# Programming in MATLAB/Octave

Lecture 6 – Monday March 27, 2017

# Outline

- MATLAB Environment
- Identifiers
- Constants
- Variables
- Vectors and Matrices
- Examples

# Outline

- **MATLAB Environment**
- Identifiers
- Constants
- Variables
- Vectors and Matrices
- Examples

# MATLAB Environment

- MATLAB is a program for doing numerical computation. It was originally designed for solving linear algebra type problems using matrices.
- It's name is derived from **MAT**rix **LAB**oratory.
- MATLAB is a software environment for interactive numerical computations.
- MATLAB is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.

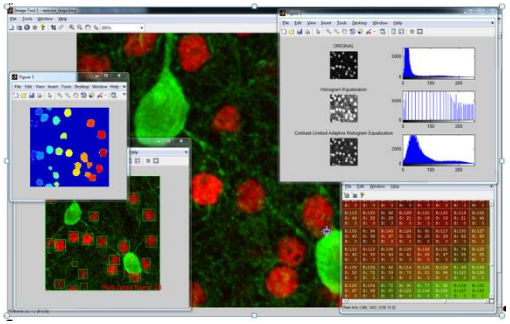
# MATLAB Environment

- **Tasks**
  - ◇ Matrix computations and linear algebra
  - ◇ Solving nonlinear equations
  - ◇ Numerical solution of differential equations
  - ◇ Mathematical optimization
  - ◇ Statistics and data analysis
  - ◇ Signal processing
  - ◇ Modelling of dynamical systems
  - ◇ Solving partial differential equations
  - ◇ Simulation of engineering systems

# MATLAB Environment

- **Usage**

  Matlab used (on a daily basis) in many engineering companies



# MATLAB Environment

- **Background**

**Matlab** = **Mat**rix **Lab**oratory

- Originally a user interface for numerical linear algebra routines (Lapak/Linpak)

- Commercialized 1984 by The Mathworks

- Since then heavily extended (defacto-standard)

### ◇ Alternatives

Matrix-X

Octave (free; GNU)

Lyme (free; Palm)

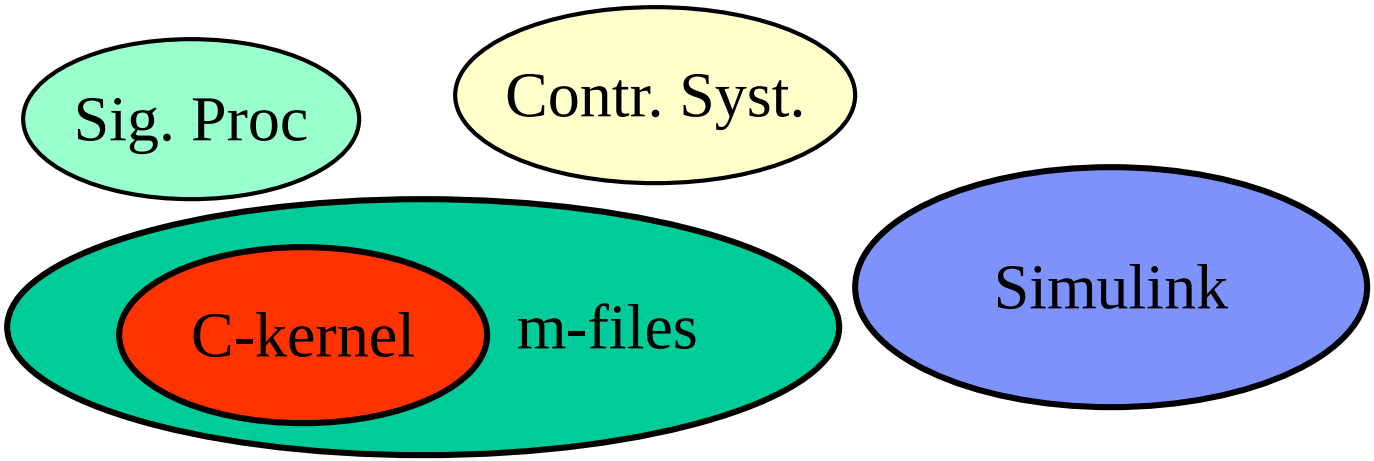
### ◇ Complements

Maple (symbolic)

