While you are waiting

MICHIGAN ENGINEERING
UNIVERSITY of MICHIGAN ■ COLLEGE of ENGINEERING

ANAGRAM

Re-arrange the letters to reveal the famous person described...

old west action

ENGR 101 – Lecture 4

Functions and Vectorization

Laura Alford, James Juett, Rick Niciejewski 9/19/17

Announcements

- LAB #2 – Current lab. Lab #2 assignments are due the day after you take the lab !!!
- WEEKLY REVIEW #3 – will be available on Thursday. Please complete, due this coming Sunday

- Project #1 – published and due Thursday, 28-SEPT-2017

- An overview of Project #1 shall be given later this lecture. It and today's lecture are stored as .pdfs on GoogleDrive, Public area, 00_Todays_Lecture

Lecture Goals

- Previous lecture: Vectors and Matrices
  - Working with *Lobster*
  - Vectors and Indexing
  - Matrices and 2-types of matrix indexing
  - Suggested readings, Attaway, Chap 2.1 and 2.3

- Today's lecture: Functions and Vectorization
  - Reducing code duplication
  - Functions
  - Vectorization
  - Project 1 Overview
  - Suggested readings, Attaway, Chap 3.7 and 6.1-6.2

Hypothetical Situation

- Let's say you have many different samples from the Proxima b probe and need to perform the ESP calculation from project 0 several times…

Sample	Na	K	Ca	Mg
1	10.9	68.2	25.4	13.8
2	13.7	66.3	26.4	13.2
3	14.3	67.0	26.7	13.0
4	14.1	72.2	25.5	17.3
5	12.3	72.3	26.8	13.1
6	12.6	67.9	26.5	17.7
7	14.1	71.5	26.9	13.0
8	12.0	72.1	26.7	15.6
9	14.5	71.4	25.7	15.0
10	12.1	73.5	25.4	13.2

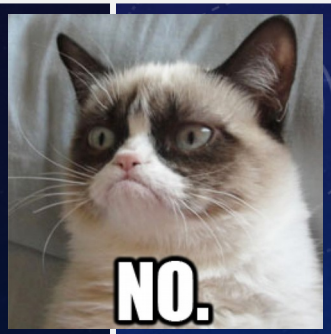
```
Na = 10.9; K = 68.2;
Ca = 25.4; Mg = 13.8;
display(Na ./ (K + Ca + Mg + Ca));

Na = 13.7; K = 66.3;
Ca = 26.4; Mg = 13.2;
display(Na ./  K + Ca + Mg + Ca);

Na = 14.3; k = 67.0;
Ca = 26.7; Mg = 13.0;
display(Na ./ (K + Ca + Mg + Ca))

Na = 14.1; K = 72.2;
Ca = 25.5; Mg = 17.3;
...
```

Is this a good approach?



Code Duplication is Bad

- Code duplication occurs when you have more than one copy of code that does "the same thing".
  - e.g. More than one copy of the formula for ESP
  - e.g. Assignments to the Na, K, Ca, Mg variables over and over

- Why is it bad?
  - Each new copy introduces more potential for mistakes.
  - It makes code hard to maintain:
    - You have to track down ALL copies if you make a change or find a bug.
  - Your code becomes cluttered and harder to understand.

Reducing Code Duplication

- Today we'll look at two important techniques used in MATLAB for reducing code duplication[1].

  - **Creating New Functions**

    Example: Instead of writing out the ESP formula each time, we create our own ESP function to use just as we would `sqrt`, `sin`, etc.

  - **Vectorization**

    Example: Instead of repeating the computation for each different sample from the probe, we put all the samples into vectors and then perform the computation on the vectors.

1 Among other things – these techniques have many benefits.

What is a Function?

- A function is an algorithm that returns data to the caller, and operates independently of the caller.
  - i.e. Data goes in, it gets processed, new data comes out.
  - It's an **algorithm** because we can use it without having to worry about the details (code) of how the computation works internally.
- Example: The `sqrt` function

```
x = 16;
y = sqrt(x);
```

- The **interface** for a function describes how we use it:
  - e.g. For `sqrt`: "Give it a number. It gives you back the square root.
  - e.g. For `size`: "Give it an array. It gives you back its dimensions.

