MATLAB

Lecture 6

Advanced Data Structures

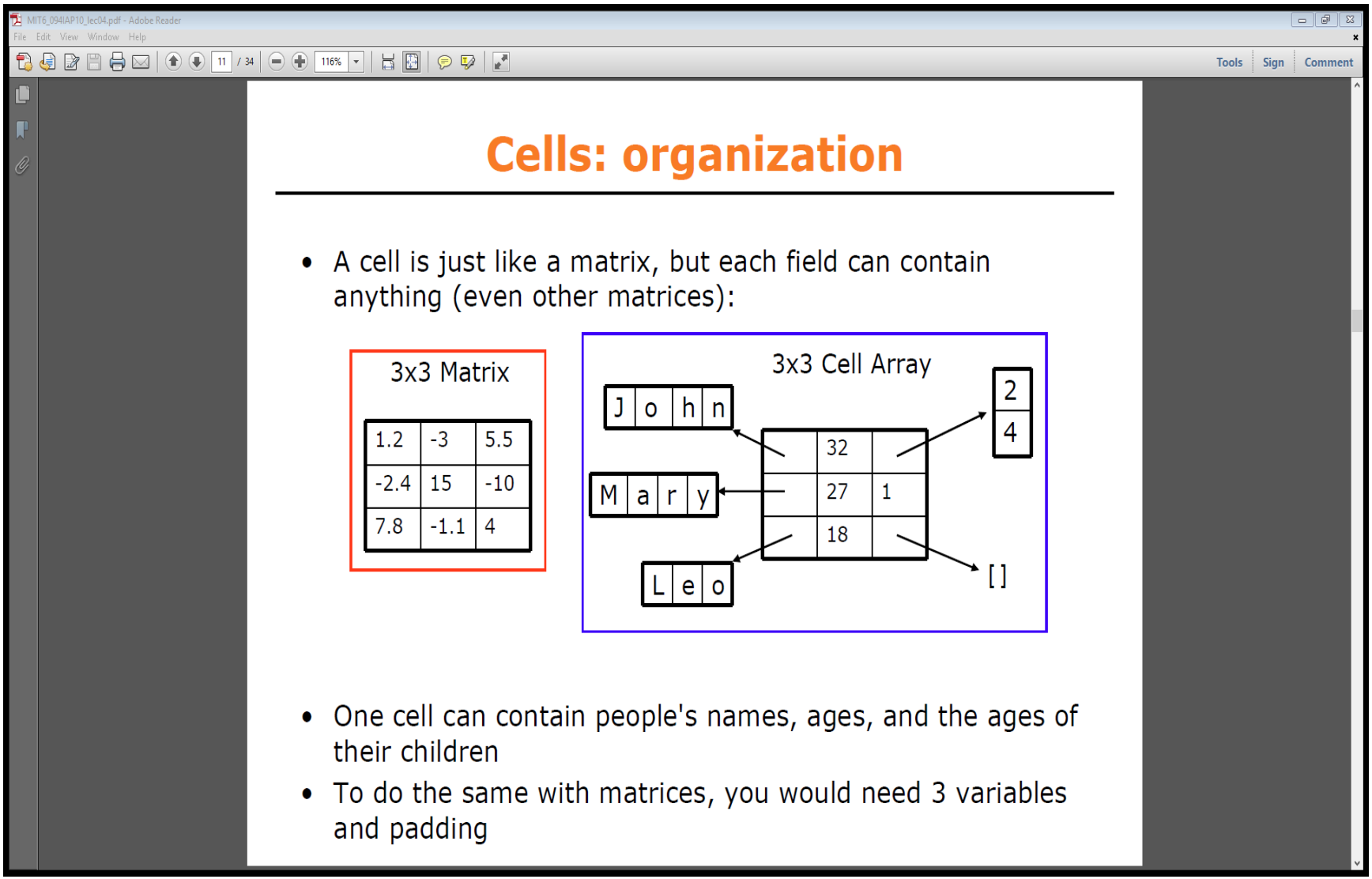
- **We have used 2D matrices**
  - Can have n-dimensions
  - Every element must be the same type (ex. integers, doubles, characters…)
  - Matrices are space-efficient and convenient for calculation
- **Sometimes, more complex data structures are more appropriate**