Mathematica (symbolic)

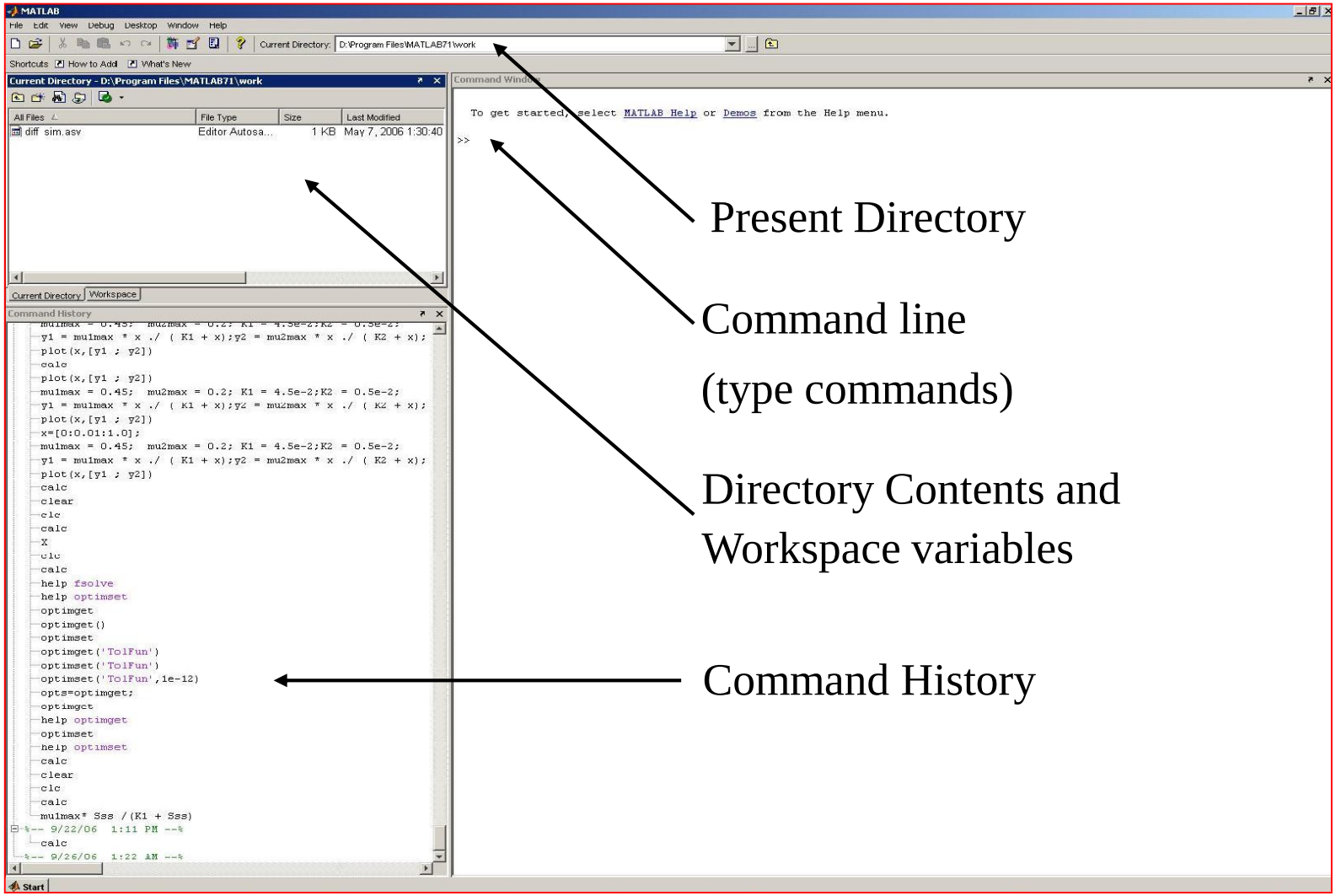
# MATLAB Environment

- **Functionality**
  - ◇ Core functionality: compiled C-routines
  - ◇ Most functionality is given as ***m-files***, grouped into toolboxes
    - – m-files contain source code, can be copied and altered
    - – m-files are platform independent (PC, Unix/Linux, MAC)
  - ◇ Simulation of dynamical systems is performed in Simulink.



