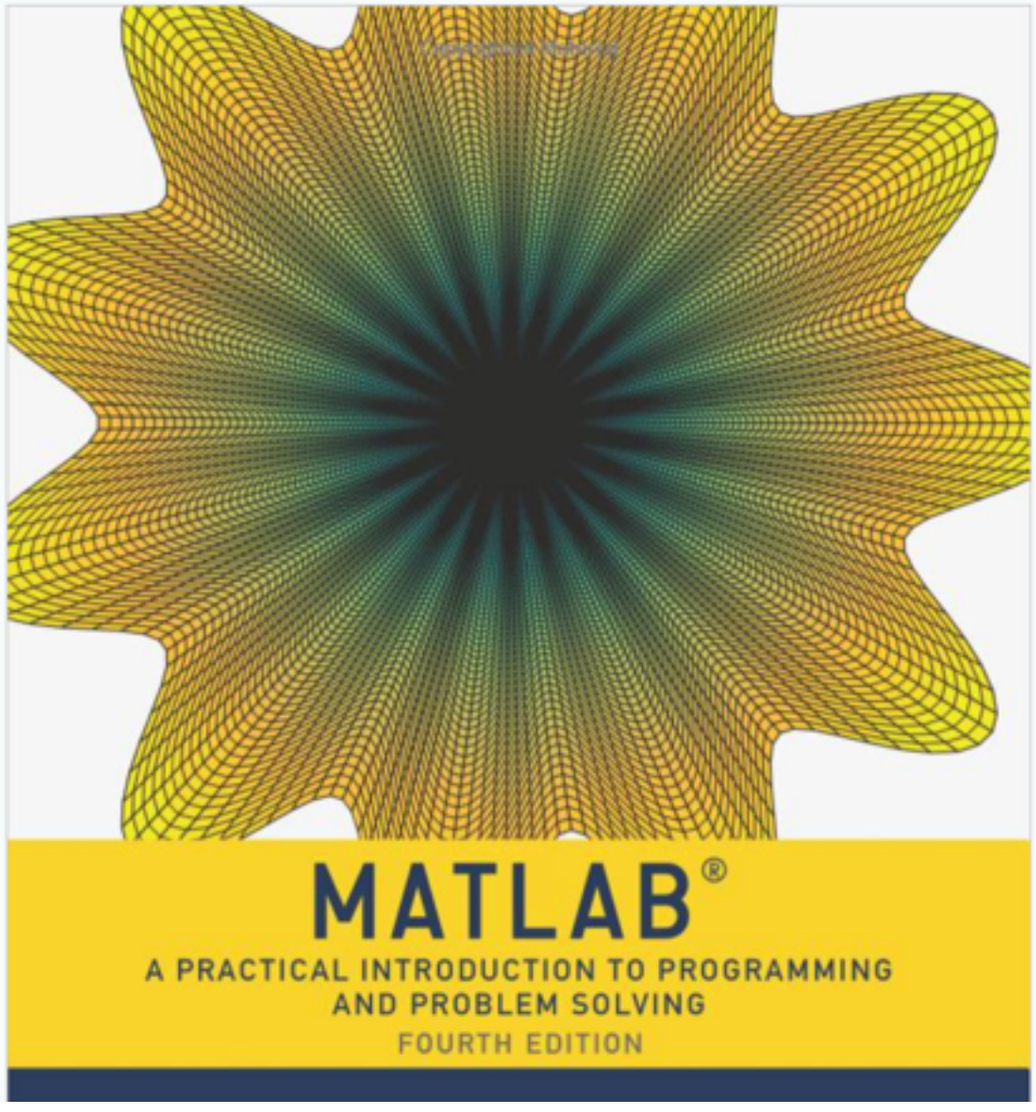
Introduction to Matlab!

NSC3270 / NSC5270
Computational Neuroscience

Tu/Th 9:35-10:50am
Wilson 316

Professor Thomas Palmeri
Professor Sean Polyn

optional suggested Matlab textbook



MATLAB: A practical introduction to programming and problem solving, 4th Edition, Stormy Attaway, Elsevier Publishing

(can be ordered from your favorite online store)

Online Matlab documentation

- Within Matlab

```
>> help some_function
>> doc some_function
```

- Online documentation
  http://www.mathworks.com/help/techdoc/

- PDF files
  http://www.mathworks.com/help/index.html

- Video Tutorials
  http://www.mathworks.com/videos/

we provide links to other Matlab-related documents as we cover specific concepts in class

Topics we will cover

- The Matlab environment
- Variables, assignment, data types, basic syntax
- Scripts and functions, paths
- Comparison, logical operators
- Control flow: if, for, while
- Making fancy figures
- Vectors and matrices
- Matlab hotdogging

HW#2: The logistic function

$$f(x) = \frac{L}{1 + e^{-k(x - x_0)}}$$

Due next Thursday 1/19 before class starts.

Create two functions that implement a logistic transformation (next slides have specifics).

Submit the m-files containing the functions through Blackboard.

The lecture slides contain all the Matlab functions and commands you need to do this assignment.

