MATLAB

Introduction

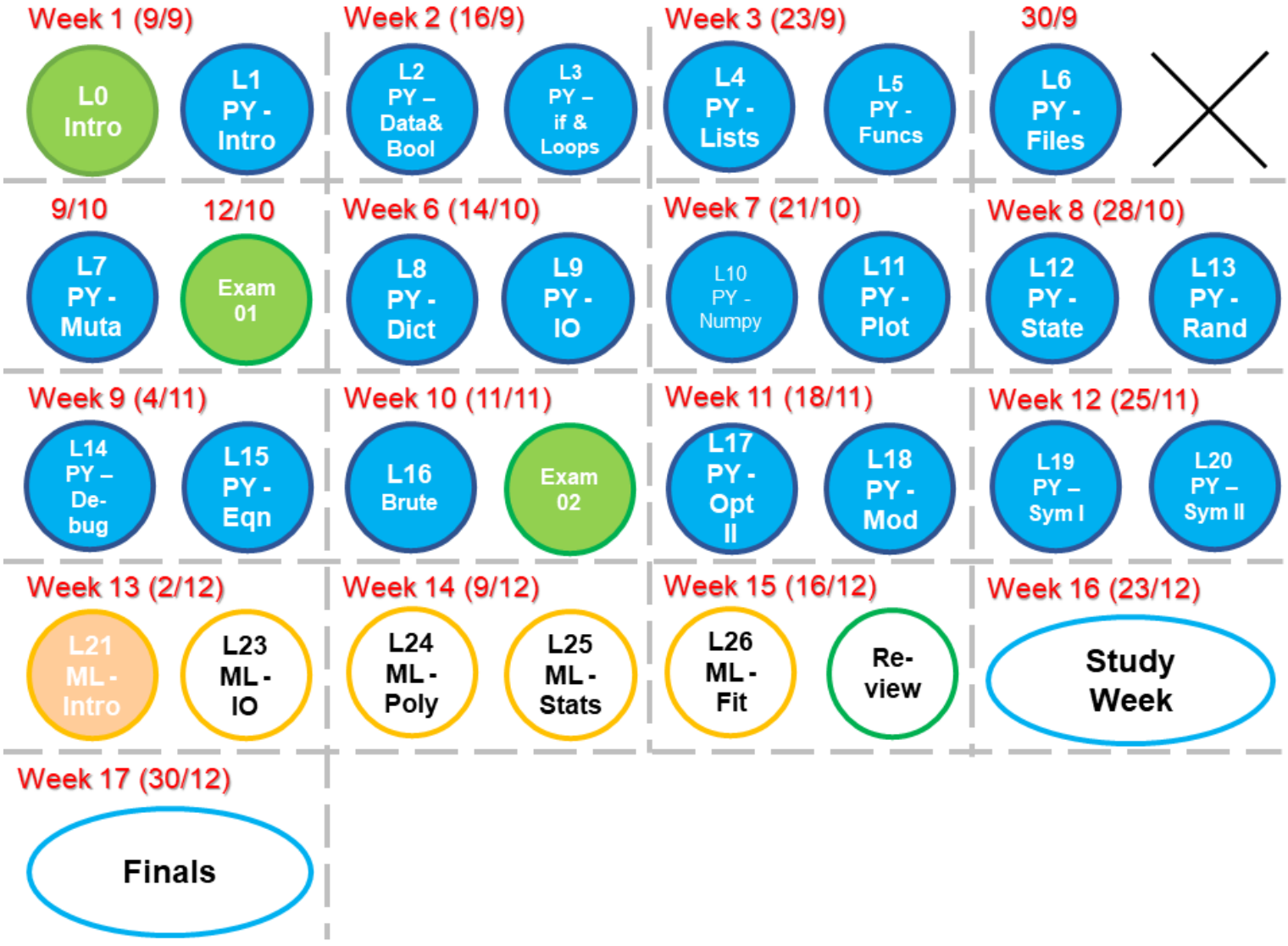
Announcements

quiz: `quiz21` due on Tues 03/12

lab: `lab` on Fri 06/12

hw: `hw11` due 04/12

Roadmap


Week 1 (9/9)
L0 Intro
L1 PY - Intro
Week 2 (16/9)
L2 PY – Data& Bool
L3 PY – if & Loops
Week 3 (23/9)
L4 PY - Lists
L5 PY - Funcs
30/9
L6 PY - Files
9/10
L7 PY - Muta
12/10
Exam 01
Week 6 (14/10)
L8 PY - Dict
L9 PY - IO
Week 7 (21/10)
L10 PY - Numpy
L11 PY - Plot
Week 8 (28/10)
L12 PY - State
L13 PY - Rand
Week 9 (4/11)
L14 PY – De-bug
L15 PY - Eqn
Week 10 (11/11)
L16 Brute
Exam 02
Week 11 (18/11)
L17 PY - Opt II
L18 PY - Mod
Week 12 (25/11)