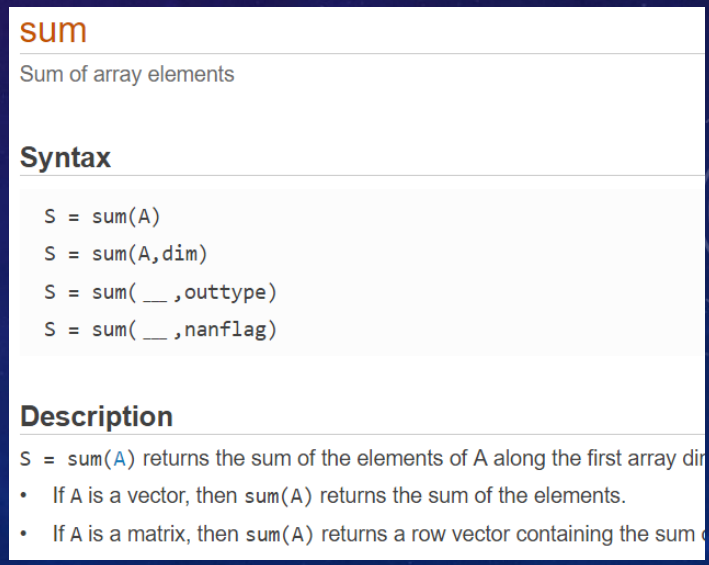
Functions for Use with Matrices

- MATLAB has many built-in functions for working with matrices.
  - We'll only see a few today. If you imagine a function that might be useful, do a quick search online to see if it already exists!
- If you need to look at the documentation for a function, use the help command in MATLAB or search for it online.

```
>> help sum
 sum Sum of elements.
    S = sum(X) is the sum of the elem
    S is a row vector with the sum ov
    sum(X) operates along the first n

    S = sum(X,DIM) sums along the dim

    S = sum(..., TYPE) specifies the
    sum is performed, and the type of
```



sum

Sum of array elements

Syntax

```
S = sum(A)
S = sum(A,dim)
S = sum( __ ,outtype)
S = sum( __ ,nanflag)
```

Description

S = sum(A) returns the sum of the elements of A along the first array dir

- If A is a vector, then sum(A) returns the sum of the elements.
- If A is a matrix, then sum(A) returns a row vector containing the sum

The sum Function

- Many functions work column-by-column. For example, the sum function yields sums of each column in a matrix.
- A row vector is a special case. We get the sum of that row.

3	7	6	8
6	2	4	1
4	2	5	3

X

13	11	15	12

sum(X)

2	3	2	1

y

8

sum(y)

3	7	6	8
6	2	4	1
4	2	5	3

X

51

sum(sum(X)), or *sum(X(:))*

Functions for Manipulating Matrices

- MATLAB includes many functions for modifying the layout and shape of data within matrices.
- Here are a few, for your reference:
  - `rot90` – rotate the data in a matrix by 90 degrees counterclockwise
  - `fliplr` – flip the matrix "left to right" (i.e. horizontal flip)
  - `flipud` – flip the matrix "up to down" (i.e. vertical flip)
  - `reshape` – keep the same data, but pack it into a different number of rows/columns to "reshape" the matrix
  - `repmat` – create a larger matrix by replicating (i.e. "tiling") this one
- See the MATLAB documentation online for more details!

In MATLAB, there are many built-in functions !!!
But what if the one you need does not exist ???

You can make your own *user-defined* function

Creating a New Function: syntax

- Use the MATLAB editor to create a new "function".m file.

```
function [ return variables ] = name( parameters )
  statement;
  statement;
  ...
end
```

- Every function has two major pieces:
  - An **interface** that specifies how it must be used by other code.
    i.e. What does this thing do?
  - An **implementation** with code that makes it work behind the scenes.
    i.e. How does this thing work, internally?