HW#2: The logistic function

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

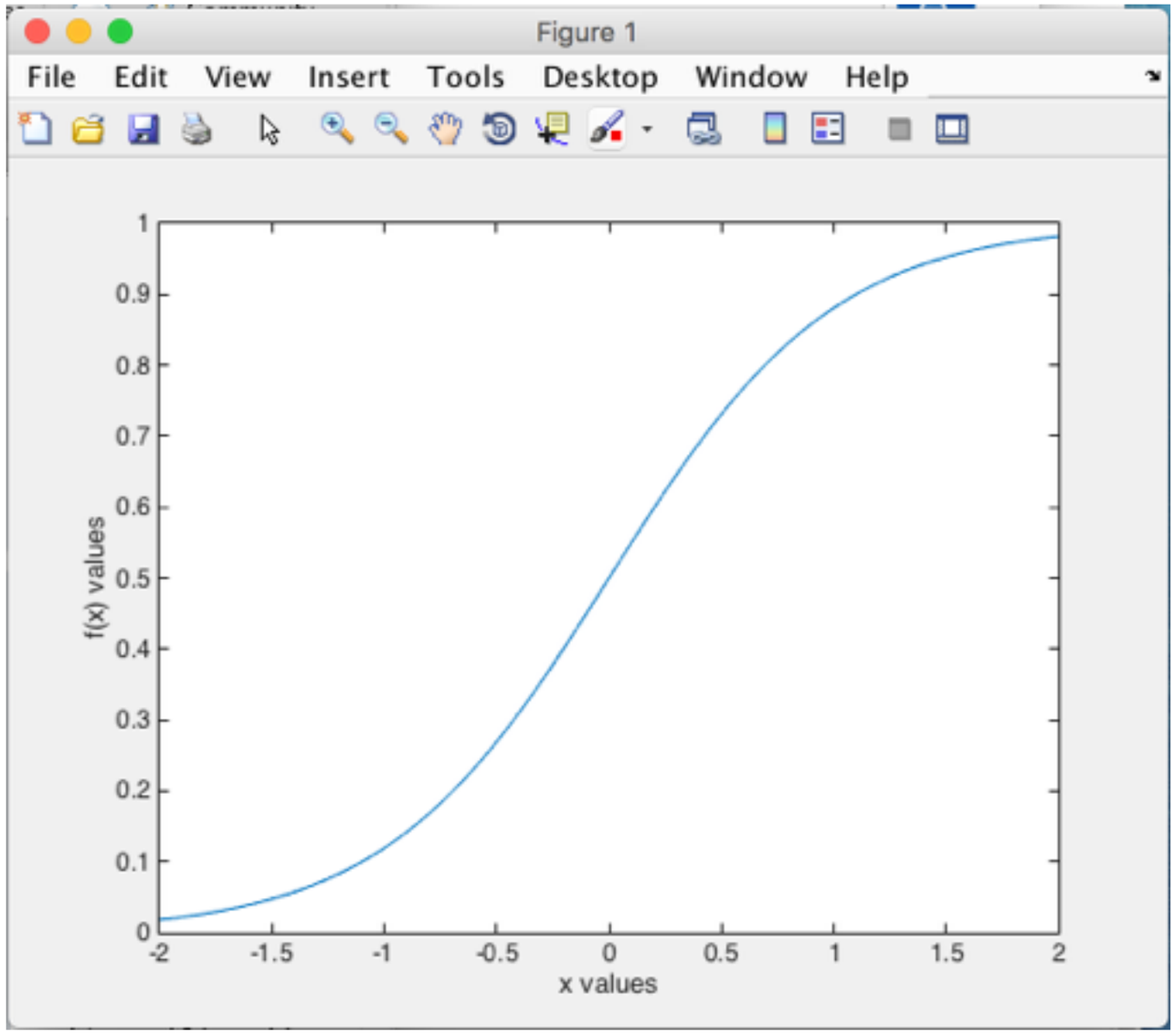
That L term can be fixed at 1. The x0 term can be fixed at 0. e is Euler's number.

The function should take a vector of x values as an input argument. Another input argument should allow the user to specify k. The output argument should be a vector of f(x) values corresponding to the x values.

The function should also create a figure plotting the input x values against the output f(x) values. The figure's axes should be labeled "x values" and "f(x) values".

HW#2: The logistic function

So if we gave it x values ranging from -2 to 2, and k=2, the plot should look something like this:



HW#2: The logistic function

The two functions

The first function should use a for loop to iterate through the vector of x values.

The second function should use vector arithmetic to transform the x values into f(x) values without a for loop.

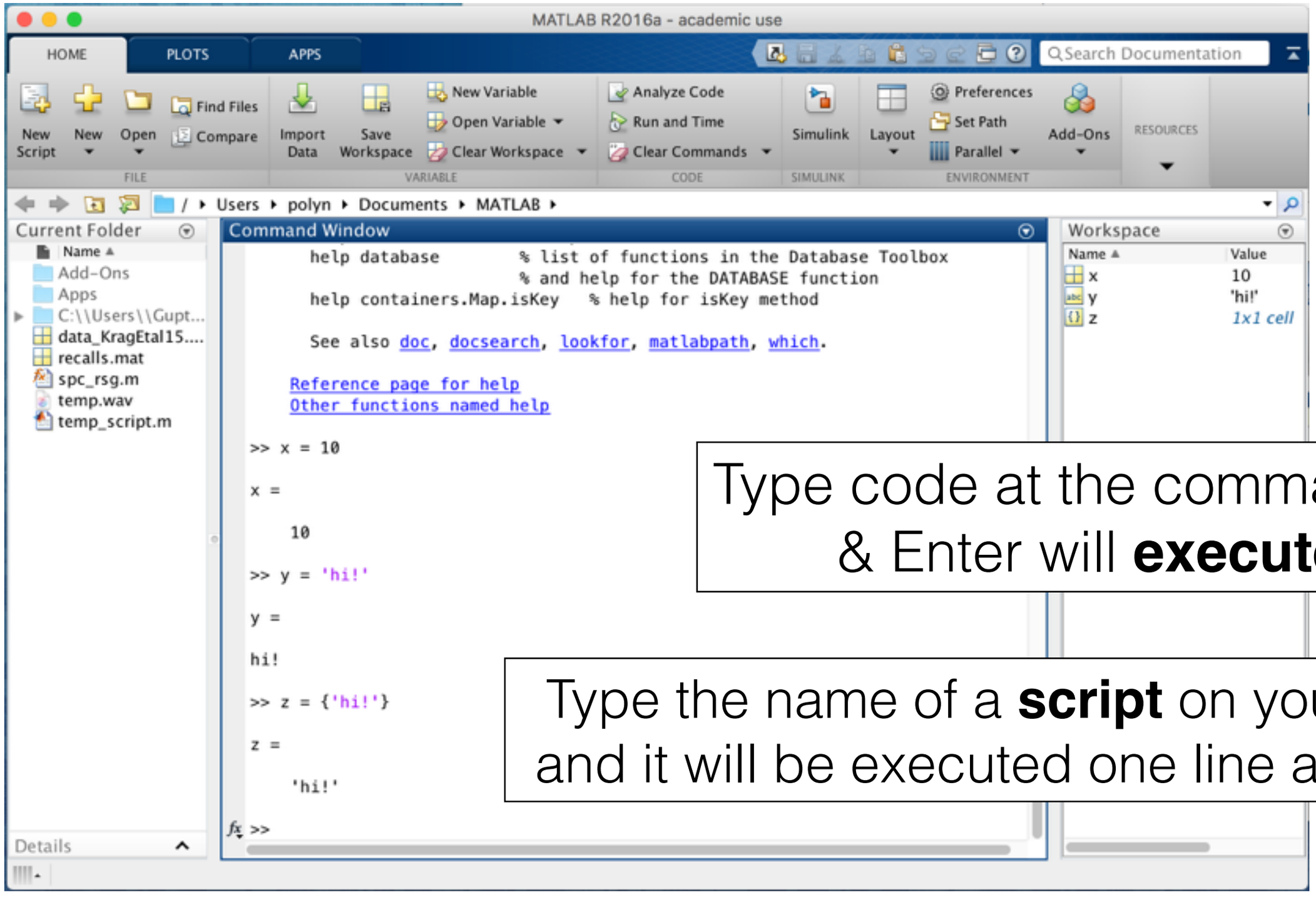
(both functions should produce the same output values for a given set of input values!)