Advanced Data Structures

- **Cell array**

- **it's like an array, but elements don't have to be the same type**

Advanced Data Structures



Advanced Data Structures

- **Cells initialization**

To initialize a cell, specify the size

a=cell(3,10);

a will be a cell with 3 rows and 10 columns

or do it manually, with curly braces {}

c={'hello world',[1 5 6 2],rand(3,2)};

c is a cell with 1 row and 3 columns

- **Each element of a cell can be anything**

Advanced Data Structures

- **Cells accessing:**

- **c={'hello world',[1 5 6 2],rand(3,2)};**
  - To access a cell element, use curly braces {}

```
>> c{1,1}
ans =

hello world

>> c{1,2}

ans =

    1    5    6    2

>> c{1,2}(2)

ans =

    5
```

Advanced Data Structures

- **Cells accessing:**

```
>> a=cell(3,10);
>> a

a =

   []   []   []   []   []   []   []   []   []   []
   []   []   []   []   []   []   []   []   []   []
   []   []   []   []   []   []   []   []   []   []

a{1,1}

ans =

   []
```

- **a is a matrix with 3 rows and 10 columns of empty “matrices(anything)”**

Advanced Data Structures

- **Cells accessing:**

```
>> a{1,1}=[1 3 4 -10];
>> a{2,1}='hello world 2';
>> a

a =

    [1x4 double]    []    []    []    []    []    []    []    []    []
    'hello world 2'    []    []    []    []    []    []    []    []    []
                []    []    []    []    []    []    []    []    []    []
```

Advanced Data Structures

- **Cells :**

Having two arrays C1 , C2

We can combine those into one cell C3:

C3 = {C1 C2}

Concatenates cell arrays C1 and C2 into a two-element cell array C3 such that C3{1} = C1 and C3{2} = C2

C3 = [C1 C2]

Concatenates the contents of cell arrays C1 and C2, assuming that the dimensions of these arrays are compatible

Advanced Data Structures

- **Struct**

- Structs allow you to name and bundle relevant variables
  Like C-structs, which are objects with fields

To initialize an empty struct:

- **»s=struct;**
  - creates a 1-by-1 structure with no fields

To add fields

```
»s.name = 'Jack Bauer';
»s.scores = [95 98 67];
»s.year = 'G3';
```

- Fields can be anything: matrix, cell, even struct
- Useful for keeping variables together

Advanced Data Structures

- **Struct**

- **>> s=struct**

- **s =**

- **1x1 struct array with no fields.**

- **>> s.name = 'Jack Bauer';**
- **s.scores = [95 98 67];**
- **s.year = 'G3';**
- **>> s**

- **s =**

- **name: 'Jack Bauer'**
- **scores: [95 98 67]**
- **year: 'G3'**

Advanced Data Structures

- **Struct**

- We can do the same at ones:

```
>> d=struct('name',{'Jack Bauer'},'scores',[95 98 67],'year',{'G3'});

>> d

    d =

     name: 'Jack Bauer'
     scores: [95 98 67]
     year: 'G3‘
```