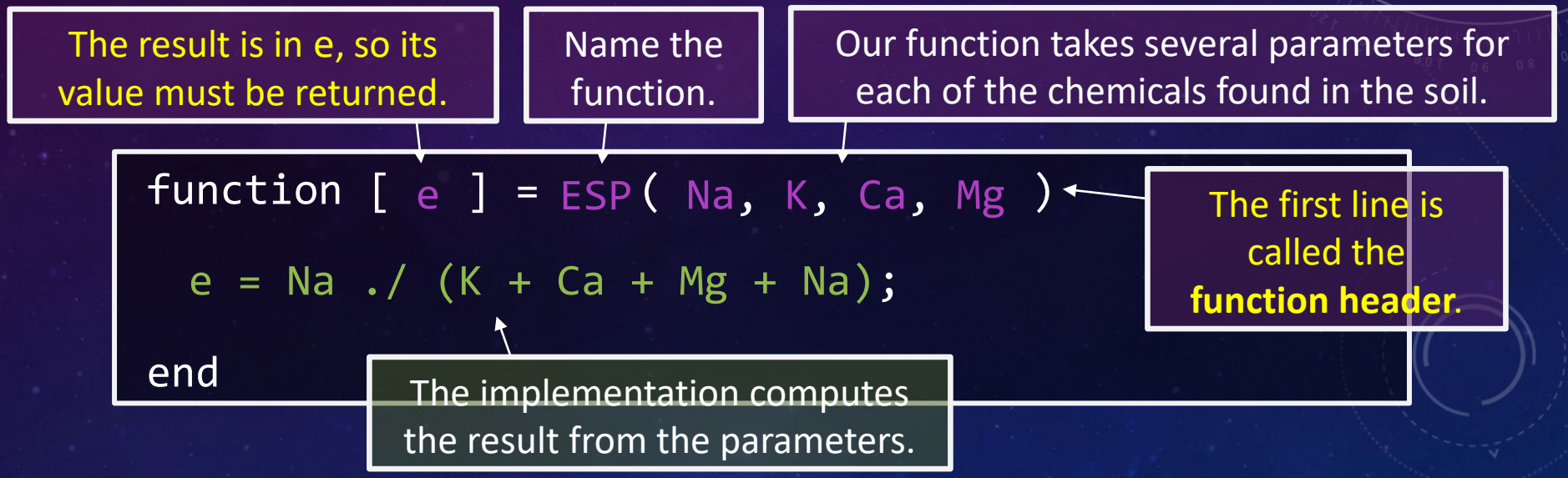
Function Interfaces

```
function [ return variables ] = name( parameters )
  statement;
  statement;
  ...
end
```

- The interface defines how we interact with the function.
  - *name* – What is the function called? (name.m)
  - *parameters* – Variables used to pass data into the function as input.
  - *return variables* – A list of variables whose values will be returned back to the outside world as output from the function.
- The implementation contains code to compute the return values.

A Function to Calculate ESP

- Let's write a function that calculates the Exchangeable Sodium Percentage (ESP) from project 0. Here's the interface:



- Now, in the implementation of the function, we write code that uses the parameters Na, K, Ca, and Mg to calculate e.

Using the ESP Function

- Now, in the "command window" **call** the ESP function.

Sample	Na	K	Ca	Mg
1	10.9	68.2	25.4	13.8
2	13.7	66.3	26.4	13.2
3	14.3	67.0	26.7	13.0
4	14.1	72.2	25.5	17.3
5	12.3	72.3	26.8	13.1
6	12.6	67.9	26.5	17.7
7	14.1	71.5	26.9	13.0
8	12.0	72.1	26.7	15.6
9	14.5	71.4	25.7	15.0
10	12.1	73.5	25.4	13.2

```
Na = 10.9; K = 68.2;
Ca = 25.4; Mg = 13.8;
display(ESP(Na, K, Ca, Mg));

Na = 13.7; K = 66.3;
Ca = 26.4; Mg = 13.2;
display(ESP(Na, K, Ca, Mg));

Na = 14.3; k = 67.0;
Ca = 26.7; Mg = 13.0;
display(ESP(Na, K, Ca, Mg));

Na = 14.1; K = 72.2;
Ca = 25.5; Mg = 17.3;
...
```

This is better, but still not great. Stay tuned for more changes…

Function Calls and Expressions

- A function call takes in **arguments** that correspond to the parameters of the function.
- The arguments could be expressions. They do not have to be variables!

```
x = 3; y = 2; z = 4;
mat = [9,2;8,7];

a = ESP( 12 + x, sin(1.4), 4 .* x, mat(1,2) );

b = ESP(x, y, z, y) + ESP(z, z, z, z);
```

These are both nonsense, but perfectly legal in MATLAB.