L19 PY – Sym I
L20 PY – Sym II
Week 13 (2/12)
L21 ML - Intro
L23 ML - IO
Week 14 (9/12)
L24 ML - Poly
L25 ML - Stats
Week 15 (16/12)
L26 ML - Fit
Re-view
Week 16 (23/12)
Study Week
Week 17 (30/12)
Finals

Objectives

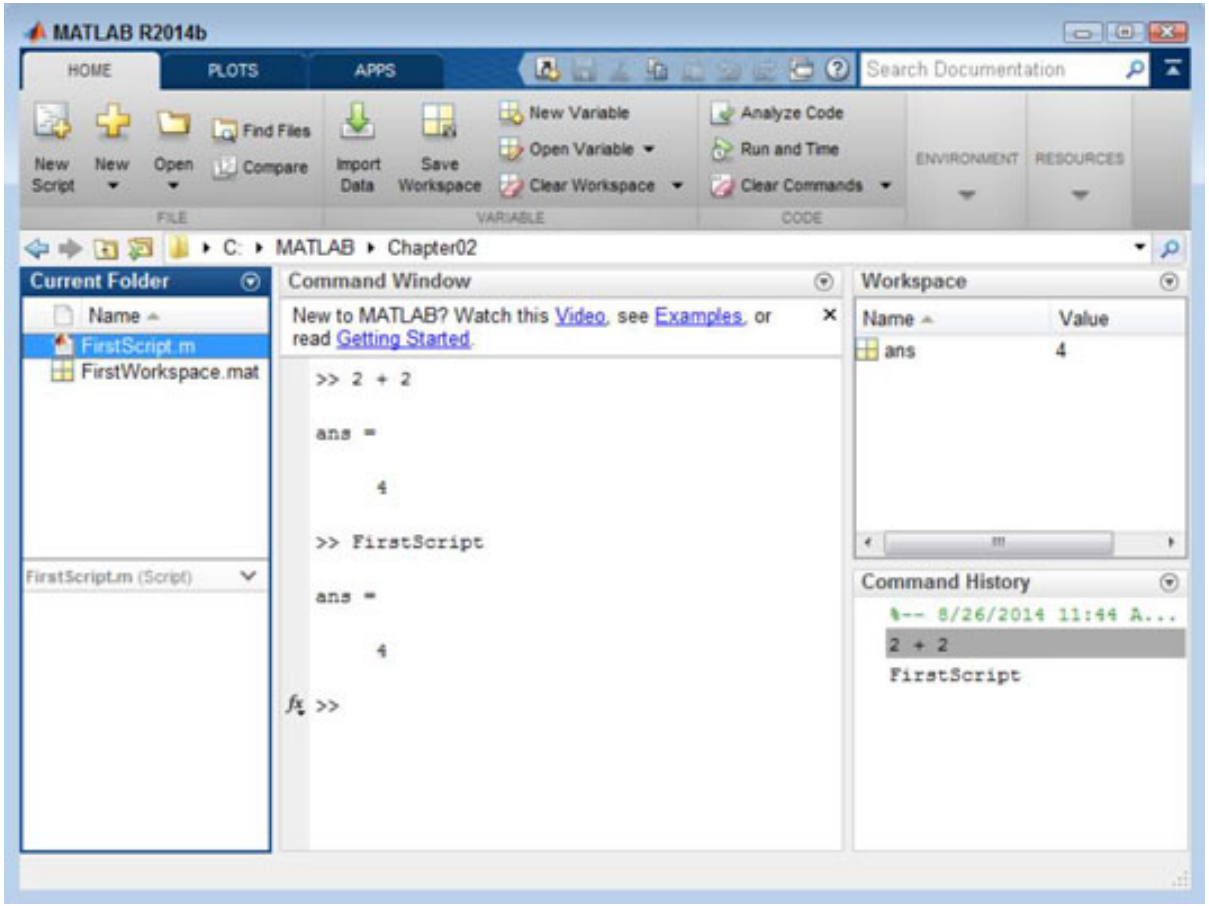
A. Explore the MATLAB user interface.
B. Index and slice arrays.
C. Compose basic functions.
D. Distinguish vector/elementwise and matrix operations.
E. Create basic loops (`for`/`while`).
F. Employ conditional logic (`if`/`else`/`end` statements).
G. Distinguish MATLAB `logical` values.
H. Utilize MATLAB-specific data types like `datetime`.


