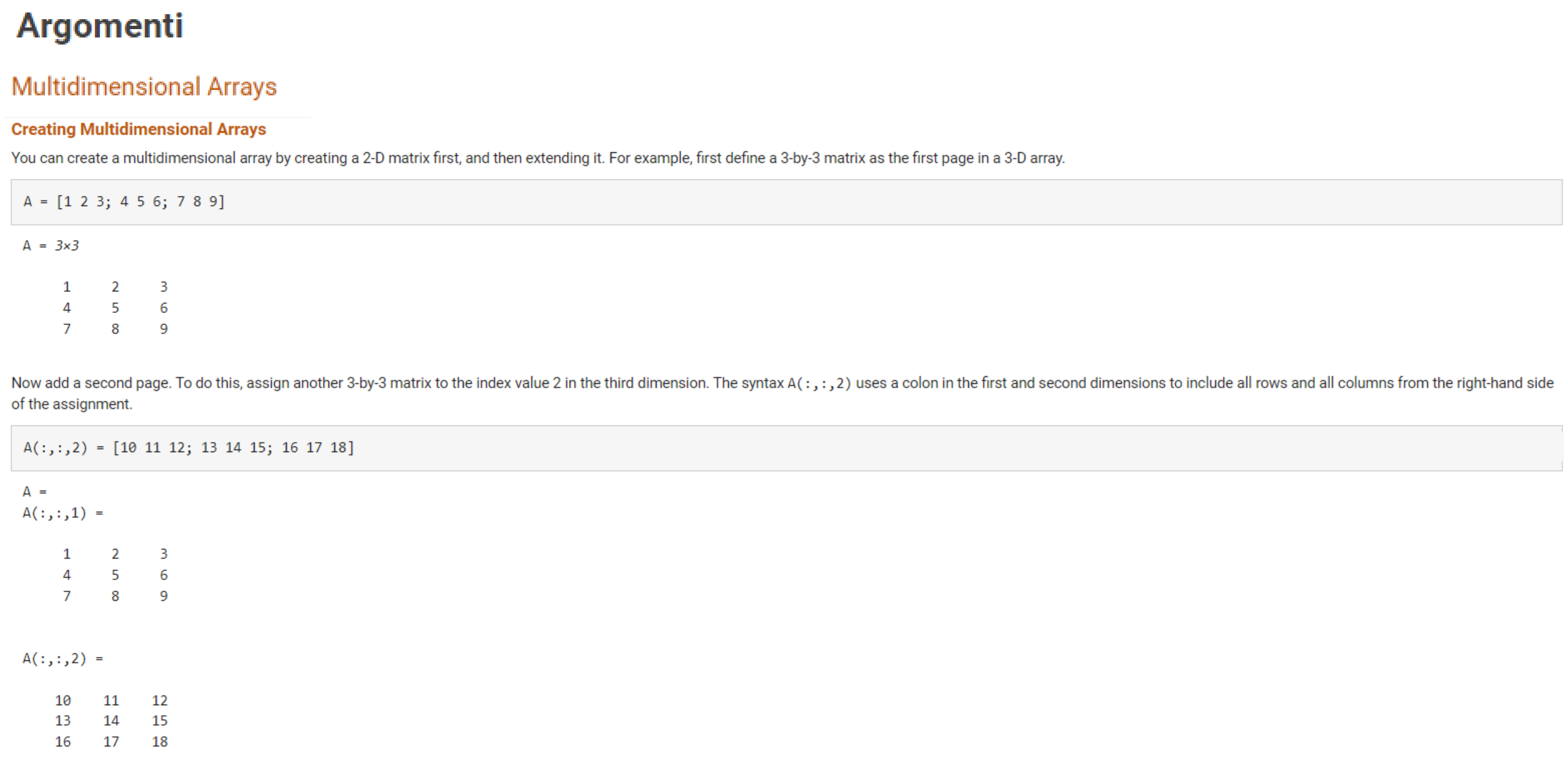


Each element is defined by two subscripts, the row index and the column index. Multidimensional arrays are an extension of 2-D matrices and use additional subscripts for indexing. A 3-D array, for example, uses three subscripts. The first two are just like a matrix, but the third dimension represents *pages* or *sheets* of elements.



# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



## Argomenti

### Multidimensional Arrays

#### Creating Multidimensional Arrays

You can create a multidimensional array by creating a 2-D matrix first, and then extending it. For example, first define a 3-by-3 matrix as the first page in a 3-D array.

```
A = [1 2 3; 4 5 6; 7 8 9]
```

```
A = 3x3

     1     2     3
     4     5     6
     7     8     9
```

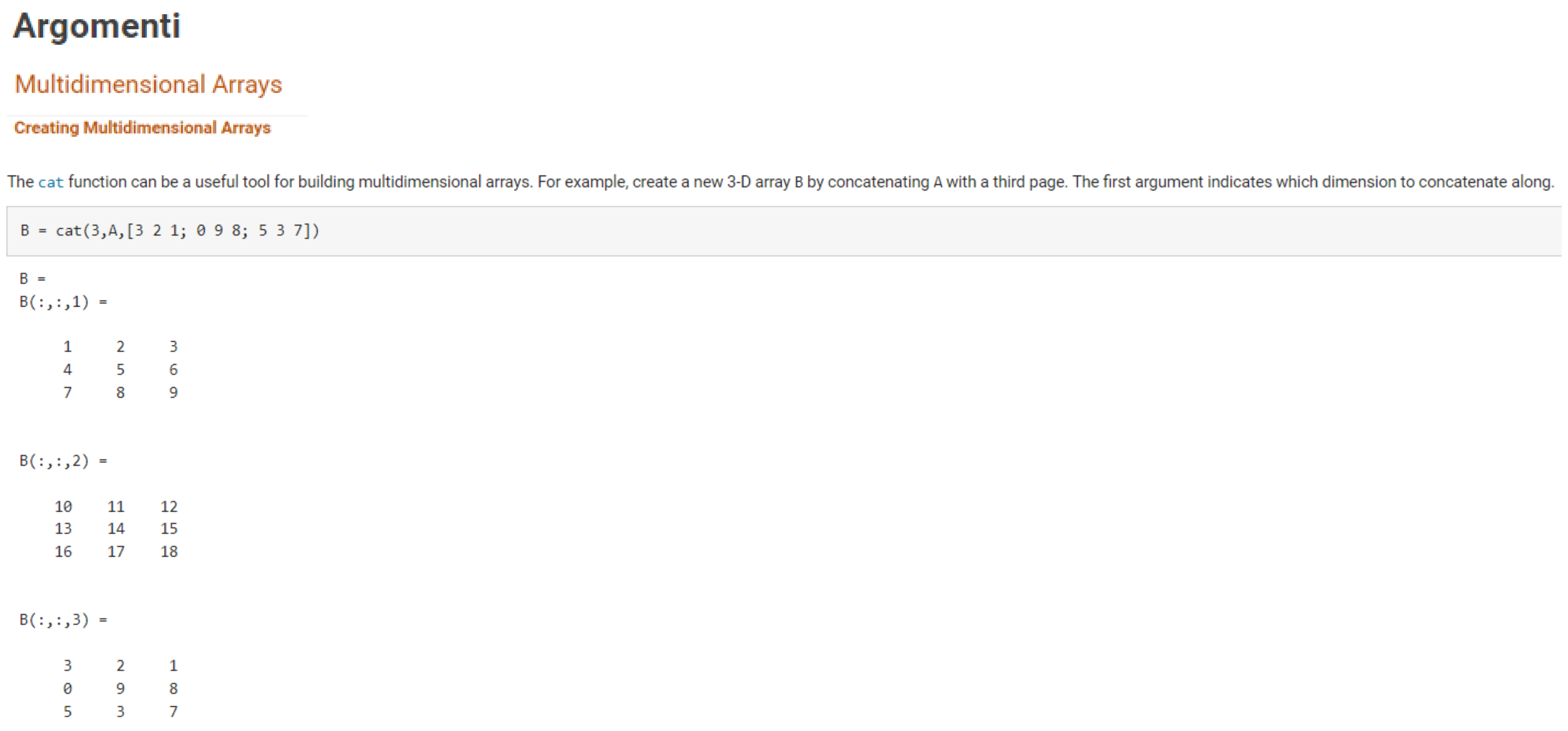
Now add a second page. To do this, assign another 3-by-3 matrix to the index value 2 in the third dimension. The syntax `A(:,:,2)` uses a colon in the first and second dimensions to include all rows and all columns from the right-hand side of the assignment.

```
A(:,:,2) = [10 11 12; 13 14 15; 16 17 18]
```

```
A =
A(:,:,1) =

     1     2     3
     4     5     6
     7     8     9


A(:,:,2) =

    10    11    12
    13    14    15
    16    17    18
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

#### Multidimensional Arrays

##### Creating Multidimensional Arrays

The `cat` function can be a useful tool for building multidimensional arrays. For example, create a new 3-D array B by concatenating A with a third page. The first argument indicates which dimension to concatenate along.

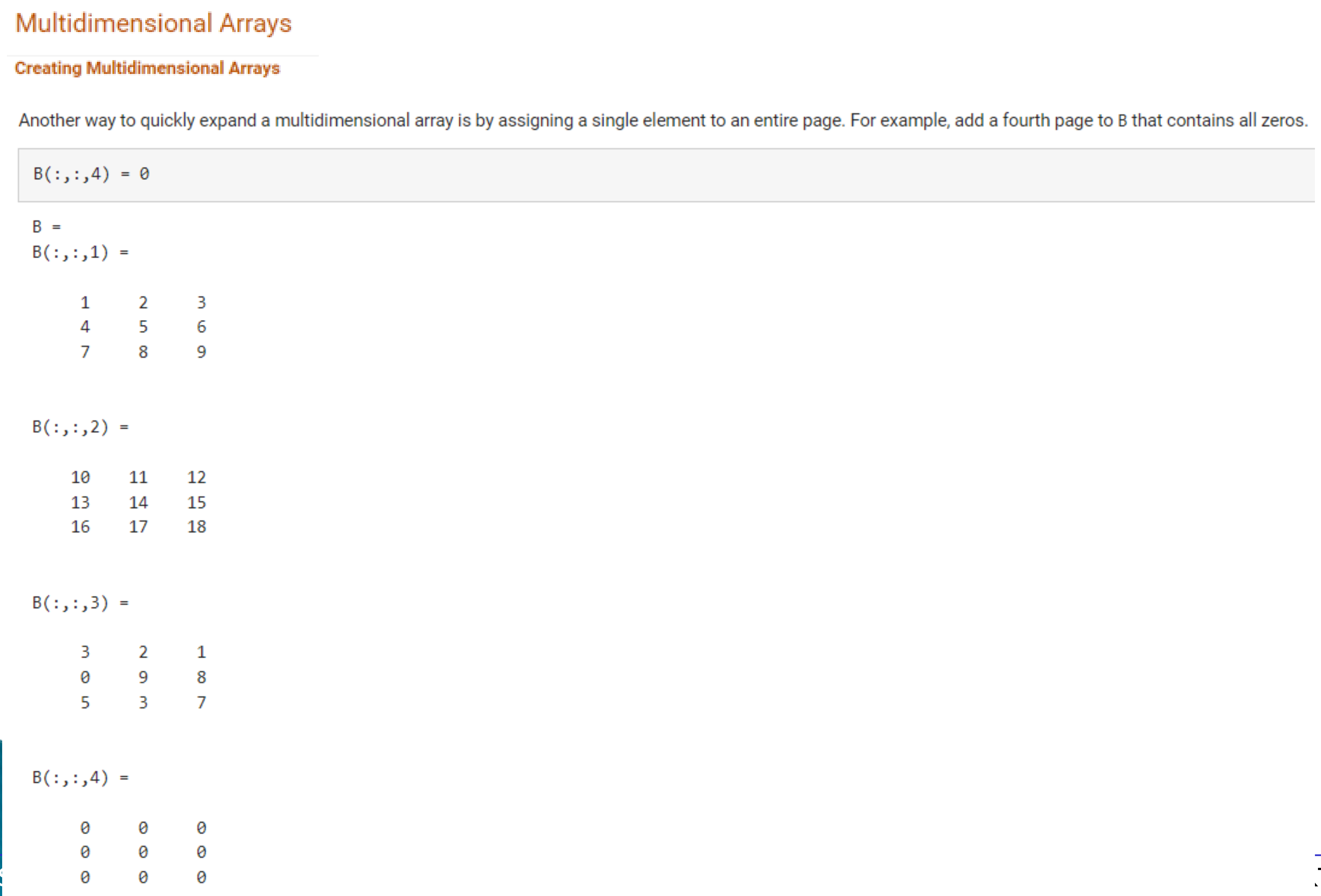
```
B = cat(3,A,[3 2 1; 0 9 8; 5 3 7])
```