Make a ZIP file with the two m-files inside

The Matlab environment - Vocabulary corner

- execute / evaluate
- path
- workspace
- script
- function
- scope

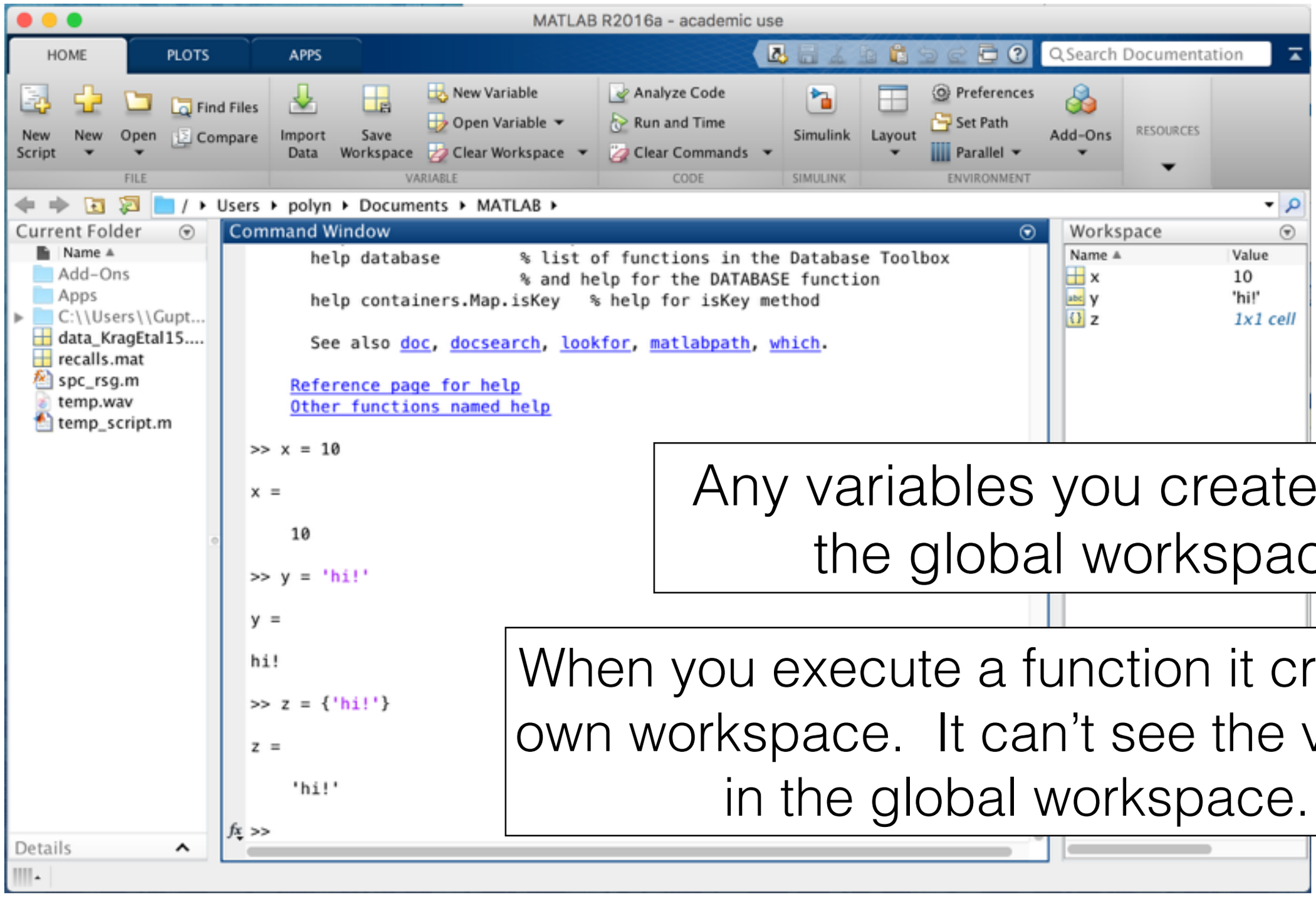
The Matlab environment



Type code at the command line & Enter will **execute** it

Type the name of a **script** on your **path** and it will be executed one line at a time

The Matlab environment - Workspace



Any variables you create live in the global workspace

When you execute a function it creates its own workspace. It can't see the variables in the global workspace.

The Matlab environment - helpful commands

```
>> whos
>> doc
>> which function_name
>> help function_name
>> doc function_name
>> pwd
>> clear all
>> cd ~
```

The Matlab environment - Miscellany

The command line:

```
>> x = 5
```

```
>> x = 5;
```

Semicolon stops Matlab from showing you the answer

If you want to stop Matlab from executing something, try CTRL-C

The Matlab environment - arithmetic

```
>> 1+1
>> 10-1
>> 10*10
>> 10/5
>> 5^2          % raise to a power
>> sqrt(2)      % can you guess?
>> exp(1)       % Euler's number: e[1]
```

1. That was meant to be an exponent, not a footnote[2]
2. BTW you need this function for your homework

The Matlab environment - order of operations

PEMDAS applies

(parenthesis, exponent, multiplication, division, addition, subtraction)

If you can't remember, just add more parentheses?

```
>> 5*4-3

>> (5*4)-3   % same thing
```

Variables, assignment, data types

```
Basic types: double, single, int,
string/character, logical

>> x1 = 5;          % ends up being a double
>> x2 = ‘hi!’; % a string
>> x3 = true;   % logical (true, 1)
>> x4 = x1==6; % logical (false, 0)
```

Variables, assignment, data types

```
>> x1 = 5;
>> x2 = 'hi!';
>> x3 = true;
>> x4 = x1==6;
>> whos
```