SS,DD,RH
single side
double density
MATLAB
Gatlinburg IX edition 1984
Please limit redistribution

MATLAB is MATrix LABoratory

Interface

Start MATLAB either at the command line, `matlab`, or by clicking the icon.



Why MATLAB?

Designed for engineering.

Excellent documentation and toolboxes.

Strong areas of application:

A. Linear algebra
B. Control dynamics
C. Numerical analysis
D. Image processing

Why MATLAB?

Can you do anything with it that you can’t do in Python?

Why MATLAB?

Can you do anything with it that you can't do in Python?

All programming languages can be made "equivalent"—so it depends on the libraries and applications, and the culture of your working group.

What is MATLAB?

Programming language + environment.

Proprietary, owned and maintained by MathWorks.

Dates from late 1970s, under active development.

Influenced NumPy/MatPlotLib, so will have familiar syntax.

Basics

Literals, variables, operators, comment

```
4 ^ 3   %what is this operator in Python?
```

Basics

Literals, variables, operators, comment

```
4 ^ 3   %what is this operator in Python?
```

Expressions

```
a = 3 * 2
b = 1 + a
```

Basics

Literals, variables, operators, comment

```
4 ^ 3    %what is this operator in Python?
```

Expressions

```
a = 3 * 2
b = 1 + a
```

Semicolon suppresses output (mutes): `;`

```
b = b + 2;
```

Basics

Literals, variables, operators, comment

```
4 ^ 3   %what is this operator in Python?
```

Expressions

```
a = 3 * 2
b = 1 + a
```

Semicolon suppresses output (mutes): `;`

```
b = b + 2;
```

`ans` is the default result variable.

```
a / 4
```

`fprintf` displays the value only.

```
fprintf( ans ); %if ans is a string
fprintf('%d', ans); %if ans is an integer
```

Numeric types

MATLAB implements:

A. integers
B. floating-point numbers
C. complex numbers

in 8-, 16-, 32-, and 64-bit versions (like NumPy).

`whos` shows type, value of all variables in workspace.

```
>> whos
  Name      Size              Bytes  Class       Attributes

  D         2x2                  32  double
  H         1x1                   8  double
  M         1x1                   8  double
  MI        1x1                   8  double
  S         1x1                   8  double
  Y         1x1                   8  double
  a         1x1                   8  double
  aa        1x11                 22  char
  ans       1x1                   8  double
  t         1x1                  17  datetime
```

Array types

Arrays are the fundamental type in MATLAB:

```
a = [ 1 2 3 ];
```

Arrays are indexed using parentheses:

```
b = a( 1 );    %what about in python?
```

Array types

Arrays are the fundamental type in MATLAB:

```
a = [ 1 2 3 ];
```

Arrays are indexed using parentheses:

```
b = a( 1 );    %what about in python?
```

MATLAB indexes from one, NOT zero!

