ENGR 101 – Chapter 3

Functions and Data

Laura Alford, James Juett, Rick Niciejewski

9/5/2020

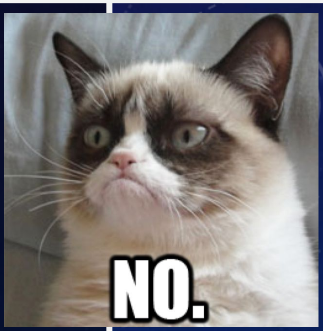
Hypothetical Situation

- Let's say you have many different samples from the Proxima b probe, taken at different sites for a potential settlement, and you need to perform the ESP calculation for each…

Site	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 = 13.8; 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:
  Multiple copies of code that do "the same thing“
  (perhaps with different data)

- 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 we would `sqrt`, `sin`, etc.

  - **Vectorization**

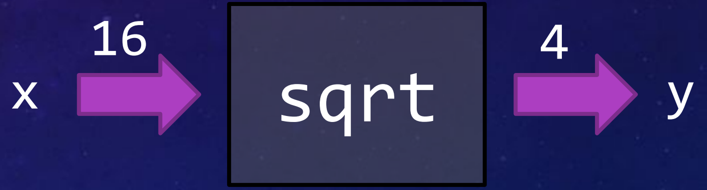
    Example: Instead of repeating the computation for each different sample from the probe, we put all the samples into vectors and then perform the computation on the vectors all at once.

[1] Among other things – these techniques have many benefits.

Recall: What is a Function?

- A **function** is an abstraction over a chunk of computation.
  - i.e. Data goes in, it gets processed, new data comes out.
  - It's an **abstraction** because we can use it without having to worry about the details of how the computation works internally.
- Example: The `sqrt` function

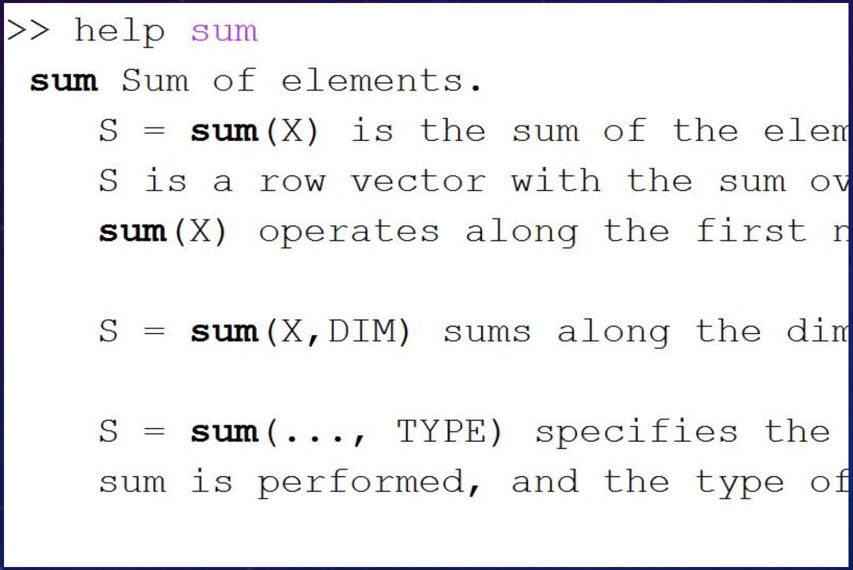
```
x = 16;
y = sqrt(x);
```



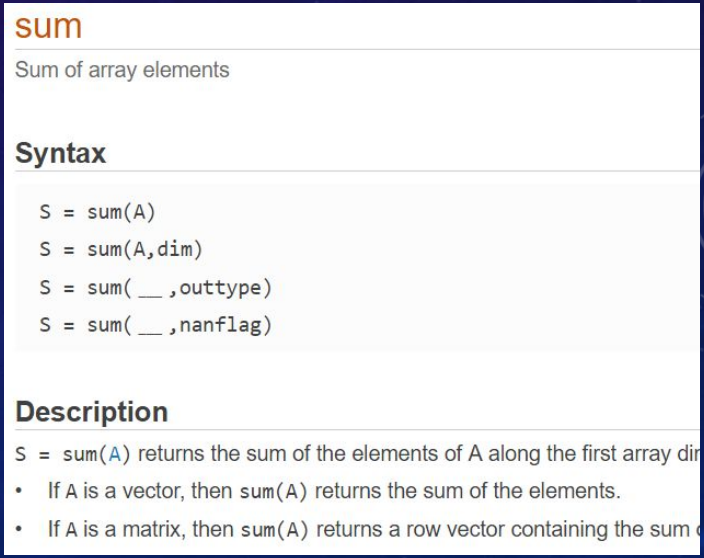
- The **interface** for a function describes how we use it:
  - e.g. For `sqrt`: "Give it a number. It gives you back the square root.
  - e.g. For `size`: "Give it an array. It gives you back its dimensions.