`whos` gives a list of the variables in your workspace

```
  Name        Size                    Bytes  Class
Attributes

  x1          1x1                         8  double
  x2          1x3                         6  char
  x3          1x1                         1  logical
  x4          1x1                         1  logical
```

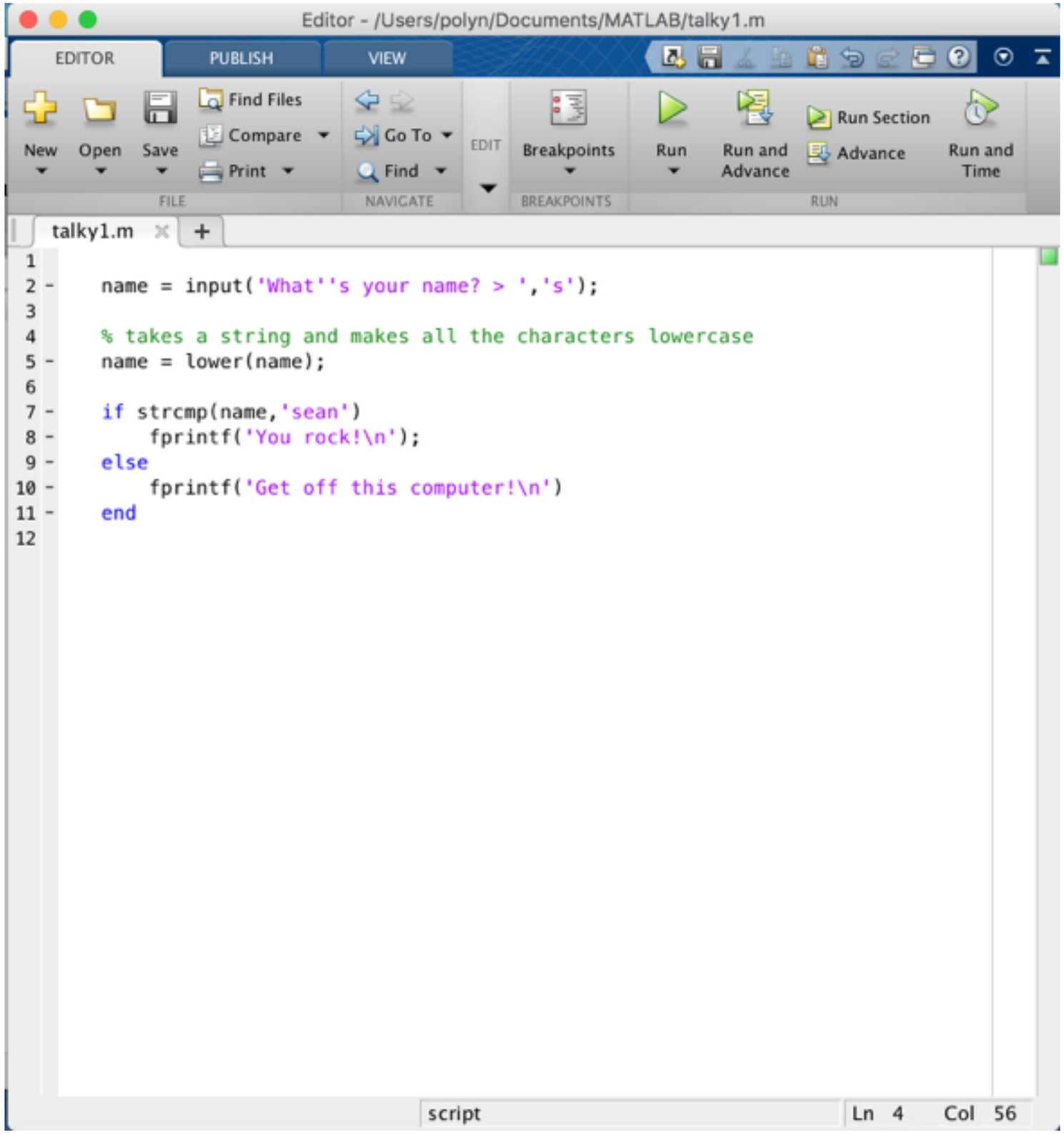
Writing scripts

The Matlab Editor allows you to write scripts and functions.

The editor is just a text editor, you could use another text editor to make scripts & functions (like Emacs).

You write some code in this window, and save it, and a text file is saved to disk.

Keep track of where (what directory) that file is saved!