```
B =
B(:,:,1) =

     1     2     3
     4     5     6
     7     8     9


B(:,:,2) =

    10    11    12
    13    14    15
    16    17    18


B(:,:,3) =

     3     2     1
     0     9     8
     5     3     7
```

## ESERCITAZIONI
## MATLAB – Nozioni fondamentali

*Matrici e array*

### Argomenti



Multidimensional Arrays

**Creating Multidimensional Arrays**

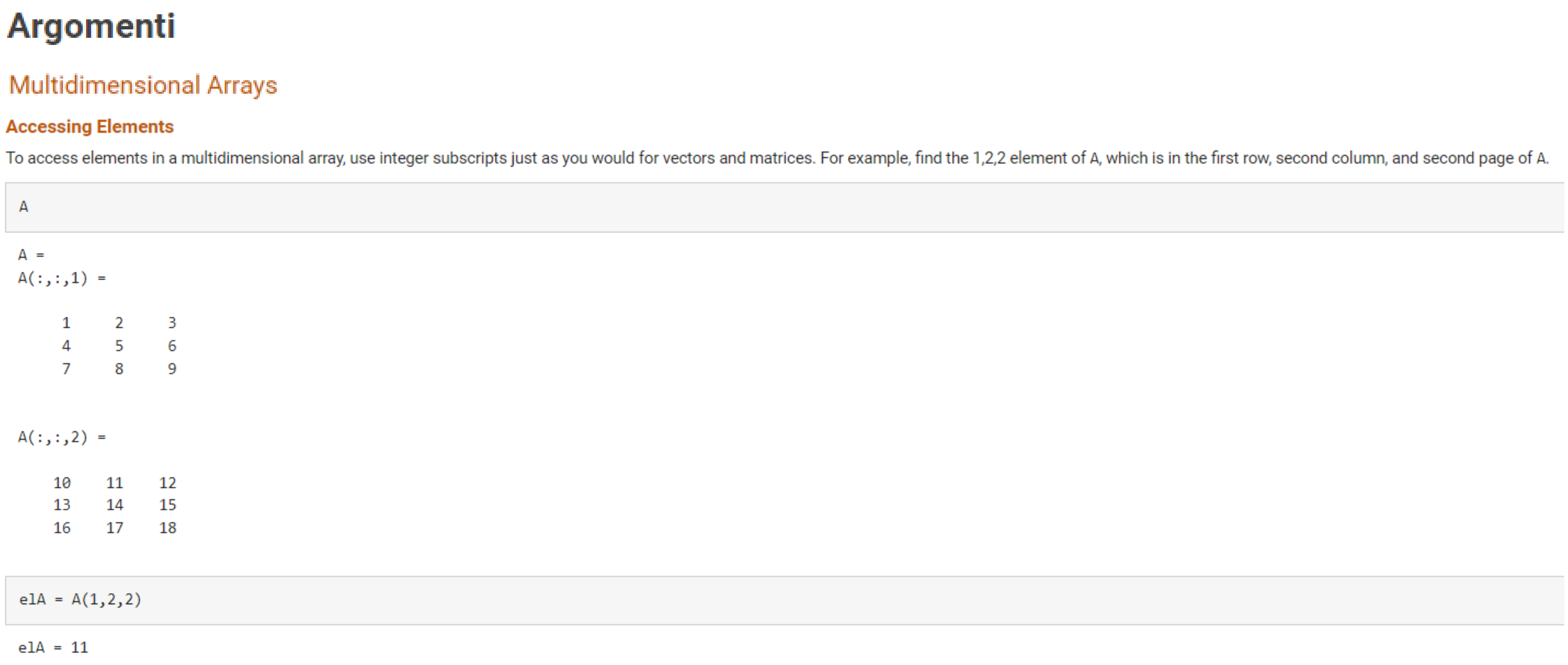
Another way to quickly expand a multidimensional array is by assigning a single element to an entire page. For example, add a fourth page to B that contains all zeros.

```
B(:,:,4) = 0
```

```
B =
B(:,:,1) =

     1     2     3
     4     5     6
     7     8     9


B(:,:,2) =

    10    11    12
    13    14    15
    16    17    18


B(:,:,3) =

     3     2     1
     0     9     8
     5     3     7


B(:,:,4) =

     0     0     0
     0     0     0
     0     0     0
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

#### Multidimensional Arrays

**Accessing Elements**

To access elements in a multidimensional array, use integer subscripts just as you would for vectors and matrices. For example, find the 1,2,2 element of A, which is in the first row, second column, and second page of A.

```
A
```

