

A faint, dotted world map is visible in the background of the slide, centered behind the text.

# Lecture 3: Array Applications, Cells, Structures & Script Files

# Euclidean Vectors

- An Euclidean vector (or geometric vector, or simply a vector) is a geometric entity that has both **magnitude** and **direction**.
- In physics, vectors are used to represent physical quantities that have both magnitude and direction, such as force, acceleration, electric field, etc.
- Vector algebra: adding and subtracting vectors, multiplying vectors, scaling vectors, etc.

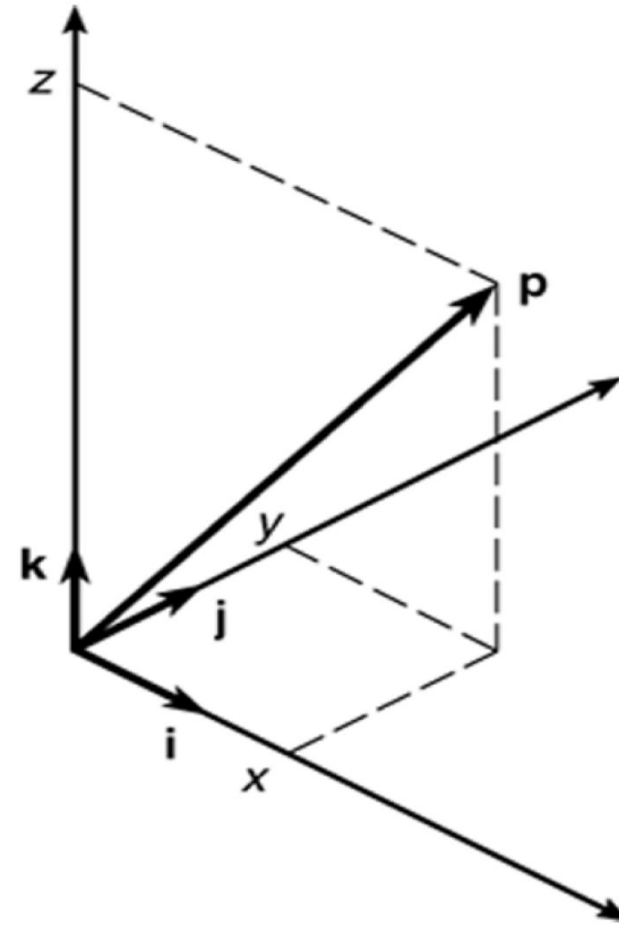


# Euclidean Vectors in MATLAB

- We specify a vector using its Cartesian coordinates.
- Hence, the vector  $\mathbf{p}$  can be specified by three components:  $x$ ,  $y$  and  $z$ , and can be written in MATLAB as:

$$\mathbf{p} = [x, y, z];$$

- MATLAB supports 2-D and 3-D vectors, and even higher dimensional ones.



# Magnitude, Length, Absolute Value

- In MATLAB, `length()` of a vector is **not** its magnitude. It is the number of elements in the vector.
- The **absolute value** of a vector **a** is a vector whose elements are the absolute values of the elements of **a**.
- The **magnitude** of a vector is its Euclidean norm or geometric length as shown:

$$\mathbf{a} = a_x \mathbf{i} + a_y \mathbf{j} + a_z \mathbf{k}$$

$$\|\mathbf{a}\| = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

```
>> a = [2, -4, 5]
a =
     2     -4     5

>> length(a)
ans =
     3

>> abs(a)
ans =
     2     4     5

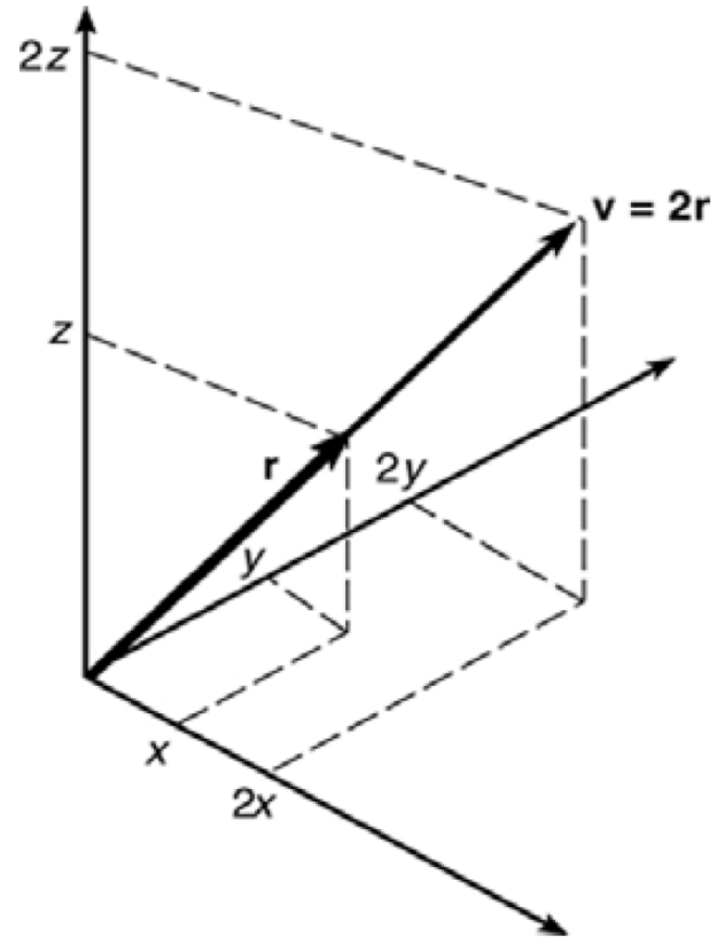
>> sqrt(a*a') % magnitude
ans =
     6.7082

>> sqrt(sum(a.*a)) %magnitude
ans =
     6.7082
```

$$\|\mathbf{a}\| = \sqrt{2^2 + (-4)^2 + 5^2} = \sqrt{[2 \quad -4 \quad 5] \begin{bmatrix} 2 \\ -4 \\ 5 \end{bmatrix}} = 6.7082$$

# Vector Scaling

- For vector:  
 $\mathbf{a} = a_x \mathbf{i} + a_y \mathbf{j} + a_z \mathbf{k}$
- Scaling this vector by a factor of 2 gives:
- $\mathbf{v} = 2\mathbf{a}$   
 $= 2a_x \mathbf{i} + 2a_y \mathbf{j} + 2a_z \mathbf{k}$
- This is just like MATLAB scalar multiplication of a vector:  
 $\mathbf{v} = 2 * [x, y, z];$



# Adding and Subtracting Vectors

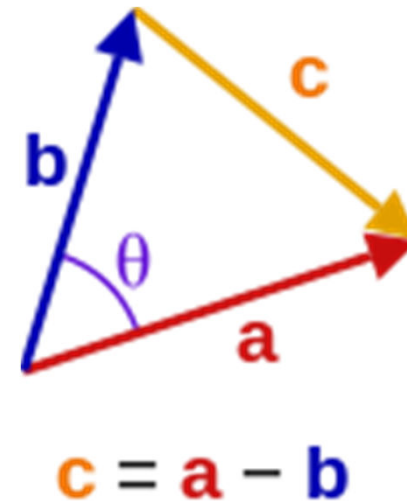
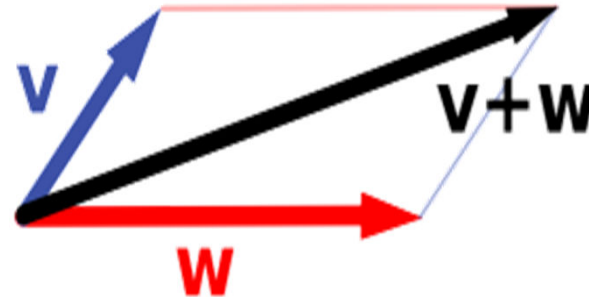
- Vector addition by geometry: The parallelogram law.
- Or, mathematically:

$$\mathbf{a} = a_x \mathbf{i} + a_y \mathbf{j} + a_z \mathbf{k}$$

$$\mathbf{b} = b_x \mathbf{i} + b_y \mathbf{j} + b_z \mathbf{k}$$

$$\begin{aligned} \mathbf{a} + \mathbf{b} &= (a_x + b_x) \mathbf{i} \\ &\quad + (a_y + b_y) \mathbf{j} \\ &\quad + (a_z + b_z) \mathbf{k} \end{aligned}$$

- Same as vector addition and subtraction in MATLAB.