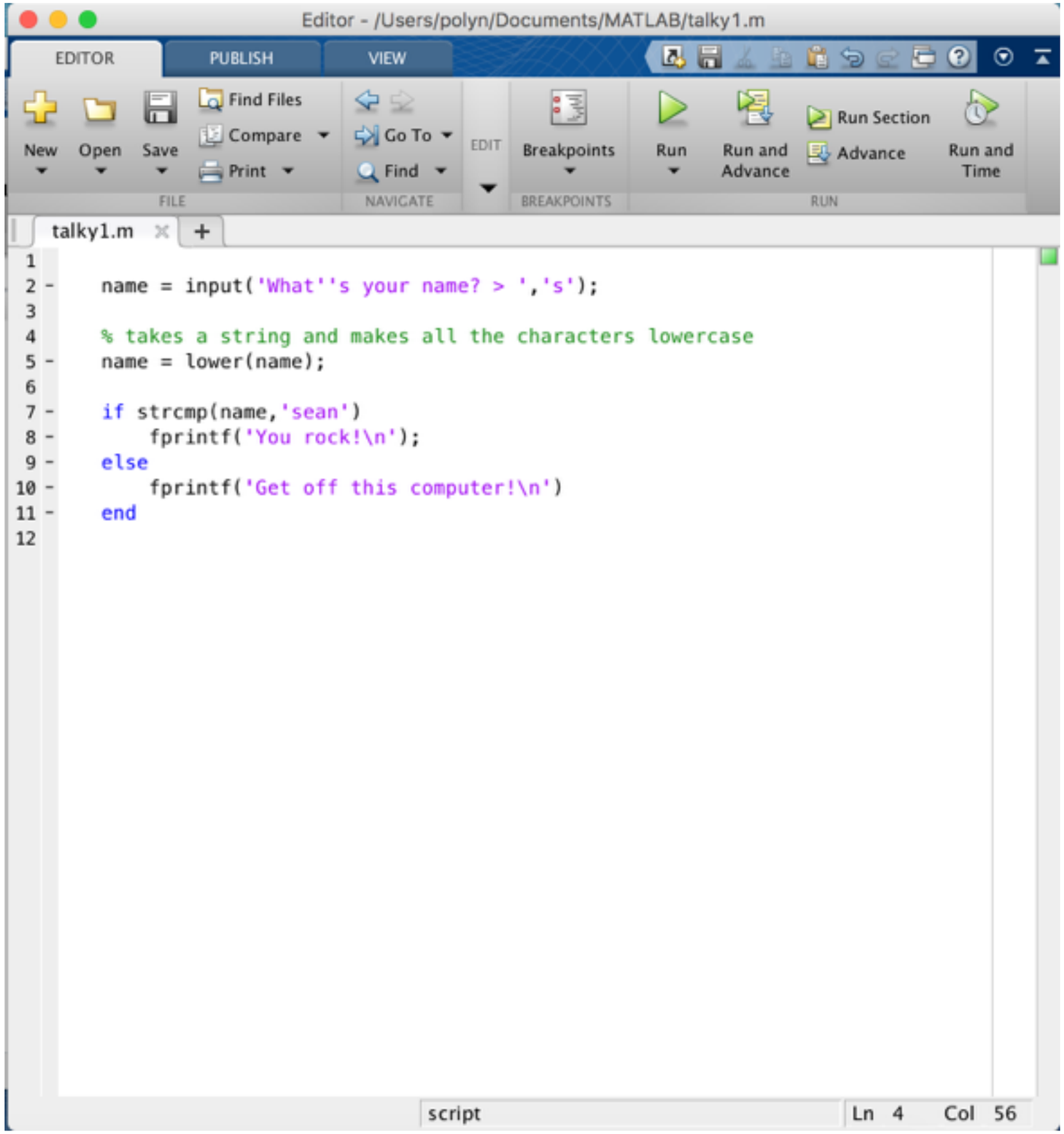
Writing scripts

Comments:

If you have a % sign in your code, that indicates the following text is a comment, that code won't be evaluated.

Useful for making notes to yourself or to us.

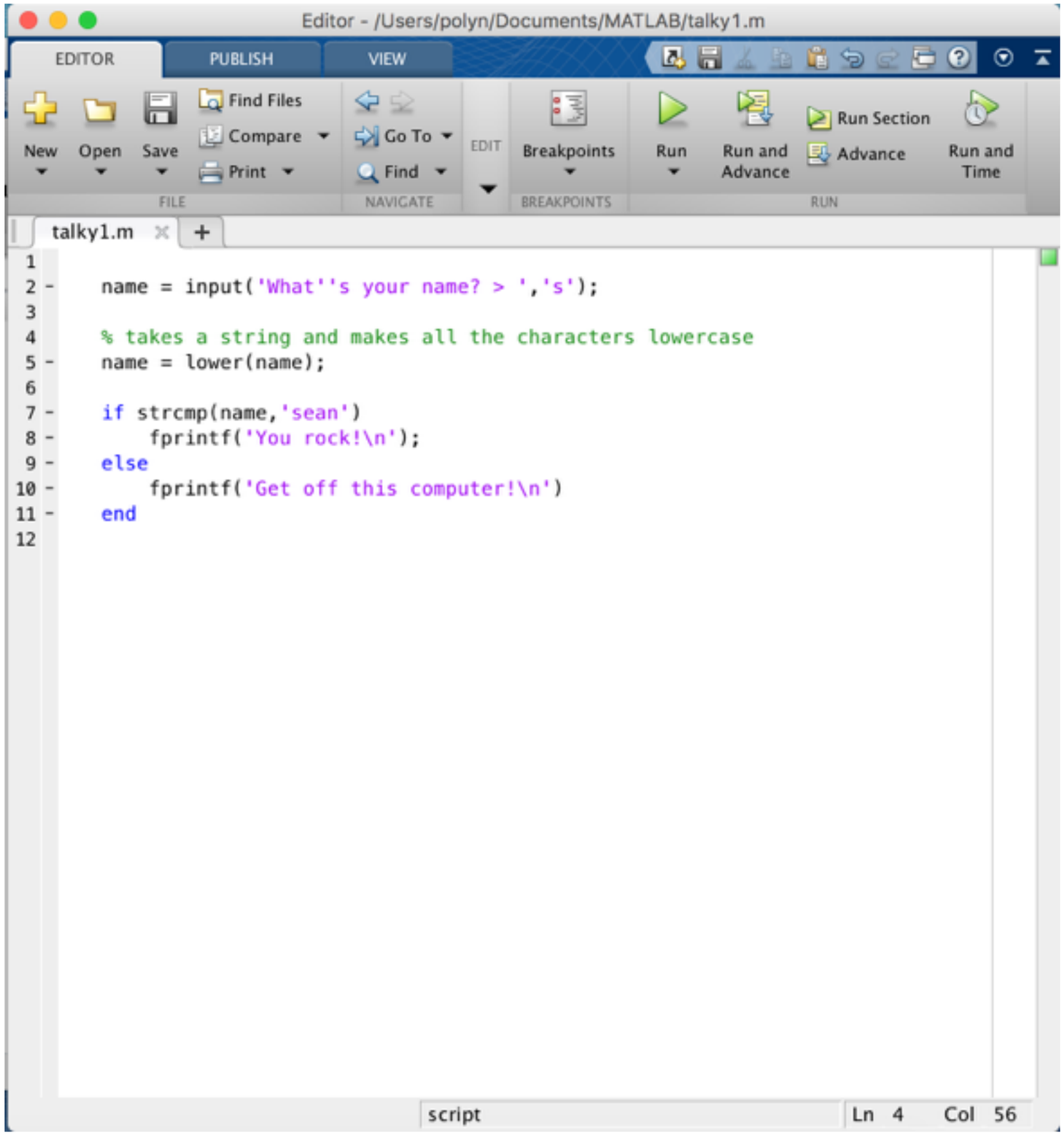
Note: A % sign inside single quotes means something different though.



Writing scripts

The extension on one of these files is .m

so I tend to call it an “m file” or a “dot m file”



Writing scripts

The path:

The path is the set of directories Matlab will look in to check for scripts or functions. Some directories are automatically on the path, like whatever directory you are currently in, as well as `~/Documents/MATLAB` (on a Mac).

If you get an error like:

```
>> talky1

Undefined function or variable 'talky1'.
```

One possibility is that you are in the wrong directory, and wherever talky1.m got saved is not on your path.

```
>> pwd % stands for 'present working directory'
```

This will tell you your current directory (or you can look on the toolbar).