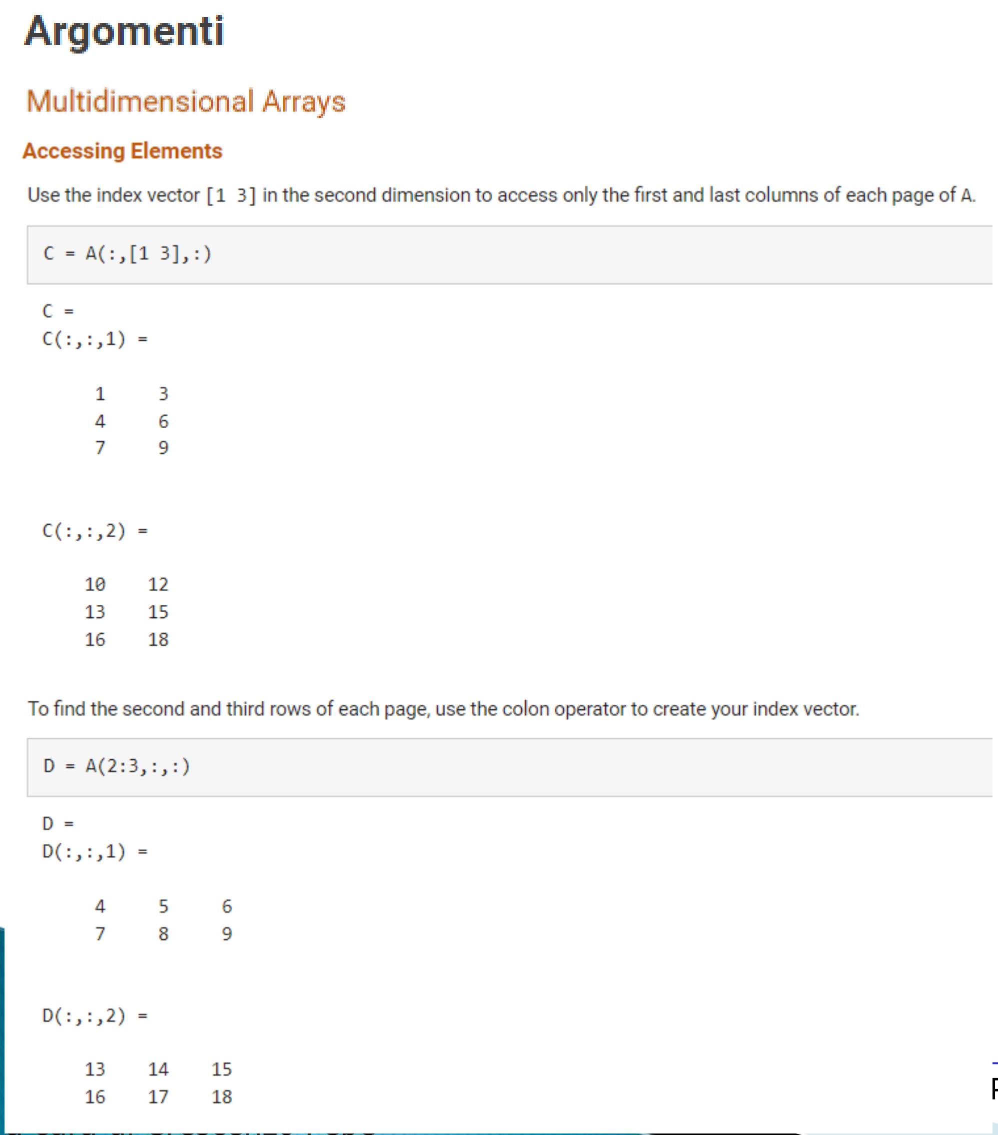
```
A =
A(:,:,1) =

     1     2     3
     4     5     6
     7     8     9


A(:,:,2) =

    10    11    12
    13    14    15
    16    17    18
```

```
elA = A(1,2,2)
```

```
elA = 11
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

#### Multidimensional Arrays

**Accessing Elements**

Use the index vector `[1 3]` in the second dimension to access only the first and last columns of each page of `A`.

```
C = A(:,[1 3],:)
```

```
C = 
C(:,:,1) =

     1     3
     4     6
     7     9


C(:,:,2) =

    10    12
    13    15
    16    18
```

To find the second and third rows of each page, use the colon operator to create your index vector.

```
D = A(2:3,:,:)
```

```
D = 
D(:,:,1) =

     4     5     6
     7     8     9


D(:,:,2) =

    13    14    15
    16    17    18
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

MathWorks®

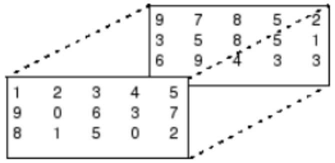
## *Matrici e array*

### Argomenti

### Multidimensional Arrays

#### Manipulating Arrays

Elements of multidimensional arrays can be moved around in many ways, similar to vectors and matrices. `reshape`, `permute`, and `squeeze` are useful functions for rearranging elements. Consider a 3-D array with two pages.



Reshaping a multidimensional array can be useful for performing certain operations or visualizing the data. Use the `reshape` function to rearrange the elements of the 3-D array into a 6-by-5 matrix.

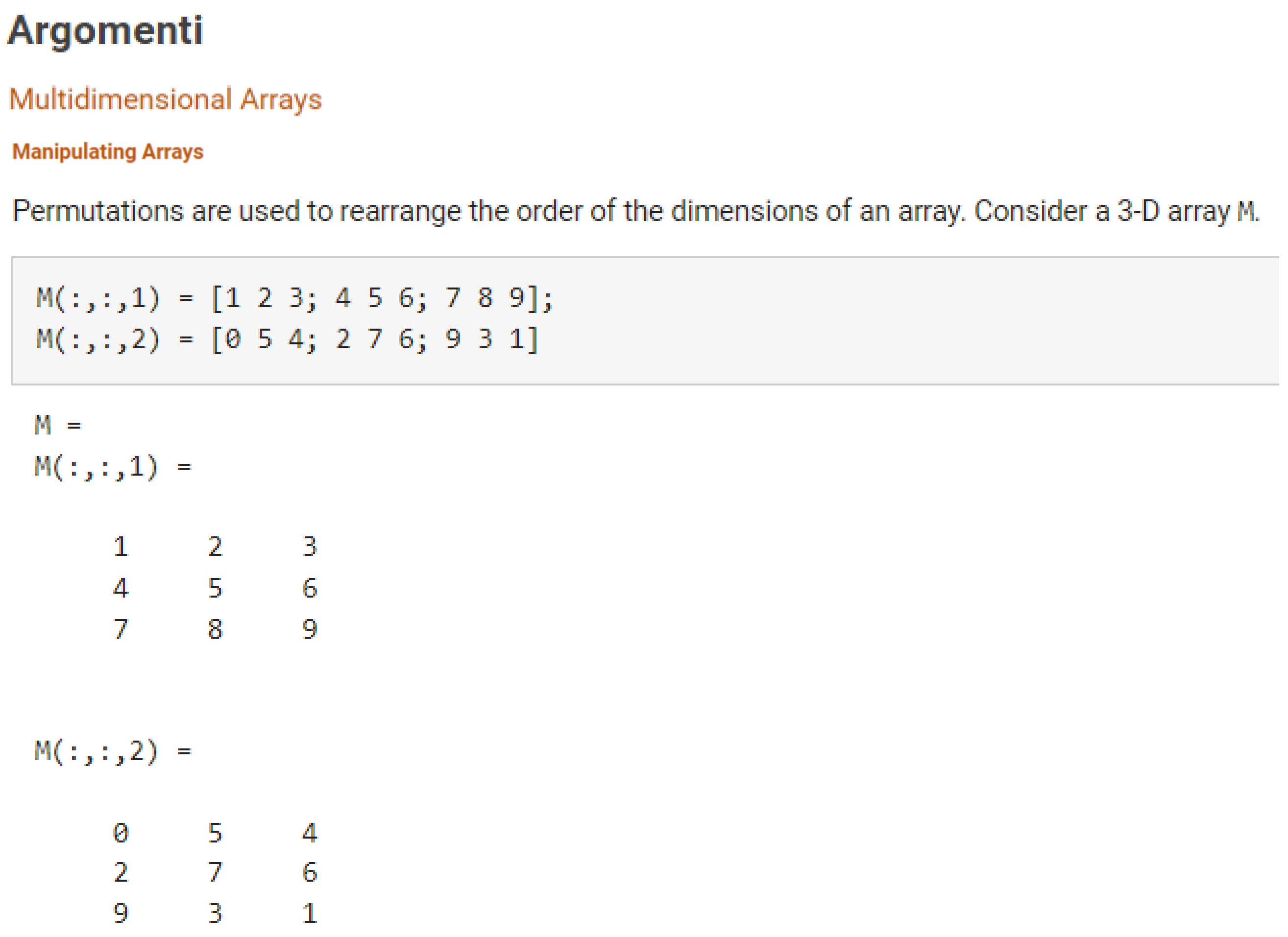
```
A = [1 2 3 4 5; 9 0 6 3 7; 8 1 5 0 2];
A(:,:,2) = [9 7 8 5 2; 3 5 8 5 1; 6 9 4 3 3];
B = reshape(A,[6 5])
```

```
B = 6×5

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

`reshape` operates columnwise, creating the new matrix by taking consecutive elements down each column of `A`, starting with the first page then moving to the second page.

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

Multidimensional Arrays

**Manipulating Arrays**

Permutations are used to rearrange the order of the dimensions of an array. Consider a 3-D array `M`.

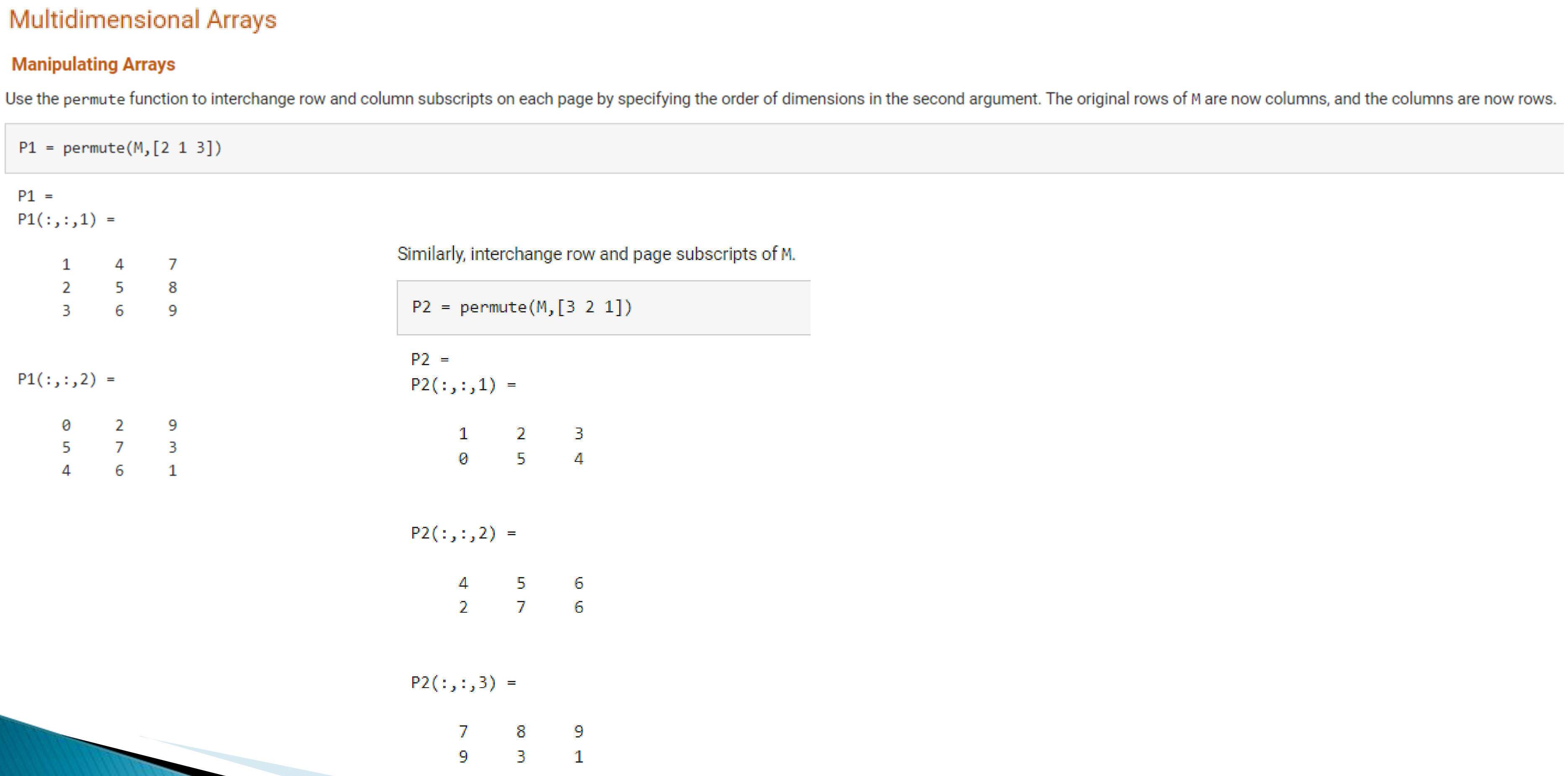
```
M(:,:,1) = [1 2 3; 4 5 6; 7 8 9];
M(:,:,2) = [0 5 4; 2 7 6; 9 3 1]
```