# MATLAB Environment

- **Windows**



# MATLAB Environment

- **MATLAB Variable Names**
  - ◇ Variable names ARE case sensitive
  - ◇ Variable names can contain up to 63 characters (as of MATLAB 6.5 and newer)
  - ◇ Variable names must start with a letter followed by letters, digits, and underscores.

# MATLAB Environment

- **MATLAB Special Variables**

| | |
|---|---|
| ans | Default variable name for results |
| pi | Value of π |
| eps | Smallest incremental number |
| inf | Infinity |
| NaN | Not a number e.g. 0/0 |
| i and j | i = j = square root of -1 |
| realmin | The smallest usable positive real number |
| realmax | The largest usable positive real number |

# MATLAB Environment

- **MATLAB Math & Assignment Operators**

| | | | | | |
|---|---|---|---|---|---|
| Power | ^ | or | .^ | a^b | or a.^b |
| Multiplication | * | or | .* | a*b | or a.*b |
| Division | / | or | ./ | a/b | or a./b |
| or | \ | or | .\ | b\a | or b.\a |

NOTE: 56/8 = 8\56

| | | | |
|---|---|---|---|
| - (unary) + (unary) | | | |
| Addition | + | a + b | |
| Subtraction | - | a - b | |
| Assignment | = | a = b | (assign b to a) |

# MATLAB Environment

- **Other MATLAB Symbols**

| | |
|---|---|
| >> | prompt |
| . . . | continue statement on next line |
| , | separate statements and data |
| % | start comment which ends at end of line |
| ; | (1) suppress output |
| | (2) used as a row separator in a matrix |
| : | specify range |

# MATLAB Environment

- **Interactive Calculations**

  Matlab is interactive, no need to declare variables

  >> 2+3*4/2

  >> a=5e-3; b=1; a+b

  Most elementary functions and constants are already defined

  >> cos(pi)

  >> abs(1+i)

  >> sin(pi)

  Last call gives answer 1.2246e-016 !?

# MATLAB Environment

- **Variable and Memory Management**

Matlab uses double precision (approx. 16 significant digits)

```
>> format long
>> format compact
```

All variables are shown with

```
>> who
>> whos
```

Variables can be stored on file

```
>> save filename
>> clear
>> load filename
```

# MATLAB Environment

- **Some Useful MATLAB commands**
  - ◇ who — List known variables
  - ◇ whos — List known variables plus their size
  - ◇ help — >> help sqrt — Help on using sqrt
  - ◇ lookfor — >> lookfor sqrt — Search for keyword sqrt in m-files
  - ◇ what — >> what a: — List MATLAB files in a:
  - ◇ clear — Clear all variables from work space
  - ◇ clear x y — Clear variables x and y from work space
  - ◇ clc — Clear the command window

# MATLAB Environment

- **Some Useful MATLAB commands**
  - ◇ what — List all m-files in current directory
  - ◇ dir — List all files in current directory
  - ◇ ls — Same as dir
  - ◇ type test — Display test.m in command window
  - ◇ delete test — Delete test.m
  - ◇ cd a: — Change directory to a:
  - ◇ chdir a: — Same as cd
  - ◇ pwd — Show current directory
  - ◇ which test — Display directory path to ‘closest’ test.m