Writing scripts

Changing directory / Where am I?

Matlab usually starts you off in `~/Documents/MATLAB`, but you can navigate around your computer using the GUI or the `cd` command.

Adding directories to your path:

If you have some scripts / functions in one directory, but need to navigate to another directory, you can add a directory to your path.

```
>> addpath('~/some_directory/sub_dir/this_one');
```

When specifying a path, the tilde ~ symbol is shorthand for your home directory.

Functions

In mathematics, a function is used to map one set of values to another set of values.

Here are some linear functions. They are equivalent to one another.

By convention, m and b are constants, but x and y are variables. Here, you can pick a value for x, plug in the values of m and b, and calculate the value of y. So x is like an input to the function, and y is an output.

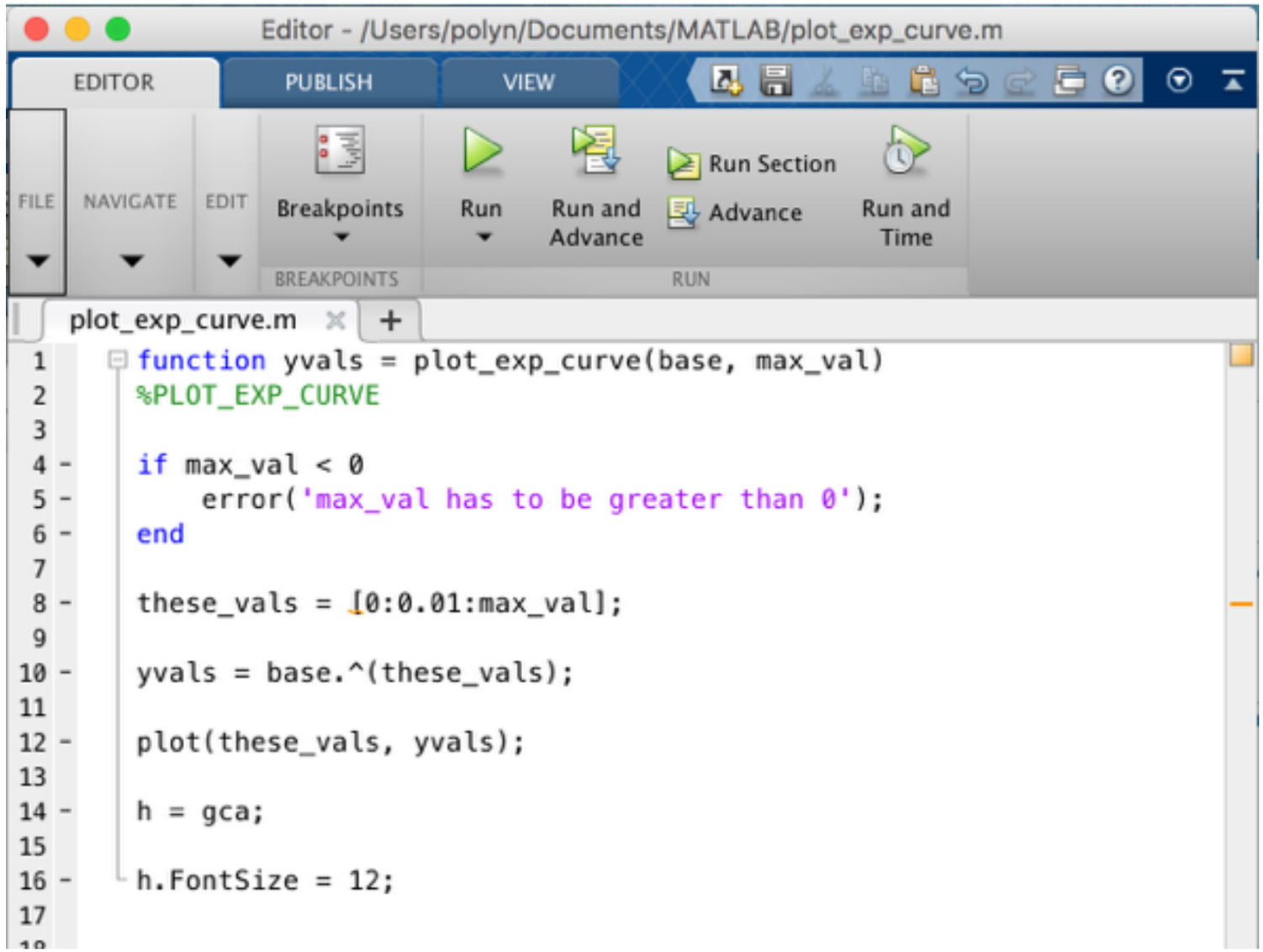
In programming, a function is very similar!

$$y = mx + b \qquad\qquad f(x) = mx + b$$

Functions and scripts

If you just start writing code in the editor, then you are writing a **script**.

To turn a script into a function, the first line of the script has to have a particular structure.

