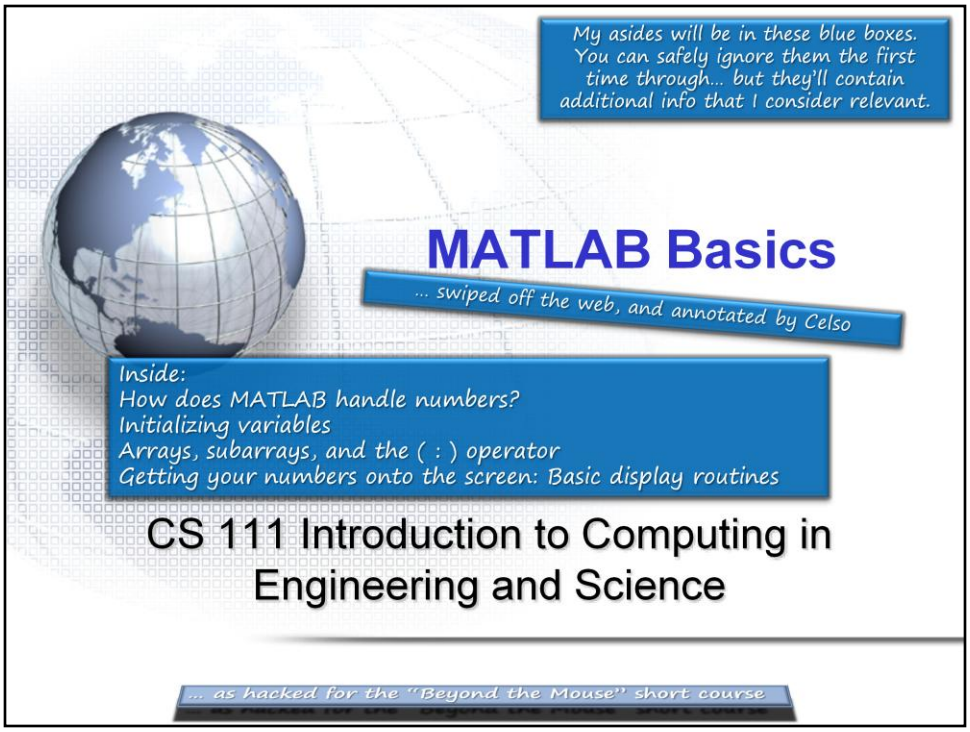


This handout set will provide the background information that will be useful in working in MATLAB. While it isn't my own creation, it hits the important stuff and (I thought) was pretty well organized. SO, I've annotated the heck out of them with the additional information that I'd include in a presentation, and am handing them out.

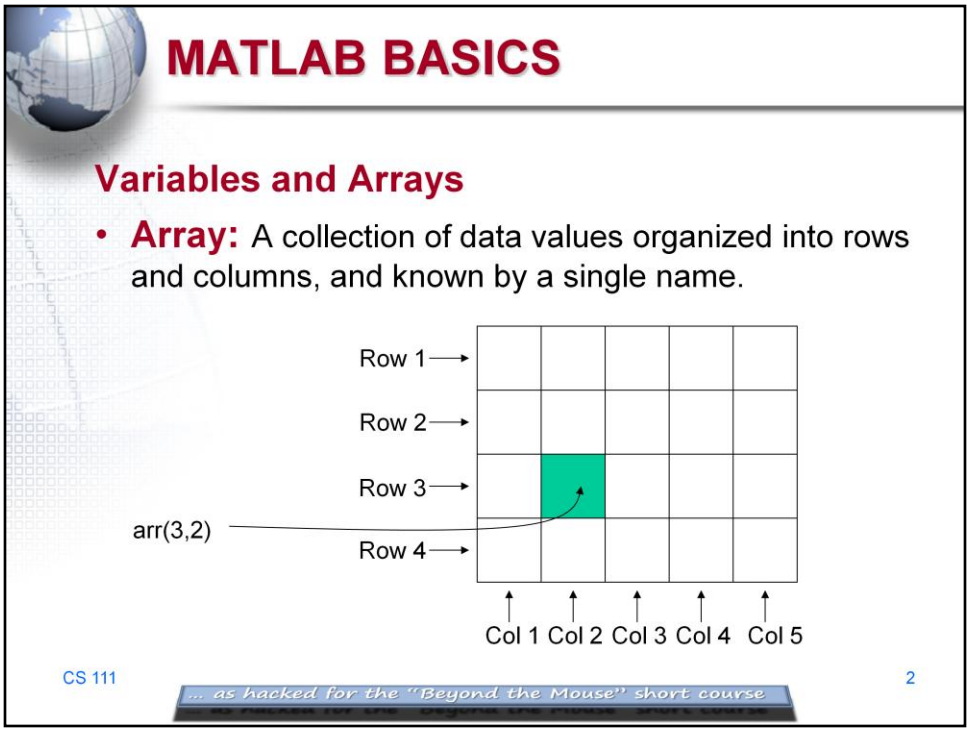
This handout doesn't go over language constructs like if-else-end or try-catch-end or loops. Instead it discusses the fundamentals: how MATLAB manages and deals with numbers. This is the core of MATLAB, and understanding this stuff will make your (programming) life easier.

Unfortunately, there's a huge gap between the amount of time I'll have in class and the material that I'd like to cover. When I start my MATLAB basics lecture, I'll take some questions based on this handout, then transition into how you can get MATLAB bend to your will. I intend to start with a (very) brief tour of the parts of the desktop environment that matter, then discuss the writing of scripts, and then transition into the creation and juggling of functions.

Below several of the slides, in the notes section (like this one), you'll find a couple comments as well as suggestions for things to try in MATLAB that might help clarify the concepts and syntax.

I recommend starting MATLAB and trying the examples as you go. *PLEASE* ask me about anything that needs clarification along the way. Frustration is your enemy when learning to program, and I want us all to have been exposed to these fundamentals before I start my class on Friday.

OK. Deep breath. **Here we go.**



If you'd like more information about any command, simply type:

help *command* OR **doc *command*** for the command in question.