# MATLAB Environment

- **The Help System**

  Search for appropriate function

  >> lookfor *keyword*

  Rapid help with syntax and function definition

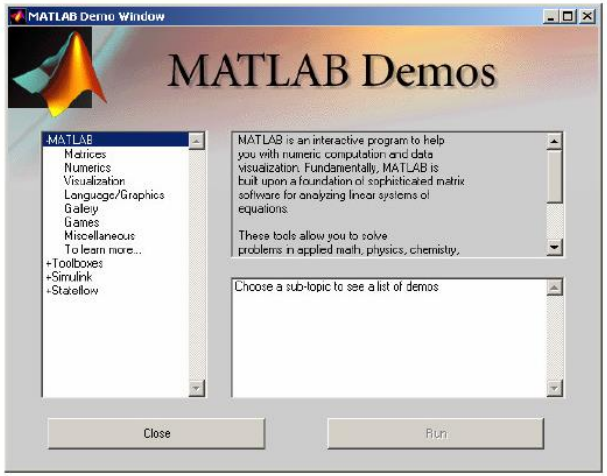
  >> help *function*

  An advanced hyperlinked help system is launched by

  >> helpdesk

  Demo launched by

  >> demo



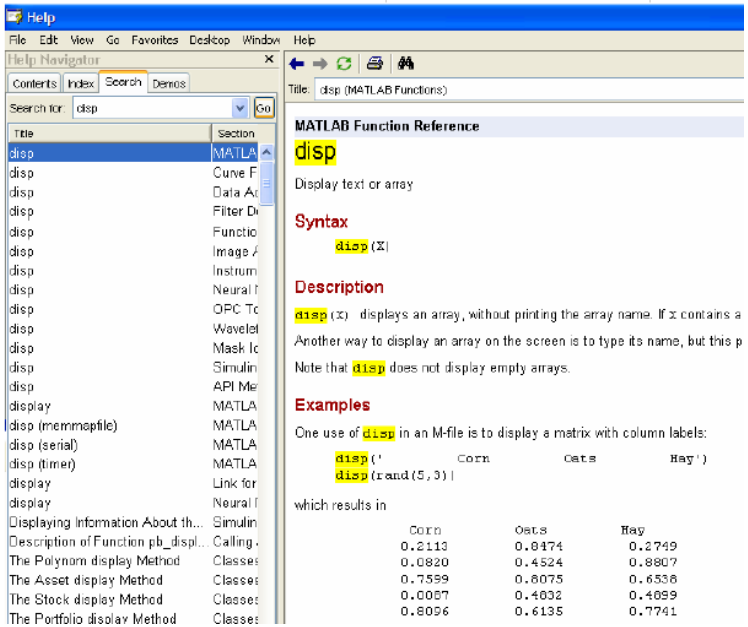
# MATLAB Environment

- **Technical Documentations**

  To get a nicer version of help with examples and easy-to-read descriptions

  >> doc *function*

  >> doc *disp*




  To search for a function by specifying keywords:

  »doc + *Search tab*

# Outline

- MATLAB Environment
- **Identifiers**
- Constants
- Variables
- Vectors and Matrices
- MATLAB Functions

# Identifiers

- Identifiers are all the words that build up the program
- An identifier is a sequence of letters, digits and underscores “ _ ”
- Maximal length of identifiers is 63 characters
- Can’t start with a digit
- Can’t be a reserved word

Legal identifiers

- time
- day_of_the_week
- bond007
- findWord

Illegal identifiers

- 007bond
- #time
- ba-baluba
- if
- while

# Identifiers

- **Reserved words**

  There are 17 reserved words:

| | |
|---|---|
| for | if |
| function | elseif |
| otherwise | continue |
| try | global |
| break | while |
| end | case |
| return | else |
| switch | persistent |
| catch | |

# Outline

- MATLAB Environment
- Identifiers
- **Constants**
- Variables
- Vectors and Matrices
- MATLAB Functions

# Constants

The value of a constant is fixed and does not change throughout the program.

Numbers

100 0.3

Chars

‘c’

Strings

‘I like to eat sushi’

‘1 + 2’

Arrays

[ 1 2 3 4 5 ]

Matrices

[5 3
 4 2]

# Outline

- MATLAB Environment
- Identifiers
- Constants
- **Variables**
- Vectors and Matrices
- Plotting with MATLAB
- MATLAB Functions

# Variables

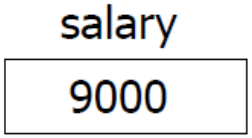
Variable

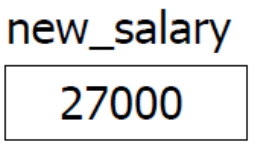
Constant

```
>> salary = 9000;
>> new_salary = salary * 3;
>> disp(new_salary);
27000
```

Library function

Computer memory




If we update salary, new_salary will NOT be updated automatically

# Outline

- MATLAB Environment
- Identifiers
- Constants
- Variables
- **Vectors and Matrices**
- MATLAB Functions