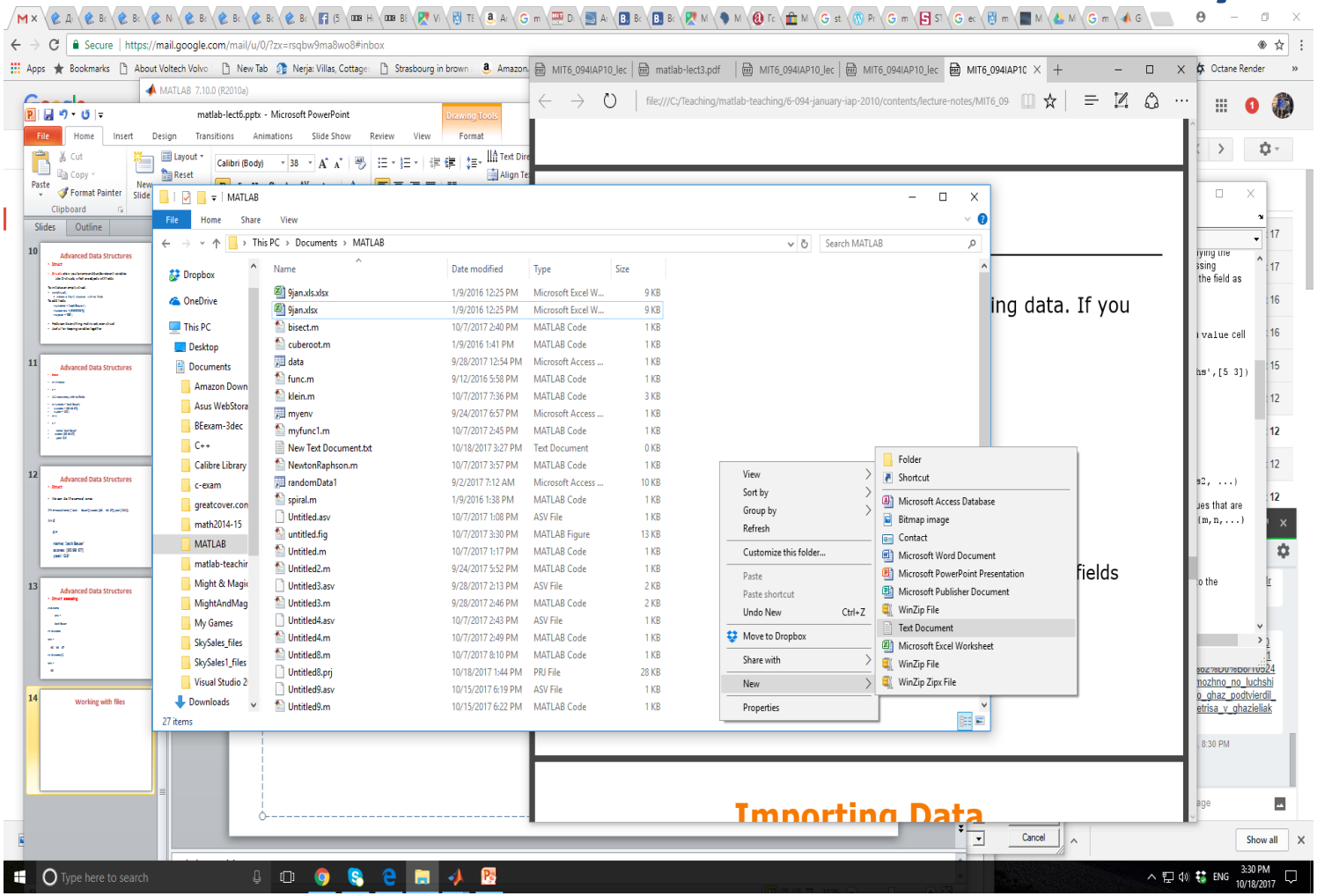
Advanced Data Structures

- **Struct assessing**

```
>>d.name

      ans =

      Jack Bauer

>> d.scores

ans =

   95    98    67

>> d.scores(1)

ans =

   95
```

Working with files

- **Create txt file in the current matlab directory:**



Working with files

- Name of my file is example1.txt
- File contains:

fprintf reapplies the conversion information to cycle through all values of the input arrays in column order.

For example, create a file named exptable.txt that contains a short table of the exponential function, and a text header:

```
% create a matrix y, with two rows
x = 0:0.1:1;
y = [x; exp(x)];

% open a file for writing
fid = fopen('exptable.txt', 'w');

% print a title, followed by a blank line
fprintf(fid, 'Exponential Function\n\n');

% print values in column order
% two values appear on each row of the file
fprintf(fid, '%f  %f\n', y);
fclose(fid);
```

Working with files

Basic technique :

Use **importdata** function:

```
>>a=importdata('example1.txt', ' ');
```

a  is a **struct**

Working with files

```
>> a=importdata('example1.txt', ' ');
>> a

a =

    [1x109 char]
    ''
    [1x121 char]
    '% create a matrix y, with two rows'
    'x = 0:0.1:1;'
    'y = [x; exp(x)];'
    ''
    '% open a file for writing'
    'fid = fopen('exptable.txt', 'w');'
    ''
    '% print a title, followed by a blank line'
    'fprintf(fid, 'Exponential Function\n\n');'
    ''
    '% print values in column order'
    '% two values appear on each row of the file'
    'fprintf(fid, '%f  %f\n', y);'
    'fclose(fid);‘
```

Working with files

```
>>a(1)

ans =

    [1x109 char]
>> a{1}(1)

ans =

F
>> a{1}(1:109)

ans =

fprintf reapplies the conversion information to cycle through all values of the input arrays in column order.
>> a(5)

ans =

    'x = 0:0.1:1;'
```

