

FROM WEEK 1 → WEEK 2





- Last time: MATLAB UI, variables, scripts, **basic plotting, debugging.**
- Warm-up (predict → run → reflect): quick scalar cone recap.

```
% Warm-up: scalar cone recap (no vectors yet)
r = 2; h = 5;
v = (pi * r^2 * h) / 3;
disp(v)
```




Question: “What if we wanted this for many r/h values?” We’ll answer that after we learn arrays & the dot operators.

WEEK 2: MATLAB FUNDAMENTALS

OBJECTIVES:

- Understand variables and the workspace 
- Build with arrays, vectors, and matrices 
- Use operators, expressions, precedence 
- Plot results clearly 

TOPICS

1. Variables & workspace
2. Numbers & text (strings), logicals
3. Arrays, vectors, matrices
4. **Operator precedence** (PEMDAS)
5. Array **access** → **slices** → **reassignment**
6. Vectorization & the **dot** operator ( ,  , )
7. Basic plotting
8. Example: Vertical motion under gravity

VARIABLES & THE WORKSPACE

- **Variables:** named containers for values (numbers/text).
- **Case-sensitive:** `var`, `Var`, `VAR` are different.

Convention vs Rule: `ALL_CAPS` suggests a constant, but MATLAB *does not enforce* constancy.

```
x = 5; y = 10; z = x * y;  
GRAVITY = 9.81;      % convention (not enforced)  
fprintf('z = %d\n', z)  
GRAVITY = GRAVITY + 1; % allowed – not a true constant
```

NAMING RULES & CONVENTIONS

- ****Must**** start with a letter; then letters, digits, underscores.
- No spaces or special characters
- avoid MATLAB keywords (e.g., `clear`).
- Conventions: `camelCase`, `snake_case`, `ALL_CAPS` for constants.

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

1mile

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

✗ 1mile

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

clear

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

! clear

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

MyVariable

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

✓ MyVariable

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

x

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

! x

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

DEBUG_ENABLED

QUICK ACTIVITY: IS THIS A GOOD NAME?

✗ invalid ! avoid ✓ valid

✓ DEBUG_ENABLED

NUMBERS & DATA TYPES

- **double** (default numeric, ~15 digits)
- **string** (modern text): `msg = "Hello"`
- **logical**: `true` / `false` (0 is false, anything else is true)

MATLAB also has legacy **char arrays** using single quotes (`'hello'`). We'll prefer **strings** (`"hello"`) in this course.

NUMBERS AND DATA TYPES IN MATLAB

DOUBLE (DEFAULT NUMERIC)

- ~15 digits of precision; default for numbers.
- Good for most engineering calculations.

```
a = 3.14;
```

STRING (MODERN TEXT)

- Preferred for text in current MATLAB.

```
msg = "Hello, MATLAB!";
```

LOGICAL (BOOLEANS)

- Represents true (**1**) or false (**0**).
- Used in conditional statements, if/else logic, etc.

```
flag = true;
```

LOGICAL (BOOLEANS)

- Represents true (**1**) or false (**0**).
- Used in conditional statements, if/else logic, etc.

```
flag = true;
```

0 is **false**, anything else is **true**.

COMPLEX NUMBERS (CAPABILITY)

- MATLAB supports complex math (e.g., $3 + 4i$).
- We will not use complex numbers in this course.

WORKSPACE: WHAT EXISTS RIGHT NOW?

```
who      % names
whos     % names, size, class
clear x
clear    % everything
```

UNDERSTANDING SHAPE/SIZE

Now that we know what variables exist, how can we get more meta?

```
size(A)      % [rows cols]
length(v)    % longest dimension
numel(A)     % total elements
```

ARRAYS, VECTORS, MATRICES

Vector: 1D (row or column). **Matrix:** 2D. Arrays generalize both.

CREATE

```
vec1 = [1, 2, 3, 4, 5];    % row vector
mat1 = [1, 2; 3, 4];      % 2x2 matrix
vec2 = 1:2:10;            % [1,3,5,7,9]
vec3 = 1:10;              % [1,2,3,4,5,6,7,8,9,10]
vec4 = linspace(1,10,5);  % [1,3.25,5.5,7.75,10]
mat2 = zeros(3,5); mat3 = ones(2,3); % zeros/ones
```

OPERATOR PRECEDENCE (PEMDAS)

1. Parentheses
2. Exponents
3. Multiply/Divide (left→right)
4. Add/Subtract (left→right)

When unsure, add parentheses.

```
result = (3 + 5) * 2^2 / (1 + 1) - 1;    % 15
```

HIERARCHY OF OPERATIONS

EXAMPLE

```
c = 2 * 3^2 + 1/(1 + 2); % Step-by-step breakdown  
c = 2 * 9 + 1/3;  
c = 18 + 0.33333;  
c = 18.33333;
```

ARRAY ACCESS → SLICES → REASSIGNMENT

STEP 1 — SINGLE ELEMENT ACCESS

```
v = [5, 6, 7, 8, 9];  
v(3)    % => 7    (MATLAB indexing starts at 1)
```

Off-by-one: there is no index 0.

