# Welcome to BME1901 Introductory Computer Sciences 2019-2020 Fall

- Instructor: Dr. Görkem SERBES (C317)
  gserbes@yildiz.edu.tr
  https://avesis.yildiz.edu.tr/gserbes/

- Lab Assistants:
  Doğan Onur ARISOY - *arisoy@yildiz.edu.tr*
  Nihat AKKAN - *nakkan@yildiz.edu.tr*
  Yeliz ERŞAN - *yelize@yildiz.edu.tr*

- Grading:
  - Midterm exams & Assignments & Quizzes 60%
  - Final exam 40%

*only individual submissions allowed!*

# BUILT-IN MATHEMATICAL FUNCTIONS in MATLAB

# Elementary math functions

The matrices A and B are defined as `A = [1,2,3], B = [1,2,3;4,5,6;7,8,9]` for the following examples. If the input is a matrix, func() treats the columns of the matrix as vectors, returning a row vector of the sums of each column.

| Function | Description | Example |
|---|---|---|
| `sum()` | Sum of array elements | `sum(A) = 6`<br>`sum(B) = [12,15,18]` |
| `prod()` | Product of array elements | `prod(A) = 6`<br>`prod(B) = [28,80,162]` |
| `max()` | Largest elements in array | `max(A) = 3`<br>`max(B) = [7,8,9]` |
| `min()` | Smallest elements in array | `min(A) = 1`<br>`min(B) = [1,2,3]` |
| `mean()` | Average or mean value of array | `mean(A) = 2`<br>`mean(B) = [4,5,6]` |

# Elementary math functions ctd.

| Function | Description | Example |
|---|---|---|
| `rand(m,n)` | returns an m-by-n matrix, containing pseudorandom values drawn from the standard uniform distribution on the open interval (0,1) | |
| `round()` | Round to nearest integer | `round([1.5,-1.4]) = [2,-1]` |
| `floor()` | Round toward negative infinity | `floor([1.5,-1.4]) = [1,-2]` |
| `ceil()` | Round toward positive infinity | `ceil([1.5,-1.4]) = [2,-1]` |
| `fix()` | Round toward zero | `fix([1.5,-1.4]) = [1,-1]` |
| `mod(x,y)` | Modulus after division | `mod(3,2) = 1, mod(3,-2) = -1` |
| `rem(x,y)` | Remainder after division | `rem(3,2) = 1, rem(3,-2) = 1` |

# Elementary math functions ctd.

| Function | Description | Example |
|---|---|---|
| `factorial()` | Factorial function | `factorial(5) = 120` |
| `perms()` | All possible permutations | `perms([3,5]) = [5,3;3,5]` |
| `factor()` | Prime factors | `factor(15) = [3,5]` |
| `primes()` | Generate list of prime numbers | `primes(7) = [2,3,5,7]` |
| `isprime()` | Array elements that are prime numbers | `isprime([6,7]) = [0,1]` |
| `lcm()` | Least common multiple | `lcm(30,20) = 60` |
| `gcd()` | Greatest common divisor | `gcd(30,20) = 10` |

# Complex number functions

| Function | Description | Example |
|---|---|---|
| `a+bi`<br>`a+bj` | Complex number input in the command line | `x = 3+5i`<br>`x = 3+5j` |
| `complex()` | Construct complex data from real and imaginary components | `complex(a,b) = a+bi` |
| `abs()` | Absolute value and complex magnitude | `abs(3+4i) = 5` |
| `angle()` | Phase angle of the complex number | `angle(3+4i) = 0.9273` |
| `conj()` | Complex conjugate | `conj(3+5i) = 3-5i` |