Working with files

- Let us consider more structured file:

- Example.txt

Exponential Function

0.000000 1.000000
0.100000 1.105171
0.200000 1.221403
0.300000 1.349859
0.400000 1.491825
0.500000 1.648721
0.600000 1.822119
0.700000 2.013753
0.800000 2.225541
0.900000 2.459603
1.000000 2.718282

Working with files

- **Much more easer to work with structured data files:**

```
 >> b=importdata('example.txt', ' ');
>> b

b =

    data: [11x2 double]
    textdata: {'Exponential'  'Function'}
    colheaders: {'Exponential'  'Function'}

>> zz=b.data

zz =

    0         1.0000
    0.1000    1.1052
    0.2000    1.2214
    0.3000    1.3499
    0.4000    1.4918
    0.5000    1.6487
    0.6000    1.8221
    0.7000    2.0138
    0.8000    2.2255
    0.9000    2.4596
    1.0000    2.7183

>> size(zz)

ans =

    11     2
```

Working with files

With importdata, you can also specify delimiters.

For example, for comma separated values, use:

```
»a=importdata('filename', ', ');
```

The second argument tells matlab that the tokens of interest are separated by commas or spaces

Importdata is very robust, but sometimes it can have trouble.

To read files with more control, use fscanf(similar to C/Java), textread, textscan.