# Exercise

```
>> a = [2 -4 6]
```

```
a =  
    2    -4    6
```

```
>> b = [3 -1 -1]
```

```
b =  
    3    -1   -1
```

```
>> c = a + b
```

```
c =  
    5    -5    5
```

```
>> d = a - b
```

```
d =  
   -1    -3    7
```

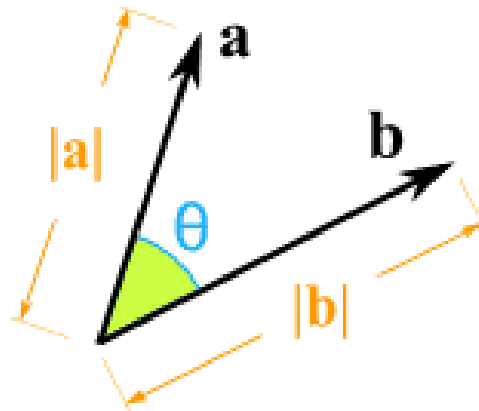
```
>> e = 2*a
```

```
e =  
    4   -8   12
```



# Dot Product

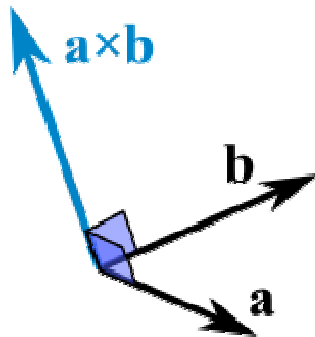
- The dot product of vectors results in a scalar value.
- $\mathbf{a} \cdot \mathbf{b}$   
 $= (a_x b_x + a_y b_y + a_z b_z)$   
 $= \|\mathbf{a}\| \|\mathbf{b}\| \cos(\theta)$



```
>> a = [2 -4 6];  
>> b = [3 -1 -1];  
>> c = a * b'  
c =  
    4  
  
>> c = sum(a .* b)  
c =  
    4  
  
>> c = dot(a, b)  
c =  
    4
```



# Cross Product



$$\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \sin(\theta) \mathbf{n}$$

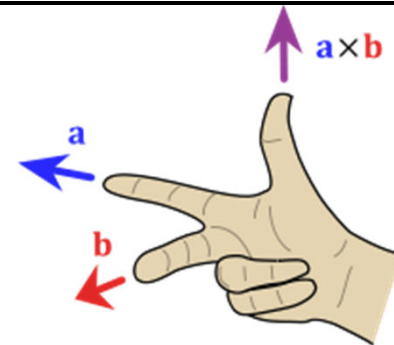
$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} a_y & a_z \\ b_y & b_z \end{vmatrix} \mathbf{i} - \begin{vmatrix} a_x & a_z \\ b_x & b_z \end{vmatrix} \mathbf{j} + \begin{vmatrix} a_x & a_y \\ b_x & b_y \end{vmatrix} \mathbf{k}$$

```
>> a = [2 -4 6];
>> b = [3 -1 -1];
>> cross(a, b)
ans =
    10    20    10

>> syms x y z
>> det([x y z; 2 -4 6; 3 -1 -1])
ans =
 10*x + 20*y + 10*z

>> cross([1 0 0], [0 1 0])
ans =
    0    0    1
```



# Complex Numbers

```
>> a = 7 + 4j
a =
    7.0000 + 4.0000i

>> [theta, rho] = cart2pol(real(a), imag(a))
theta =
    0.5191
rho =
    8.0623

>> rho = abs(a) % magnitude of complex number
rho =
    8.0623

>> theta = atan2(imag(a), real(a))
theta =
    0.5191
% atan2 is four quadrant inverse tangent

>> b = 3 + 4j
b =
    3.0000 + 4.0000i

>> a+b
ans =
   10.0000 + 8.0000i

>> a*b
ans =
    5.0000 + 40.0000i
```



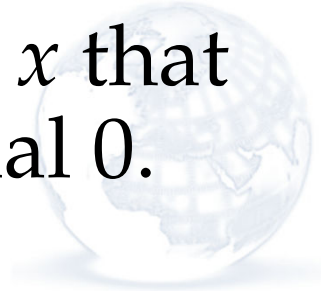
# Polynomials

- A polynomial can be written in the form:  
$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0$$

- Or more concisely:

$$\sum_{i=0}^n a_i x^i$$

- We can use MATLAB to find all the roots of the polynomial, i.e., the values of  $x$  that makes the polynomial equation equal 0.