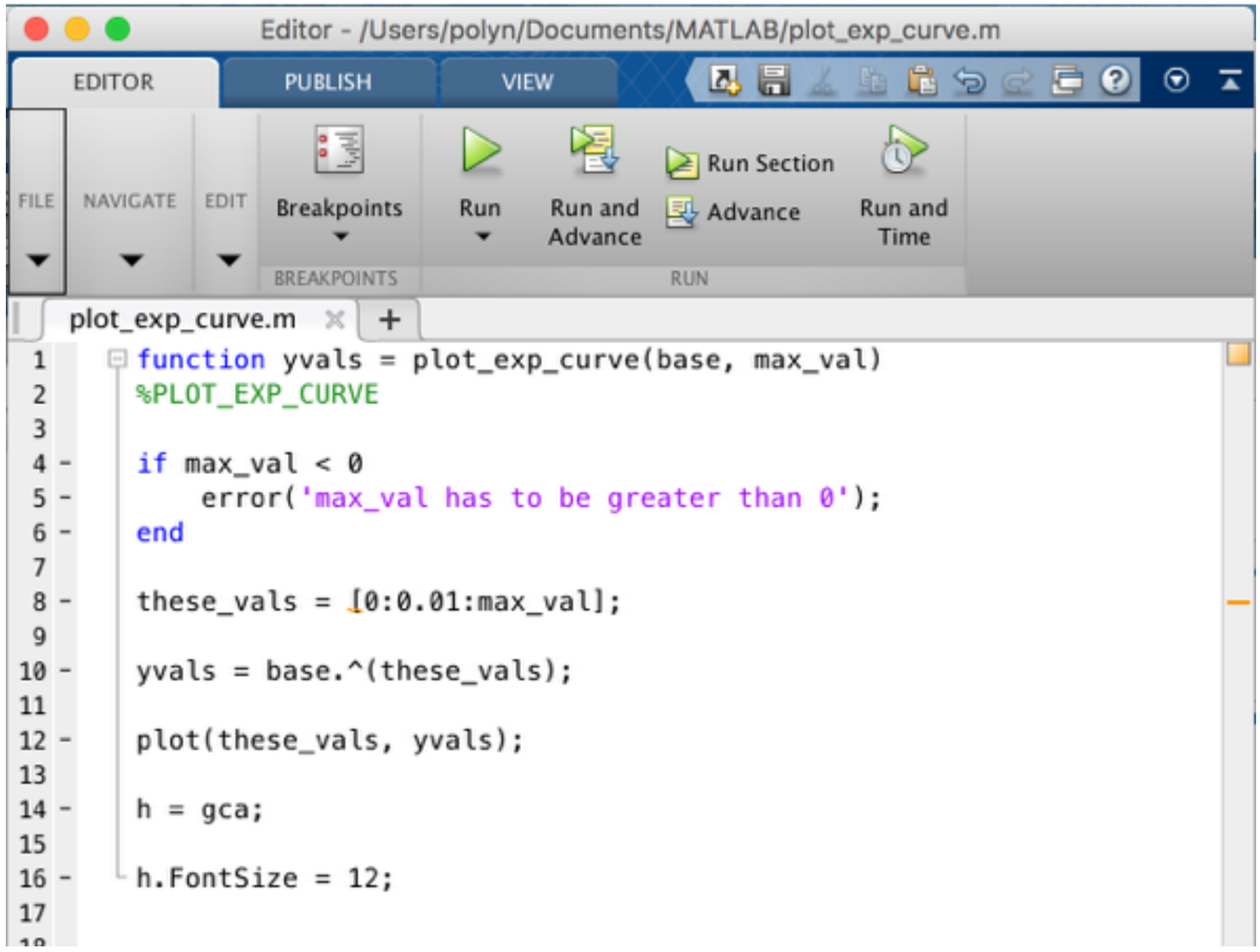



```
function [out_args] = function_name(input_args)
```

Functions and scripts

Once your script is a function, it behaves a bit differently.

Now it has its own workspace: The only variables it has access to are the ones specified in the set of input arguments.




```
function [out_args] = function_name(input_args)
```

Functions and scripts

This is called the function definition statement:

```
function [out_args] = function_name(input_args)
```

You can have however many input args and output args as you like.

```
function [perf, data] = run_network(n_units, n_layers)
```

At the command line, you specify the values of the input arguments, and specify names for the output arguments (these don't have to match what you wrote in the function definition statement):

```
>> [perfv1, datav1] = run_network(10, 3);

>> [perfv2, datav2] = run_network(100, 4);

>> run_network(100, 4);
```

If you don't specify the output arguments, the function won't be able to pass its results out to the global workspace.

Relational operations (on numbers)

Single = means assignment, Double == means comparison

produces a logical: TRUE (1) or FALSE (0)

```
>> x=5; y=6;
>> x==y  % these two things are equal: FALSE
>> x~=y  % these two things are not equal: TRUE
>> val = x==y;      % val is a logical truth value
>> val = eq(x,y);   % the functional form equivalent
>> val = x < y;     % x is less than y: TRUE
>> val = x > y;     % x is greater than y: FALSE
>> x >= y; x <= y;
```

Logical operators

```
>> x=true; y=false;

>> val = x & y;     % logical AND: true if both x and y
are true

>> val = x && y;    % short-circuit version, if x is
false then y isn't evaluated

>> val = and(x,y); % functional form, equivalent

>> val = x | y;     % logical OR, also or(x,y): TRUE

>> val = ~y;     % logical NOT, also not(y): TRUE
```

Comparison (strings)

```
>> x='hi'; y='ho'; z='hi';

>> val = strcmp(x,y);   % FALSE

>> val = strcmp(x,z);   % TRUE
```

Data types - Arrays (vectors)

```
Creating an array / vector

>> x = zeros(1,10); % one row, 10 columns

>> y = zeros(10,1); % 10 columns, 1 row

>> z = [1:10];  % colon operator constructs a vector
counting up from 1 to 10 in increments of 1

>> x = [8 1 4 2 3 1]; % 1 row, 6 cols

>> x(1) > x(2) % 8 > 1, TRUE
```

Data types - Arrays (vectors)

```
>> z = [1:10];

>> zt = z';  % single quote transposes the array, if
no complex numbers are involved, equivalent to z.'

transposing flips the row and column indices for each
element
```

```
>> x = [1:3:10]; % counts up by 3's: 1 4 7 10

>> x = [10:-1:5];

>> x = [1:0.01:3];  % also check out function linspace

>> length(x)

>> size(x)
```

Data types - Arrays (vectors)

```
>> x = [];  % the empty array

>> length(x); % is zero

>> x = [1 2 3];

>> x(1) = 6; % now x is [6 2 3]

>> x = [1:10];  % makes a row vector
```

```
>> x(7:10)   % will give you [7 8 9 10]

>> x(7:end)  % same thing

>> x = [1:10]'  % single quote here transposes the
vector, now it is a column vector

>> x'   % this will print out the column vector but
won't change x

>> x = x';  % this will change x
```

Data types - Arrays (vectors)

```
>> x = [1 2 3];
>> x(1) = 6; % now x is [6 2 3]
>> x(end) = 4;  % now x is [6 2 4]
>> x(end+1) = 5; % now x is [6 2 4 5]
>> x(end+4) = 5; % now x is [6 2 4 5 0 0 0 5]
>> x(3)==4      % this is TRUE
>> x(1,3)==4    % this is equivalent, and TRUE
>> x(3,1)==4    % this will give an error
Index exceeds matrix dimensions.
```

Common error messages

```
>> x = zeros(1,10);

>> x(1:5) = ones(5,1);

In an assignment  A(:) = B, the number of elements in
A and B must be the same.
```

Data types - Arrays (vectors)

```
>> x = [1 2 3];
>> x .* 5      % displays 5 10 15
% multiply each element of an array by a certain number
>> x ./ 2      % displays 0.5 1 1.5
% divide each element of an array by a certain number
```