STEP 2 — SLICE (READ-ONLY)

```
v = [5, 6, 7, 8, 9];  
v(2:4) % => [6, 7, 8]
```

STEP 3 — SLICE → REASSIGN (VECTOR)

Before

`v = [5, 6, 7, 8, 9]`

Replacement *lengths must match* the slice size.

```
v(2:4) = [99, 100, 101];
```

After

`v = [5, 99, 100, 101, 9]`

STEP 4 — MATRIX ACCESS (READ-ONLY)

```
m = [1, 2; 3, 4]; % 2x2
m(2,1) % => 3 (row 2, col 1)
m(:,2) % => [2; 4] entire column 2
```

: means “all indices” along that dimension.

STEP 5 — MATRIX SLICE → REASSIGN

Before

```
m = [1, 2; 3, 4]
```

The replacement must be a 2x1 column to
fit `m(:, 2)`.

```
m(:, 2) = [9; 9];
```

After

```
m = [1, 9; 3, 9]
```

STEP 6 — DIMENSION MISMATCH (GUARANTEED)

```
A = [1, 2, 3, 4];  
B = [9, 9, 9];      % <-- different length  
A .* B              % error: dimensions must agree
```

Newer MATLAB supports *implicit expansion* in some cases, but mismatched lengths like 1x4 vs 1x3 will still error. Check `size(A)` and `size(B)`.

VECTORIZATION & THE DOT

```
A = [1, 2, 3]; B = [4, 5, 6];  
A .* B      % [4, 10, 18] (element-wise)  
A .^ 2      % [1, 4, 9]
```

No dot = linear algebra multiply:

```
[1,2,3] * [4,5,6]
```

is invalid (1x3 * 1x3). You'd need $[1, 2, 3]$
 $* [4; 5; 6]$.

BACK TO THE WARM-UP: VECTORIZED CONE

```
% Warm-up: scalar cone recap (no vectors yet)
r = [2, 3, 4]; h = [5, 6, 7];
v = (pi * r.^2 .* h) / 3;
disp(v) % [20.9440  56.5487  117.2861]
```

Why only one dot on `.^`? Because `r` and `h` are vectors, so every multiply with them should be element-wise: `.*`, `.^`. The final `/ 3` is scalar.

INTERACTIVE EXERCISE:

1. Create vectors **A** and **B** of the same length.
2. Perform **element-wise multiplication** and exponentiation.
3. Observe the differences if you omit the dot.