```
M =
M(:,:,1) =

     1     2     3
     4     5     6
     7     8     9


M(:,:,2) =

     0     5     4
     2     7     6
     9     3     1
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*

## Argomenti



### Multidimensional Arrays

#### Manipulating Arrays

Use the `permute` function to interchange row and column subscripts on each page by specifying the order of dimensions in the second argument. The original rows of `M` are now columns, and the columns are now rows.

```
P1 = permute(M,[2 1 3])
```

```
P1 =
P1(:,:,1) =

     1     4     7
     2     5     8
     3     6     9


P1(:,:,2) =

     0     2     9
     5     7     3
     4     6     1
```

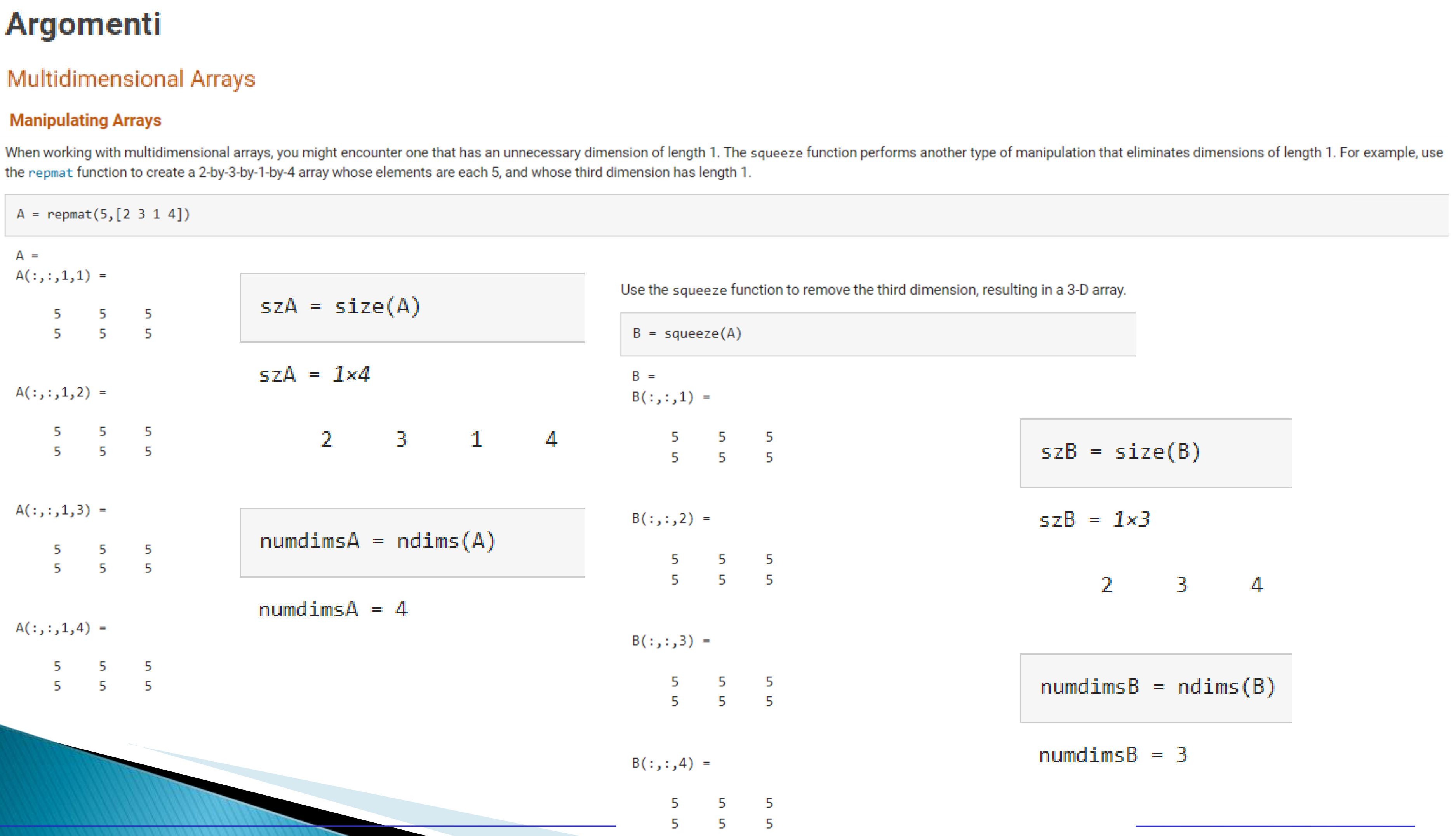
Similarly, interchange row and page subscripts of `M`.

```
P2 = permute(M,[3 2 1])
```

```
P2 =
P2(:,:,1) =

     1     2     3
     0     5     4


P2(:,:,2) =

     4     5     6
     2     7     6


P2(:,:,3) =

     7     8     9
     9     3     1
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Multidimensional Arrays

#### Manipulating Arrays

When working with multidimensional arrays, you might encounter one that has an unnecessary dimension of length 1. The `squeeze` function performs another type of manipulation that eliminates dimensions of length 1. For example, use the `repmat` function to create a 2-by-3-by-1-by-4 array whose elements are each 5, and whose third dimension has length 1.

```
A = repmat(5,[2 3 1 4])
```

```
A =
A(:,:,1,1) =

     5     5     5
     5     5     5


A(:,:,1,2) =

     5     5     5
     5     5     5


A(:,:,1,3) =

     5     5     5
     5     5     5


A(:,:,1,4) =

     5     5     5
     5     5     5
```

```
szA = size(A)
```

```
szA = 1×4

     2     3     1     4
```

```
numdimsA = ndims(A)
```

```
numdimsA = 4
```

Use the `squeeze` function to remove the third dimension, resulting in a 3-D array.