See help or doc for information on how to use these functions

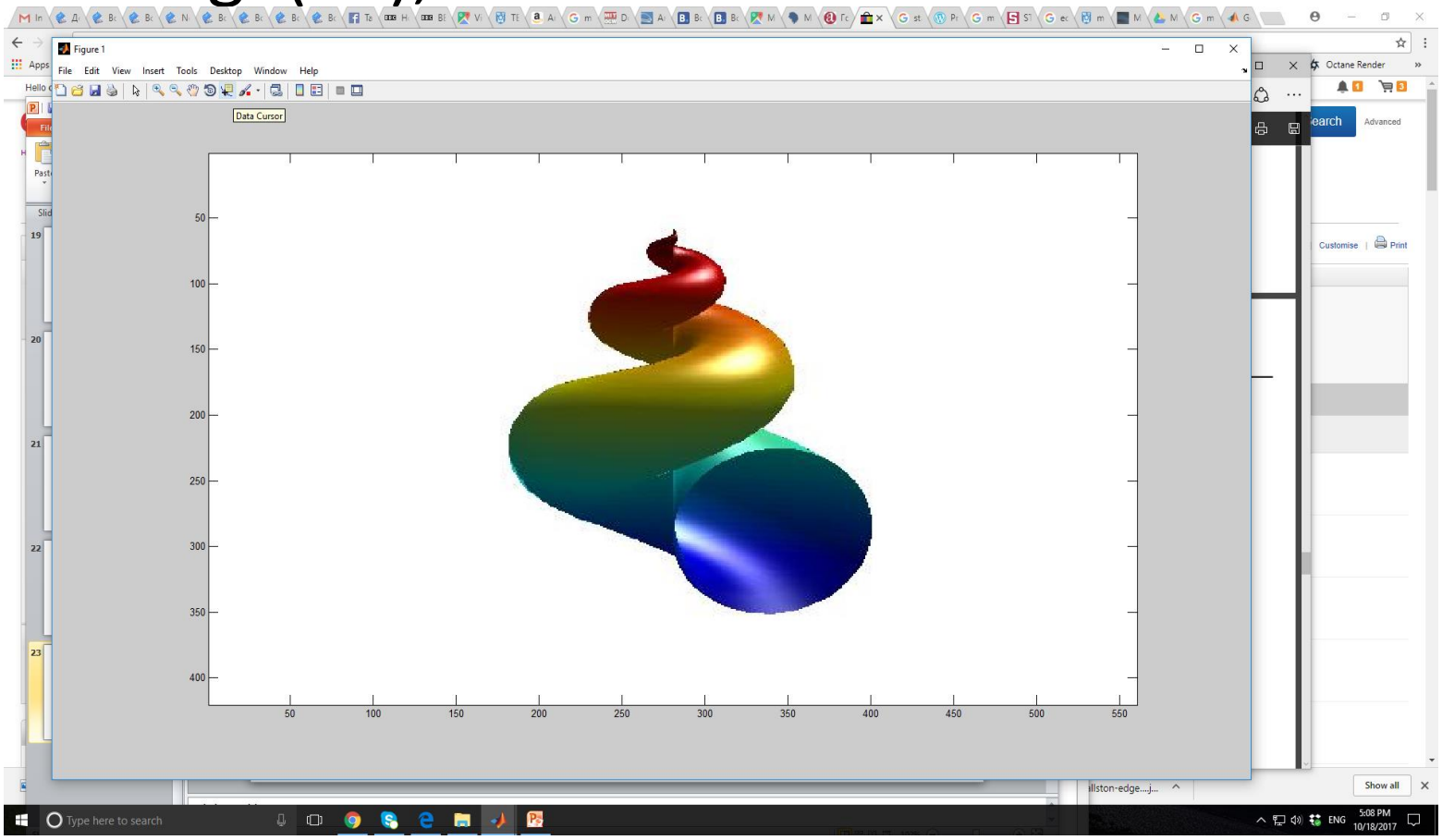
Working with files

- Writing to a file: **fprintf**

```
% create a matrix y, with two rows
x = 0:0.1:1;
y = [x; exp(x)];

% open a file for writing
fid = fopen('example.txt', 'w');

% print a title, followed by a blank line
fprintf(fid, 'Exponential Function\r\n\r\n');

% print values in column order
% two values appear on each row of the file
fprintf(fid, '%f  %f\r\n', y);
fclose(fid);
```

Working with files

```
>> ne = importdata('untitled1.jpg');
>> image(ne);
```



Working with Excel files

- **Excel files are structured data files**
- Easy to deal with in Matlab
- **xlswrite**
- **xlsread**

Working with Excel files

- **Xlswrite**

Stores numeric array or cell array in Excel workbook.

[SUCCESS,MESSAGE]=XLSWRITE(FILE,ARRAY,SHEET,RANGE)

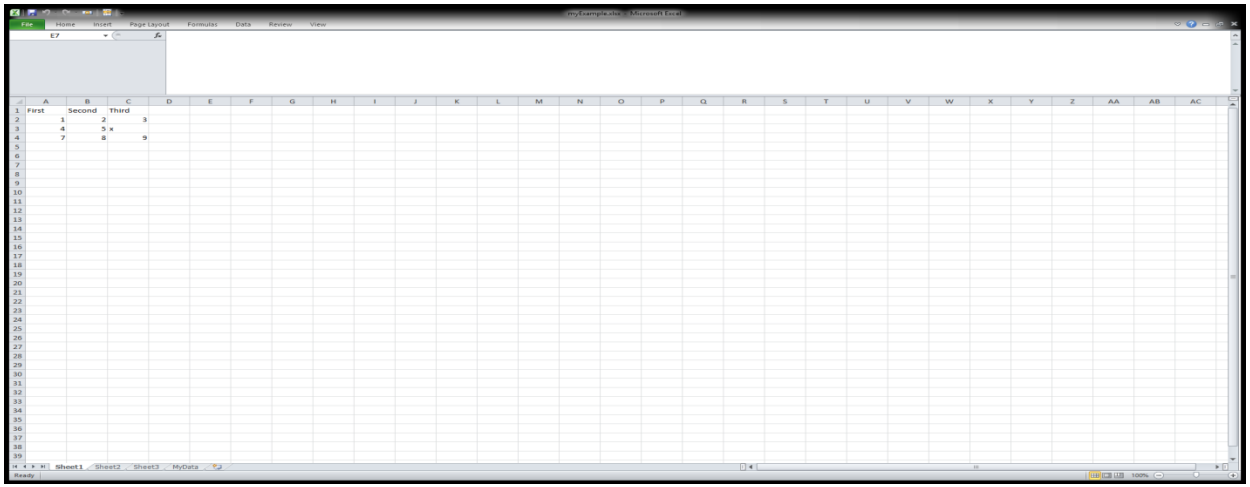
writes ARRAY to the Excel workbook, FILE, into the area, RANGE in the worksheet specified in SHEET.

- FILE and ARRAY must be specified.