Parameter Passing via the "Command Window"

- The **values** of the **arguments** to the function call are used for the **parameter** variables in the function definition.
- The **function call** evaluates to the returned value.

```
>> Na = 10.9; K = 68.2;
>> Ca = 25.4; Mg = 13.8;
>> result = ESP(Na, K, Ca, Mg);
```

The ESP function is a file in your MATLAB area

```
>> Na = 10.9; K = 68.2;
>> Ca = 25.4; Mg = 13.8;
>> result = ESP(Na, K, Ca, Mg);
```

```
function [ e ] = ESP( Na, K, C, Mg )

  e = Na ./ (K + C + Mg + Na)

end
```

Here, the "outside world" or COMMAND WINDOW, interacts via the **interface**. It does not know about the **implementation**.

Variable Scope

- Variables in a function are **completely independent** from those in the base workspace (goto COMMAND WINDOW).
- Even if they have the same name! (e.g. Na, K, Ca, Mg)

```
>> Na = 10.9; K = 68.2;
>> Ca = 25.4; Mg = 13.8;
>> result = ESP(Na, K, Ca, Mg);
```

```
function [ e ] = ESP( Na, K, C, Mg )

  e = Na ./ (K + C + Mg + Na)

end
```

Global Scope (Workspace)

```
Na, K, Ca, Mg, result
```

ESP Local Scope

```
Na, K, C, Mg, e
```

What's in a name?

- We could change the names of either the parameters or returns and the code would still run just the same.
- The ordering of the arguments/parameters is what matters.
- It's just a coincidence they often end up named similarly.

```
>> A = 10.9; B = 68.2;
>> C = 25.4; D = 13.8;
>> Monday = ESP(A, B, C, D);
```

```
function [ e ] = ESP( Na, K, C, Mg )

  e = Na ./ (K + C + Mg + Na)

end
```

Global Scope (Workspace)

```
A, B, C, D, Monday
```

ESP Local Scope

```
Na, K, C, Mg, e
```

Your turn: work with partner(s) having a laptop

- 1) CREATE: in the MATLAB editor, *write and save* a general function that calculates the hypotenuse of any right-angled triangle
  - Input parameters: the two shorter sides of the triangle
  - Return variable: the length of the hypotenuse