# Exercise

- Polynomial Roots:  
 $x^3 - 7x^2 + 40x - 34 = 0$
- Roots are  $x = 1, x = 3 \pm 5i$ .
- We can also build polynomial coefficients from its roots.
- We can also multiply (convolution) and divide (deconvolution) two polynomials.

```
>> a = [1 -7 40 -34];
```

```
>> roots(a)
```

```
ans =
```

```
3.0000 + 5.0000i
```

```
3.0000 - 5.0000i
```

```
1.0000
```

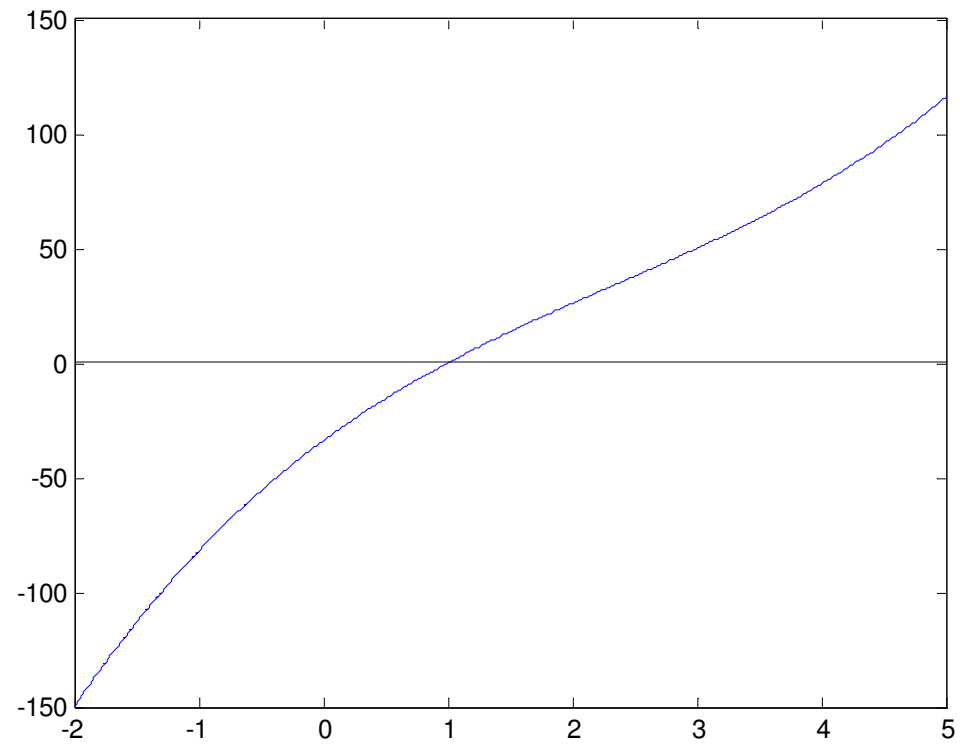
```
>> poly([1 3+5i 3-5i])
```

```
ans =
```

```
1 -7 40 -34
```

# Just for fun... Plot...

```
>> x = -2:0.01:5;  
>> f = x.^3 - 7*(x.^2) + 40*x - 34;  
>> plot(x, f)
```



# Cell Array

- The cell array is an array in which each element is a cell. Each cell can contain an array.
- So, it is an array of different arrays.
- You can store different classes of arrays in each cell, allowing you to group data sets that are related but have different dimensions.
- You access cell arrays using the same indexing operations used with ordinary arrays, but using `{ }` not `( )`.



# Useful functions

<code>C = cell(n)</code>	Creates $n \times n$ cell array $C$ of empty matrices.
<code>C = cell(n, m)</code>	Creates $n \times m$ cell array $C$ of empty matrices.
<code>celldisp(C)</code>	Displays the contents of cell array $C$ .
<code>cellplot(C)</code>	Displays a graphical representation of the cell array $C$ .
<code>C = num2cell(A)</code>	Converts a numeric array $A$ into a cell array $C$ .
<code>iscell(C)</code>	Returns a 1 if $C$ is a cell array; otherwise, returns a 0.