Reminder: Getting Help

- If you want to look at the documentation for a function:
  - Use the help command in the command window
  - Use the "Search Documentation" box in MATLAB
  - 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 c

There are tons of built-in functions,
but what if the one we want isn't there?

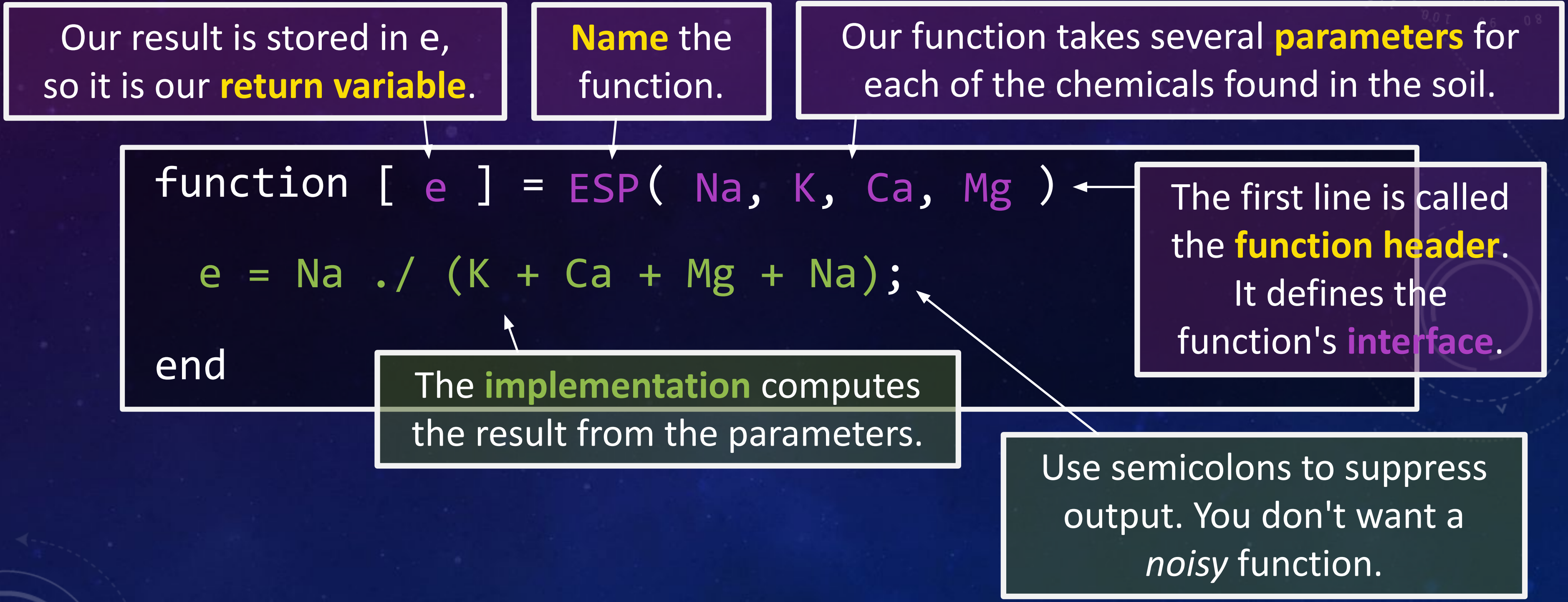
We can make our own!

MATLAB DEMO – ESP FUNCTION

Debrief: A Function to Calculate ESP

- Let's write a function that calculates the Exchangeable Sodium Percentage (ESP) from the practice project.
- This **function definition** gets saved in the file `ESP.m`

Matches function name.



Our result is stored in e, so it is our **return variable**.

Name the function.

Our function takes several **parameters** for each of the chemicals found in the soil.

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

  e = Na ./ (K + Ca + Mg + Na);