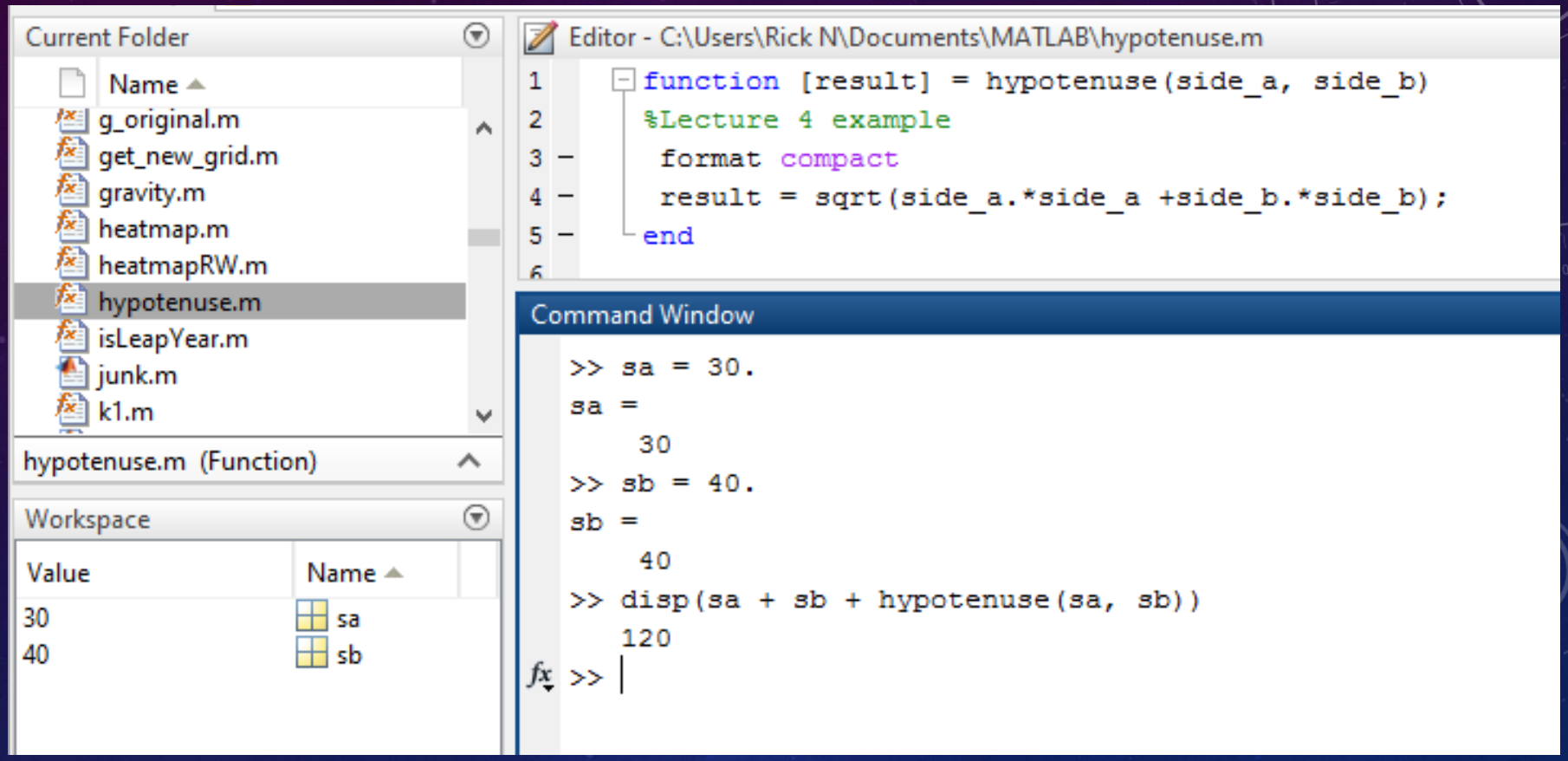
  function [result] = hypotenuse(side_a, side_b)
- 2) TEST: in the MATLAB command window, call hypotenuse with

  side_a = 3, **side_b = 4**; return should equal 5
- 3) EXAM STYLE: while in the command window, use **hypotenuse** and find the perimeter of a triangle with **side_a=30** and **side_b=40** .