# Exercise

```
>> C = cell(3)
C =
     []     []     []
     []     []     []
     []     []     []

>> D = cell(1, 3)
D =
     []     []     []

>> A(1,1) = {'Walden Pond'};
>> A(1,2) = {[1+2i 5+9i]};
>> A(2,1) = {[60, 72, 65]};
>> A(2,2) = {[55, 57, 56; 54, 56, 55; 52, 55, 53]};

>> A
A =
    'Walden Pond'    [1x2 double]
    [1x3 double]    [3x3 double]
```





# Exercise (Continue)

```
>> celldisp(A)
A{1,1} =
Walden Pond

A{2,1} =
    60    72    65

A{1,2} =
    1.0000 + 2.0000i    5.0000 + 9.0000i

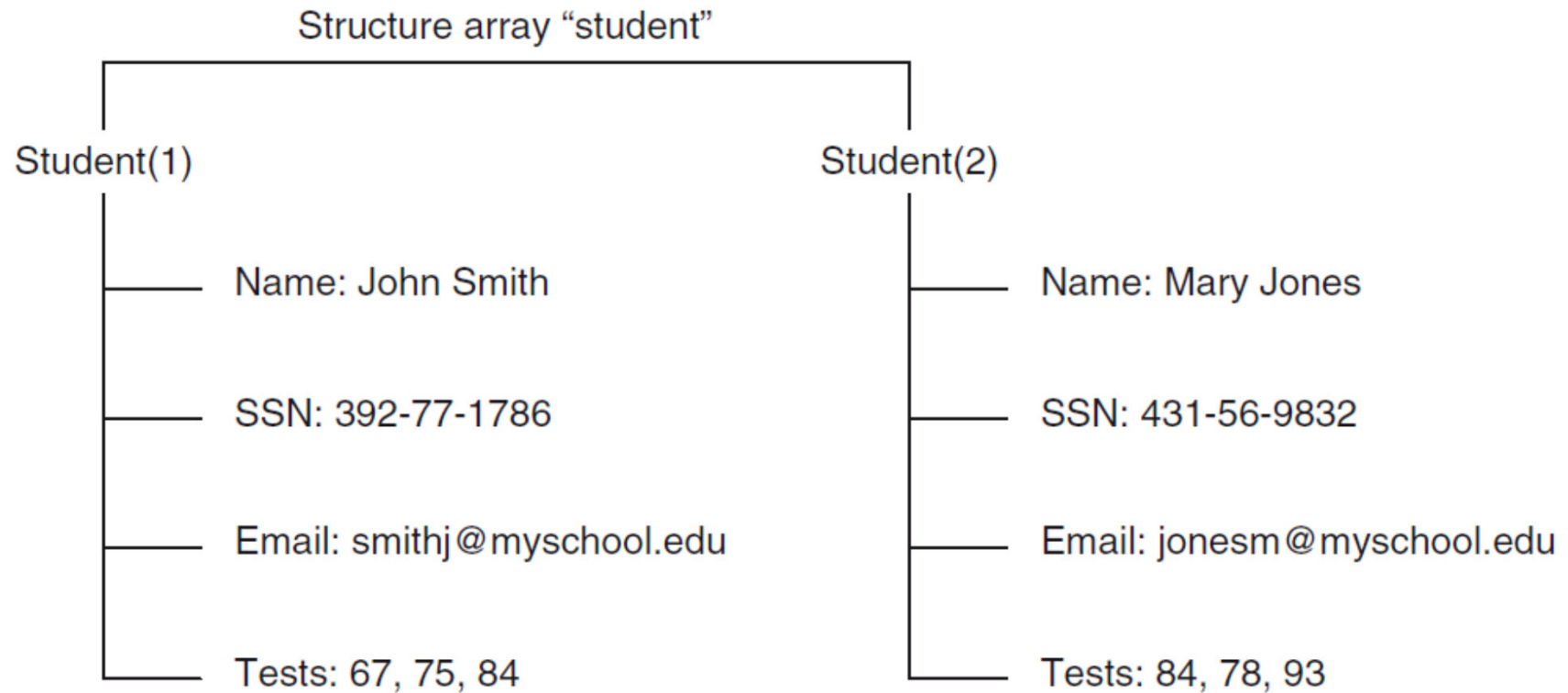
A{2,2} =
    55    57    56
    54    56    55
    52    55    53

>> B = {[2,4], [6,-9;3,5]; [7;2], 10}
B =
    [1x2 double]    [2x2 double]
    [2x1 double]    [          10]

>> B{1,2}
ans =
    6    -9
    3     5
```