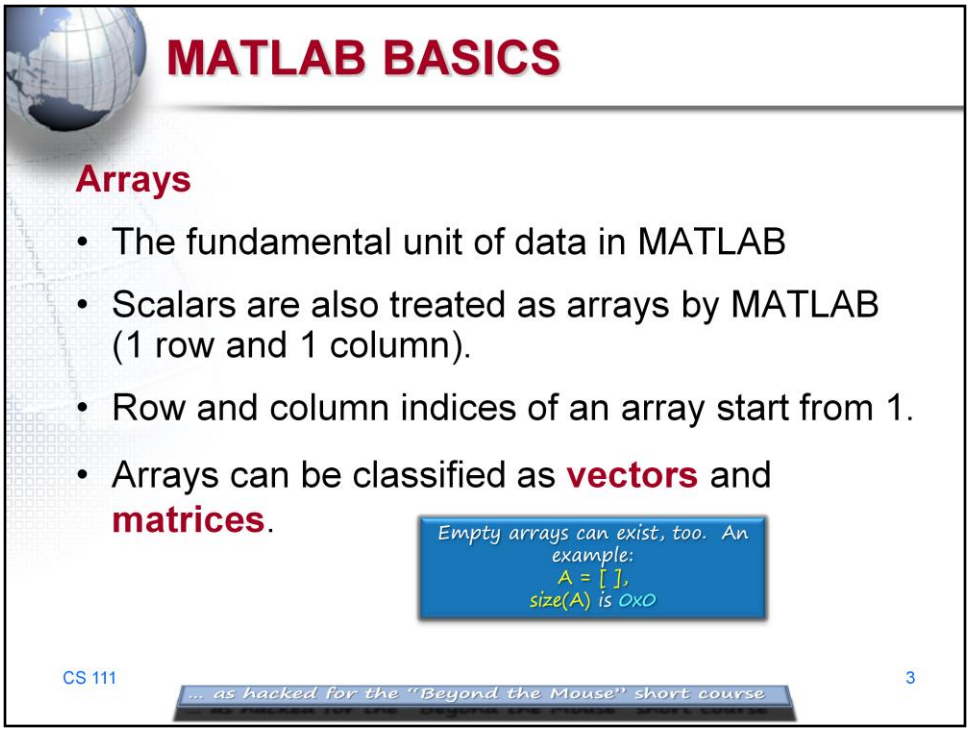
You'll find that the MATLAB help is more detailed and friendly than the MAN pages (unix help) with which you may be familiar.

• in each example, >> is the MATLAB prompt, type in the stuff that comes after it.

• comments in MATLAB start with the percent sign (%) and don't need to be typed in

Examples of arrays and their representation.

```
>> a = [1 2 3]
>> b = [1 2 3]'
>> c = [b (b * 2) ]
>> d = 'hello'
>> whos  %See the detail for these variables.
```



Examples of matrices of different sizes and dimensions

```
>> clear   % clear out all variables in memory
>> a = []  % empty array
>> b = 1  % scalar
>> c = [1 2; 3 4] %2x2 matrix
>> c (2,2)

>> whos
```

Notice, that the variable "ans" contains the answer from c(2,2)

What's the difference between [] and ()?

- The square brackets are used when creating an array
- parenthesis either groups operations (like in regular math), or references to locations within arrays.

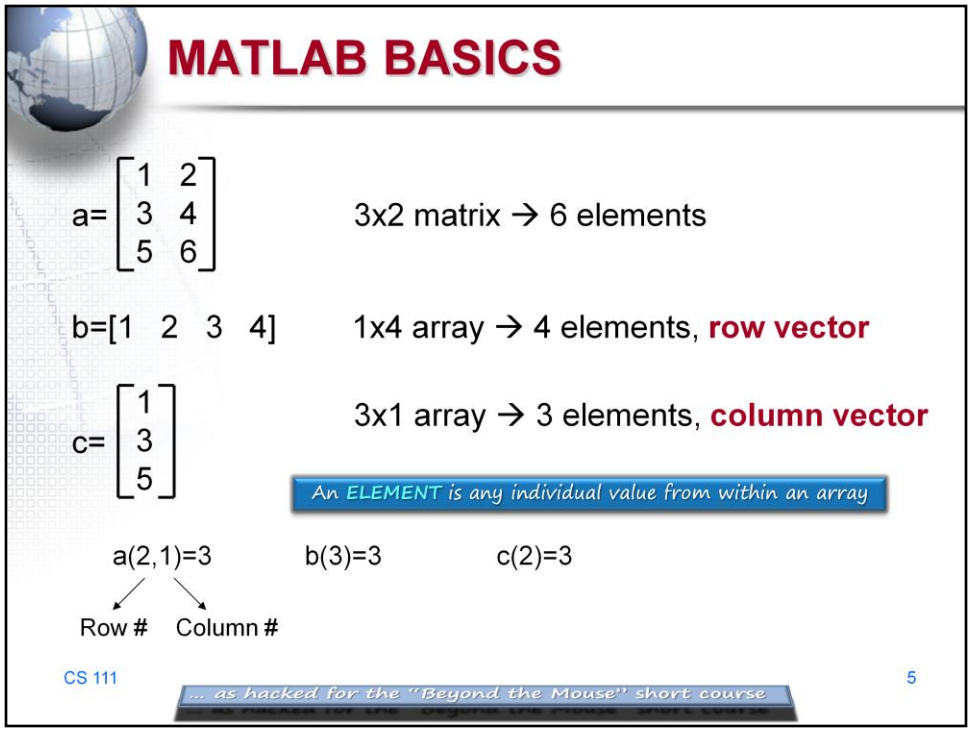
```
>> a(5)       % produces an error.  You're trying to access something out of bounds
>> a(5) = 5  % put "5" into the fifth position of the empty matrix a.
>>a  = [a 6]
```

MATLAB BASICS

- **Vector:** Array with one dimension
- **Matrix:** Array with more than one dimension
- **Size** of an array is specified by the number of rows and the number of columns, with the number of rows mentioned first (For example: n x m array).

  Total number of elements in an array is the product of the number of rows and the number of columns.

... Matlab can have N-dimensional arrays. So, instead of just being limited to X (n,m), you can have X(n,m,p,q,r,t,z). Not always recommended- but it can be done!



An example of accessing an element of an array.
>> d = magic(4) % creates a 4x4 magic-square matrix.
>> d(2,3) % should give you 10
>> d(10) % see explanation below

Even though there are multiple dimensions, you can access any element with a single index number. This is where it matters how MATLAB stores the values of its array.