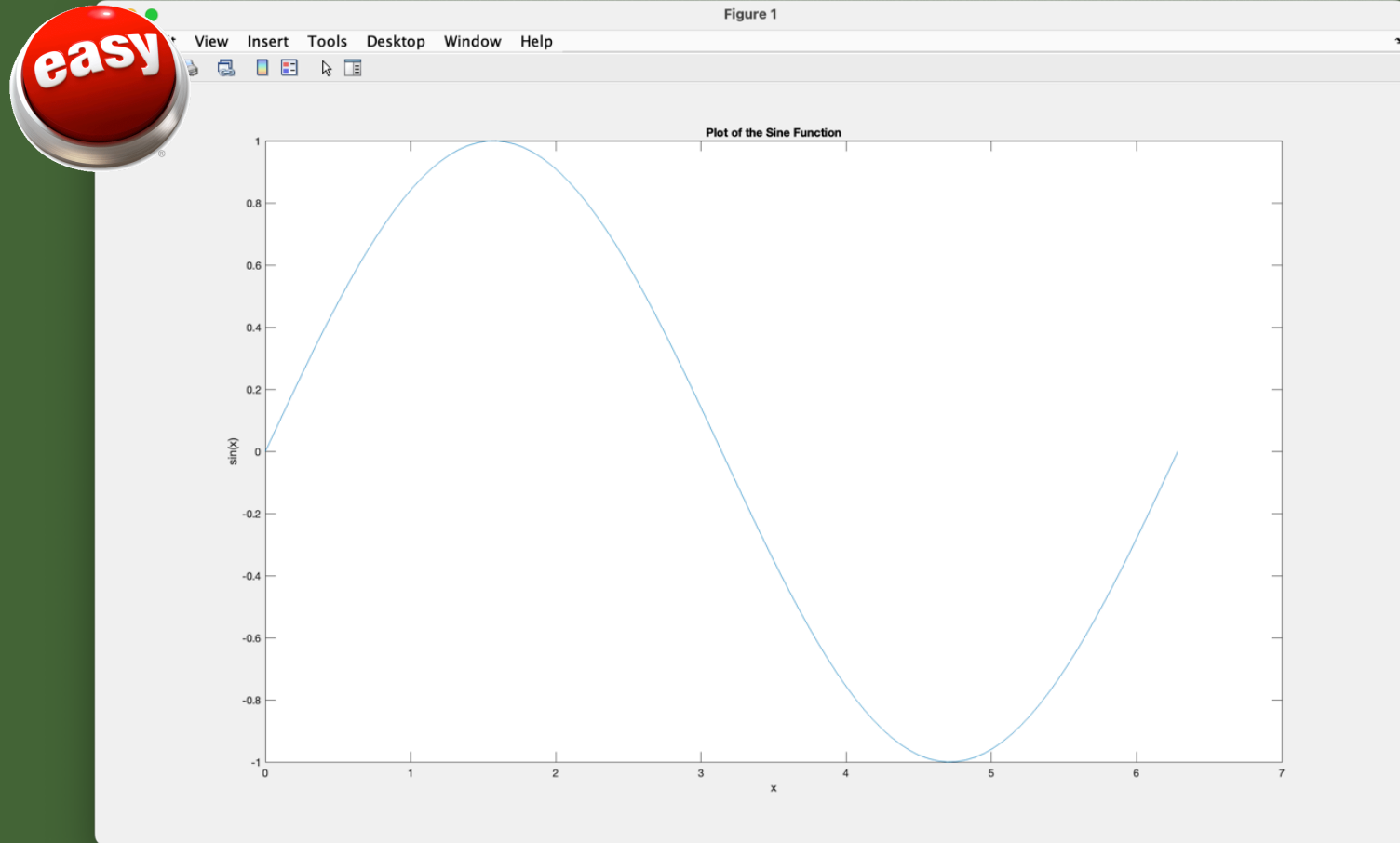
ARITHMETIC (RECAP)

```
a = 5; b = 2;  
[a+b, a-b, a*b, a/b, a^b] % [7, 3, 10, 2.5, 25]
```

BASIC PLOTTING (REVIEW)

```
x = linspace(0, 2*pi, 100);  
y = sin(x);  
  
plot(x, y, '-o', 'LineWidth', 1.5);  
xlabel('x');  
ylabel('sin(x)');  
title('Sine');  
legend('sin(x)');  
grid on
```

BASIC EXAMPLE



VERTICAL MOTION UNDER GRAVITY EXAMPLE

THE PROBLEM:

Calculate vertical motion of an object under gravity — how things fall.

APPROACH:

1. Inputs - What data do you need to solve this problem

```
GRAVITY = 9.81;    % (m/s^2)
time = 0:0.1:10;   % 0 to 10s
v0 = 50;           % initial velocity (m/s)
```

2. Manipulation - Perform operations to get to your destination

```
y = v0 * time - 0.5 * GRAVITY * time.^2;
```

3. Output - Produce clear and concise output Let's see what the raw output looks like

```
time
y
```

Raw output can be messy. Let's plot for clarity:

```
plot(time, y);  
xlabel('Time (s)');  
ylabel('Height (m)');  
title('Vertical Motion Under Gravity');  
grid on;
```



GOTCHAS

- Variables are case sensitive.
- Remember the difference between `*` `/` `^` and `.*` `./` `.^`
- 0 is false but anything \neq 0 is true.
- MATLAB indexing starts at 1.

PREVIEW: COMING WEEKS

- Logical vectors (1s and 0s)
- Character arrays (legacy) vs strings (modern)
- `struct` for grouped fields
- Cell arrays for mixed types

KEY TAKEAWAYS

- Arrays are the building blocks of MATLAB.
- Think of arrays in terms of: access → slice → reassign.
- Dots (`.*`, `.^`, `./`) are for element-wise operations.
- Use parentheses to clarify precedence.
- Plot early to sanity-check results.

WHY THIS MATTERS

- Cleaner mental model → fewer debugging surprises.
- Vectorization is both speed and clarity.
- These skills carry to data analysis, simulation, controls.

SOFTWARE ENGINEERING