```
B = squeeze(A)
```

```
B =
B(:,:,1) =

     5     5     5
     5     5     5


B(:,:,2) =

     5     5     5
     5     5     5


B(:,:,3) =

     5     5     5
     5     5     5


B(:,:,4) =

     5     5     5
     5     5     5
```

```
szB = size(B)
```

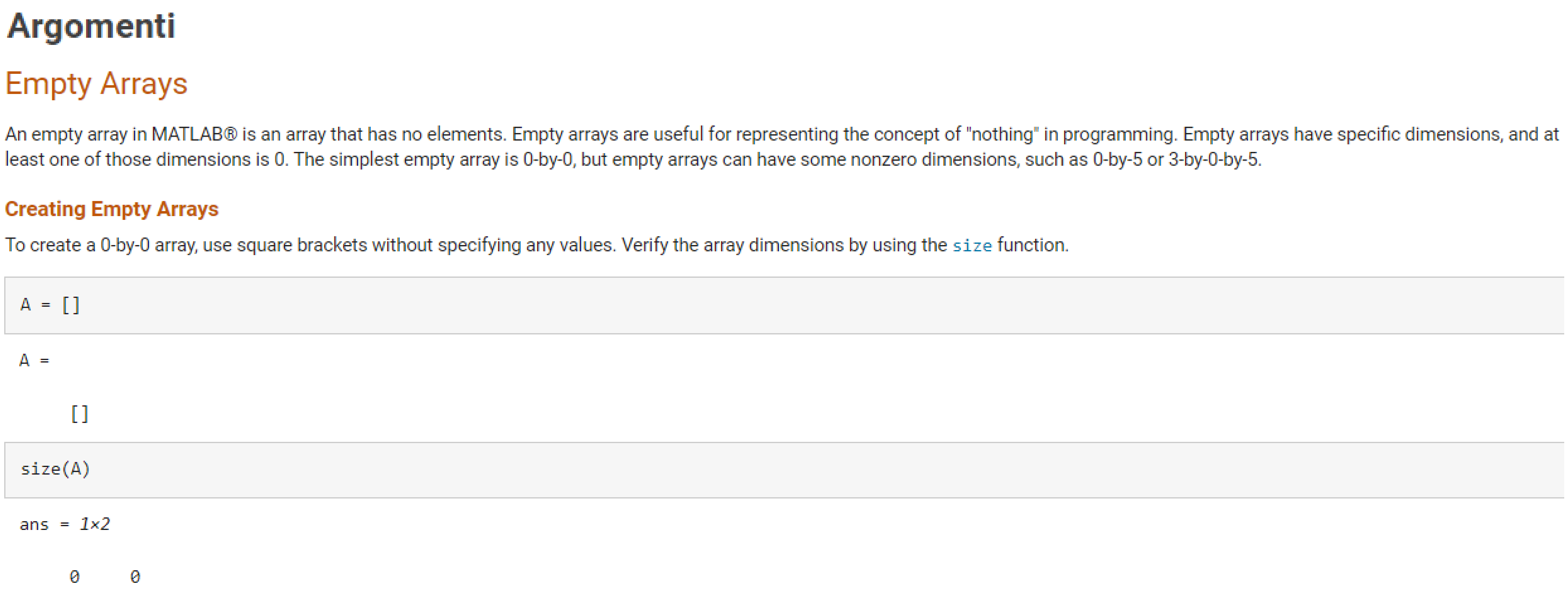
```
szB = 1×3

     2     3     4
```

```
numdimsB = ndims(B)
```

```
numdimsB = 3
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Empty Arrays

An empty array in MATLAB® is an array that has no elements. Empty arrays are useful for representing the concept of "nothing" in programming. Empty arrays have specific dimensions, and at least one of those dimensions is 0. The simplest empty array is 0-by-0, but empty arrays can have some nonzero dimensions, such as 0-by-5 or 3-by-0-by-5.

#### Creating Empty Arrays

To create a 0-by-0 array, use square brackets without specifying any values. Verify the array dimensions by using the `size` function.

```
A = []
```

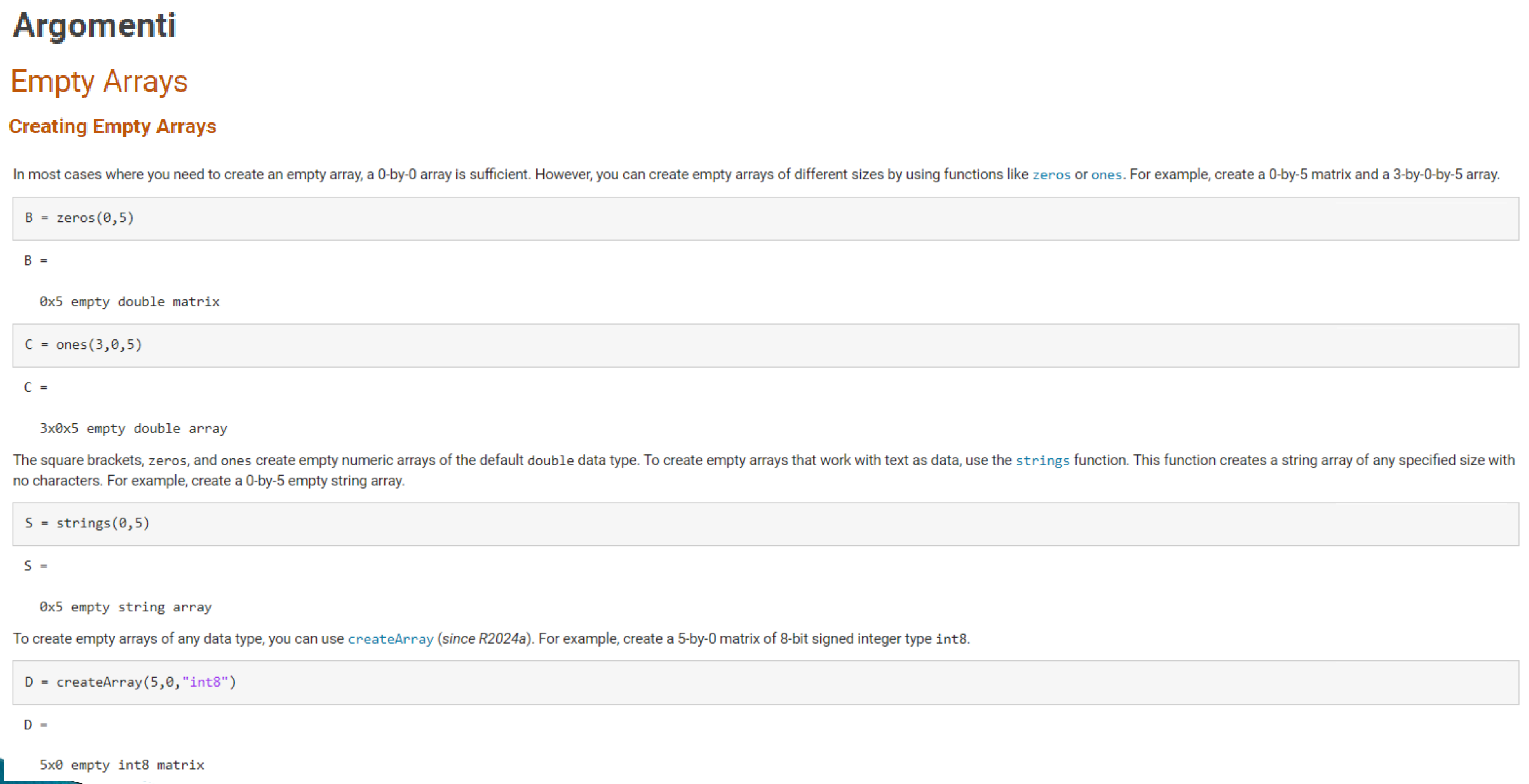
```
A =

     []
```

```
size(A)
```

```
ans = 1×2

     0     0
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Empty Arrays

#### Creating Empty Arrays

In most cases where you need to create an empty array, a 0-by-0 array is sufficient. However, you can create empty arrays of different sizes by using functions like `zeros` or `ones`. For example, create a 0-by-5 matrix and a 3-by-0-by-5 array.

```
B = zeros(0,5)
```

```
B =

  0x5 empty double matrix
```

```
C = ones(3,0,5)
```

```
C =

  3x0x5 empty double array
```

The square brackets, `zeros`, and `ones` create empty numeric arrays of the default `double` data type. To create empty arrays that work with text as data, use the `strings` function. This function creates a string array of any specified size with no characters. For example, create a 0-by-5 empty string array.

```
S = strings(0,5)
```

```
S =

  0x5 empty string array
```

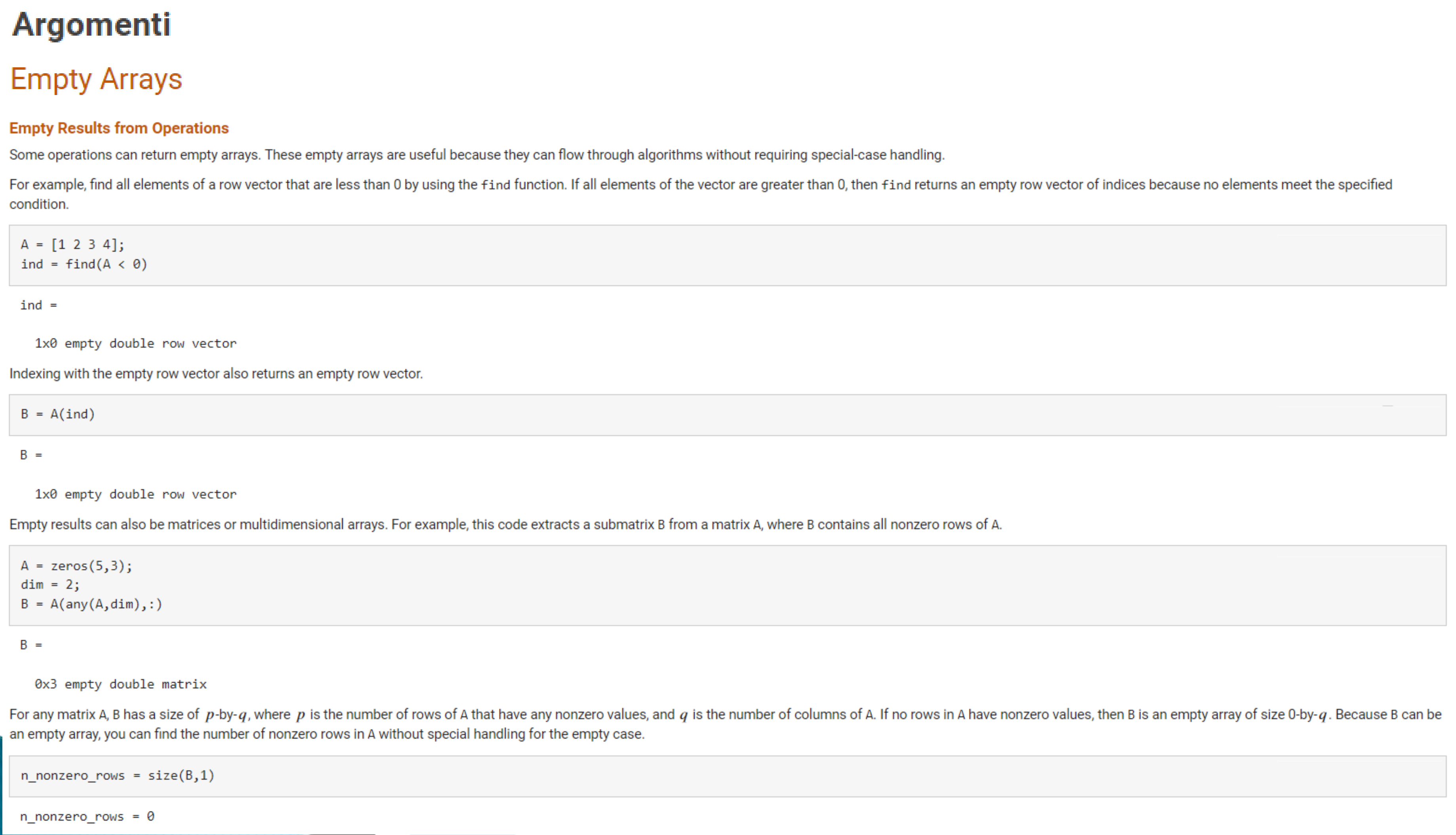
To create empty arrays of any data type, you can use `createArray` (*since R2024a*). For example, create a 5-by-0 matrix of 8-bit signed integer type `int8`.