# Vectors and Matrices

Vectors (arrays) are defined as

```
>> v = [1, 2, 4, 5]
>> w = [1; 2; 4; 5]
```

$$v = \begin{bmatrix} 1 & 2 & 4 & 5 \end{bmatrix}$$

$$w = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 5 \end{bmatrix}$$

Matrices (2D arrays) defined similarly

```
>> A = [1,2,3;4,-5,6;5,-6,7]
```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & -6 & 7 \end{bmatrix}$$

# Vectors and Matrices

- **Matrix Operators**

All common operators are overloaded

```
>> v + 2
```

Common operators are available

```
>> B = A'
>> A*B
>> A+B
```

Note:

Matlab is case-sensitive

A and a are two different variables

# Vectors and Matrices

- **Indexing Matrices**

  Indexing using parentheses

  ```
  >> A(2,3)
  ```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

  Index submatrices using vectors of row and column indices

  ```
  >> A([2 3],[1 2])
  ```

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

  Ordering of indices is important!

  ```
  >> B=A([3 2],[2 1])
  >> B=[A(3,2),A(3,1);A(2,2);A(2,1)]
  ```

$$B = \begin{bmatrix} 6 & 5 \\ -5 & 4 \end{bmatrix}$$

# Vectors and Matrices

- **Indexing Matrices**

  Index complete row or column using the colon operator

  >> A(1,:)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

  Can also add limit index range

  >> A(1:2,:)

  >> A([1 2],:)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

  General notation for colon operator

  >> v=1:5

  >> w=1:2:5

$$v = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

$$w = \begin{bmatrix} 1 & 3 & 5 \end{bmatrix}$$

# Vectors and Matrices

- **Numerical Linear Algebra**

Basic numerical linear algebra

```
>> z=[1;2;3]; x=inv(A)*z
>> x=A\z
```

$$Ax = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$x = A^{-1} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Many standard functions predefined

```
>> det(A)
>> rank(A)
>> eig(A)
```

The number of input/output arguments can often be varied

```
>> [V,D]=eig(A)
```

# Outline

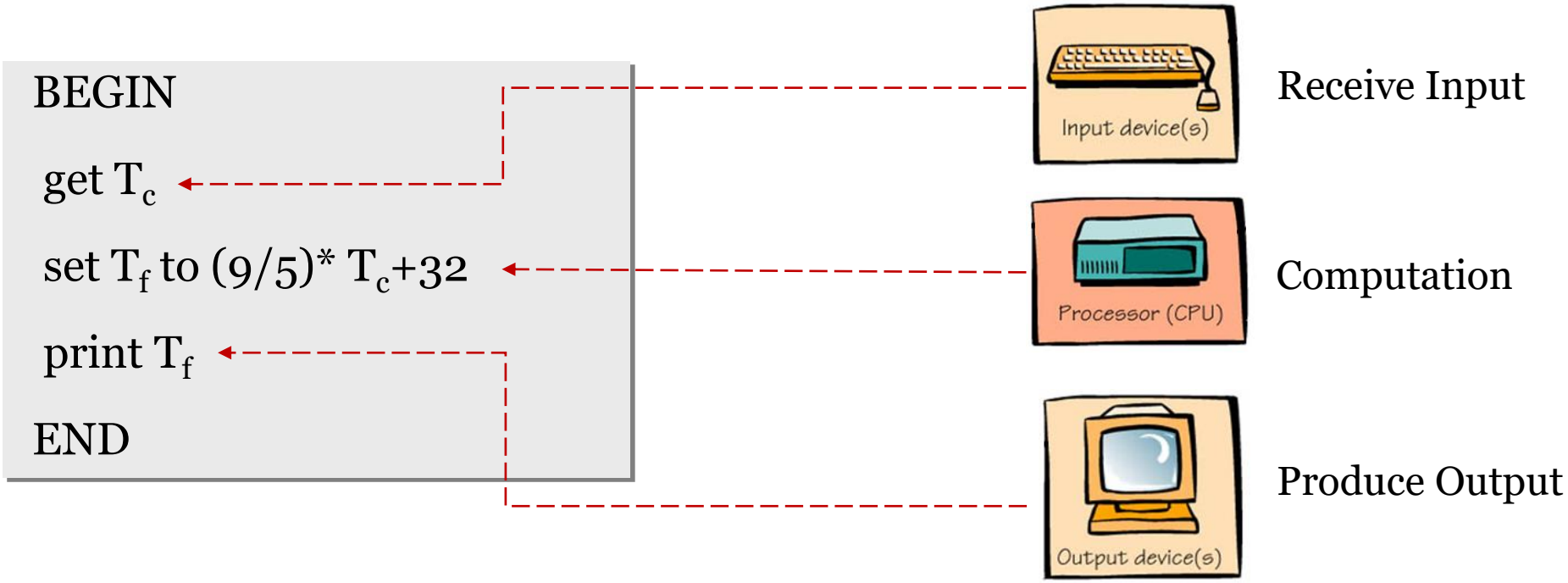
- MATLAB Environment
- Identifiers
- Constants
- Variables
- Vectors and Matrices
- **Examples**