# Complex number functions ctd

| Function | Description | Example |
|---|---|---|
| `real()` | Real part of complex number | `real(3+5i) = 3` |
| `imag()` | Imaginary part of complex number | `imag(3+5i) = 5` |
| `isreal()` | Check if input is real array | `x = 3+5i`<br>`isreal(x) = 0`<br>`isreal(x+conj(x))=1` |
| `sign()` | Signum function<br>$X : X \in \mathbb{R}$ sign(x) = 1; x > 0; 0; x = 0; 1; x < 0<br>$X : X \in \mathbb{C}$ sign(x) = x./abs(x) | `sign(3+4i)=0.6+0.8i` |

# Trigonometric functions

- sine, cosine, tangent, cotangent, secant and cosecant functions are defined in MATLAB as `sin(), cos(), tan(), cot(), sec(), csc()`, respectively for inputs in radians.

- Each trig function can be used for the inputs in degrees by adding “d” at the end of the function.

  Example: `sind(90)=1`.

- “h” is added to the end of a function for hyperbolic functions.

  Example : `sinh(1)=1.1752`.

- “a” is added to the beginning of a function for inverse trigonometric functions.

  Example: `acos(1)=0`.

# Exponential functions

| Function | Description | Example |
|---|---|---|
| `sqrt()` | Square root | `sqrt(4) = 2`<br>`sqrt([1,4]) = [1,2]` |
| `exp()` | Exponential | `exp(1) =`<br>`2.718281828459046` |
| `expm1(x)` | Compute `exp(x)-1`<br>accurately for small values of `x` | |
| `nextpow2()` | `p=nextpow2(A)` gives the value of *p* where $2^p > A$ | `nextpow2(8) = 3`<br>`nextpow2(9) = 4` |

# Exponential functions ctd.

| Function | Description | Example |
|---|---|---|
| `log()` | Natural logarithm | `log(exp(2)) = 2` |
| `log10()` | Common (base 10) logarithm | `log10(100) = 2` |
| `log2()` | Base 2 logarithm | `log2(4) = 2` |
| `log1p(x)` | Compute `log(1+x)` accurately for small values of `x` | |
| `reallog()` | Natural logarithm for nonnegative real arrays | `a = exp(2);`<br>`reallog([a,a^2])=[2,4]` |

# Polynomial functions

- An n-degree polynomial with constant coefficients ($a_i$) is defined as,

$$p_1 x^n + \ldots + p_n x + p_{n+1} = 0.$$

- Polynomials are defined as row vectors in MATLAB, i.e.

For example, `p = [3 2 -2]` represents the polynomial $3x^2 + 2x - 2$.

`p1=[1,2,3]; p2 = [2,3,4];` are considered to be defined for the examples given in this subsection.

| Function | Description | Example |
|---|---|---|
| `roots()` | Polynomial roots | `roots(p1) = [-1+1.4142i,-1-1.4142i]` |
| `polyval(p,x)` | Polynomial evaluation | `polyval(p1,[3,5]) = [18,38]` |

# Polynomial functions ctd.

| Function | Description | Example |
|---|---|---|
| `conv()` | Convolution and polynomial multiplication | `conv(p1,p2) = [2,7,16,17,12]` |
| `deconv()` | Deconvolution and polynomial division | `deconv(p1,p2) = 0.5` |
| `poly(A)` | If A is an n-by-n matrix, returns an n+1 element row vector whose elements are the coefficients of the characteristic polynomial (*det(sI-A)*). If A is a vector, returns a row vector whose elements are the coefficients of the polynomial whose roots are the elements of A. | |
| `polyder()` | Polynomial derivative | `polyder(p2) = [4,3]` |
| `polyint(p,k)` | Returns a polynomial representing the integral of polynomial p, using a scalar constant of integration k. | `polyint(p2,0) = [0.666,1.5,4,0]` |

# Matrix operation functions

`v = [1,2,3], A = [1,2,3;4,5,6;7,8,10]` are considered to be defined for the examples given in this subsection.