Here's the array of interest, as displayed on the screen:
d=

```
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
```

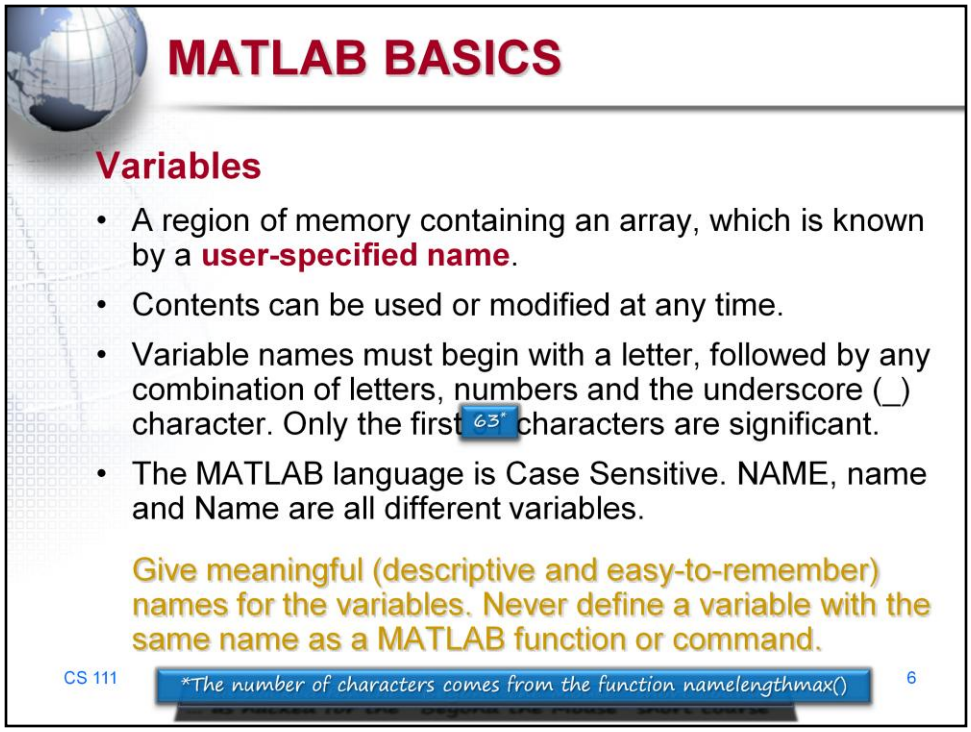
Internally, the numbers are stored like this…
16 5 9 4 2 11 7 14 3 10 6 15 13 8 12 1

So, when I access the 10th element, it turns out to be the number 10, just the same as if I accessed Row 2, Col 3.

All elements can be accessed either by their row-column position or through their continuous index.

>> for col = 1: size(d,2); for row = 1:size(d,1); disp(d(row,col)); end; end

>>for idx = 1 : numel(d); disp(d(idx)); end;



I've been using simple variable names (A, B, c, etc.) in the examples, but that is their values don't have any specific meaning.

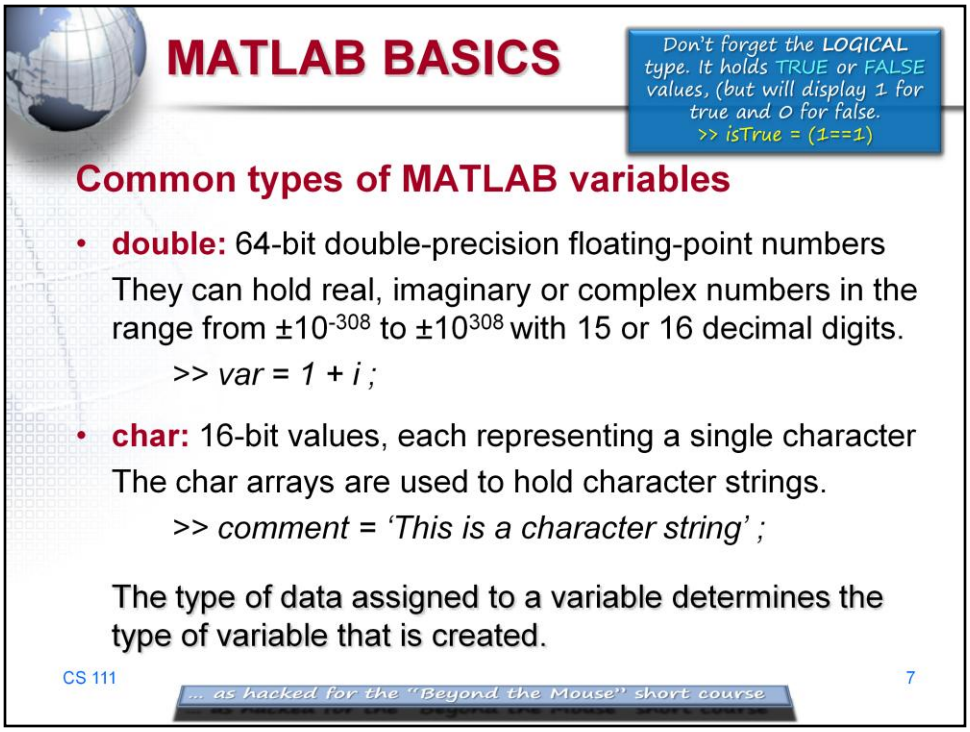
In a program where the variable actually MEANS something, it behooves you to give it a name that will mean something, too.

Which variable name would be the easiest to decipher later on??

- x
- s1
- sig1
--sigma1
-majorPrincipalStress

In some situations, a variable is created and then used right away – ten times in the very next equation, and then never seen again. Then, maybe, it makes sense to use a shorter version.

In general, the more abstract a variable name is, the fewer lines of code should use it. Geophysics, for example, has a notorious number of uses for "sigma", so you're only asking for trouble by declaring a sigma somewhere, and then hoping it will be understood (or have the expected value) a couple hundred lines of code later.

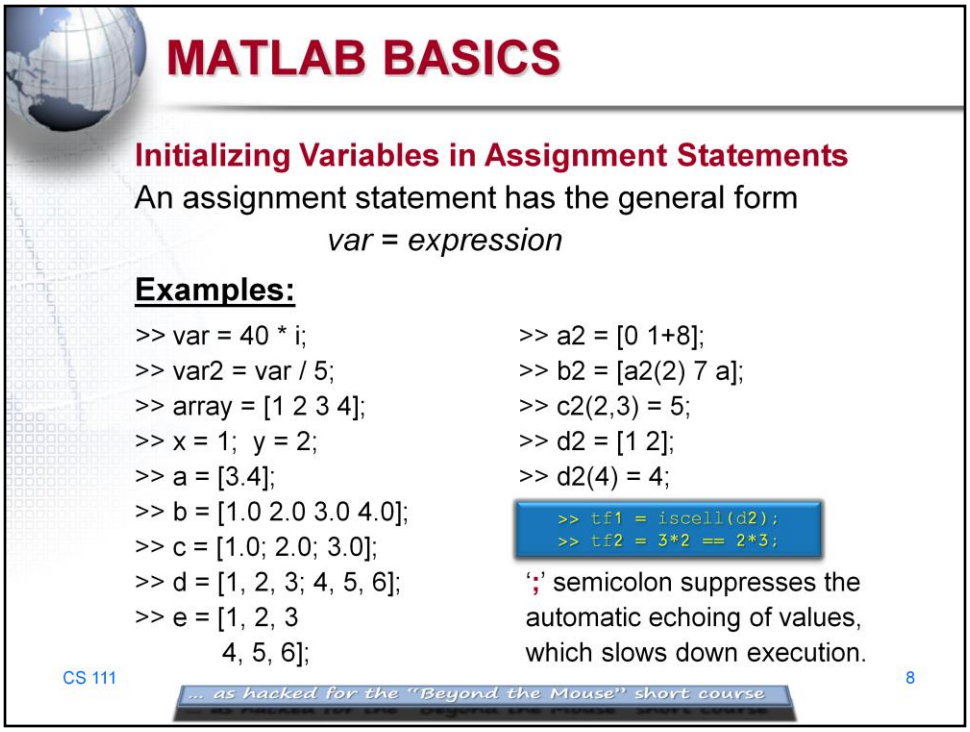


Examples of converting one type to another

```
>> clear
>> a = 75.3
>> b = int32(a)
>> d = char(a)
>> c = logical(a)
```