```
D = createArray(5,0,"int8")
```

```
D =

  5x0 empty int8 matrix
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



## Argomenti

## Empty Arrays

**Empty Results from Operations**

Some operations can return empty arrays. These empty arrays are useful because they can flow through algorithms without requiring special-case handling.

For example, find all elements of a row vector that are less than 0 by using the `find` function. If all elements of the vector are greater than 0, then `find` returns an empty row vector of indices because no elements meet the specified condition.

```
A = [1 2 3 4];
ind = find(A < 0)
```

```
ind =

  1x0 empty double row vector
```

Indexing with the empty row vector also returns an empty row vector.

```
B = A(ind)
```

```
B =

  1x0 empty double row vector
```

Empty results can also be matrices or multidimensional arrays. For example, this code extracts a submatrix B from a matrix A, where B contains all nonzero rows of A.

```
A = zeros(5,3);
dim = 2;
B = A(any(A,dim),:)
```

```
B =

  0x3 empty double matrix
```

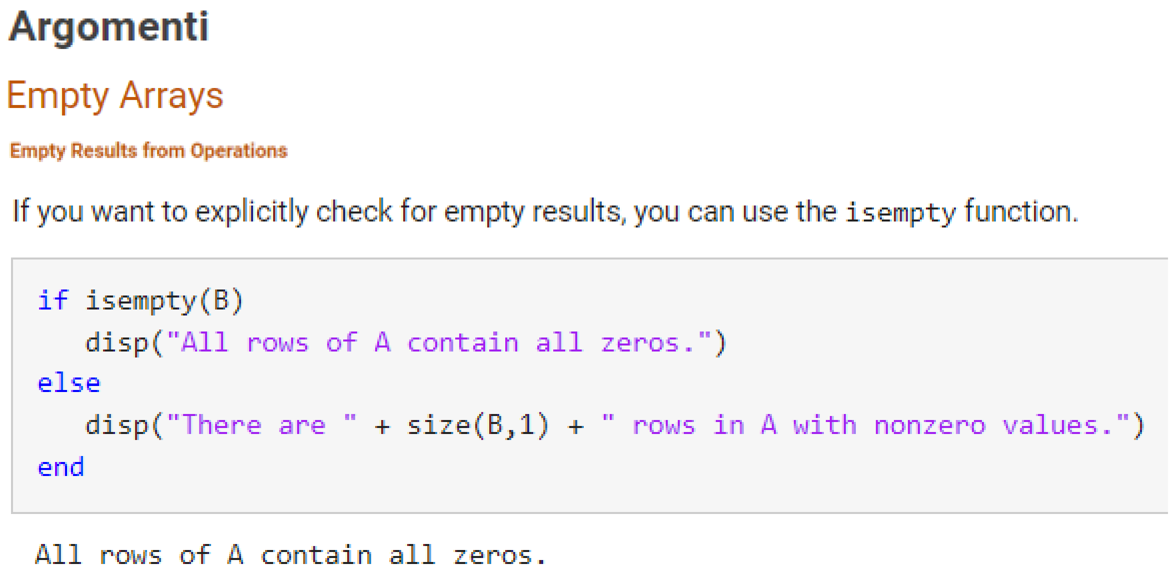
For any matrix A, B has a size of $p$-by-$q$, where $p$ is the number of rows of A that have any nonzero values, and $q$ is the number of columns of A. If no rows in A have nonzero values, then B is an empty array of size 0-by-$q$. Because B can be an empty array, you can find the number of nonzero rows in A without special handling for the empty case.

```
n_nonzero_rows = size(B,1)
```

```
n_nonzero_rows = 0
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



## Argomenti

### Empty Arrays

#### Empty Results from Operations

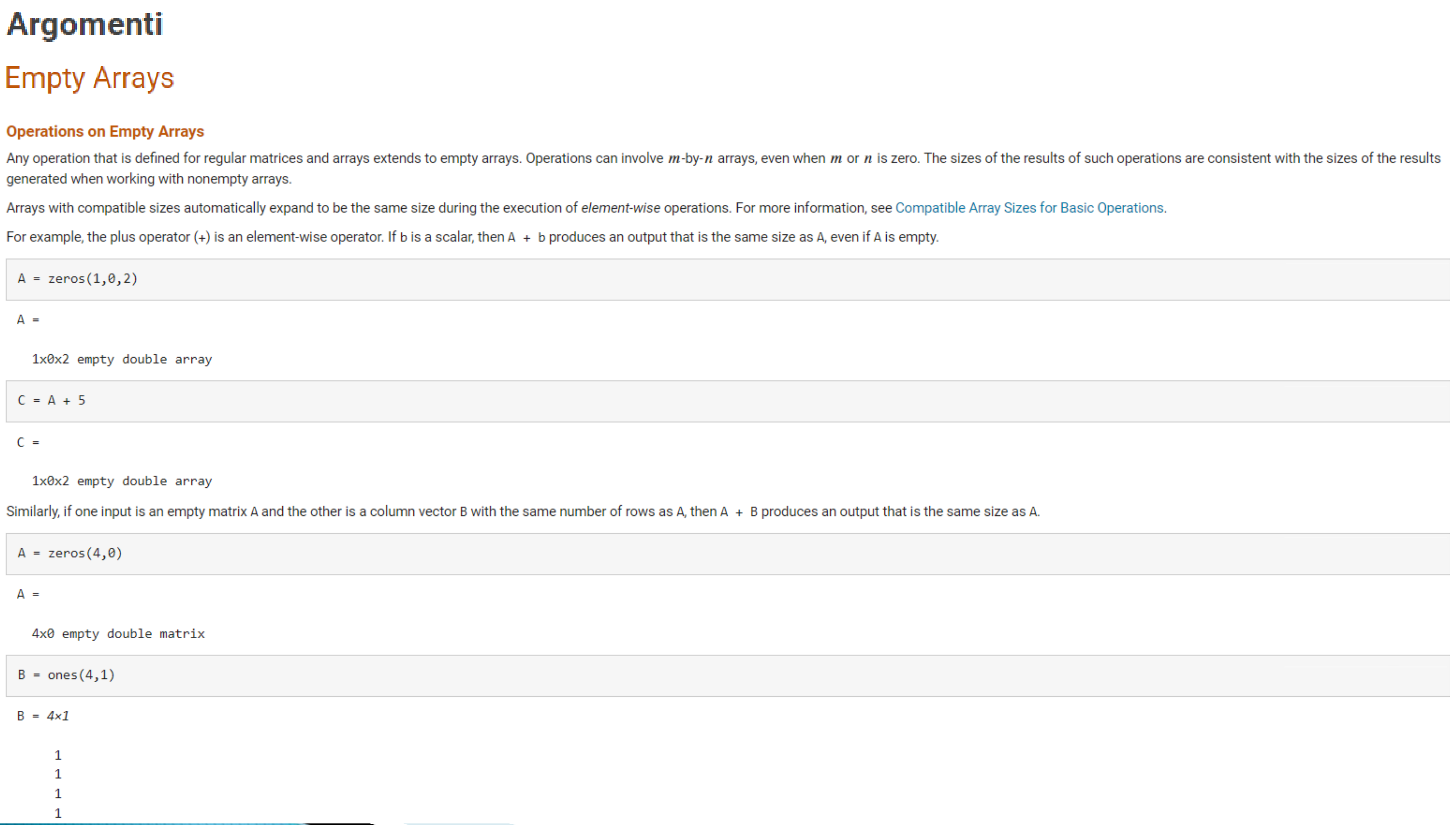
If you want to explicitly check for empty results, you can use the `isempty` function.

```
if isempty(B)
   disp("All rows of A contain all zeros.")
else
   disp("There are " + size(B,1) + " rows in A with nonzero values.")
end
```

```
All rows of A contain all zeros.
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Empty Arrays

#### Operations on Empty Arrays

Any operation that is defined for regular matrices and arrays extends to empty arrays. Operations can involve $m$-by-$n$ arrays, even when $m$ or $n$ is zero. The sizes of the results of such operations are consistent with the sizes of the results generated when working with nonempty arrays.

Arrays with compatible sizes automatically expand to be the same size during the execution of *element-wise* operations. For more information, see Compatible Array Sizes for Basic Operations.