Multidimensional arrays

More dimensional arrays use semicolons to separate rows:

```
A = [first row; second row; ...]
A = [ 1 2 3 ; 4 5 6 ];  %How about in Python?
```

Multidimensional arrays

More dimensional arrays use semicolons to separate rows:

```
A = [first row; second row; ...]
A = [ 1 2 3 ; 4 5 6 ];  %How about in Python?
```

Arrays are indexed using parentheses and commas:

```
a = A( 1,2 );
```

Helper functions are available:

```
B = ones( 3,3 ) + eye( 3,3 ) + zeros( 3,3 );
```

what is `eye`?

Question 1

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$

Which of the following will produce this array?

A. `[ 1 1 1 ] ; [ 2 2 2 ]`

B. `[ 1 1 1 ; 2 2 2 ]`

C. `[ 1 2 ] ; [ 1 2 ] ; [ 1 2 ]`

D. `[ 1 2 ; 1 2 ; 1 2 ]`

E. `[ [ 1 1 1 ] , [ 2 2 2 ] ]`

Question 1

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix}$$

Which of the following will produce this array?

A. `[ 1 1 1 ] ; [ 2 2 2 ]`

B. `[ 1 1 1 ; 2 2 2 ]` ★★★

C. `[ 1 2 ] ; [ 1 2 ] ; [ 1 2 ]`

D. `[ 1 2 ; 1 2 ; 1 2 ]`

E. `[ [ 1 1 1 ] , [ 2 2 2 ] ]`

Question 2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Which of the following will access `4` in this array?

A. `A( 1,0 )`

B. `A[ 2,1 ]`

C. `A( 2,1 )`

D. `A( 1 )( 0 )`

Question 2

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Which of the following will access `4` in this array?

A. `A( 1,0 )`

B. `A[ 2,1 ]`

C. `A( 2,1 )` ★★★

D. `A( 1 )( 0 )`

Array operations

Basic (scalar) mathematics:

```
A = ( ones( 3,3 ) + 1 ) / 2
B = sin( ones( 3,3 ) * pi )
C = B'   % transpose with '
```

or use ”dot” operator for `*` and `/`

```
A = ( ones( 3,3 ) + 1 ) ./ 2
B = sin( ones( 3,3 ) .* pi )
```

Array operations

Basic (scalar) mathematics:

```
A = ( ones( 3,3 ) + 1 ) / 2
B = sin( ones( 3,3 ) * pi )
C = B'  % transpose with '
```

or use ”dot” operator for `*` and `/`

```
A = ( ones( 3,3 ) + 1 ) ./ 2
B = sin( ones( 3,3 ) .* pi )
```

Matrix multiplication:

```
D = eye( 3,4 ) * ones( 4,5 ) * pi
```

Matrix vs element operations

"Matrix dimensions must agree for Matrix operations."

It is necessary to distinguish *elementwise* operations and *matrix* operations.

```
A = 2 * ones( 2,2 )   %same as in numpy
B = A .* eye( 2,2 )   %same as in numpy. Use * only
C = A  * eye( 2,2 )   %we never did this in numpy.
```

These are distinguished by a dot `.` in front of the operator.

Question 3

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Which of the following will produce this array?

A. `3*ones( 2,2 ) - 2*eye( 2,2 )`

B. `2*ones( 2,2 ) + eye( 2,2 )`

C. `3*ones( 2,2 ) - eye( 2,2 )`

D. `ones( 2,2 ) + eye( 2,2 )`

Question 3

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Which of the following will produce this array?