Working with Excel files

- **Xlswrite**

```
values = {1, 2, 3 ; 4, 5, 'x'; 7, 8, 9};
headers = {'First','Second','Third'};
xlswrite('myExample.xlsx',[headers; values]);
```



Working with Excel files

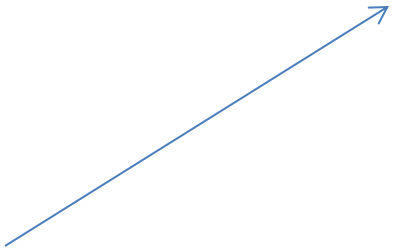
- **Xlswrite**

First	Second	Third		
1	2	3		
4	5	x		
7	8	9		

Working with Excel files

- **Xlswrite**

- A = rand(5);
- xlswrite(‘myExample.xlsx’,A,’MyData’)



- Specific Excel sheet

Working with Excel files

- Read numeric data

Example:

```
Filename = 'myExample.xlsx';
A = xlsread(Filename)
```

Example:

Read a specific range of data from the Excel file

```
num = xlsread(filename,sheet,xlRange)
```

```
filename = 'myExample.xlsx';
sheet = 1;
xlRange = 'B2:C3';
subsetA = xlsread(Filename,sheet,xlRange)
```

Working with Excel files

- Read numeric data

- Complete format:

[num,txt,raw]=xlsread(…….)

Reads data

Num contains numbers,

Txt contains strings,

Raw is the entire cell array containing everything

Characters and String

- Character arrays and string arrays provide storage for text data in MATLAB.

- A character array is a sequence of characters, just as a numeric array is a sequence of numbers.

  A typical use is to store short pieces of text as *character vectors*, such as c = 'Hello World'.

- A string array is a container for pieces of text.

- String arrays provide a set of functions for working with text as data.

Characters and String

Creating a character string is quite simple in MATLAB.

In fact, we have used it many times.