# Examples

- **Example-1: Temperature Conversion**

Convert temperature from Celsius to Fahrenheit using the following formula :

$$T_f = 9/5 * T_c + 32$$



# Examples

- **Example-1: Temperature Conversion**

```
%Script file: temp_conversion.m
%
% Purpose:
% To convert an input temperature from degrees Celsius
% to an output temperature in Fahrenheit
%
%Record of revisions:
%Date       Programmer      Description of change
%=====      ==========      =====================
%30/10/2017 Alaa Khamis     Orignial Code
%
%Define variables:
% temp_c: Temperature in Celsius
% temp_f: Temperature in Fahrenheit

%Prompt the user for the input temperature
temp_c=input('Enter the temperature in degrees Celsius:');

%Convert to Fahrenheit
temp_f=(9/5)*(temp_c)+32;

%Write the results
fprintf('%6.2f degrees Celsius %6.2f Fahrenheit.\n',temp_c,temp_f);
```

```
>> temp_conversion
Enter the temperature in degrees Celsius:50
 50.00 degrees Celsius 122.00 Fahrenheit.
>>
```

# Examples

- **Example-2: Flying Time**

Write an algorithm in pseudo-code to determine the flying time between two cities given the distance between them and the average speed of the airplane. Convert this pseudo-code into a Matlab program to calculate the flying time between Cairo and Aswan (distance is 850 Km) if the average speed of the airplane is 550 km/hr.

**Inputs:** Distance "distance"
and Average Speed
"speed"

**Outputs:** Flying Time
"time"

**Pseudo-code:**

```
BEGIN
get distance, speed
set time to distance/speed
print time
END
```

# Examples

- **Example-2: Flying Time**

**Matlab Program:**

```
%Script file: flying_time.m
% Purpose:
% To determine the flying time between two cities given the distance
% between them and the average speed of the airplane
%Record of revisions:
%Date       Programmer      Description of change
%=====      ==========      =====================
%03/05/2017 Alaa Khamis     Orignial Code

%Define variables:
% distance: Distance between two cities (Km)
% speed:    Average speed of the airplane (Km/hr)
% time:     Flying time betwee the two cities

%Prompt the user for the distance
distance=input('Enter the distance between the two cities (Km):');

%Prompt the user for the distance
speed=input('Enter the average speed of the airplane (Km/hr):');

%Calculate the flying time
time=distance/speed;

%Write the results
fprintf('The flying time between the two cities is %6.2f Hours.\n',time);
```

# Examples

- **Example-2: Flying Time**

**Program run:**

```
>> flying_time
Enter the distance between the two cities (Km):850
Enter the average speed of the airplane (Km/hr):550
The flying time between the two cities is   1.55 Hours.
>>
```

# Examples

- **Example-3: Quadratic Roots**

Write an algorithm in pseudo-code that computes the roots of the quadratic equation.

$$as^2 + bs + c = 0$$

Implement this algorithm using MATLAB.