| Function | Description | Example |
|---|---|---|
| `zeros(m,n)` | returns *mxn* matrix of zeros where *m* and *n* are integers | `zeros(2,2) = [0,0;0,0]` |
| `ones(m,n)` | returns *mxn* matrix of zeros where *m* and *n* are integers | `ones(2,2) = [1,1;1,1]` |
| `eye(n)` | returns *nxn* identity matrix | `eye(2) = [1,0;0,1]` |
| `diag(x)` | When x is a vector of n components, returns a square matrix with the elements of x on the diagonal. When x is a matrix, returns a column vector formed from the elements of the diagonal of x. | `diag(A) = [1,5,10]`<br><br>`diag(v) = [1,0,0;0,2,0;0,0,3]` |
| `det()` | Matrix determinant | `det(A) = -3` |

# Matrix operation functions ctd.

| Function | Description | Example |
|---|---|---|
| `inv()` | Matrix inverse | |
| `norm()` | Vector and matrix norms | `norm(v,1) = 6,`<br>`norm(v,2) = 3.7417` |
| `null()` | Null space | `null(A) =`<br>`Empty matrix: 3-by-0` |
| `rank()` | Rank of matrix | `rank(A) = 3` |
| `rref()` | Reduced row echelon form | |
| `trace()` | Sum of diagonal elements | `trace(A)=16` |
| `sqrtm()` | Matrix square root | `X = sqrtm(A)` → `X*X = A` |

# Matrix operation functions ctd.

| Function | Description | Example |
|---|---|---|
| `eig()` | `[w,D]=eig(X)` produces matrices of eigenvalues (D) and eigenvectors (V) of matrix X | `[w,D]=eig(A)`<br><br>`w =`<br><br>`-0.2235 -0.8658 0.2783`<br>`-0.5039 0.0857 -0.8318`<br>`-0.8343 0.4929 0.4802`<br><br>`D =`<br><br>`16.7075 0 0`<br>`0 -0.9057 0`<br>`0 0 0.1982` |
| `expm(A)` | Matrix exponential | |
| `linsolve(A,B)` | Solve the linear system *A*x=B* | |

# Linear system example

Let

$$\begin{aligned} 3x_1 + 5x_2 + x_3 &= 16 \\ x_1 + x_2 &= 3 \\ + 3x_2 + 5x_3 &= 21 \end{aligned}$$

The solution to that linear system can be obtained in MATLAB as given below.

```
>> A = [3,5,1;1,1,0;0,3,5];
>> B = [16;3;21];
>> linsolve(A,B)

ans =

    1.0000
    2.0000
    3.0000
```