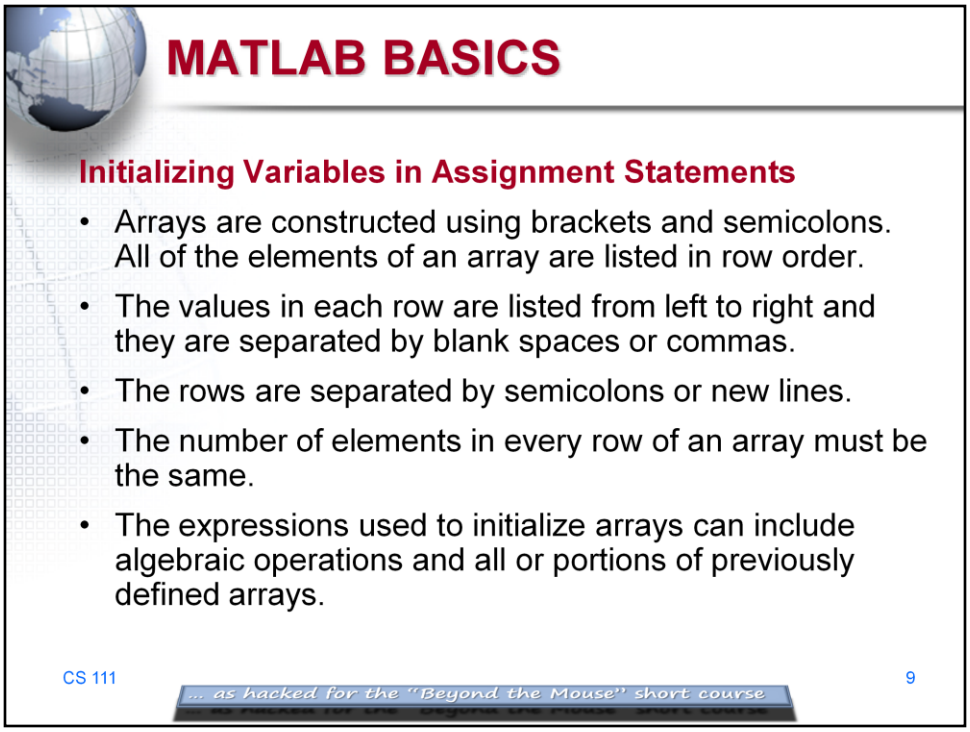
Matlab handles these pretty well… but how about these…

```
>> x = char(-1)      % characters are generally represented with values from 1 to 255…
>> x = logical(nan)
```



OK, so a semicolon actually has two uses:

1. At the end of a MATLAB command, it will suppress the echoing of values.
2. Within the [] of a matrix, where it says “what follows is on the next row”



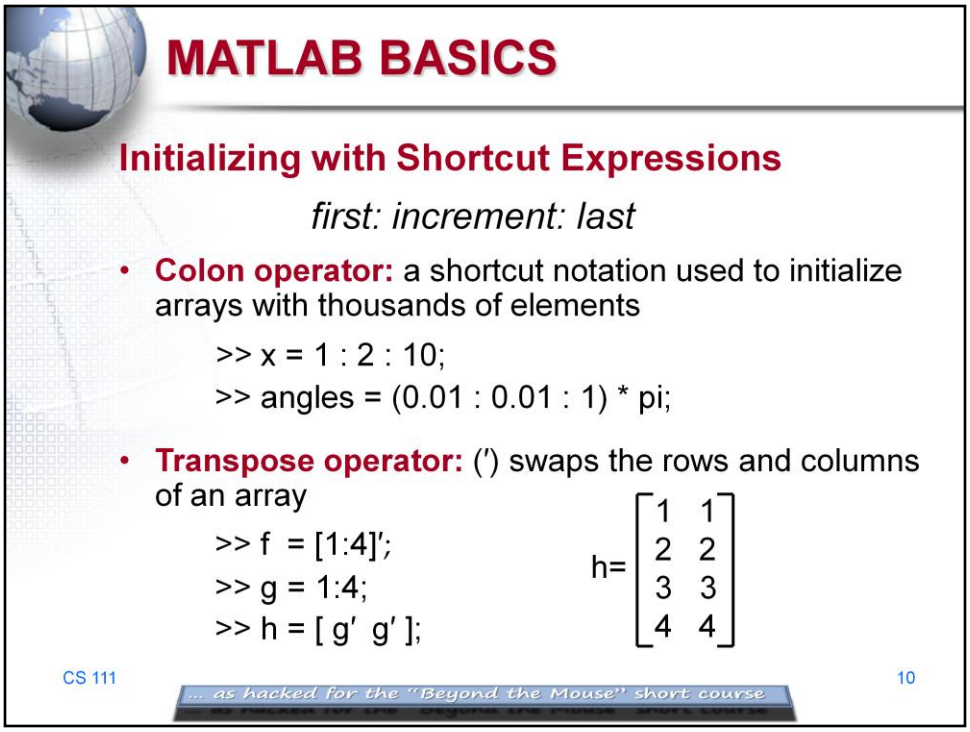
These are the same:

>> A = [1 2 3; 4 5 6]

And

>> A = [1 2 3
4 5 6]

(yes, that was supposed to be on two lines…)