1. Prompting for input of coefficients a, b, and c;
2. Format the display of the computed roots s1 and s2;

# Examples

- **Example-3: Quadratic Roots**

$$as^2 + bs + c = 0$$

Inputs: a, b and c

Outputs: roots s

Expression:

$$s = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Algorithm in Pseudo-code:

BEGIN

get a,b and c

set $s = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

print s

END

# Examples

- **Example-3: Quadratic Roots**

Implementation in Matlab:

```
%Script file: rqroots.m.m
% Purpose:
% To that computes the roots of the quadratic equation
%Record of revisions:
%Date       Programmer      Description of change
%=====      ==========      =====================
%03/05/2017 Alaa Khamis     Orignial Code

%Define variables:
% a,b,c: input coefficients
% x,y:   output quadratic roots

% prompt for coefficient input
a = input('Enter quadratic coefficient a: ');
b = input('Enter quadratic coefficient b: ');
c = input('Enter quadratic coefficient c: ');
disp('')

% compute intermediate values x & y
x = -b/(2*a);
y = sqrt(b^2-4*a*c)/(2*a);

% compute roots
s1 = x+y;

% display roots
disp('Value of first quadratic root: '),disp(s1);
s2 = x-y;
disp('Value of second quadratic root: '),disp(s2);
```

```
Enter quadratic coefficient a: 1
Enter quadratic coefficient b: -5
Enter quadratic coefficient c: 3
Value of first quadratic root:
    4.3028
Value of second quadratic root:
    0.6972
```

# Examples

- **Example-4: Carbon 14 Dating**

A radioactive isotope of an element is a form of the element that is not stable. Instead, it spontaneously decays into another element over a period of time. Radioactive decay is an exponential process. If $Q_o$ is the initial quantity of a radioactive substance at time t=0, then the amount of that substance which will be present at any time t in the future is given by:

$$Q(t) = Q_o e^{-\lambda t}$$

Where $\lambda$ is the radioactive decay constant.

# Examples

- **Example-4: Carbon 14 Dating**

  Because radioactive decay occurs at a known rate, it can be used as a clock to measure the time that has elapsed since the decay started. If we know the initial amount of radioactive material $Q_o$ present in a sample an the amount of material Q left at the current time t, we can solve for t in the previous equation to determine how long the decay has been going on. The resulting equation is:

$$t_{decay} = -\frac{1}{\lambda} \log_e \frac{Q}{Q_o}$$

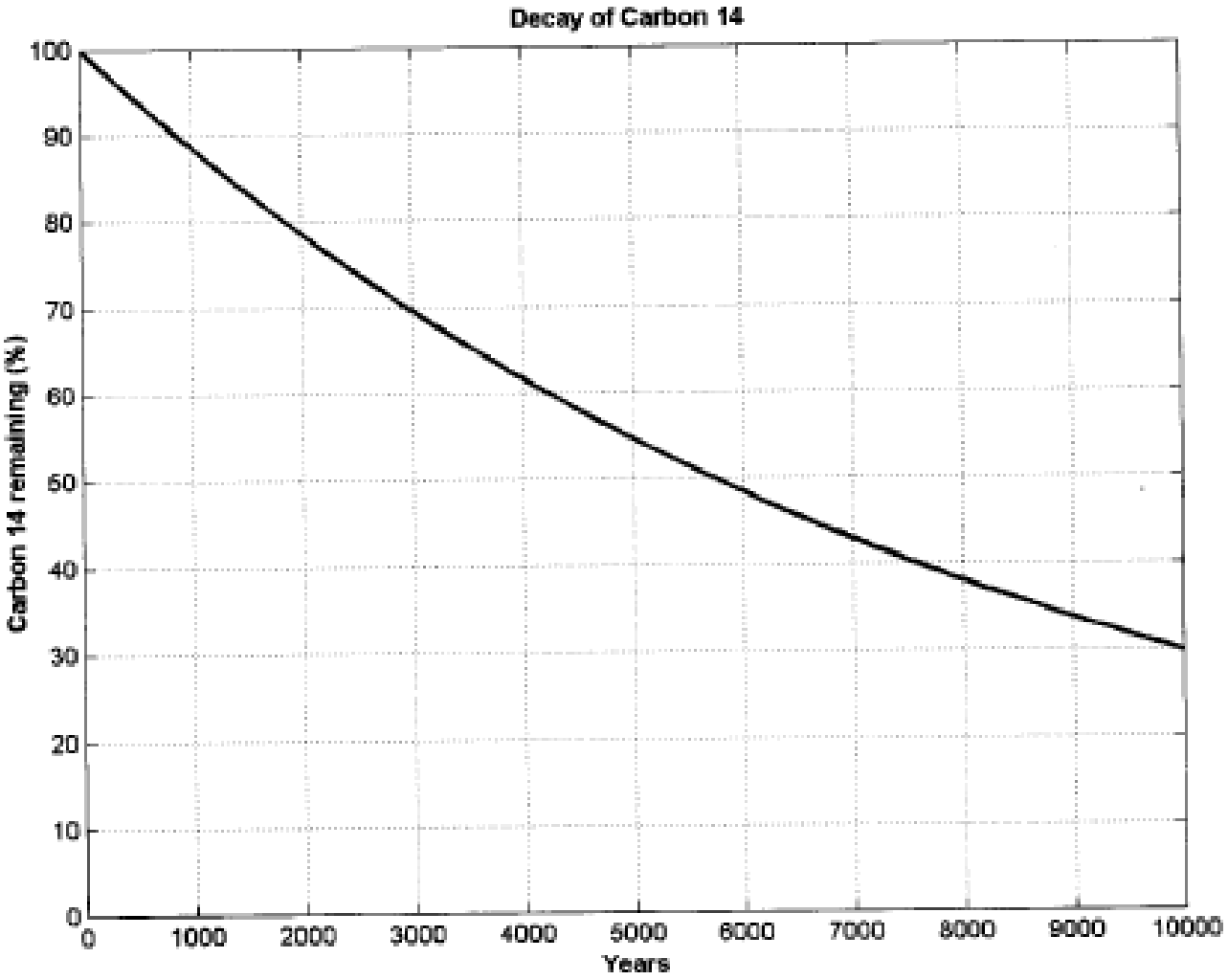
# Examples

- **Example-4: Carbon 14 Dating**

  Archaeologists use a radioactive clock based on carbon 14 to determine the time that has passed since a once-living thing died. Carbon 14 is continually taken into the body while a plant or animal is living, so the amount of it present in the body at the time of death is assumed to be known. The decay rate of carbon 14 is well known to be 0.00012097/year. Therefore, the amount of carbon 14 remaining now can be accurately measured, and the previous equation can be used to determine how long ago the living thing died.

# Examples

- **Example-4: Carbon 14 Dating**



# Examples

- **Example-4: Carbon 14 Dating**

Write an algorithm in pseudo-code that reads the percentage of carbon 14 remaining in a sample, calculates the age of the sample from it, and prints out the result with proper units.

Inputs: $Q/Q_o$

Outputs: age in years

Expression:

$$t_{decay} = -\frac{1}{\lambda} \log_e \frac{Q}{Q_o}$$

Algorithm in Pseudo-code:

BEGIN

get $Q/Q_o$

set $t_{decay} = -\frac{1}{\lambda} \log_e \frac{Q}{Q_o}$

print $t_{decay}$

END

# Examples

- **Example-4: Carbon 14 Dating**

Implementation in Matlab:

```
%Script file: c14_date.m

%Purpose:
% To calculate teh age of an organic sample from the
% percentaeg of the orginial carbon 14 remaining in the sample

%Record of revisions:
%Date       Programmer      Description of change
%=====      ==========      =====================
%03/05/2017 Alaa Khamis     Orignial Code

%Define variables
% age: the age of the sample in years
% lambda: the radioactive decay constant for carbon 14, in units 1/years.
% percent: the percentage of the carbon 14 remaining at time of the measurement
% ratio: the ratio of the carbon 14 remaining at time of the measurement
%        to the time of the orignal amount of carbon 14.

%Set the decay constant for Carbon-14
lambda=0.00012097;

%Prompt the user for the percntage of C-14 remaining.
percent=input('Enter the percntage of Carbon-14 remaining:\n');

%Perform calculations
ratio=percent/100;   % Convert to fractional ratio
age=(-1.0/lambda)*log(ratio);  % Get age in years

%tell the user about the age of the sample.
string=['The age of the sample is '  num2str(age)  ' years'];
disp(string);
```

# Examples

- **Example-4: Carbon 14 Dating**

Implementation in Matlab:

```
Enter the percntage of Carbon-14 remaining:
50
The age of the sample is 5729.9097 years
```