end
```

The first line is called the **function header**. It defines the function's **interface**.

The **implementation** computes the result from the parameters.

Use semicolons to suppress output. You don't want a *noisy* function.

Your custom function files need to be in your current MATLAB folder.

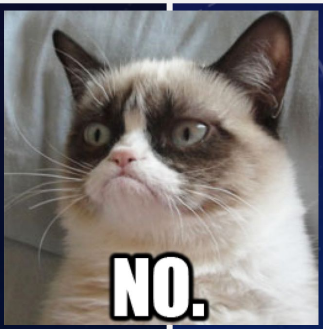
Debrief: Using the ESP Function

- We replace the formula with a **call** to the ESP function.

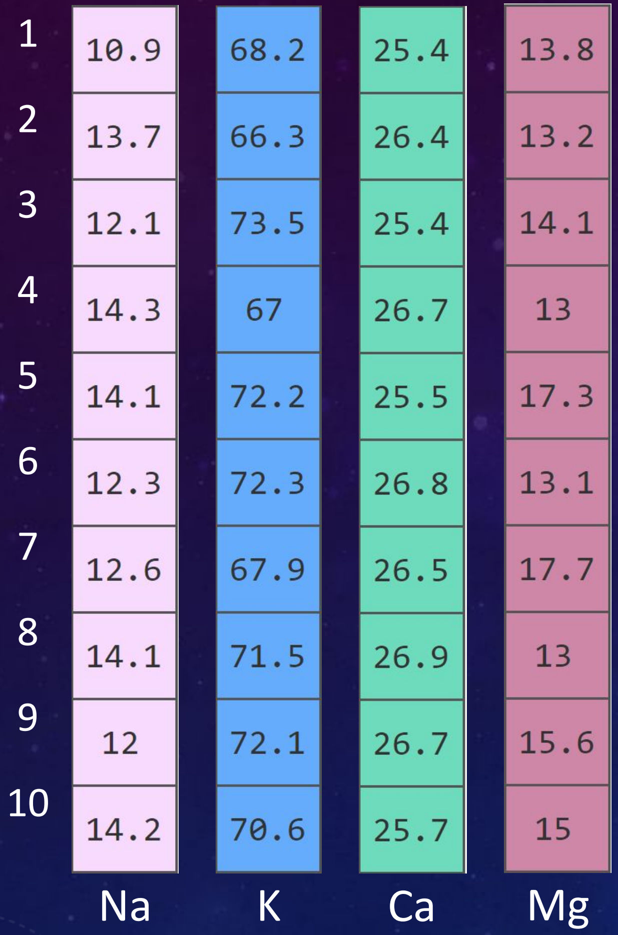
Site	Na	K	Ca	Mg
1	10.9	68.2	25.4	13.8
2	13.7	66.3	26.4	13.2
3	14.3	67.0	26.7	13.0
4	14.1	72.2	25.5	17.3
5	12.3	72.3	26.8	13.1
6	12.6	67.9	26.5	17.7
7	14.1	71.5	26.9	13.0
8	12.0	72.1	26.7	15.6
9	14.5	71.4	25.7	15.0
10	12.1	73.5	25.4	13.2

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

Na = 13.8; 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