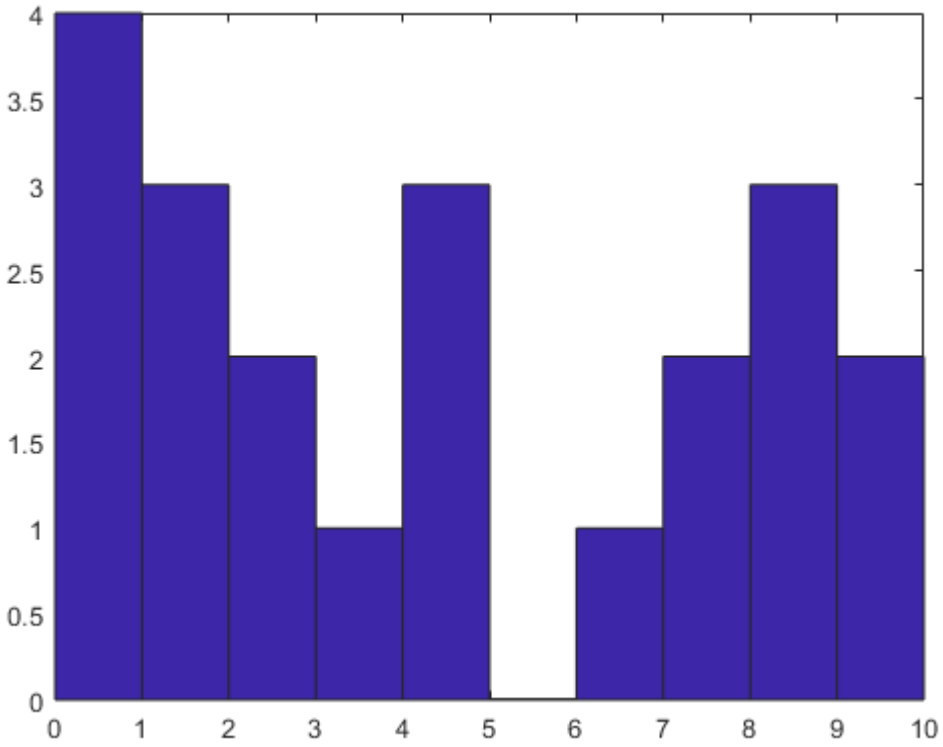
# Histogram in Matlab

- hist(x) (or histogram(x) in recent versions) creates a histogram bar chart of the elements in vector x.
- The elements in x are sorted into 10 equally spaced bins along the x-axis between the minimum and maximum values of x.
- hist displays bins as rectangles, such that the height of each rectangle indicates the number of elements in the bin.
- **Important Note**:

**hist is not recommended. Use histogram instead in recent versions of Matlab.**

# Histogram in Matlab

```
x = [0 2 9 2 5 8 7 3 1 9 4 3 5 8 10 0 1 2 9 5 10];
hist(x)
```



# find function

- **find** is a very important function
  - Returns indices of nonzero values
  - Can simplify code and help avoid loops

- Basic syntax: index=find(cond)

```
» x=rand(1,100);
» inds = find(x>0.4 & x<0.6);
```

- `inds` will contain the indices at which x has values between 0.4 and 0.6. This is what happens:
  - x>0.4 returns a vector with 1 where true and 0 where false
  - x<0.6 returns a similar vector
  - The & combines the two vectors using an **and**
  - The find returns the indices of the 1's

# Element-Wise Functions

- All the functions that work on scalars also work on vectors
  - `» t = [1 2 3];`
  - `» f = exp(t);`
    - is the same as
  - `» f = [exp(1) exp(2) exp(3)];`

- If in doubt, check a function's help file to see if it handles vectors elementwise

- Operators (* / ^) have two modes of operation
  - element-wise
  - standard

# Operators: element-wise

- To do element-wise operations, use the dot: `.` (`.*`, `./`, `.^`). BOTH dimensions must match (unless one is scalar)!
  - `» a=[1 2 3];b=[4;2;1];`
  - `» a.*b, a./b, a.^b → all errors`
  - `» a.*b', a./b', a.^(b') → all valid`

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix}.*\begin{bmatrix}4\\2\\1\end{bmatrix} = ERROR$$

$$\begin{bmatrix}1\\2\\3\end{bmatrix}.*\begin{bmatrix}4\\2\\1\end{bmatrix} = \begin{bmatrix}4\\4\\3\end{bmatrix}$$

$$3\times 1 .* 3\times 1 = 3\times 1$$

$$\begin{bmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{bmatrix}.*\begin{bmatrix}1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{bmatrix} = \begin{bmatrix}1 & 2 & 3\\2 & 4 & 6\\3 & 6 & 9\end{bmatrix}$$

$$3\times 3 .* 3\times 3 = 3\times 3$$

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}.\hat{}\,2 = \begin{bmatrix}1^2 & 2^2\\3^2 & 4^2\end{bmatrix}$$

*Can be any dimension*

# Operators: standard

- Multiplication can be done in a standard way or element-wise
- Standard multiplication (*) is either a dot-product or an outer-product
  - Remember from linear algebra: inner dimensions must MATCH!!
- Standard exponentiation (^) can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix} * \begin{bmatrix}4\\2\\1\end{bmatrix} = 11$$

$$1\times 3 * 3\times 1 = 1\times 1$$

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \wedge 2 = \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} * \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$$