# Structures (*struct.member*)



# Create and Add to Structure

```
>> student.name = 'John Smith';
>> student.SSN = '392-77-1786';
>> student.email = 'smithj@myschool.edu';
>> student.exam_scores = [67,75,84];

>> student
student =
    name: 'John Smith'
    SSN: '392-77-1786'
    email: 'smithj@myschool.edu'
    exam_scores: [67 75 84]

>> student(2).name = 'Mary Jones';
>> student(2).SSN = '431-56-9832';
>> student(2).email = 'jonesm@myschool.edu';
>> student(2).exam_scores = [84,78,93];

>> student
student =
1x2 struct array with fields:
    name
    SSN
    email
    exam_scores
```



# Investigate Structure

```
>> student(2)
ans =
    name: 'Mary Jones'
    SSN: '431-56-9832'
    email: 'jonesm@myschool.edu'
    exam_scores: [84 78 93]

>> fieldnames(student)
ans =
    'name'
    'SSN'
    'email'
    'exam_scores'

>> max(student(2).exam_scores)
ans =
    93

>> isstruct(student)
ans =
    1
```



# Script files

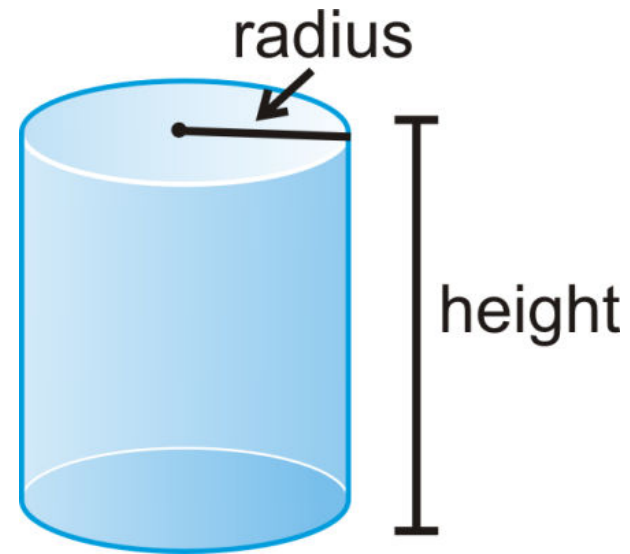
- You can save a particular sequence of MATLAB commands for reuse later in a script file (.m file)
- Each line is the same as typing a command in the command window.
- From the main menu, select File | New | Script, then save the file as `mycylinder.m`



```
File Edit Text Go Cell Tools Debug Desktop Window
: [Icons]
: [Icons]
1 - r = 5
2 - h = 13
3 - V = pi * r^2 * h
4 - A = 2 * pi * r * (r + h)
```

# Remember Example?

- Develop MATLAB code to find Cylinder volume and surface area.
- Assume radius of 5 m and height of 13 m.