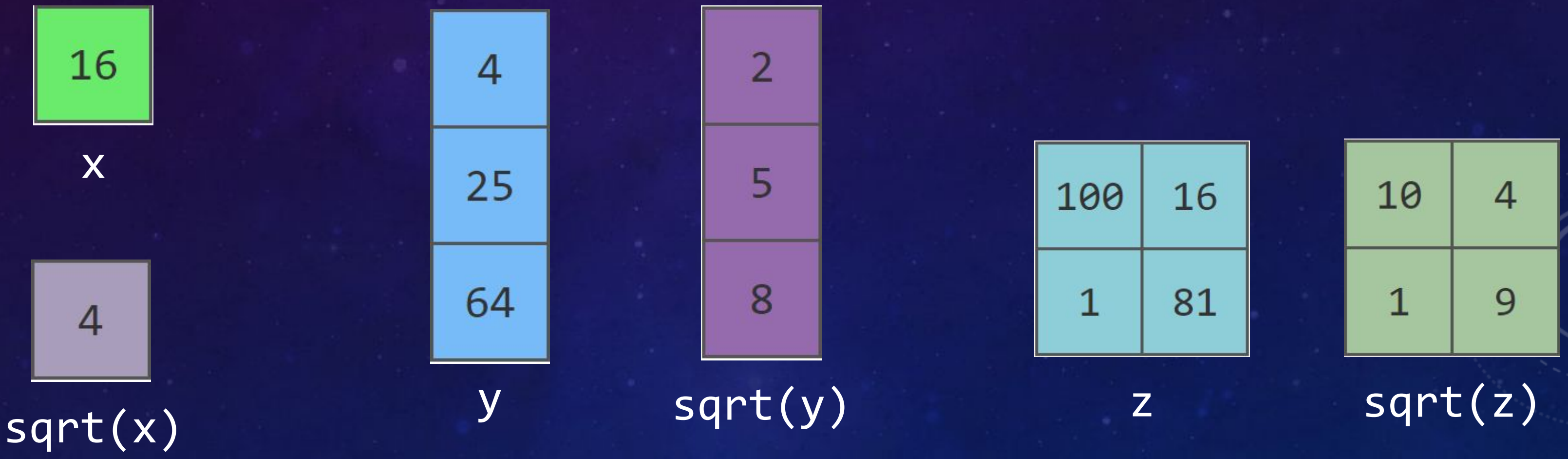


	Na	K	Ca	Mg
1	10.9	68.2	25.4	13.8
2	13.7	66.3	26.4	13.2
3	12.1	73.5	25.4	14.1
4	14.3	67	26.7	13
5	14.1	72.2	25.5	17.3
6	12.3	72.3	26.8	13.1
7	12.6	67.9	26.5	17.7
8	14.1	71.5	26.9	13
9	12	72.1	26.7	15.6
10	14.2	70.6	25.7	15

- Columns generally correspond to different variables in the experiment
- Each "row" within these columns corresponds to a different sample.
- Now, we need our ESP function to work with column vectors instead of just single values…

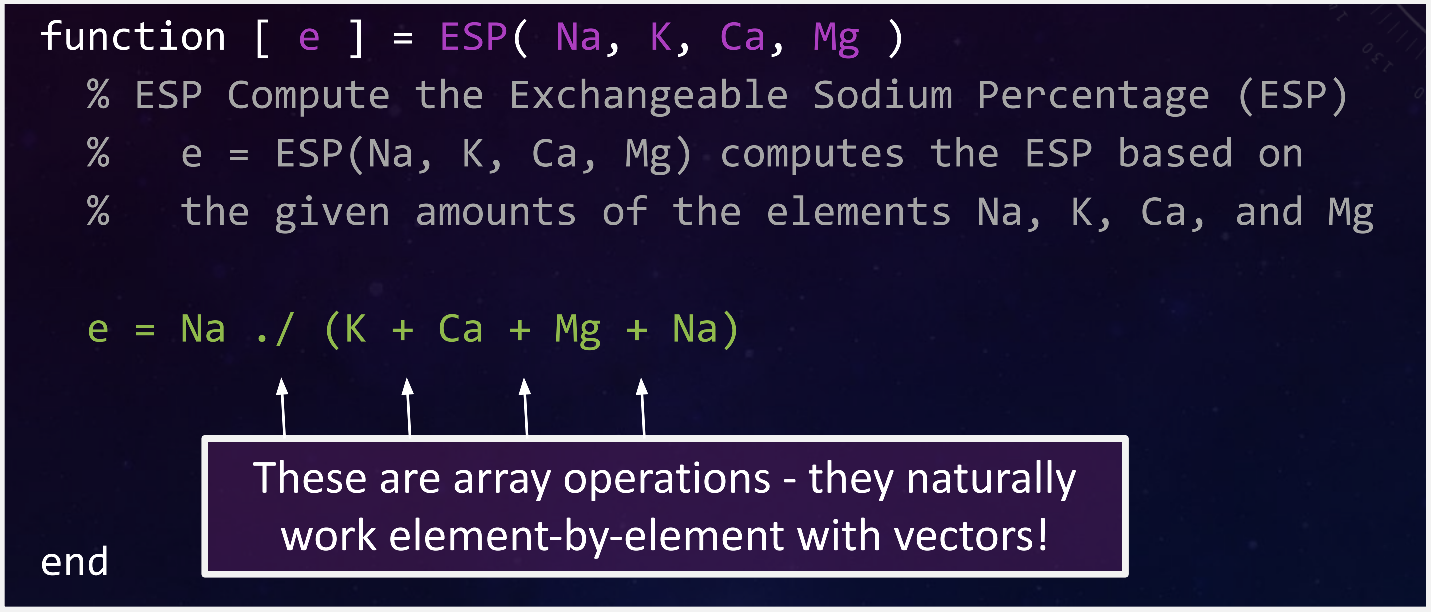
Vectorized Functions

- A vectorized function can work on vectors or matrices (in addition to just plain old scalars).
- The function's operation is applied element-by-element.



The ESP Function is Already "Vectorized"

- MATLAB makes it easy to write vectorized code.



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

end
```