*Must be square to do powers*

$$\begin{bmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{bmatrix} * \begin{bmatrix}1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{bmatrix} = \begin{bmatrix}3 & 6 & 9\\6 & 12 & 18\\9 & 18 & 27\end{bmatrix}$$

$$3\times 3 * 3\times 3 = 3\times 3$$

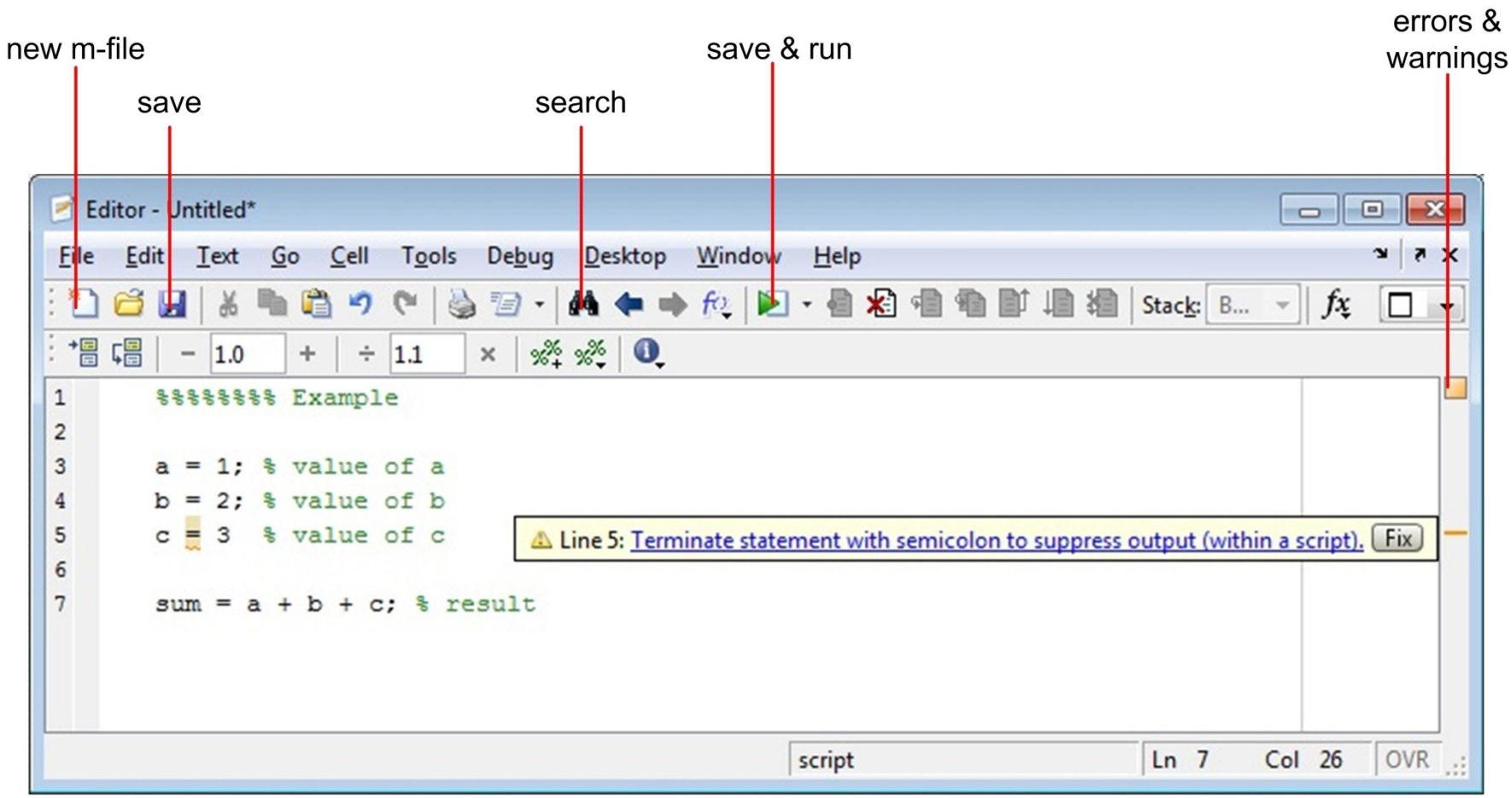
# M-FILES, USER-DEFINED INPUT and OUTPUT