$$V = \pi r^2 h$$

$$A = 2\pi r^2 + 2\pi r h = 2\pi r(r + h)$$

# Solution

```
>> r = 5
```

```
r =  
    5
```

```
>> h = 13
```

```
h =  
   13
```

```
>> V = pi * r^2 * h
```

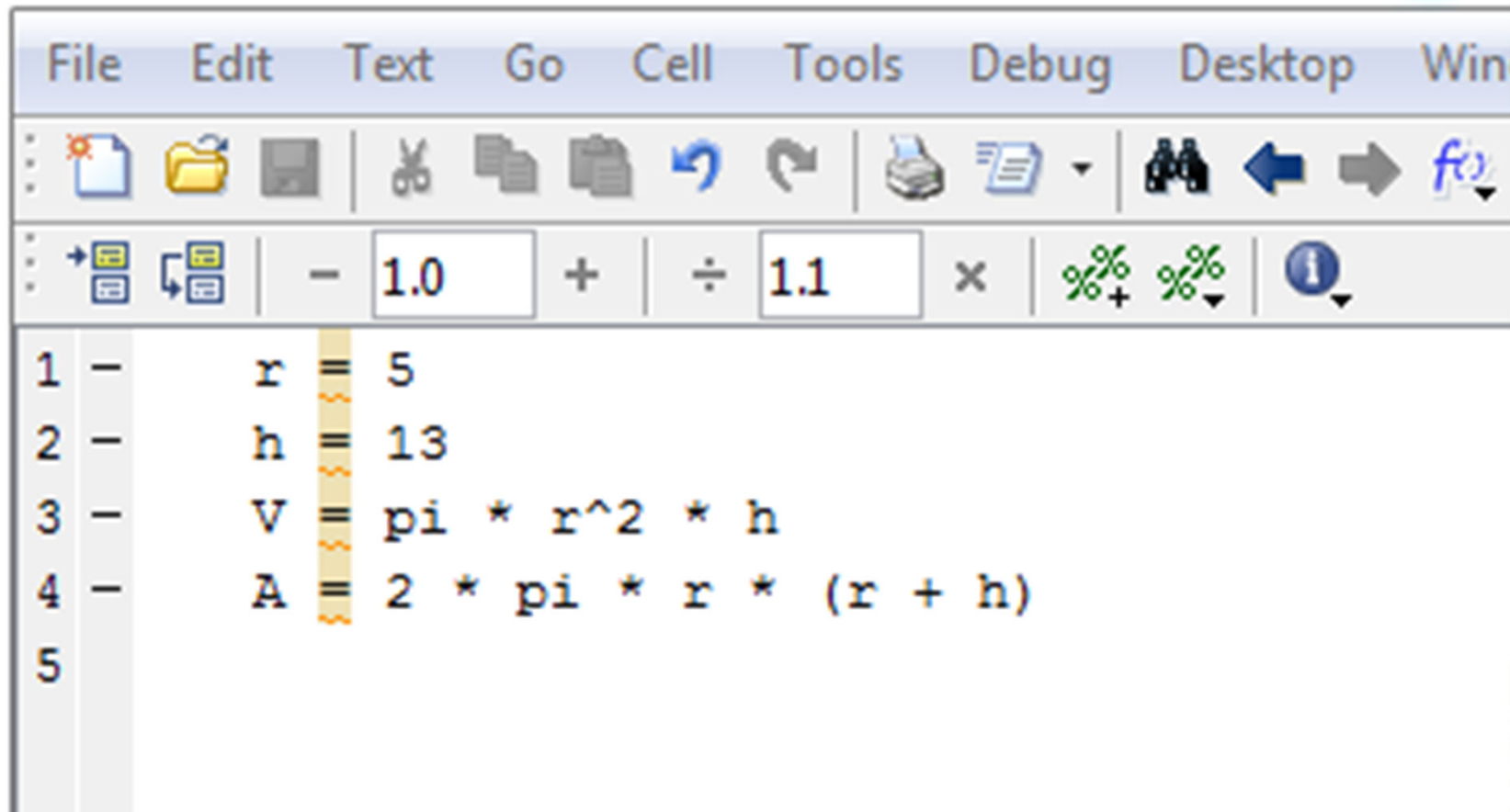
```
V =  
 1.0210e+003
```

```
>> A = 2 * pi * r * (r + h)
```

```
A =  
 565.4867
```



# Exercise



The image shows a screenshot of a code editor window. The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, Desktop, and Win. The toolbar contains icons for file operations, editing, and navigation. Below the toolbar is a calculator with a display showing 1.0 and 1.1. The main editor area contains the following code:

```
1 - r = 5
2 - h = 13
3 - V = pi * r^2 * h
4 - A = 2 * pi * r * (r + h)
5
```





# Be ware...

- Script File names **MUST** begin with a letter, and may include digits and the underscore character.
- Script File names should **NOT**:
  - include spaces
  - start with a number
  - use the same name as a variable or an existing command
- If you do any of the above you will get unusual errors when you try to run your script.
- You can check to see if a command, function or file name already exists by using the `exist` command.