These are array operations - they naturally work element-by-element with vectors!

- 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.

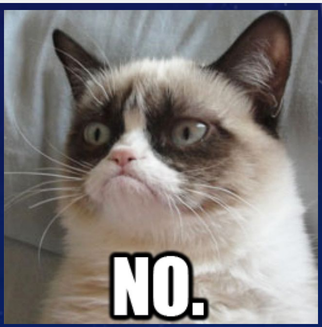
```
Na = [10.9; 13.7; 14.3; 14.1; 12.3; 12.6; 14.1; 12.0; 14.5; 12.1];
K = [68.2; 66.3; 67.0; 72.2; 72.3; 67.9; 71.5; 72.1; 71.4; 73.5];
Ca = [25.4; 26.4; 25.4; 26.7; 25.5; 26.8; 26.5; 26.9; 26.7; 25.7];
Mg = [13.8; 13.2; 14.1; 13; 17.3; 13.1; 17.7; 13; 15.6; 15];

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

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));
```

Is this a good approach yet?



Data Files

- Data should never live in your code. Put it in a separate data file.*

```
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));
```

Skip 1 header row.
Don't skip any columns.

Site	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
…	…	…	…	…
10	12.1	73.5	25.4	13.2

site_samples.csv

```
samples = csvread('site_samples.csv', 1, 0);
Na = samples(:,2);
K = samples(:,3);
Ca = samples(:,4);
Mg = samples(:,5);

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

Call the built-in csvread function, which reads data from the given file and returns it as a matrix.

*Data often already lives in files, generated by some other program.

The `sum` Function

- The `sum` function yields the sum of the elements in a vector.

- Applied to a 2D matrix, sum works column-by-column. The result is a row vector containing the sums of each column.

3
9
4

X

2	3	2	1

y

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

z

16

`sum(X)`

8

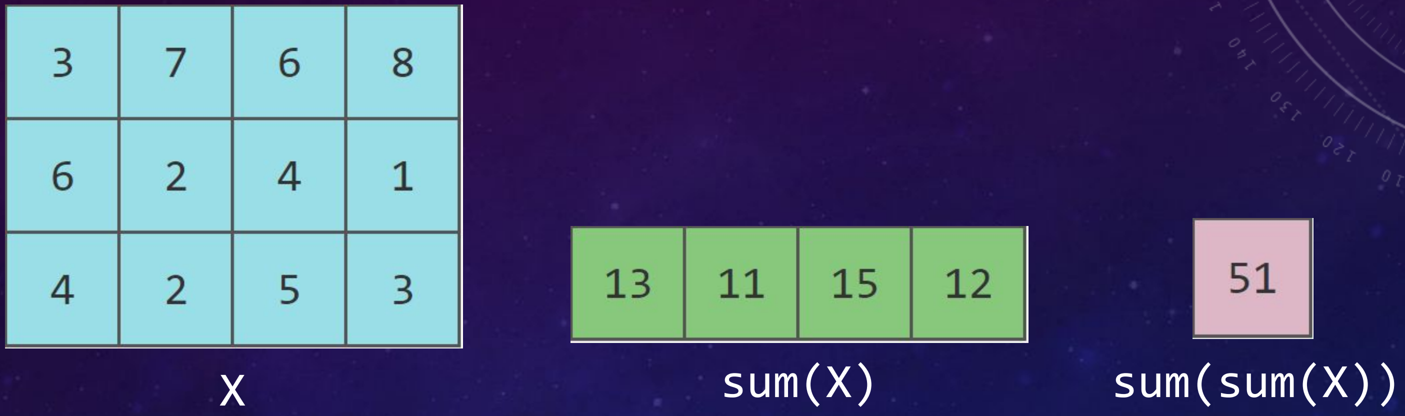
`sum(y)`

13	11	15	12

`sum(z)`

Finding the Sum of All Elements

- Option 1: Apply the `sum` function twice.
  - First - find sums of the columns. Then - add those sums to get the overall.



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

`X`

13	11	15	12

`sum(X)`

51

`sum(sum(X))`

- Option 2: Use the `:` to select all elements, then `sum`.

```
sum(x(:))
```

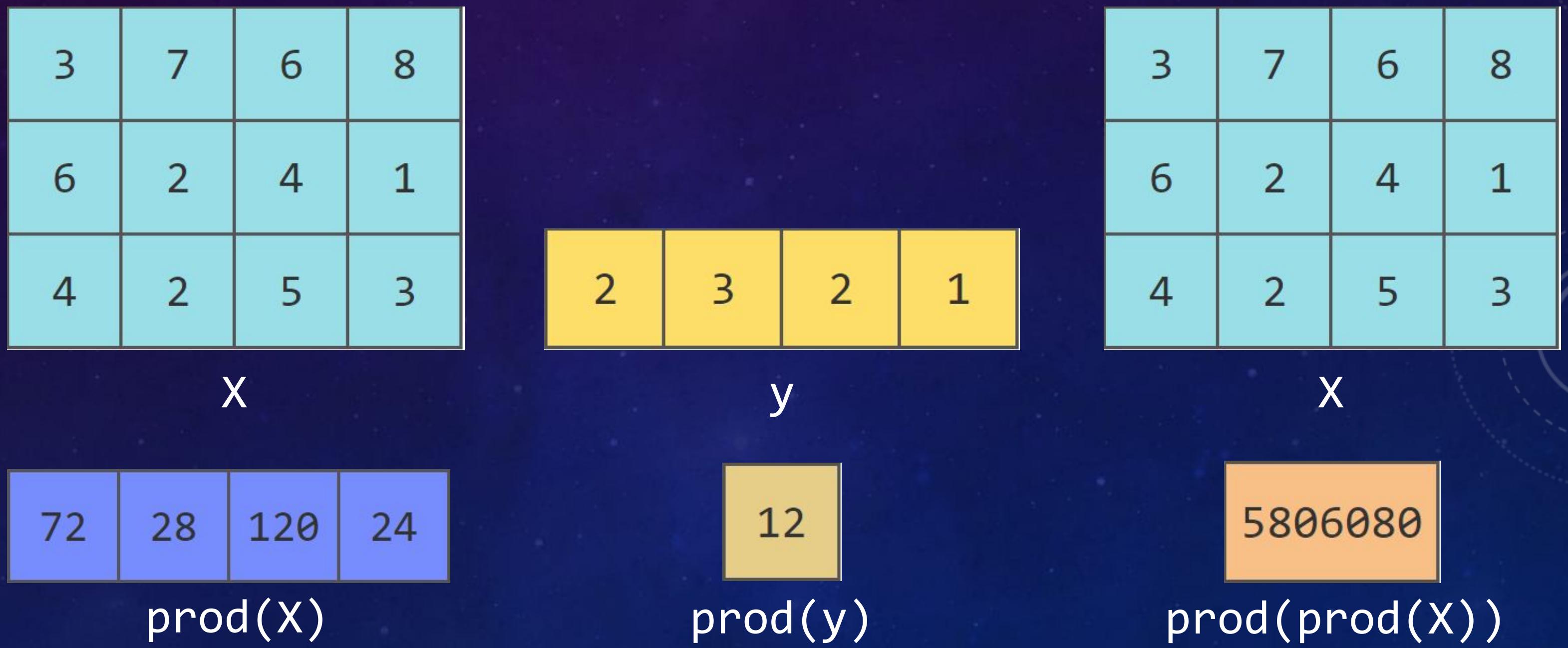
- Option 3*: Use the `'all'` option.