These are the same:
>> 1 : 10
>> 1 : 1 : 10

You can go from high-number to low, too:
>> 10 : -1 : 1

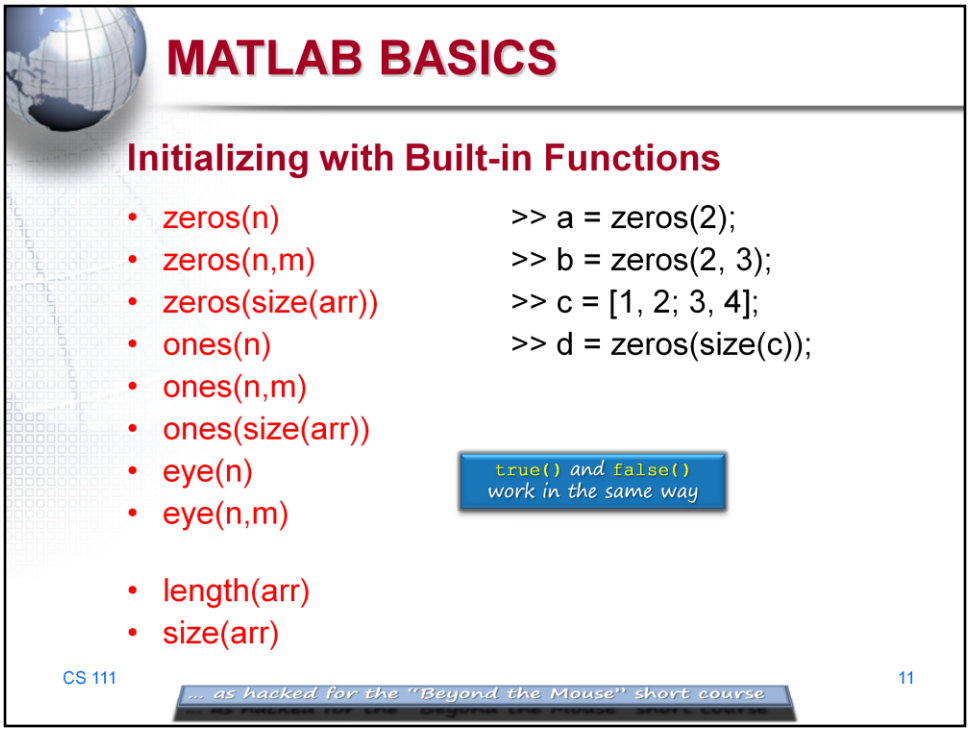
What does this one do?
>> 1 : -1 : 10

Also, you can force any array into a column by using the colon operator.
>>A = [1 2 3; 4 5 6]
>> A(:)

Or into a row with the colon and transpose operators…
>> A(:)'

>> B = 1 : 1e6 % one to a million.

oops. Forgot to suppress the output.
When you get tired of scrolling numbers, hit ctrl-C to stop it.
>> B = 1: 1e6; % the semicolon suppresses output, so this assignment is nearly instantaneous.



These functions all create arrays of values that should be self-explanatory, except for eye().

>> help eye

When you give each of these functions a single argument, such as ones(*n*), then MATLAB will create an *n* x *n* array.

MATLAB BASICS

Initializing with Keyboard Input

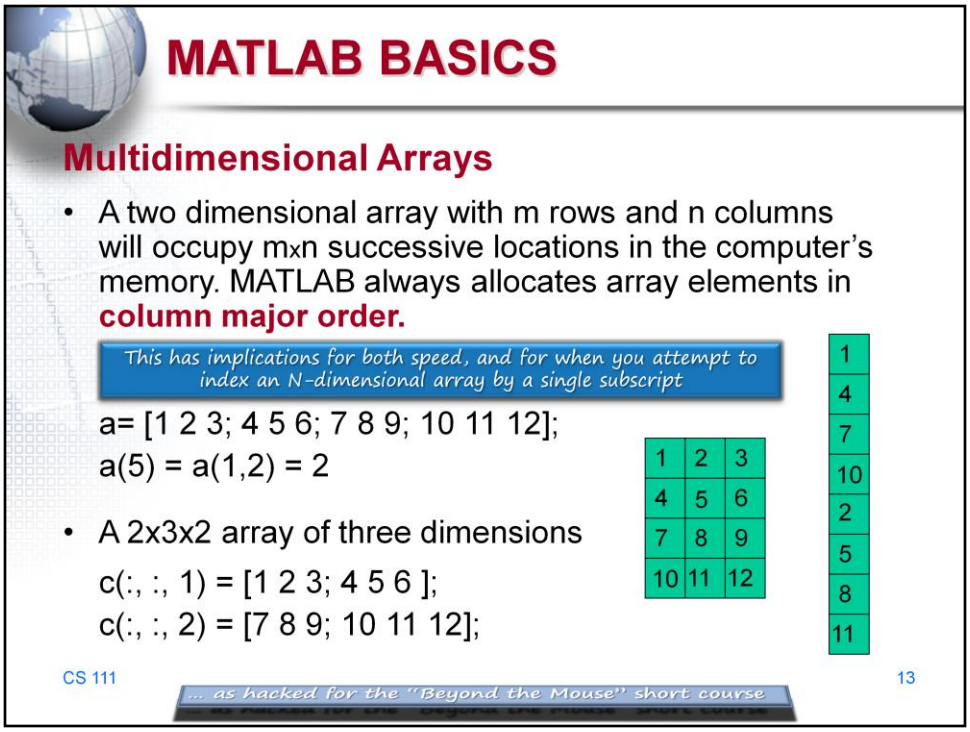
- The **input** function displays a prompt string in the Command Window and then waits for the user to respond.

```
my_val = input( 'Enter an input value: ' );

in1 = input( 'Enter data: ' );

in2 = input( 'Enter data: ' ,`s`);
```

This isn't very useful for batch files, though!



Using the example above:

>>C(:,:,1) = [1 2 3; 4 5 6];

>>C(:,:,2) = [7 8 9; 10 11 12];

>>disp(C)

>>disp(C(:));

What do these show?

>> size(C)

>> numel(C)

>> length(C)

MATLAB BASICS

Subarrays

- It is possible to select and use subsets of MATLAB arrays.

  arr1 = [1.1 -2.2 3.3 -4.4 5.5];

  arr1(3) is 3.3

  arr1([1 4]) is the array [1.1 -4.4]

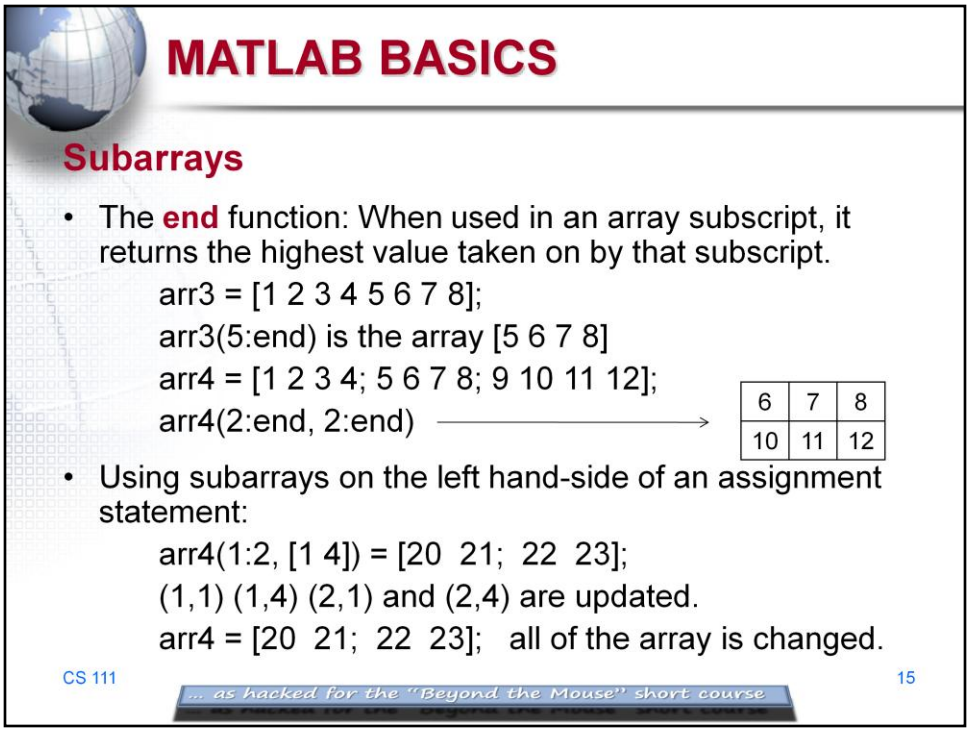
  arr1(1 : 2 : 5) is the array [1.1 3.3 5.5]