A. `3*ones( 2,2 ) - 2*eye( 2,2 )`

B. `2*ones( 2,2 ) + eye( 2,2 )`

C. `3*ones( 2,2 ) - eye( 2,2 )`

D. `ones( 2,2 ) + eye( 2,2 )` ⋆⋆⋆

Array operations

Concatenating arrays

```
A = [  eye( 3,4 ),  eye( 3,5 );
       ones( 2,4 ), ones( 2, 5) ]
```

what does this look like?

Question 4

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

How can we produce this array?

A. `[ [ 1 3 5 ] [ 2 4 6 ] ]`

B. `[ [ 1 2 ] [ 3 4 ] [ 5 6 ] ]`

C. `[ [ 1 3 5 ] ; [ 2 4 6 ] ]`

D. `[ [ 1 2 ] ; [ 3 4 ] ; [ 5 6 ] ]`

Question 4

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

How can we produce this array?

A. `[ [ 1 3 5 ] [ 2 4 6 ] ]`

B. `[ [ 1 2 ] [ 3 4 ] [ 5 6 ] ]`

C. `[ [ 1 3 5 ] ; [ 2 4 6 ] ]`

D. `[ [ 1 2 ] ; [ 3 4 ] ; [ 5 6 ] ]` ★★★

Scripting + Functions

MATLAB uses `.m` files for two purposes:

A. Scripts
B. Functions.

Comments are indicated as follows:

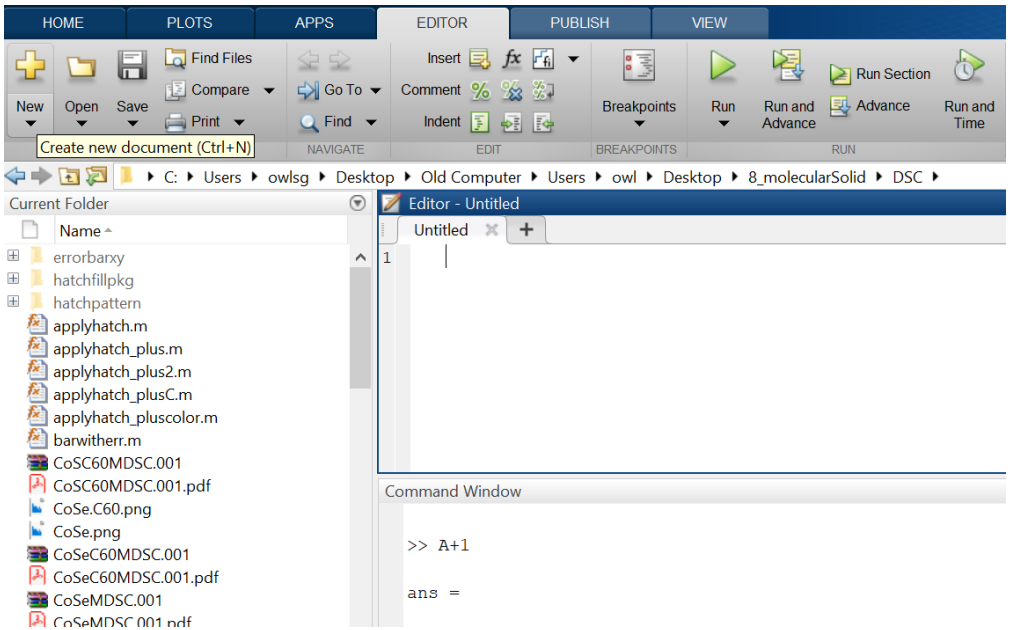
```
% this is a comment
%{
  this is a block comment
%}
```

Scripting

Scripts contain regular commands in order of execution.

Use the built-in editor to create these.

Make sure you have the correct working directory. Use `pwd` to see where you are



Functions

Functions must be located in a file of the same name as the function.

```
function [ output ] = function_name( input )
    % ... do what you want
end
```

No explicit `return` statements—rely on values in output variable list.

