

The background of the slide features a complex network diagram. It consists of numerous small, dark grey circular nodes connected by thin, light grey lines. These lines form a web-like structure that fills the entire frame. A large, solid dark teal rectangle is positioned on the right side of the image, partially overlapping the network diagram. The text is centered within this teal rectangle.

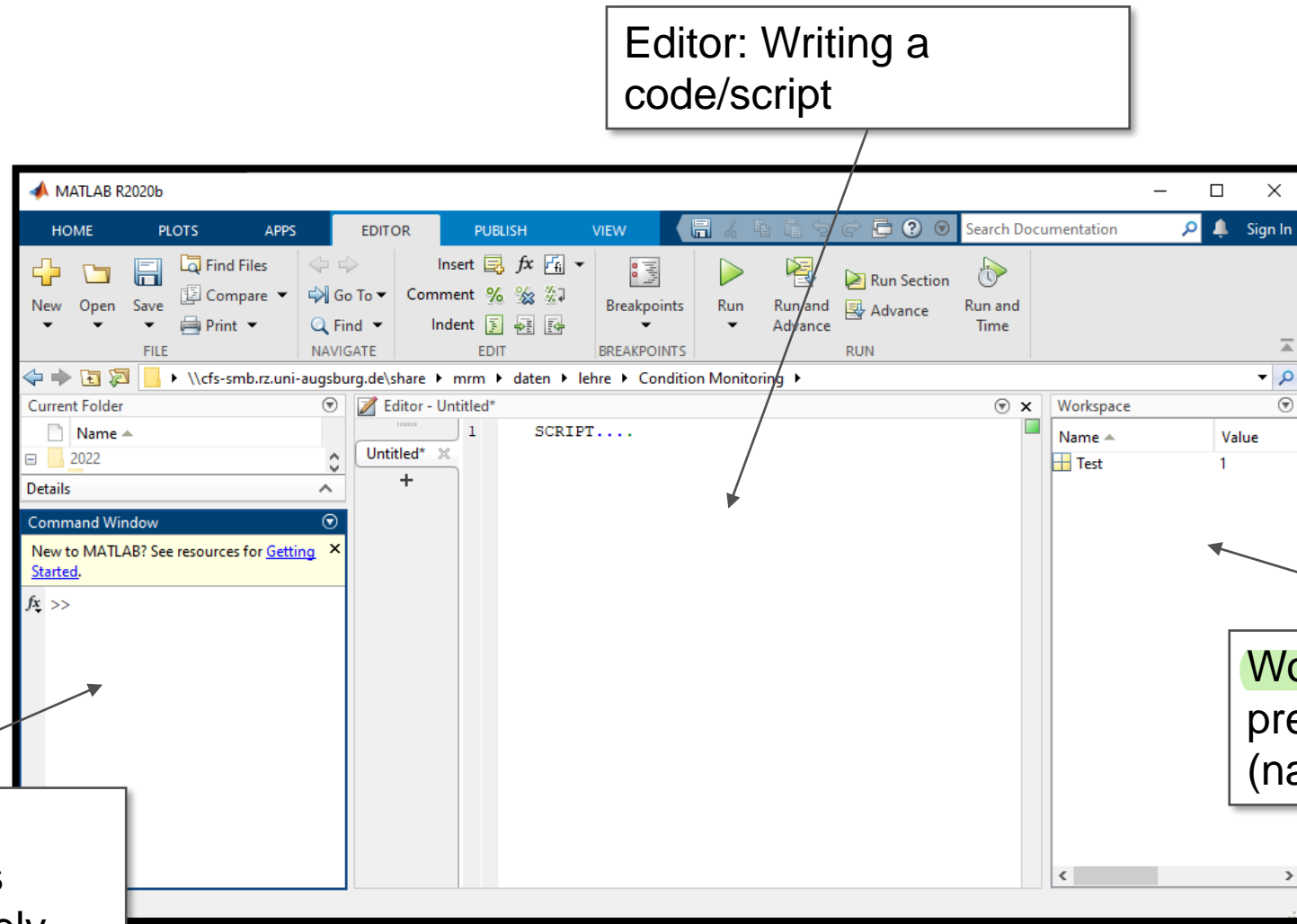
# Condition Monitoring of Structures, Machines and Processes

## Tutorial

# 1.1

## INTRODUCTION TO MATLAB

## 1.1 Desktop



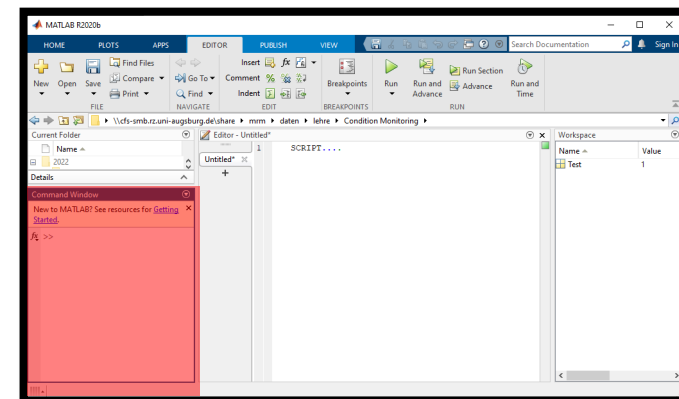
# Exercise 1

## Simple Operations

- Addition (+), Subtraction (-), Multiplication (\*), Division (/), Power (^)
- Order of Operations (same rules you should already know from math class and using a calculator)

### Task:

- Use the command window as a calculator and execute simple calculations
  - Notice the creation of the variable `ans` in the Workspace, that is, unless you specify output variables, MATLAB stores current results in that variable.