```
sum(x, 'all')
```

* The autograder does not support `'all'`, so don't use this in projects.

The `prod` Function

- The `prod` function works just like `sum`, but with multiplication instead of addition.



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

X

72	28	120	24

`prod(X)`

2	3	2	1

y

12

`prod(y)`

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

X

5806080

`prod(prod(X))`

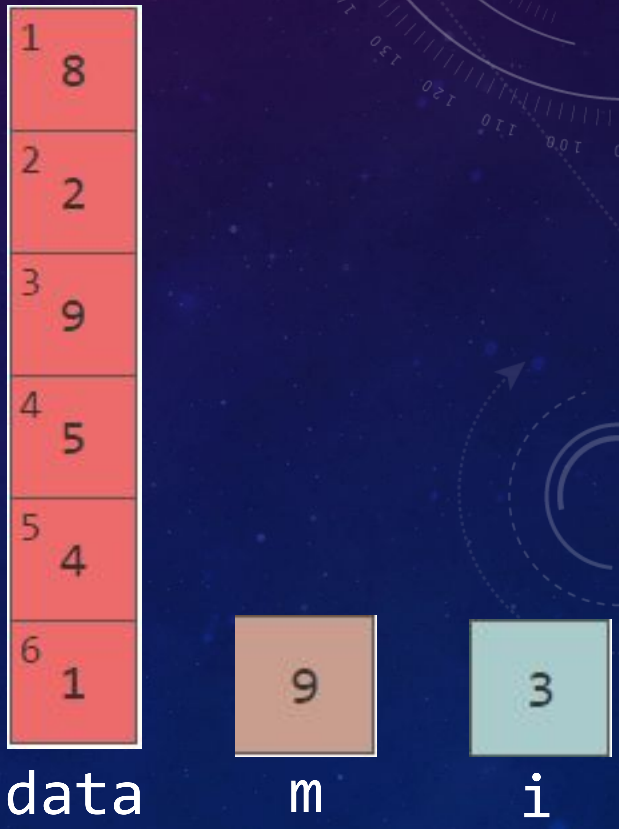
Capturing Multiple Return Variables

- Consider the interface for the built-in max function:

```
function [ m, i ] = max( X )
  % m Returns the maximum value in
X.
  % i Returns the index where the
  %   maximum was found.
  % implementation not shown
end
```

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

```
[m, i] = max(data)
```



The `min()` function works the same way.

Aggregator Functions

- Many functions work like `sum()` and `prod()`
- They compute some aggregate information about a dataset.
- When applied to a matrix, they work column-by-column.

```
sum()
prod()
mean()
median()

mode()
min()
```

Exercise: Monthly Average ESP

- The nutrient cycle of Proxima B is not yet fully understood.
- Scientists want to check whether ESP changes over time.
- We'll use a larger dataset with samples taken over a year.[1]
- Write a function that finds the average ESP in a given month.

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
…	…	…	…	…
360	…	…	…	…

daily_samples.csv

```
dailySamples = csvread('daily_samples.csv', 1, 0);
Na = dailySamples(:,2);
K = dailySamples(:,3);
Ca = dailySamples(:,4);
Mg = dailySamples(:,5);
esp = ESP(Na, K, Ca, Mg);

avgMonth1 = monthlyAverage(esp, 1);
disp('Month 1 avg:');
disp(avgMonth1);