Control flow - if statements

Pseudocode version:

If this condition is true

    run this code

Otherwise

    run this code

```
if x==y
   disp('they are equal!');
else
   disp('they are not equal!);
end
```

Control flow - if statements

```
if x==y
   disp(‘x and y are equal!’);
elseif x==z
   disp(‘well, at least x and z are equal!);
else
   disp(‘x is not equal to anything :(‘);
end
```

Control flow - for loops

The workhorse of programming. Run the enclosed code for each element in a list. On the left side of the equals sign you specify the index variable you'll use, on the right side, you provide the list.

```
count = 0;

for i = 1:10

   count = count + i;

end

% another way of writing the same thing

count = 0; x = [1:10];

for i = x

   count = count + i;

end
```

Control flow - for loops

```
x = randn(1,10);
for i = 1:length(x)
    sum = sum + x(i);
end
% this code will work, but it is dangerous! why?
```

Control flow - nested for loops

```
for i = 1:10
    for j = 2:5
        fprintf('%d', j);
    end
    fprintf('\n');
end
% what will this code do?
```

Control flow - while loops

```
% a while loop will just keep looping as long as the condition is true
count = 0; flag = true;
while flag
   count = count + 1;
   if count > 100
      flag = false;
   end
end
```

```
% same thing
count = 0;
while true
   count = count + 1;
   if count > 100
      break;
   end
end
```

Making fancy figures

```
>> figure(1);   % create a blank figure or select an existing figure
>> clf;      % clear the figure
>> plot([0:0.01:2*pi],sin([0:0.01:2*pi]))
>> h = gca; % create a handle for the current axis
>> h.FontSize = 12;
>> h.Children(1).LineWidth = 4;
>> h.XLabel.String = 'time'
>> xlabel('time')  % same thing
```

Making fancy figures

```
>> figure(1);
>> clf;
>> plot([1 2 3 4], [0.5 1.5 0.75 1.0],'rx-')
% 'rx-' specifies line color, marker type, and line style
>> axis([0 5 0 2])
% Specify [xmin xmax ymin ymax]
>> h = gca
>> h.XLim = [0 4.5];
% can also access these things with the handle
>> h.Children(1).Marker = 'o';
```

Data types - Arrays (matrices)

```
>> x = [1 2 3; 4 5 6; 7 8 9];
```

Semicolons inside square brackets will start a new row. Now you can index the elements of the matrix by specifying (row, column). A colon can be used to grab all the elements, or a subset of them

```
>> x(2,2)
% what will get displayed for each of these?
>> x(2,:)
>> x(:,2)
>> x(2:3,2:3)
```

Data types - Arrays (matrices)

```
>> x = [1 2 3; 4 5 6]
x =
     1     2     3
     4     5     6
>> x(2,1)==x(1,2)+2    % TRUE or FALSE?
>> x(3)
% only one subscript?  it still works, but which
element will it give you?
```

Data types - Arrays (matrices)

```
>> x = [1 2 3; 4 5 6]

x =

     1     2     3

     4     5     6

>> x(:)     % unraveling

ans =

     1

     4

     2

     5

     3

     6
```

Data types - Arrays (matrices)

Logical operators work on vectors and matrices, and can be used to create masks.

```
>> x = [1 2 3; 4 5 6; 7 8 9];

>> x >= 5

ans =

   0   0   0

   0   1   1

   1   1   1

>> x(x >= 5)
% what values will it give you?
% hotdog: in what order will it give them?
```

Data types - Arrays (matrices)

```
>> condition = [1 2 1 1 2 2];
>> rt = [100 300 150 200 400 300];
>> c1mask = condition==1;
>> c1rts = rt(c1mask);
>> meanc1rt = mean(c1rts); % what's the answer?

>> meanc1rt = mean(rt(condition==1));
% same thing in one line, hot dog
```

Data types - Arrays (N-d matrices)

You can have as many dimensions as you want in a matrix, but it can be confusing if you have too many

```
>> x = ones(3,4,2,8)

>> x = ones(8,4,1,3,12)  % you can do this but why?

>> x = randn(10,10,10)
```

Data types - Fun with arrays

find returns the indices of the nonzero elements in an array

```
>> x = [0 0 84; 0 0 0; 0 12 0];

>> ind = find(x)

>> [row,col] = find(x)

>> x(ind)
```

Data types - Fun with arrays

```
>> x = [1 1 1 8 8 1 2 2 4 4 4 4 4 8 8 8 1 1 8 8 2 4];
>> unique(x)
ans =
     1     2     4     8
```

Data types - Fun with arrays

```
>> x = [0 0 1]; y = [1 2 3];

>> y(x)

Subscript indices must either be real positive
integers or logicals.

>> x = logical(x);

>> y(x)

ans =

     3
```

Data types - Fun with arrays

```
>> x = [0 0 1]; y = [3 1 2];
>> x(y)
ans =
     1     0     0
>> z = [3 3 3 3 3];
>> x(z)
```

Data types - Fun with arrays

```
>> x = [0 0 1];  y = [1 1 1]; z = [0 0 0];
>> any(x)     % TRUE
>> any(z)     % FALSE
>> all(y)     % TRUE
>> all(x)     % FALSE
```

Data types - Fun with arrays

```
>> rts = [380 2140 600 630 840 3560 4510];
>> mask = rts > 1000;
>> rts(mask) = 1000;  % sets the TRUE indices to 1000

>> rts = [380 2140 600 630 840 3560 4510];
>> rts(rts>1000) = 1000;  % same thing
```

Data types - Structures

Structures are another kind of data type, they are good for bookkeeping, keeping track of many variables, parameters, results of analyses. Different fields on a structure can hold different data types.

```
>> network = struct;  % this isn't necessary
>> network.n_units = 100;
>> network.transfer_function = 'logistic';
>> network.wts = randn(100,100);
```

Data types - Cell arrays

Cell arrays allow you to store different data types in a single array, they use curly braces instead of square braces.

```
>> x{1} = 5;

>> x{2} = ‘hi!’

>> x

ans =

    [5]    ‘hi’

>> x(1) == 5
Undefined operator '==' for input arguments of type ‘cell'.

>> x{1} == 5

ans =

     1
```

Other miscellaneous junk

Ellipses continue long statements across multiple lines:

```
>> long_output_name = absurdly_long_function_name(input1, …
           input2, input3);
```

Good luck! Have fun storming the castle!