For example, you type the following in the command prompt

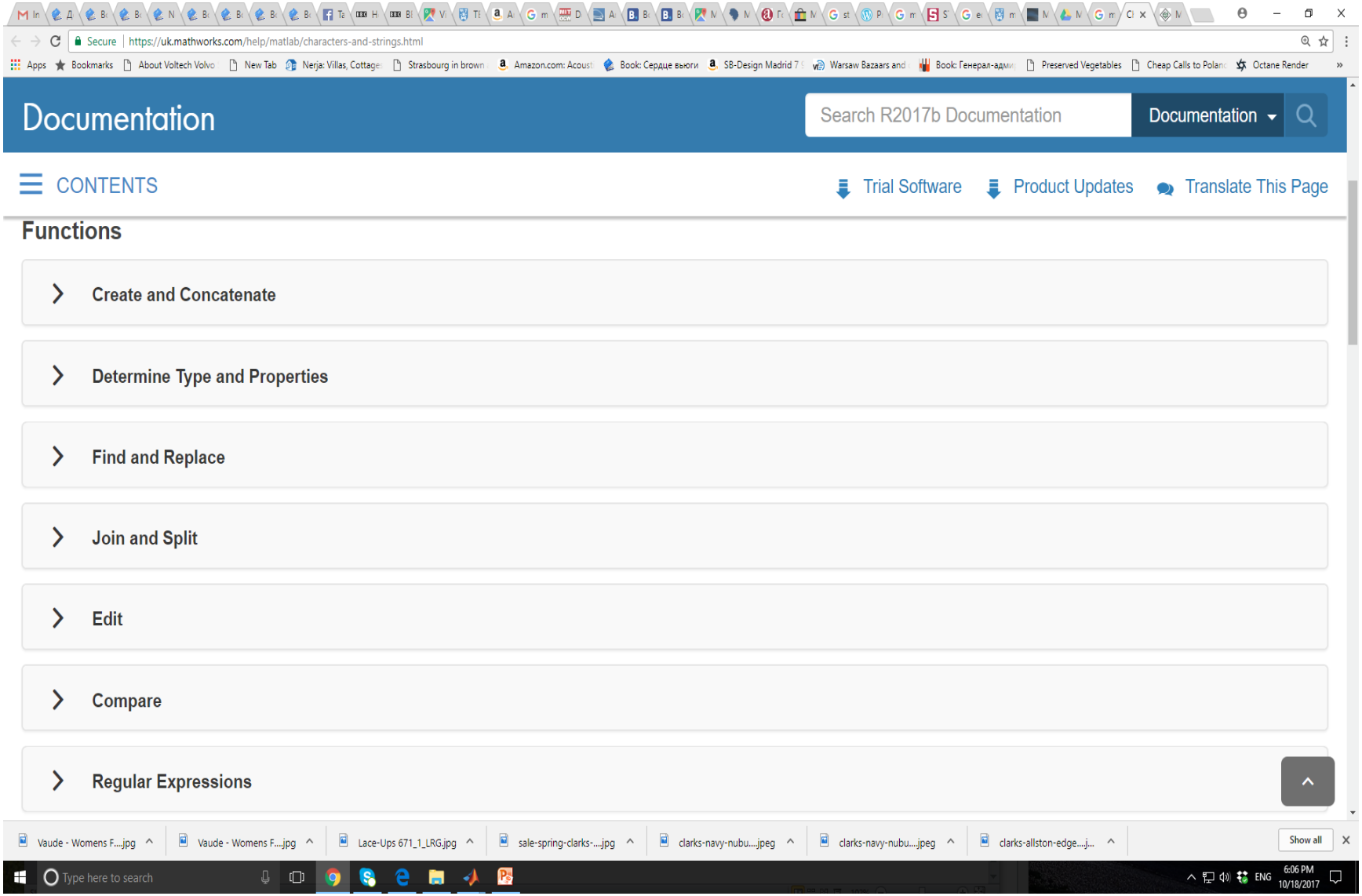
```
my_string = 'Tutorials Point‘
```

MATLAB will execute the above statement and return the following result

```
my_string = Tutorials Point
```

MATLAB considers all variables as arrays, and strings are considered as character arrays.

Characters and String



Characters and String

- Combining strings.
- You can combine strings vertically in either of the following way:
- Using the MATLAB concatenation operator **[]** and separating each row with a semicolon (;).
- Please note that in this method each row must contain the same number of characters.
- For strings with different lengths, you should pad with space characters as needed.
- example

```
>>doc_profile = ['Zara Ali                                    '; ...
                 'Sr. Surgeon                                ‘; ...
                 'R N Tagore Cardiology Research Center'];
```

Characters and String

- Combining strings.

- Using the **char** function.
- If the strings are of different lengths, char pads the shorter strings with trailing blanks so that each row has the same number of characters.
- Example:

```
>>doc_profile = char('Zara Ali', 'Sr. Surgeon', 'RN Tagor Cardiology Research Center')
```

Characters and String

- Combining Strings into a Cell Array

- From our previous discussion, it is clear that combining strings with different lengths could be a pain as all strings in the array has to be of the same length.
- We have used blank spaces at the end of strings to equalize their length.
- However, a more efficient way to combine the strings is to convert the resulting array into a cell array.

Characters and String

- Combining Strings into a Cell Array
- MATLAB cell array can hold different sizes and types of data in an array.
- Cell arrays provide a more flexible way to store strings of varying length.
- The **cellstr** function converts a character array into a cell array of strings.
- Example:

```
name = 'Zara Ali ';
position = 'Sr. Surgeon ';
worksAt = 'R N Tagore Cardiology Research Center';
profile = char(name, position, worksAt);
profile = cellstr(profile);
 disp(profile)
```

Characters and String

- Main problems:

- Find string
- Replace string
- Compare strings

Characters and String

- **strfind**
- k = **strfind**(str,pattern) searches **str** for occurrences of pattern.
- The output, **k**, indicates the starting index of each occurrence of pattern in **str**

Characters and String

- **strfind**

- Find the starting indices for occurrences of patterns in a character vector.

- First, create a character vector.

```
>>str = 'Find the starting indices of a pattern in a character vector';
```

- Find the pattern in.

```
>>k = strfind(str,'in')
```

```
k = 2 15 19 40
```

- There are four instances of the pattern in str.

Characters and String

- **strrep**

- newStr = **strrep**(str,old,new)

- replaces all occurrences of **old** in **str** with **new**.

Characters and String

- **strcmp**

- **tf = strcmp(s1,s2)** compares **s1** and **s2** and returns 1 (true) if the two are identical and 0 (false) otherwise.

- Text is considered identical if the size and content of each are the same. The return result **tf** is of data type logical.

- The input arguments can be any combination of string arrays, character vectors, and cell arrays of character vectors.