disp('Month 7 avg:');
disp(monthlyAverage(esp, 7));
```

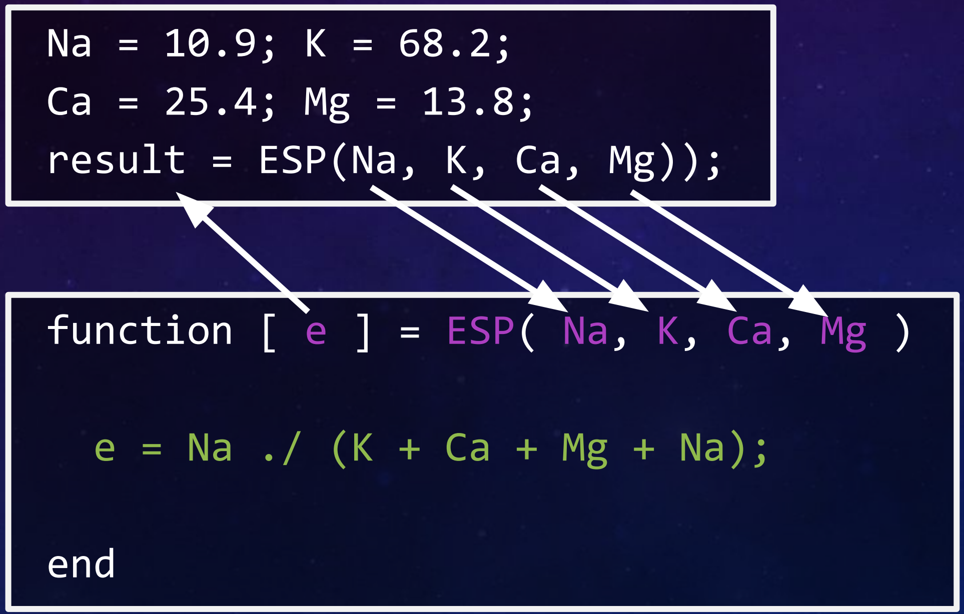
TestMonthlyAverage.m

Interface: Take ESP vector and month number, return average for that month.

[1] Assume 1 month = 30 days for simplicity. There are 360 samples total. Month numbering starts at 1 (i.e. 1 is January, 12 is December)

Parameter Passing

- 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));
```

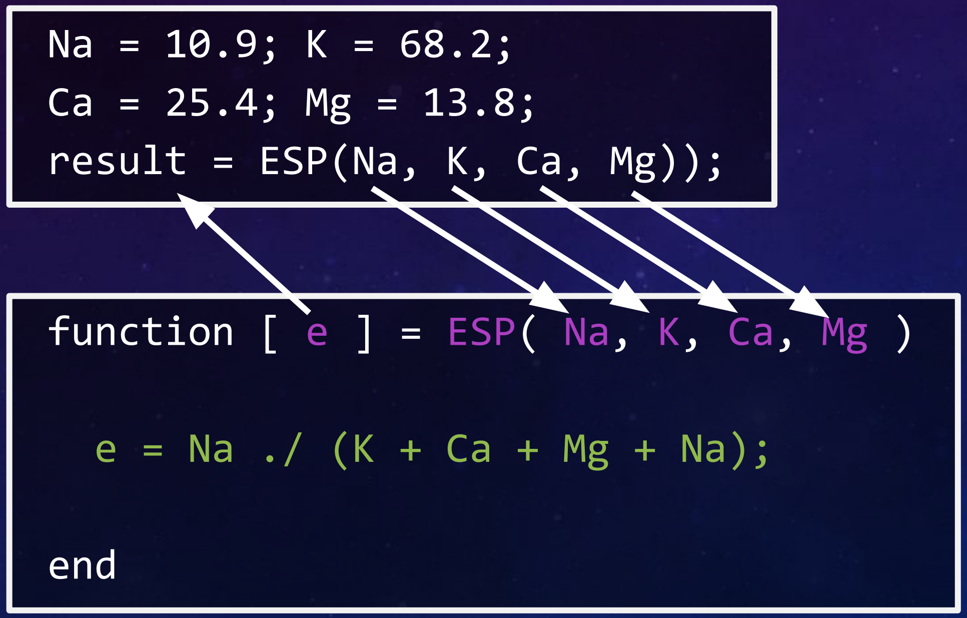
```
function [ e ] = ESP( Na, K, Ca, Mg )

  e = Na ./ (K + Ca + Mg + Na);

end
```

Note that the "outside world" only ever interacts with the **interface**. It doesn't have to worry about the **implementation**.

[1] This follows the rules of "pass-by-value". We'll come back to other mechanisms for parameter passing in the 2nd half of the course.