# m-files

- Useful utilities translated into MATLAB (either sequences of command lines or *functions) and saved as m-files in* the working directory are our primary goals.
- In the working directory, we will begin to accumulate a set of m-files that we have created as you use MATLAB.
- The goals in designing a software tool are that it works, it can easily be read and understood, and, hence, it can be systematically modified when required.
- For programs to work well they must satisfy the requirements associated with the problem or class of problems they are intended to solve.
- The program must be readable and hence clearly understandable. Thus, it is useful to decompose major tasks (or the main program) into subtasks (or subprograms) that do specific parts of it. It is much easier to read subprograms, which have fewer lines, than one large main program that doesn't segregate the subtasks effectively, particularly if the problem to be solved is relatively complicated.

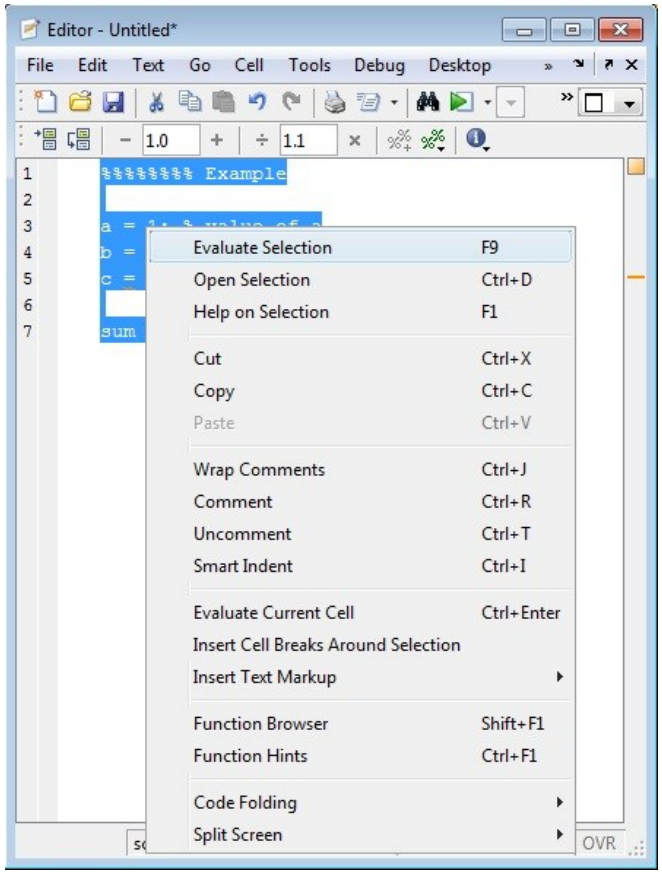
# MATLAB Editor

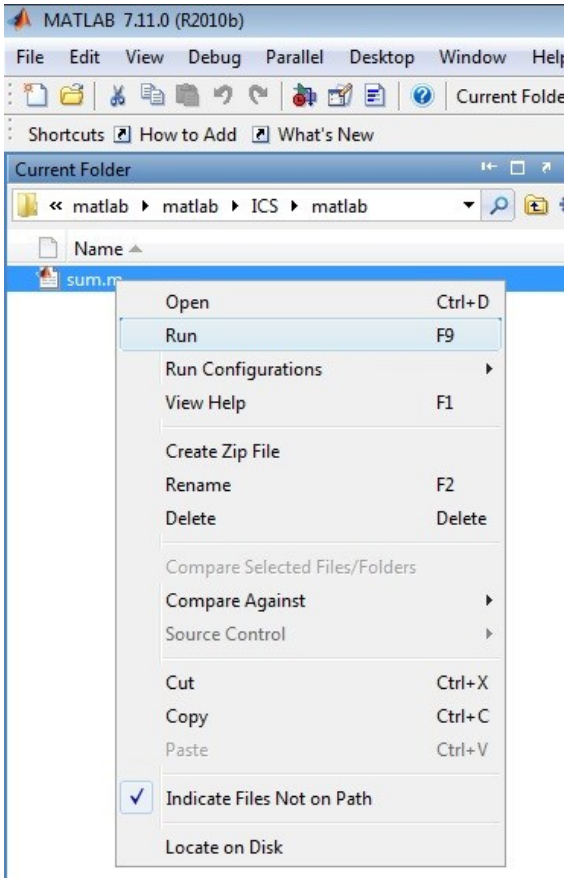
- Example: `a = 1;b = 2;c = 3;sum = a+b+c;`



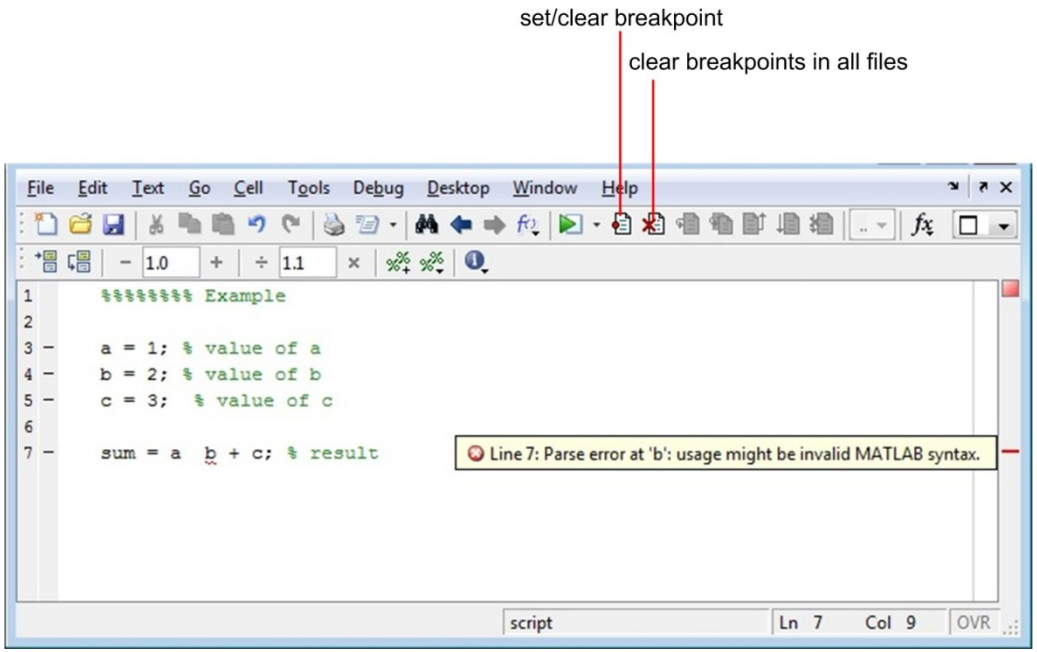
# Running m-files

- In addition to save&run button in MATLAB editor, m-files can also be run as depicted in the figures below.