For example, the plus operator (+) is an element-wise operator. If b is a scalar, then A + b produces an output that is the same size as A, even if A is empty.

```
A = zeros(1,0,2)
```

```
A =

  1x0x2 empty double array
```

```
C = A + 5
```

```
C =

  1x0x2 empty double array
```

Similarly, if one input is an empty matrix A and the other is a column vector B with the same number of rows as A, then A + B produces an output that is the same size as A.

```
A = zeros(4,0)
```

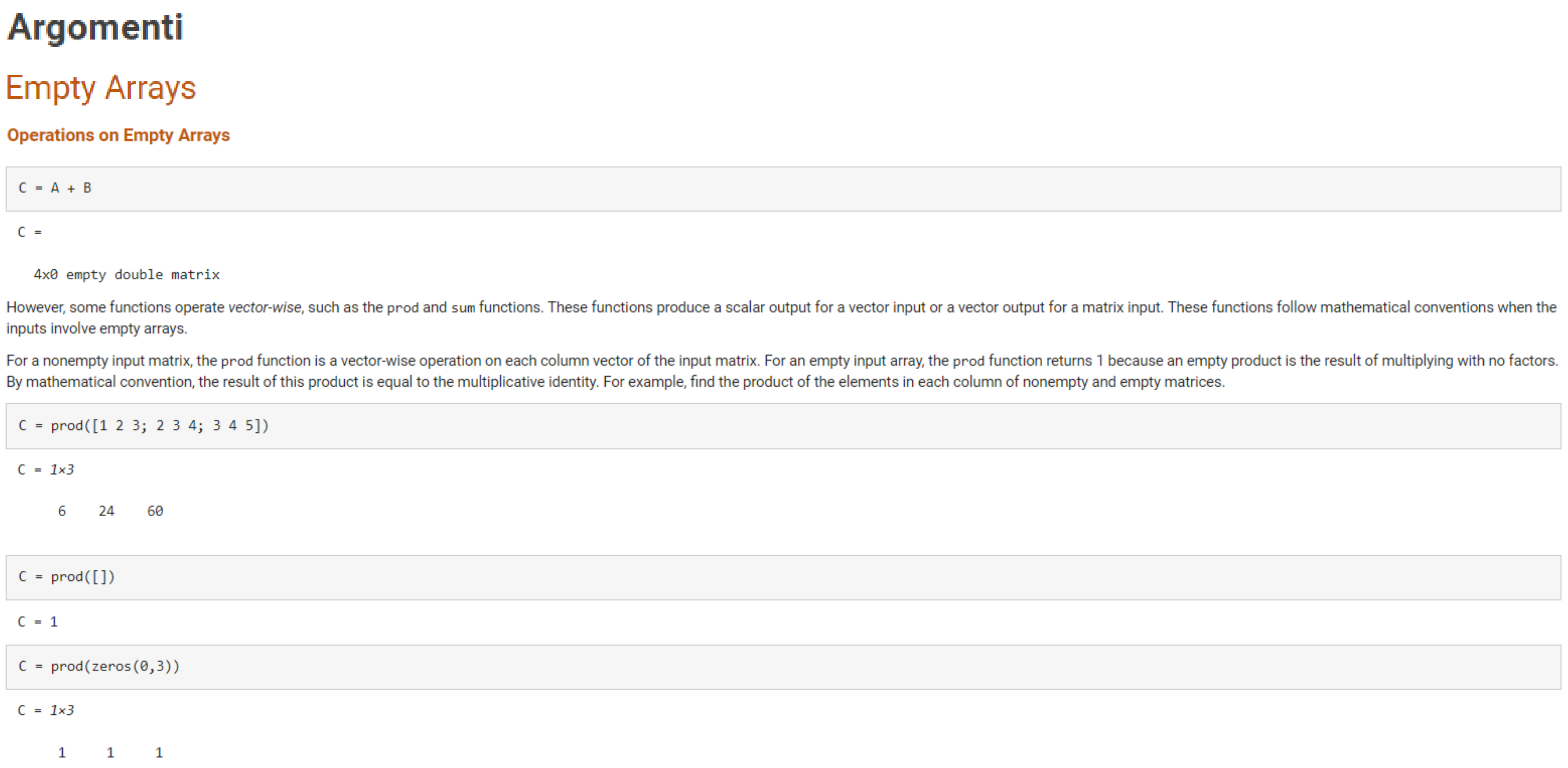
```
A =

  4x0 empty double matrix
```

```
B = ones(4,1)
```

```
B = 4x1

     1
     1
     1
     1
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Empty Arrays

#### Operations on Empty Arrays

```
C = A + B
```

```
C =

  4x0 empty double matrix
```

However, some functions operate *vector-wise*, such as the `prod` and `sum` functions. These functions produce a scalar output for a vector input or a vector output for a matrix input. These functions follow mathematical conventions when the inputs involve empty arrays.

For a nonempty input matrix, the `prod` function is a vector-wise operation on each column vector of the input matrix. For an empty input array, the `prod` function returns 1 because an empty product is the result of multiplying with no factors. By mathematical convention, the result of this product is equal to the multiplicative identity. For example, find the product of the elements in each column of nonempty and empty matrices.

```
C = prod([1 2 3; 2 3 4; 3 4 5])
```

```
C = 1×3

     6    24    60
```

```
C = prod([])
```

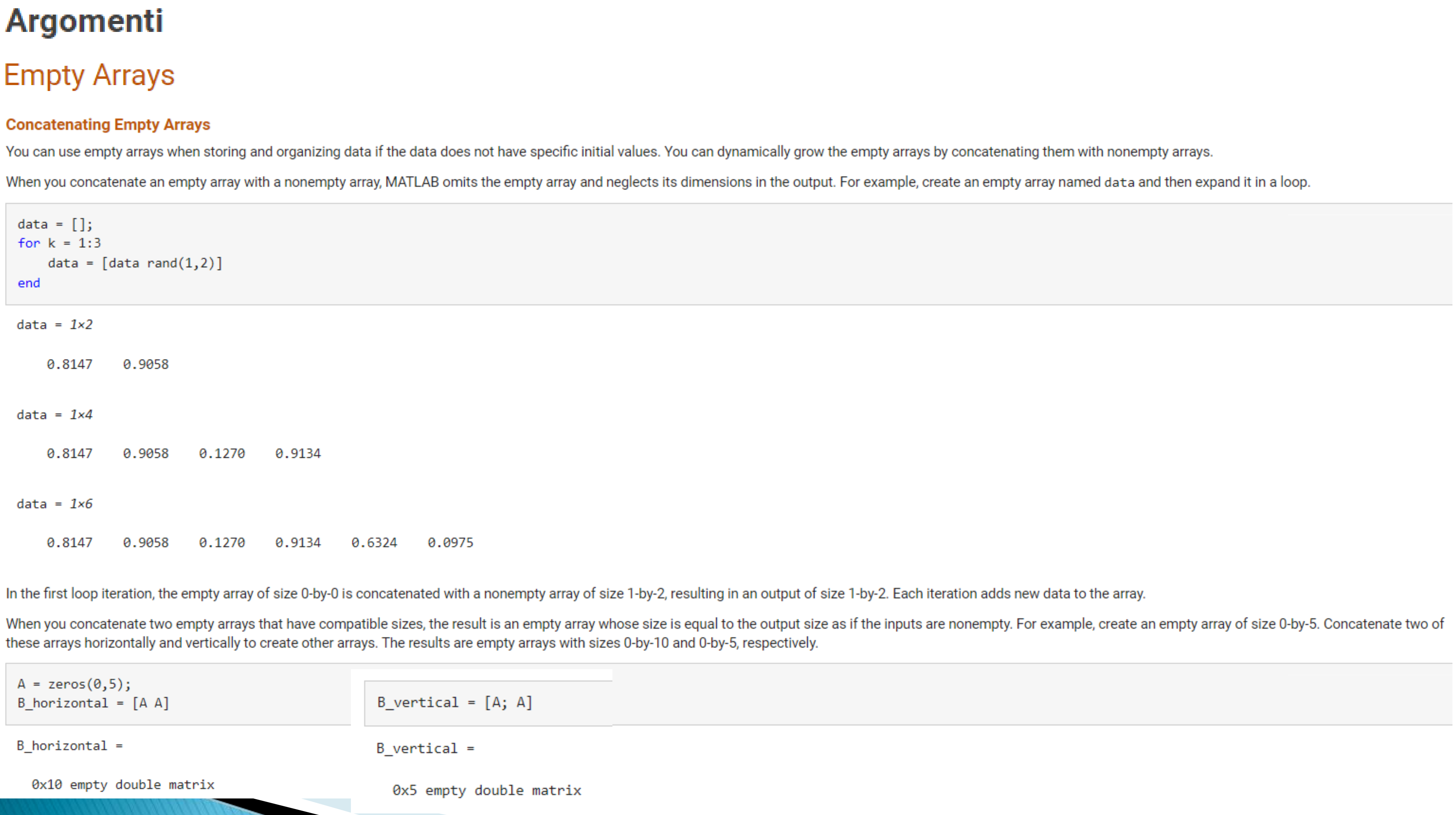
```
C = 1
```

```
C = prod(zeros(0,3))
```

```
C = 1×3

     1     1     1
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

*Matrici e array*



**Argomenti**

## Empty Arrays

### Concatenating Empty Arrays

You can use empty arrays when storing and organizing data if the data does not have specific initial values. You can dynamically grow the empty arrays by concatenating them with nonempty arrays.

When you concatenate an empty array with a nonempty array, MATLAB omits the empty array and neglects its dimensions in the output. For example, create an empty array named `data` and then expand it in a loop.

```
data = [];
for k = 1:3
    data = [data rand(1,2)]
end
```

```
data = 1×2

    0.8147    0.9058


data = 1×4

    0.8147    0.9058    0.1270    0.9134


data = 1×6

    0.8147    0.9058    0.1270    0.9134    0.6324    0.0975
```

In the first loop iteration, the empty array of size 0-by-0 is concatenated with a nonempty array of size 1-by-2, resulting in an output of size 1-by-2. Each iteration adds new data to the array.