## Exercise 2

### Variables

- To avoid overwriting results in `ans`, you can create your own variables displayed in the workspace. The equal sign (`=`) in MATLAB is the assignment operator, meaning that the expression on the right of the equal sign is assigned to the variable on the left.
- Semicolons suppress the output.
- Variables are case sensitive! `A`  $\neq$  `a`
- Mind the syntax `Aa`  $\neq$  `A*a`

#### Task:

- Assign the calculation `6*7` to a variable `n`
- Try `n = n + 1;`
- Define new variables: `y = n/4;` and `k = 2;`
- What happens to the values of `n` and `k` by executing `n = 3*k;`
- Did `y` change? If not update the value for `y` by entering the right command. Note: Use the up-arrow to reenter recent commands.

## Exercise 3

### Saving and Loading Variables

- You can save variables in your workspace to a MATLAB specific file format called a MAT-file using the `save` function. The input variable is optionally without which the whole workspace is saved. Unless you specify the full path MATLAB saves the data in the current folder, which you can inquiry using the command `pwd`.

#### Task:

- Save the workspace under `myData.mat`
- Clear the workspace using the command `clear`
- Reload the workspace with the variable `k` of `myData.mat` using `load filename`
- Load the rest of the variables.
- Specify the saving path by using the function `fullfile`

## Exercise 4

### Built-in Functions and Constants

MATLAB provides built-in Functions and Constants:

- Constants:
  - `pi`
  - `physconst('LightSpeed')`
  - ...
- Elementary functions:
  - `exp,pow2`: Exponential function
  - `log,log10,log2`: Logarithmic functions
  - `sqrt`: Squarefunction
  - `sin,cos,tan`: trigonometric functions
  - `abs`: absolute value
  - ...

#### Task:

- Create a variable `x` with the value of  $\frac{\pi}{2}$ 
  - Try display more numbers using the command `format long/short`
- Calculate `sin(x)`
- Calculate `sqrt(-9)`
  - Note the use of `i` as the imaginary number
- Create random numbers using the function `rand` and extract the maximum values using a appropriate built in function

## Exercise 5

### Running Scripts

The Editor provides the space for writing a script which being executed displays your output and storing results in the workspace.

Task:

- Write a script which displays the area of a circle with a given radius  $r$

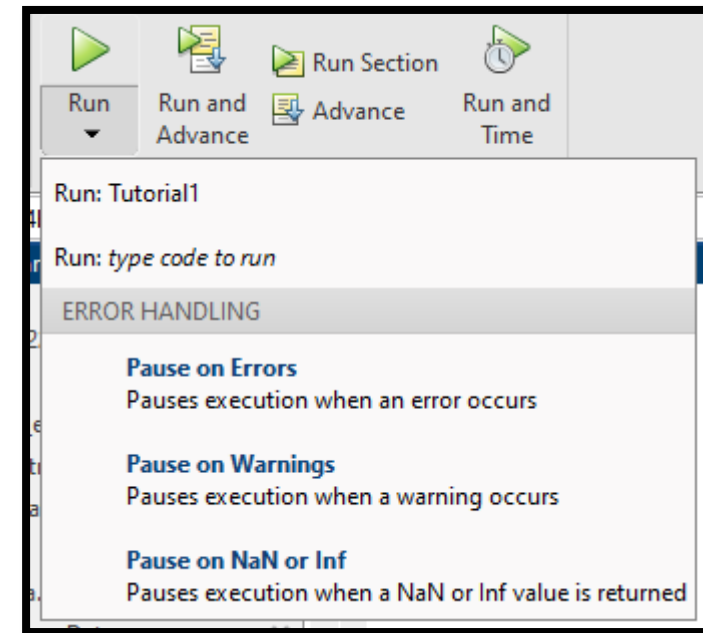


## Exercise 6

### Debugging

Consider the following script containing an error. Usually MATLAB detects issues in your code, underlining it in red. Searching for errors you could run the corrupted script using appropriate Error handling settings:

```
r = 4;  
r = 5;  
r = pi**2;
```