- For two-dimensional arrays, a colon can be used in a subscript to select all of the values of that subscript.

  arr2 = [1 2 3; -2 -3 -4; 3 4 5];

  arr2(1, :)

  arr2(:, 1:2:3)



Examples of **end** in use

>> A = magic(3)

>> A(end)

>> A (1, end)

Subtract one row from another

>>A(2,:) – A(1,:)

This will find the differences between the columns

>> A(:,2:end) – A(:,1:end-1)

MATLAB BASICS

Subarrays

- Assigning a Scalar to a Subarray: A scalar value on the right-hand side of an assignment statement is copied into every element specified on the left-hand side.

```
>> arr4 = [1 2 3 4; 5 6 7 8; 9 10 11 12];
>> arr4(1:2, 1:2) = 1
arr4 =
     1   1   3   4
     1   1   7   8
     9  10  11  12
```


... as hacked for the "Beyond the Mouse" short course

MATLAB BASICS

Special Values

- MATLAB includes a number of predefined special values. These values can be used at any time without initializing them.
- These predefined values are stored in ordinary variables. They can be overwritten or modified by a user.
- If a new value is assigned to one of these variables, then that new value will replace the default one in all later calculations.

```
>> circ1 = 2 * pi * 10;
>> pi = 3;
>> circ2 = 2 * pi * 10;
```

Don't change the value of functions, either! This can give WEIRD errors. Ex.

```
>> disp = 5;
>> disp('Hello'); %supposed to show"Hello"
??? Index exceeds matrix dimensions
```

Never change the values of predefined variables.

MATLAB BASICS

Special Values

Warning: Inf and NaN will screw up some calculations like "mean". Otherwise, NaN is great for representing missing data.

- pi: π value up to 15 significant digits
- i, j: sqrt(-1)
- Inf: infinity (such as division by 0)
- NaN: Not-a-Number (division of zero by zero)
- clock: current date and time in the form of a 6-element row vector containing the year, month, day, hour, minute, and second
- date: current date as a string such as *16-Feb-2004*
- eps: epsilon is the smallest difference between two numbers
- ans: stores the result of an expression

MATLAB BASICS

The next few slides are about Displaying stuff...

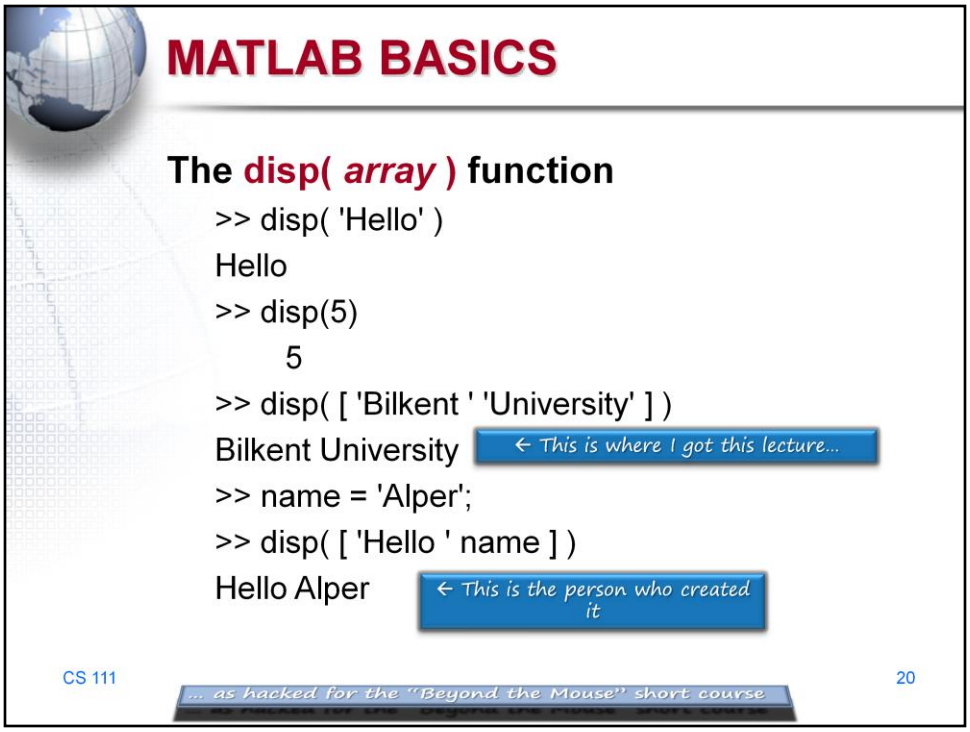
Changing the data format

>> value = 12.345678901234567;

...You'll See numbers displayed like This

After Typing This...

format short → 12.3457
format long → 12.34567890123457
format short e → 1.2346e+001
format long e → 1.234567890123457e+001
format short g → 12.346
format long g → 12.3456789012346
format rat → 1000/81