  A) 50 B) 2500

  C) 120 D) 70

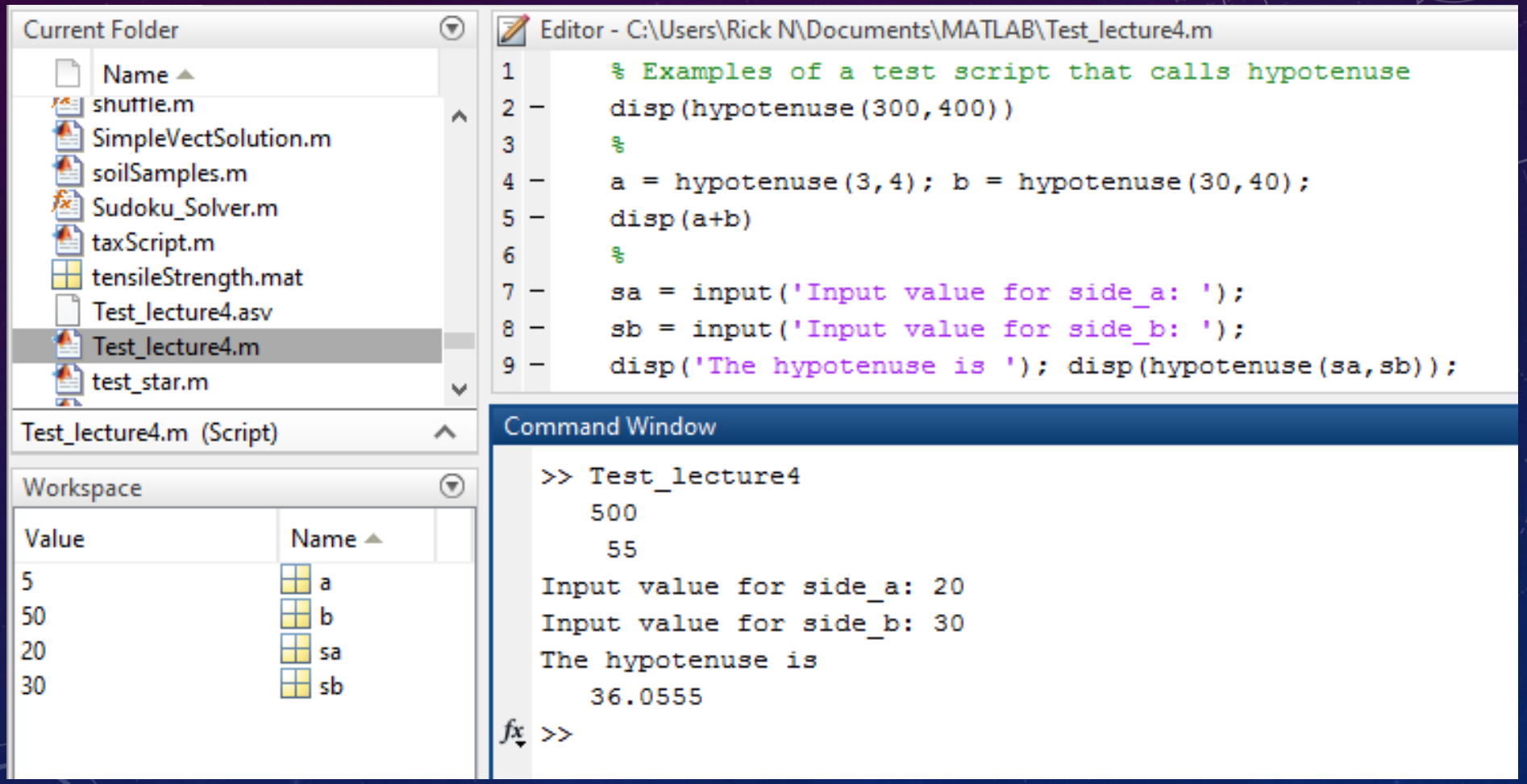
Your turn: work with partner(s) having a laptop