## Exercise 7

### Arrays

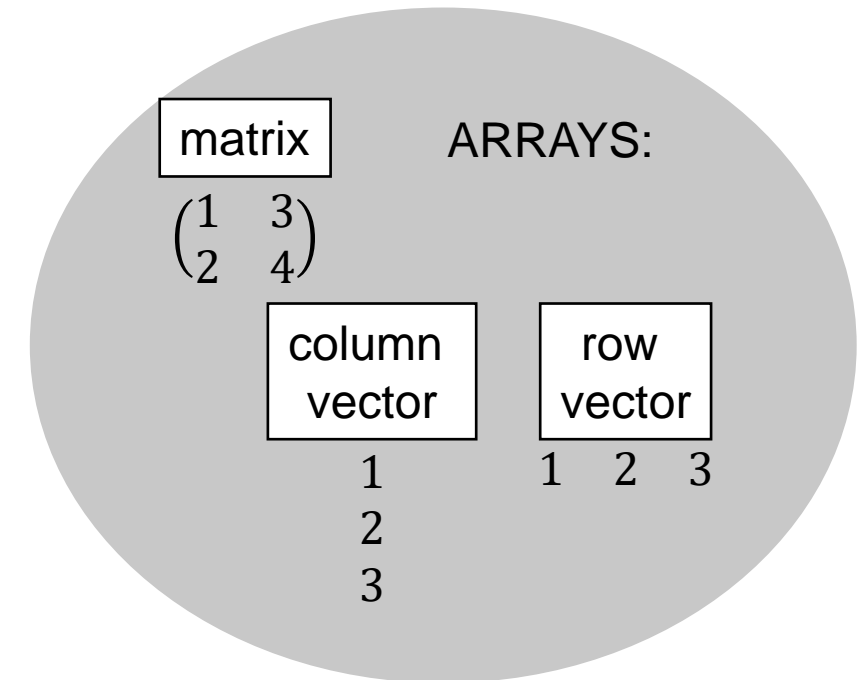
All MATLAB variables are *arrays*. Use square brackets to assign multiple elements i.e. `[3 4 6]` creates a row vector and using semicolons as separators i.e. `[3;4;6]` creates a column vector.

#### Task:

- Create a variable `x` and assign the scalar value 4.
- Create an row vector named `y` with two elements 7 and 9.
- Create a column vector `z` with two elements 7 and 9.

- Create the Matrix 

1	4	6
2	3	4
5	6	9



## Exercise 8.1

### Creating Vectors:

For long vectors, entering individual numbers is not practical. An alternative, shorthand method for creating evenly-spaced vectors is to use the `:` operator in `start:step:end` or the `linspace(first, last, number_of_elements)` function. Both `linspace` and the `:` operator create row vectors. But what if you need a linearly spaced column vector? The transpose operation (`'`) converts a row vector into a column vector.

#### Task:

- Create a row vector named `y` with integer values from 1 to 10 but this time using the `:` operator.
- Create a row vector named `z` that starts at 1, ends at 5, and each element is separated by 0.5.
- Create a row vector named `a` that starts at 3 and ends at 13, with each element separated by 2.
- Create a row vector named `b` that starts at 1, ends at 10, and contains 5 elements.
- Create an evenly-spaced vector from 1 to  $2\pi$  with 100 elements.
- Transpose `b` from a row vector to a column vector using the transpose operator.

## Exercise 8.1

### Creating Matrices:

MATLAB provides multiple way to create matrices. I.e. use the commands `rand()`, `ones()`, and `zeros()` to create matrices containing random numbers, 1's and 0's respectively. The argument designates the size of the matrix. Note the `size()` command extract the arguments dimensions.

#### Task:

- Create a variable named `x` that is a 5-by-5 matrix of random numbers. Create a row vector named `a` that starts at 3 and ends at 13, with each element separated by 2.
- Use `rand` to create a column vector that contains 5 rows and 1 column. Assign the result to a variable named `y`. Transpose `b` from a row vector to a column vector using the transpose operator.
- Use the `ones` function to create a matrix `x` with all values set to 2 and that has 6 rows and 3 columns (6-by-3). Assign the result to a variable named `z`.
- Create a matrix of random numbers which dimensions correspond to `x`.



## Exercise 8.1

### Indexing into Arrays

You can extract values from an array using row, column indexing.

I.e.  $M(5, 6)$  outputs the 5th row and 6th column of  $M$ . You could also use arrays to address multiple elements or the keyword `end` to refer to the last element of the corresponding dimension, i.e.  $M(1:2, 6)$  or  $M(\text{end}, 6)$ . Addressing the whole row use the `:` operator i.e.  $M(1, :)$ .

#### Tasks:


- Create a 7x3 Matrix  $m$  of random numbers.
- **Extract the 5th row and 6th column of  $m$ .** 
- Try  $m(4)$  and  $m(9)$ . Explain the output.
- Extract the second column of the Matrix.
- Extract the second last row and third last column element.
- Change the value of the 5th row and 6th column element to 1. 
- Change the whole 5<sup>th</sup> row to a array of 0's.

## Exercise 8.3 (Optionally)

### Find Array Elements That Meet a Condition

For an advanced technique to address matrix elements is the use of logical values. This data type represents true and false states using the numbers 1 and 0, respectively. Certain MATLAB functions and operators return logical values to indicate fulfillment of a condition. I.e. `<`, `>`, `&`, `true`, `false`, `==`, `<=`...

#### Tasks:

- Create a 5x5 matrix `m` of random numbers.
- Try `a=m<0.5` and explain the output. 
- Try `b = m(a)`.