Variable Scope

- Variables in a function are **completely different** than those in the base workspace (i.e. your main program).
- 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, Ca, Mg )

  e = Na ./ (K + Ca + Mg + Na);

end
```

Global Scope

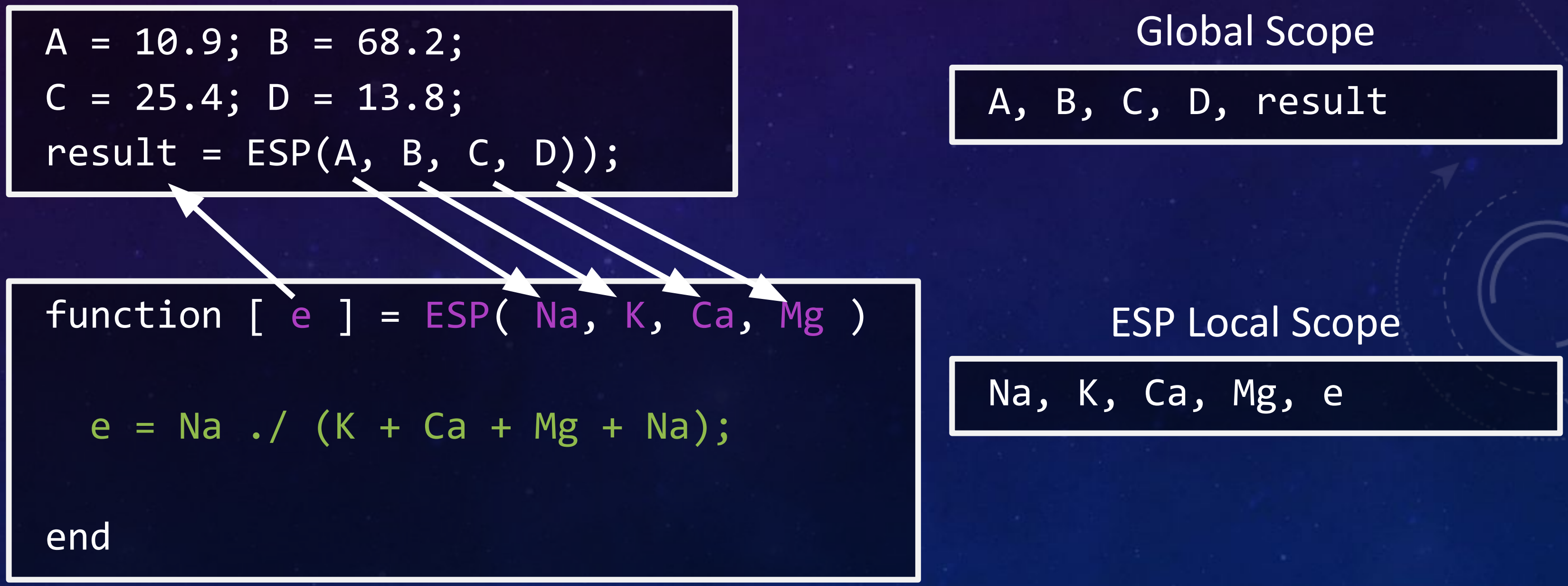
```
Na, K, Ca, Mg, result
```

ESP Local Scope

```
Na, K, Ca, Mg, e
```

What's in a name?

- We could change the names of either the parameters or arguments 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;
result = ESP(A, B, C, D));
```

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

  e = Na ./ (K + Ca + Mg + Na);

end
```

Global Scope

```
A, B, C, D, result
```

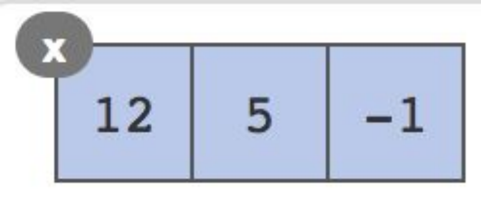
ESP Local Scope

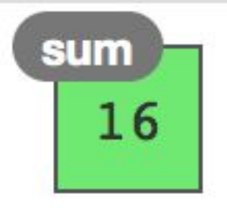
```
Na, K, Ca, Mg, e
```

Be careful when you name variables!

DoNotOverwriteBuiltInFunctions.m

```
x = [12 5 -1];

% Calculate the sum of these numbers by adding them up
sum = x(1) + x(2) + x(3);
disp('The sum of the numbers is: ');
disp(sum);

% Now, calculate the sum using the sum() function to see if it matches
disp(sum(x));
```





Command Window

```
>> DoNotOverwriteBuiltInFunctions
The sum of the numbers is:
   16

Array indices must be positive integers or logical values.

Error in DoNotOverwriteBuiltInFunctions (line 12)
disp(sum(x));
```

If you name a variable the same thing as a built-in MATLAB function, the variable “shadows” the built-in function, and you can no longer call the function.

A Function with no Parameters or Return Variables

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

```
function [ ] = fightSong( )
  % size Returns the dimensions of an array.
  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();
```

Calling Functions in Other Functions

- Functions can call each other. Each function is in its own file.

ESP.m

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

  e = Na ./ (K + Ca + Mg + Na);

end
```

meanESP.m

```
function [ m ] = meanESP( Na, K, Ca, Mg )

  m = mean(ESP(Na, K, Ca, Mg));

end
```

Your custom function files need to be in your current MATLAB folder.

Unit Testing

- Programs often use many functions working together.
- In **unit testing**, we test each function individually to make sure it behaves as it should according to its **interface**.
- Generally, this amounts to:
  - Running the function with a bunch of inputs
  - Verifying it produces the right output for each one