When you concatenate two empty arrays that have compatible sizes, the result is an empty array whose size is equal to the output size as if the inputs are nonempty. For example, create an empty array of size 0-by-5. Concatenate two of these arrays horizontally and vertically to create other arrays. The results are empty arrays with sizes 0-by-10 and 0-by-5, respectively.

```
A = zeros(0,5);
B_horizontal = [A A]
```

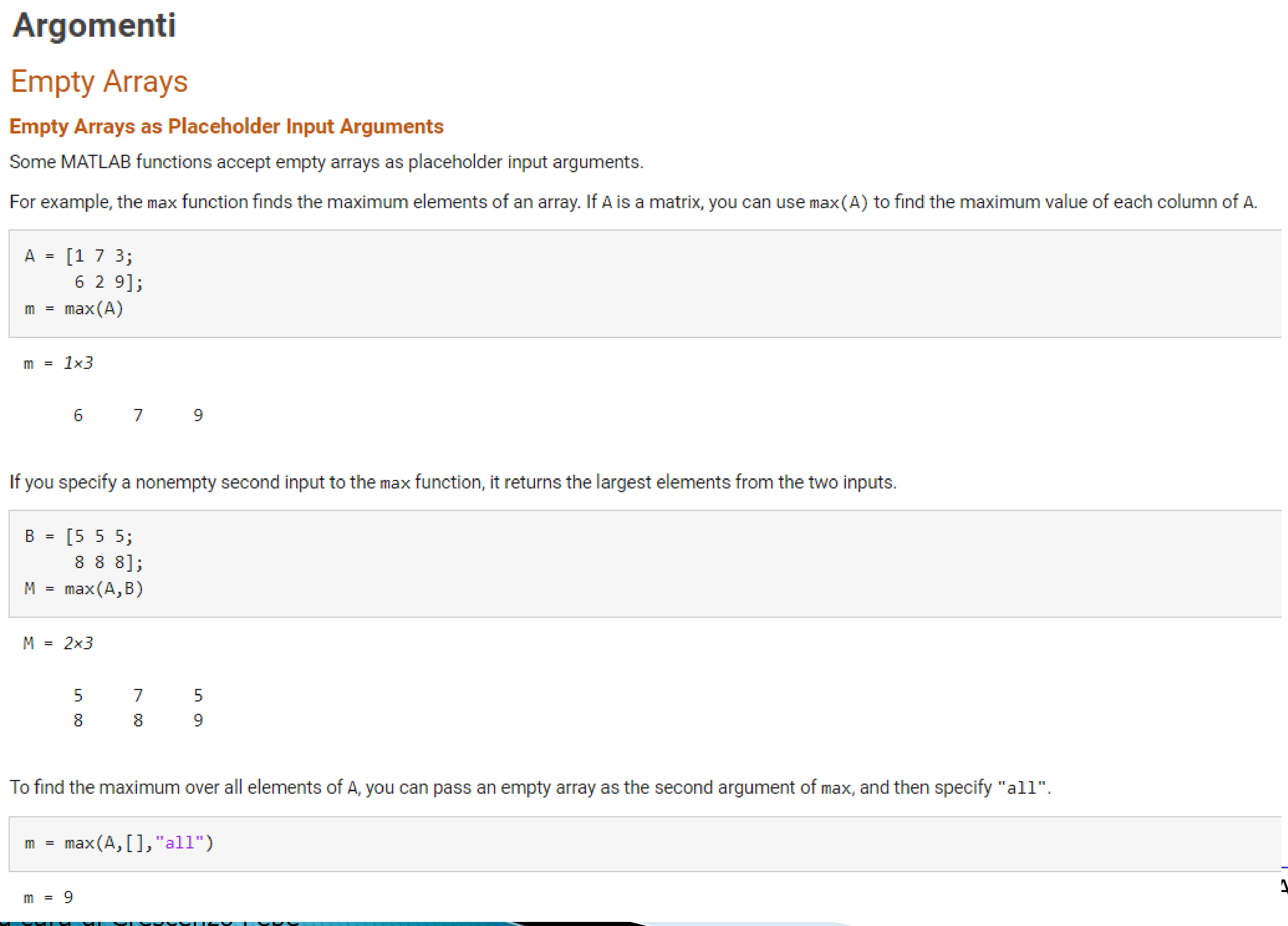
```
B_horizontal =

  0x10 empty double matrix
```

```
B_vertical = [A; A]
```

```
B_vertical =

  0x5 empty double matrix
```

## ESERCITAZIONI
## MATLAB – Nozioni fondamentali

*Matrici e array*



**Argomenti**

### Empty Arrays

#### Empty Arrays as Placeholder Input Arguments

Some MATLAB functions accept empty arrays as placeholder input arguments.

For example, the `max` function finds the maximum elements of an array. If `A` is a matrix, you can use `max(A)` to find the maximum value of each column of `A`.

```
A = [1 7 3;
     6 2 9];
m = max(A)
```

```
m = 1×3

     6     7     9
```

If you specify a nonempty second input to the `max` function, it returns the largest elements from the two inputs.

```
B = [5 5 5;
     8 8 8];
M = max(A,B)
```

```
M = 2×3

     5     7     5
     8     8     9
```

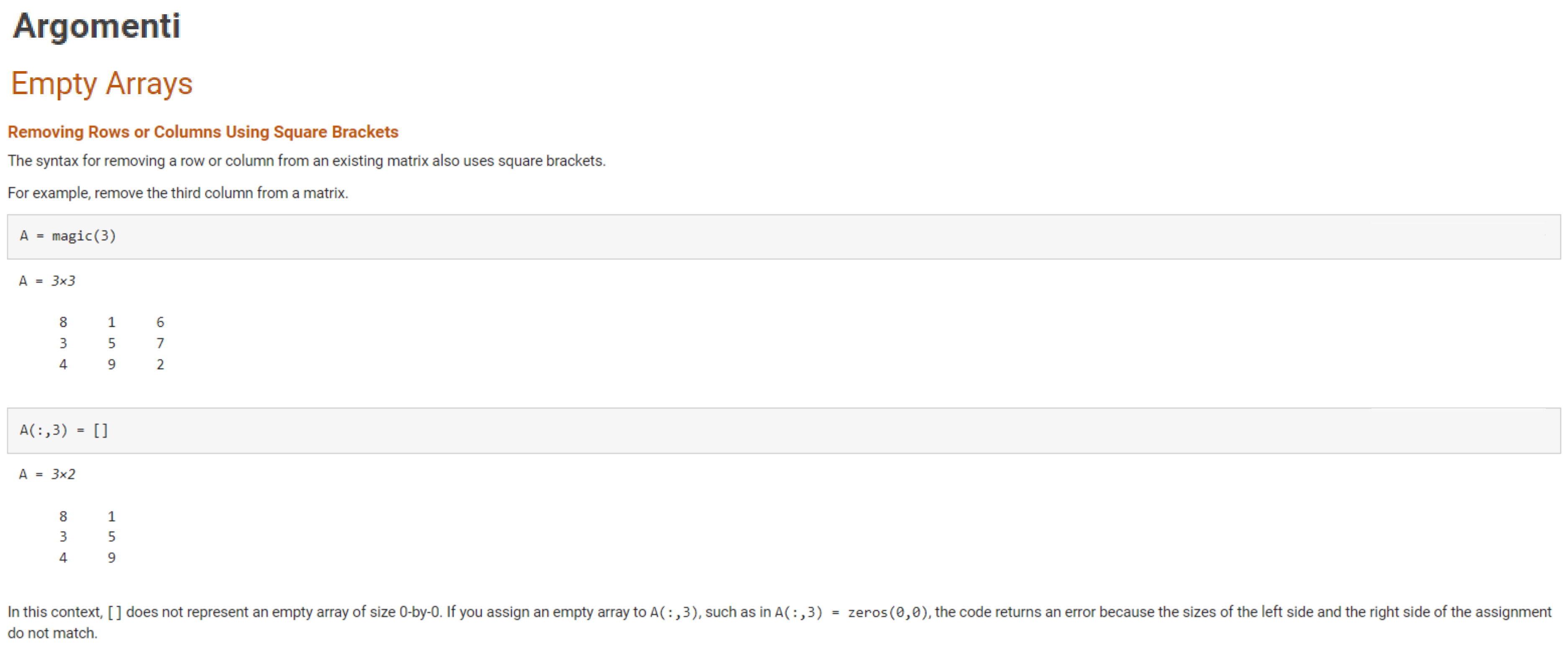
To find the maximum over all elements of `A`, you can pass an empty array as the second argument of `max`, and then specify `"all"`.

```
m = max(A,[],"all")
```

```
m = 9
```

# ESERCITAZIONI
# MATLAB – Nozioni fondamentali

## *Matrici e array*



### Argomenti

### Empty Arrays

#### Removing Rows or Columns Using Square Brackets

The syntax for removing a row or column from an existing matrix also uses square brackets.

For example, remove the third column from a matrix.

```
A = magic(3)
```

```
A = 3×3

     8     1     6
     3     5     7
     4     9     2
```

```
A(:,3) = []
```

```
A = 3×2

     8     1
     3     5
     4     9
```

In this context, [] does not represent an empty array of size 0-by-0. If you assign an empty array to `A(:,3)`, such as in `A(:,3) = zeros(0,0)`, the code returns an error because the sizes of the left side and the right side of the assignment do not match.

## *Riferimenti Bibliografici*

[1] https://it.mathworks.com