A) 50 B) 2500

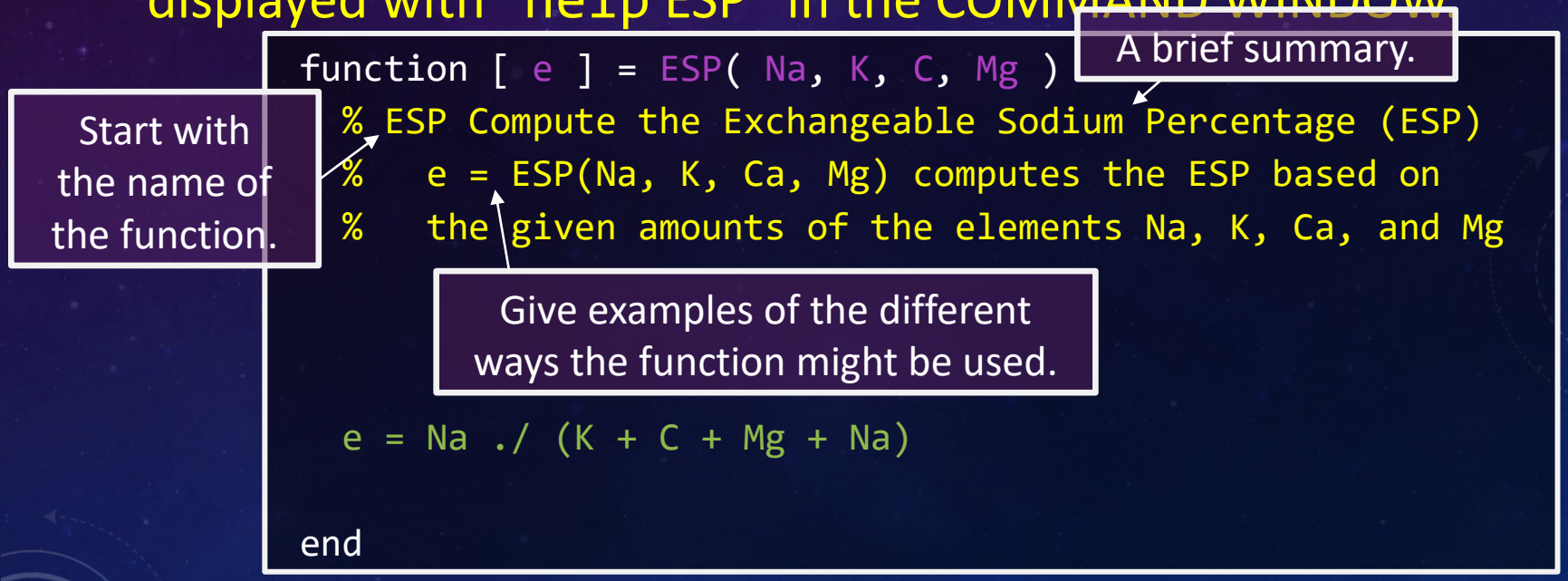
C) 120 D) 70

BONUS: an *interactive* Test Script



Guidelines for Writing Functions

- Use meaningful names for the parameters and return variables.
- Indent the body of the function using spaces or tabs.
- Write a comment at the top of the function body. This will be displayed with "`help ESP`" in the COMMAND WINDOW

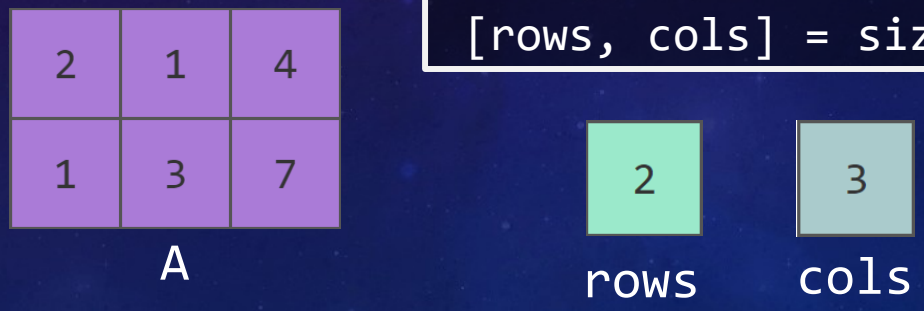


Capturing Multiple Return Variables

- Consider the interface[1] for the built-in `size` function:

```
function [ m, n ] = size( X )
  % size Returns the dimensions of an array.
  % implementation not shown
end
```

- To capture the multiple return variables, use MATLAB's compound assignment notation.



1 The real interface is more complex, but this is good enough for now.

The `min` and `max` Functions

- The `min` and `max` functions can be used to find the smallest or largest elements in a vector. They too, work in each column first and must be applied twice for a whole matrix.
- `min` and `max` have a compound return value. They return both the value found, and also the index where it was found.

A Function with no Parameters or Return Variables

- If you need to use a function to "do something" rather than "compute something", we don't need a return value.

```
function [ ] = fightSong( )
  % Part of the UM fight song.
  display('Hail! to the victors valiant');
  display('Hail! to the conquering heroes');
  display('Hail! Hail! To Michigan');
  display('the leaders and best');
end
```

```
% print the fight song twice
fightSong();
fightSong();
```

Using the ESP Function

- So how do we feel about this code now?

Sample	Na	K	Ca	Mg
1	10.9	68.2	25.4	13.8
2	13.7	66.3	26.4	13.2
3	14.3	67.0	26.7	13.0
4	14.1	72.2	25.5	17.3
5	12.3	72.3	26.8	13.1
6	12.6	67.9	26.5	17.7
7	14.1	71.5	26.9	13.0
8	12.0	72.1	26.7	15.6
9	14.5	71.4	25.7	15.0
10	12.1	73.5	25.4	13.2

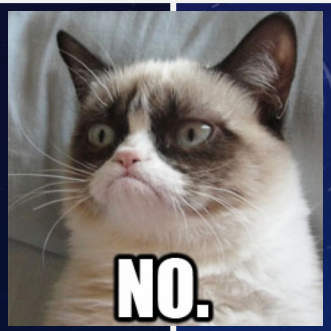
```
Na = 10.9; K = 68.2;
Ca = 25.4; Mg = 13.8;
display(ESP(Na, K, Ca, Mg));

Na = 13.7; K = 66.3;
Ca = 26.4; Mg = 13.2;
display(ESP(Na, K, Ca, Mg));

Na = 14.3; k = 67.0;
Ca = 26.7; Mg = 13.0;
display(ESP(Na, K, Ca, Mg));

Na = 14.1; K = 72.2;
Ca = 25.5; Mg = 17.3;
...
```

Is this a good approach yet?