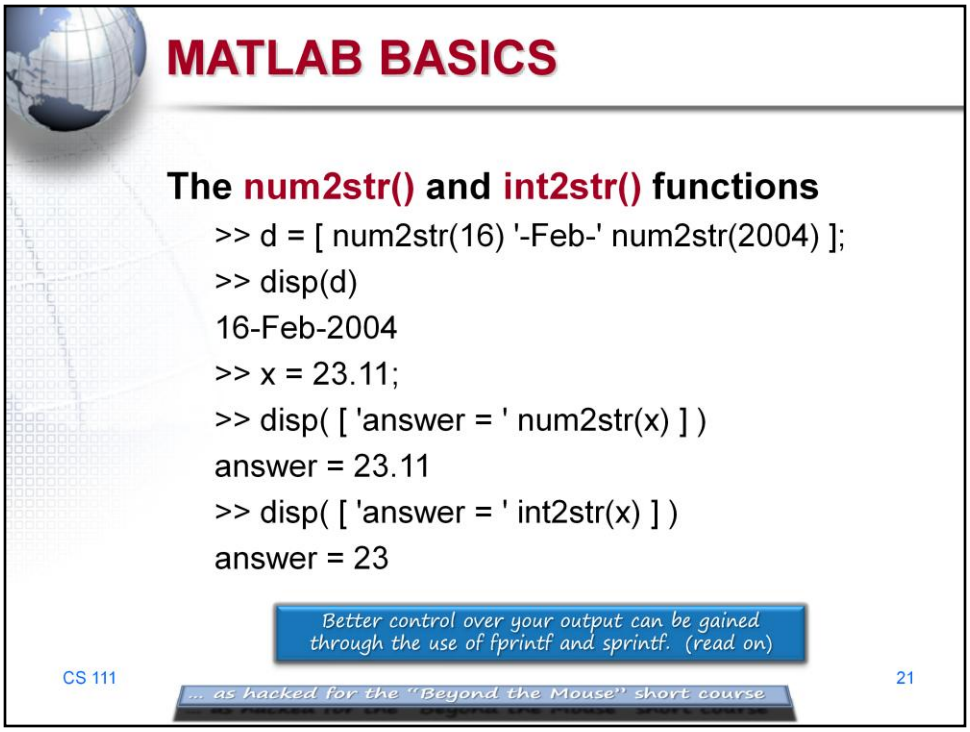


What's the difference between disp and display?

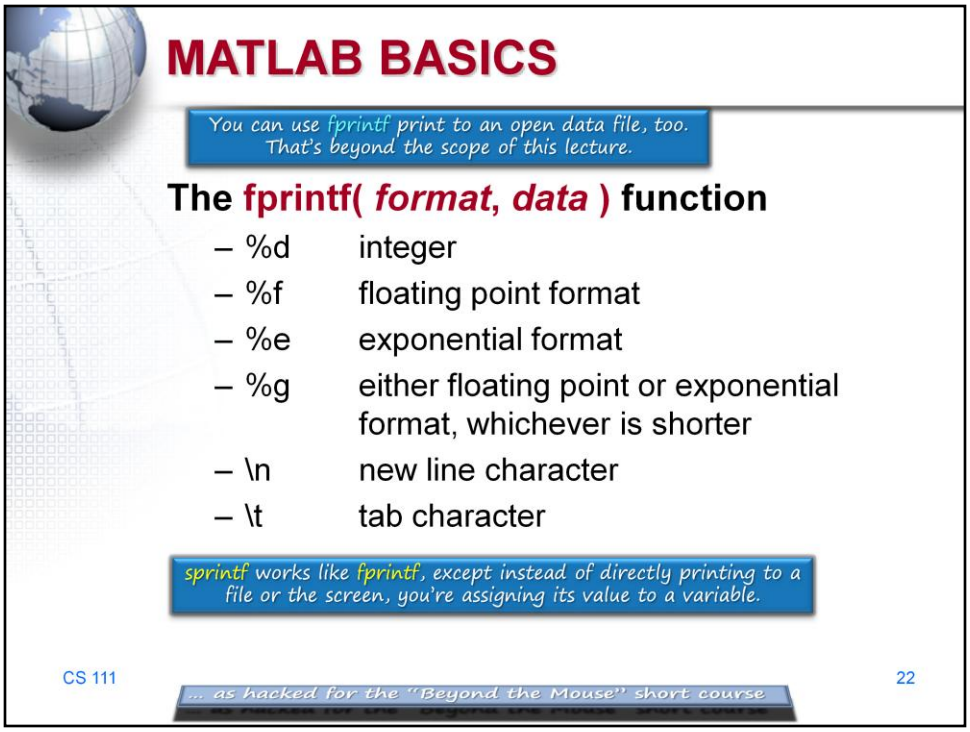
>> A = 'oy'

>> disp(A)

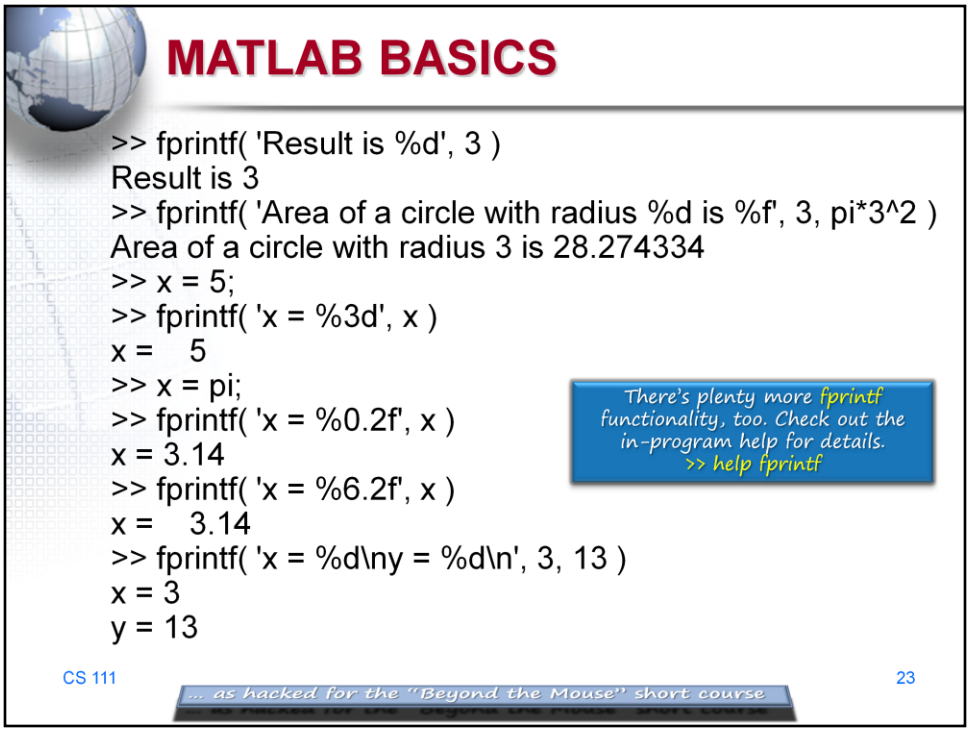
>> display(A)



>> lookfor 2str



The new line character is like pressing “return”; the subsequent characters start being displayed at the beginning of the next line below.