# Running .m files

- Run sequence of commands by typing

`mycylinder`

in the command window

- Make sure the current folder is set properly

```
>> mycylinder
r =
    5

h =
   13

V =
 1.0210e+003

A =
 565.4867
```

# When you type `mycylinder`

When multiple commands have the same name in the current scope (scope includes current file, optional private subfolder, current folder, and the MATLAB path), MATLAB uses this precedence order:

1. **Variables** in current workspace: Hence, if you create a variable with the same name as a function, MATLAB cannot run that function until you clear the variable from memory.
2. **Nested functions** within current function
3. **Local functions** within current file
4. **Functions** in current folder
5. **Functions** elsewhere on the path, in order of appearance

Precedence of functions within the same folder depends on file type:

1. MATLAB **built-in** functions have precedence
2. Then **Simulink** models
3. Then program files with **.m extension**



# Comments in MATLAB

- Comment lines start with a % not //
- Comments are not executed by MATLAB; it is there for people reading the code.
- Helps people understand what the code is doing and why!
- Comments are VERY IMPORTANT.
- Comment anything that is not easy to understand.
- Good commenting is a huge help when maintaining/fixing/extending code.
- Header comments show up when typing the help command.



# Bad vs. Good Comments/Code

```
% set x to zero  
x = 0  
% calculate y  
y = x * 9/5 + 32
```

```
% Convert freezing point of  
% water from celsius to  
% farenheit  
c = 0  
f = c * 9/5 + 32
```



# Exercise

Editor - D:\EE 201 Computer Applications\Book Chapters\Lecture3 Arrays and Script Files\tem

```
File Edit Text Go Cell Tools Debug Desktop Window Help
[Icons: New, Open, Save, Cut, Copy, Paste, Undo, Redo, Print, Run, Stop, Refresh, etc.]
[Calculator: 1.0, 1.1, %, etc.]
1      % temperature.m Convert the boiling point for
2      % water from degrees Celsius (C) to Farenheit (F)
3      % Author: Dr. Mohammed Hawa
4
5      % Convert freezing point of water
6 -    C = 100
7 -    F = C * 9/5 + 32
```



# Header comments

```
>> help temperature
temperature.m Convert the boiling point for
water from degrees Celsius (C) to Farenheit (F)
Author: Dr. Mohammed Hawa
```

```
>> temperature
```

```
C =
    100
```

```
F =
    212
```



# Simple User Interaction: I/O

- Use `input` command to get input from the user and store it in a variable:

```
h = input('Enter the height:')
```

- MATLAB will display the message enclosed in quotes, wait for input and then store the entered value in the variable





# Simple User Interaction: I/O

- Use `disp` command to show something to a user

```
disp('The area of the cylinder is: ')  
disp(A)
```

- MATLAB will display any message enclosed in quotes and then the value of the variable.



# Exercise

```
r = input('Enter the radius:');  
h = input('Enter the height:');  
  
V = pi * r^2 * h;  
A = 2 * pi * r * (r + h);  
  
disp('The volume of the cylinder is: ');  
disp(V);  
disp('The area of the cylinder is: ');  
disp(A);
```

```
>> mycylinder  
Enter the radius:5  
Enter the height:13  
The volume of the cylinder is:  
    1.0210e+003  
  
The area of the cylinder is:  
    565.4867
```



# Summary

```
disp(A)
```

**Displays the contents, but not the name, of the array *A*.**

```
disp('text')
```

**Displays the text string enclosed within quotes.**

```
x = input('text')
```

**Displays the text in quotes, waits for user input from the keyboard, and stores the value in *x*.**

```
x = input('text', 's')
```

**Displays the text in quotes, waits for user input from the keyboard, and stores the input as a string in *x*.**



# Homework

- The speed  $v$  of a falling object dropped with zero initial velocity is given as a function of time  $t$  by  $v = gt$ , where  $g$  is the gravitational acceleration.
- Plot  $v$  as a function of  $t$  for  $0 \ll t \ll t_f$ , where  $t_f$  is the final time entered by the user.
- Use a script file with proper comments.



# Solution

```
% Plot speed of a falling object
% Author: Dr. Mohammed Hawa

g = 9.81; % Acceleration in SI units

tf = input('Enter final time in seconds:');

t = [0:tf/500:tf]; % array of 501 time instants
v = g*t; % speed

plot(t,v);
xlabel('t (seconds)');
ylabel('v m/s');
```



# Homework

- Solve as many problems from Chapter 2 as you can
- Suggested problems:
- 2.33, 2.34, 2.35, 2.36, 2.39, 2.41, 2.45, 2.48