Organizing Experimental Data in MATLAB

	Na	K	Ca	Mg
1	10.9	68.2	25.4	13.8
2	13.7	66.3	26.4	13.2
3	12.1	73.5	25.4	14.1
4	14.3	67	26.7	13
5	14.1	72.2	25.5	17.3
6	12.3	72.3	26.8	13.1
7	12.6	67.9	26.5	17.7
8	14.1	71.5	26.9	13
9	12	72.1	26.7	15.6
10	14.2	70.6	25.7	15

- Columns generally correspond to different variables in the experiment
  - e.g. One column vector for Na levels
    One column vector for Mg levels
    etc.
- Each row within these columns corresponds to a different observation of that variable
  - e.g. Across all the columns, the first element (i.e. the one in the first row) corresponds to soil sample #1, the next to sample #2, and so on…

Writing a "Vectorized" ESP Function

	Na	K	Ca	Mg
1	10.9	68.2	25.4	13.8
2	13.7	66.3	26.4	13.2
3	12.1	73.5	25.4	14.1
4	14.3	67	26.7	13
5	14.1	72.2	25.5	17.3
6	12.3	72.3	26.8	13.1
7	12.6	67.9	26.5	17.7
8	14.1	71.5	26.9	13
9	12	72.1	26.7	15.6
10	14.2	70.6	25.7	15

- In MATLAB, we say code is "**vectorized**" if it can work with vectors of data just as well as scalars.
- Today we'll just give a brief preview of vectorization – we'll keep coming back to it throughout the term.
- **Question:**
  What do we need to do to our ESP function in order that it will work now that our Na, K, Ca, and Mg arguments are vectors?

Nothing

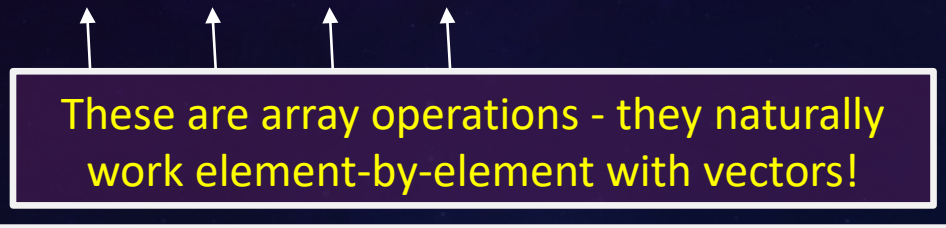
The ESP Function is Already "Vectorized"

- MATLAB makes it easy to write vectorized code.

```
function [ e ] = ESP( Na, K, C, Mg )
  % ESP Compute the Exchangeable Sodium Percentage (ESP)
  %   e = ESP(Na, K, Ca, Mg) computes the ESP based on
  %   the given amounts of the elements Na, K, Ca, and Mg

  e = Na ./ (K + C + Mg + Na)

end
```



- Don't forget the dot!
  If you did, this would work for scalars but break with vectors. ☹

Calculating ESP From Data Vectors

- Our measurements of chemicals in the soil are encoded into column vectors, which are passed into the ESP function.

```
Na = [10.9; 13.7; 14.3; 14.1; 12.3; 12.6; 14.1; 12.0; 14.5; 12.1];
K = [68.2; 66.3; 67.0; 72.2; 72.3; 67.9; 71.5; 72.1; 71.4; 73.5];
Ca = [25.4; 26.4; 25.4; 26.7; 25.5; 26.8; 26.5; 26.9; 26.7; 25.7];
Mg = [13.8; 13.2; 14.1; 13; 17.3; 13.1; 17.7; 13; 15.6; 15];

display(ESP(Na, K, Ca, Mg));
```

- A final improvement would be to read these data vectors from a file. We'll come back to this…

Break Time

We'll start again in 5 minutes.

Project 1 Introduction