Additional examples, using sprintf

>> s = sprintf('Hello\nThere\n')

>> s = sprintf('#[%d], month [%02d], change[%+d]' , 5, 5, 5)

MATLAB BASICS

Data files

- save *filename var1 var2 …*

  >> save myfile.mat x y → binary

  >> save myfile.dat x –ascii → ascii

- load *filename*

  >> load myfile.mat → binary

  >> load myfile.dat –ascii → ascii

There's also uiload and uisave, which will bring up a dialog box for interactively choosing the file names & locations.
(UI, in this case, stands for user interface)

MATLAB BASICS

- *variable_name* = *expression*;
  - addition — a + b → a + b
  - subtraction — a - b → a - b
  - multiplication — a x b → a * b
  - division — a / b → a / b
  - exponent — a^b → a ^ b

Warning: since MATLAB is matrix based, these are matrix operations. If you want element-wise calculations, use .* ./ and .^ . The typical symptom will be an error something like:
???error using --> mpower. Matrix must be square
An examlpe of the difference is you'll get different answers for:
[5 6; 7 2] * [2 3; 2 1] and [5 6; 7 2] .* [2 3; 2 1]

... as hacked for the "Beyond the Mouse" short course

MATLAB BASICS

Hierarchy of operations

- x = 3 * 2 + 6 / 2
- Processing order of operations is important
  - parentheses (starting from the innermost)
  - exponentials (from left to right)
  - multiplications and divisions (from left to right)
  - additions and subtractions (from left to right)

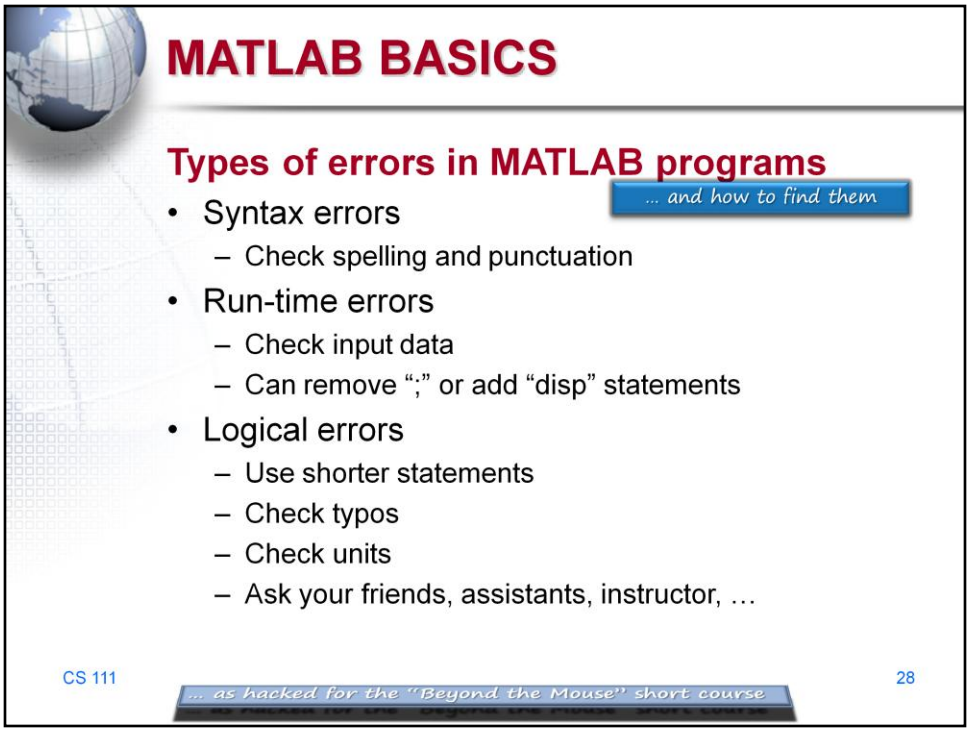
```
>> x = 3 * 2 + 6 / 2
x =
     9
```

MATLAB BASICS

Built-in MATLAB Functions

- *result = function_name(input)*;
  - abs, sign
  - log, log10, log2
  - exp
  - sqrt
  - sin, cos, tan
  - asin, acos, atan
  - max, min
  - round, floor, ceil, fix
  - mod, rem
- help elfun → help for elementary math functions

... as hacked for the "Beyond the Mouse" short course



Debugging could easily be a couple class sessions.

First of all, you can get a list of debugging instructions by typing:

>> help debug

If you're getting strange runtime errors, you could type:

>> dbstop if error

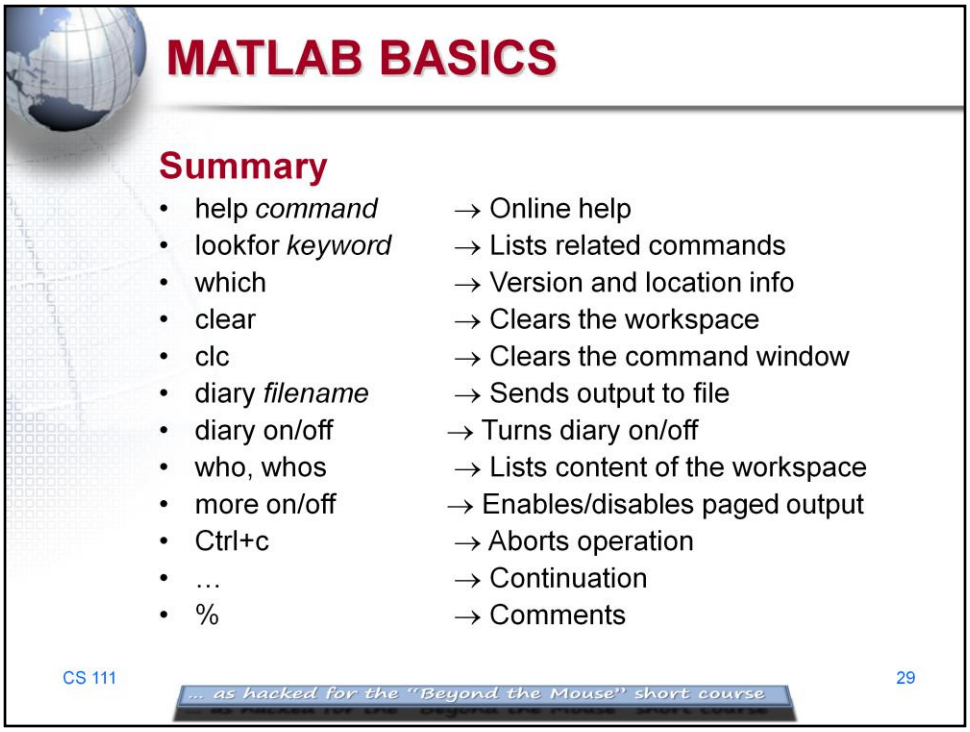
before running your program.

This will cause matlab to stop at any instruction that causes an error. A new debug prompt will show:

K>>

At this moment, you're seeing all variables with the values as they were right before the error happened. You can look to see if some variable is incorrect.

Debugging will be talked about more in the I/O class.



Interested in finding out what functions deal with, oh, distance, for example?

>> lookfor distance