# Error handling & debugging




```
>> %%%%%%%% Example

a = 1; % value of a
b = 2; % value of b
c = 3; % value of c

sum = a  b + c; % result
??? sum = a  b + c; % result
              |
Error: Unexpected MATLAB expression.
```

# User defined input

- "`input`" function displays a text string in the command window and then waits for the user to provide the requested input. The input value can then be used in subsequent calculations

```
>> x = input('enter a value: ')
enter a value: 14
x =
    14
```

- The same approach can be used to enter a one- or two-dimensional matrix. The user must provide the appropriate brackets and delimiters.

```
>> x = input('enter the input vector: ')
enter the input vector: [1,2,3]
x =
     1     2     3
```

# Output options

- There are several ways to display the contents of an output.
- The general rule of displaying the result is scaled fixed-point format, with 4 digits after the decimal point.
- In MATLAB, format function controls the output format of numeric values displayed in the Command Window.

| Function | Description | Example (π) |
|---|---|---|
| `format short` (default) | Scaled fixed-point format, with 4 digits after the decimal point | `3.1416` |
| `format long` | Scaled fixed-point format, with 15 digits after the decimal point for double | `3.14159265358979` |
| `format shortE` `(longE)` | Floating-point format, with 4 (15) digits after the decimal point for double | `3.1416e+000` `3.141592653589793e+000` |
| `format shortG` `(longG)` | Fixed- or floating-point, whichever is more readable, with 4 (15) digits after the decimal point for double | `3.1416` `3.14159265358979` |
| `format rat` | Ratio of small integers. | `355/113` |

# Output options ctd.

- "`disp()`" displays an array, without printing the array name. If the input contains a text string, the string is displayed.
- The `fprintf` function (formatted print function) gives you even more control over the output than you have with the `disp()` function. In addition to displaying both text and matrix values, you can specify the format to be used in displaying the values, and you can specify when to skip to a new line.
- The general form of the `fprintf` command contains two arguments, one a string and the other a list of variables:

```
fprintf(format-string, var,…)
```

# Output options ctd.

- Consider the following example:

```
>> x = 5;
>> fprintf('The value of x is %f \n', x);
The value of x is 5.000000
>>
```

- The string, which is the first argument inside the `fprintf` function, contains a placeholder (%) where the value of the variable (in this case, x) will be inserted. The placeholder also contains formatting information (see next slide).
- "`\n`" causes MATLAB to start a new line, we need to use \n, called a *linefeed*, at the end of the string.

# Output options ctd.

Other special format commands used for "`fprintf`" are listed in the table below

| Format | Description |
|---|---|
| `%d` | Base 10 values |
| `%e` | Exponential notation (3.141593e+00) |
| `%f` | Fixed-point notation |
| `%g` | The more compact of %e or %f, with no trailing zeros |
| `%E` | Same as %e, but uppercase (3.141593E+00) |
| `%G` | The more compact of %E or %f, with no trailing zeros |
| `%c` | Single character |
| `%s` | String of characters |
| `%o` | Base 8 (octal) |
| `%u` | Base 10 |
| `%x` | Base 16 (hexadecimal), lowercase letters |
| `%X` | Same as %x, uppercase letters |

# Examples

```
>> B = [8.8  7.7 ; ...
    8800 7700];
fprintf('X is %f meters or %f mm\n', 9.9, 9900, B)

X is 9.900000 meters or 9900.000000 mm
X is 8.800000 meters or 8800.000000 mm
X is 7.700000 meters or 7700.000000 mm
```

```
>> a = [1.02 3.04 5.06];
fprintf('%d\n', a);
1.020000e+000
3.040000e+000
5.060000e+000
```

```
>> a = [1 9 23];
>> fprintf('%u\n', a);
1
9
23
>> fprintf('%o\n', a);
1
11
27
>> fprintf('%x\n', a);
1
9
17
```

```
>> x = [2 987654321];
>> fprintf('x'' in degeri: %f \n',x)
x' in degeri: 2.000000
x' in degeri: 987654321.000000
>> fprintf('The value of x: %e \n',x)
The value of x: 2.000000e+000
The value of x: 9.876543e+008
>> fprintf('The value of x: %g \n',x)
The value of x: 2
The value of x: 9.87654e+008
```