Best to indent for your eyes but not needed as MATLAB uses `end` to tell where to stop.

Functions

$$T_F = \frac{180}{100} T_C + 32$$

Filename should have the same as function name:

TempC2F.m

```
function [ Tf ] = TempC2F( Tc )
    Tf = Tc * ( 180/100 ) + 32;
end
```

You can have more than one functions in the same .m file but only the first function can be access from outside!

Functions

$$T_F = \frac{180}{100} T_C + 32$$

Filename should have the same as function name:

TempC2F.m

```
function [ Tf ] = TempC2F( Tc )
    Tf = Tc * ( 180/100 ) + 32;
end
```

You can have more than one functions in the same .m file but only the first function can be access from outside!

This first function will be called using the name of the .m file. If other rest of functions have the same name as the .m file, MATLAB will complain!

Strings

Single quotes creates `char array`

Double quotes creates `string`

```
s = 'I know this'
bigS = "Different? What? Confused!"
```

Strings

Single quotes creates `char array`

Double quotes creates `string`

```
s = 'I know this'
bigS = "Different? What? Confused!"
```

`s(1)` shows `'I'`

`bigS(1)` shows `"Different? What? Confused!"`

Strings

Single quotes creates `char array`

Double quotes creates `string`

```
s = 'I know this'
bigS = "Different? What? Confused!"
```

`s(1)` shows `'I'`
`bigS(1)` shows `"Different? What? Confused!"`
`bigS{1}` shows `'Different? What? Confused!'`

Strings

Single quotes creates `char array`

Double quotes creates `string`

```
s = 'I know this'
bigS = "Different? What? Confused!"
```

`s(1)` shows `'I'`
`bigS(1)` shows `"Different? What? Confused!"`
`bigS{1}` shows `'Different? What? Confused!'`
then `bigS{1}(1:2)` for `'Di'`

Print formatted strings with `fprintf`:

```
fprintf( '%f %f', sin(pi/3), cos(pi/4) );
```

loop

`for` statement

We create a `for` loop as follows:

- statement `for var = range`, where you create `var`
- and provide `range`
- one or more statements
- closing statement `end`

Also have `continue` and `break` available.

`for` statement

```
%% loop through time steps
for i = 1:2:10
    fprintf( 'The number is %i.' , i );
end
```

`for` statement

The `for` loop ranges over a set of possible values.

`for` statement

The `for` loop ranges over a set of possible values.

This is *not* as flexible as Python's `for ... in ...:` syntax—think of always having to loop over the *index* rather than the item.

Ranges are straightforward: `1:10`, `1:2:10`, `0.1:0.1:0.5`. Also have `linspace` available.

NOTE: 1:1:10 *NOT* the same as linspace(1, 1, 10)

`for` statement

The `for` loop ranges over a set of possible values.

This is *not* as flexible as Python's `for ... in ...:` syntax—think of always having to loop over the *index* rather than the item.

Ranges are straightforward: `1:10`, `1:2:10`, `0.1:0.1:0.5`. Also have `linspace` available.

NOTE: 1:1:10 *NOT* the same as linspace(1, 1, 10)

1:1:10 has same output as linspace(1, 10, 10)

while loop

```
%% loop until condition is met
i = 0;
while i < 10
    i = i + 1;
    fprintf( 'The number is %i.' , i );
end
```

if and logic

`if`/`else` statement

We create an `if`/`else` statement as follows:

- the keyword `if`
- a logical comparison (more on these!)
- a **block** of code

`if`/`else` statement

We create an `if`/`else` statement as follows:

- the keyword `if`
- a logical comparison (more on these!)
- a **block** of code
- the keyword `elseif` (note this!)
- a new logical comparison
- a different **block** of code

`if`/`else` statement

We create an `if`/`else` statement as follows:

- the keyword `if`
- a logical comparison (more on these!)
- a **block** of code
- the keyword `elseif` (note this!)
- a new logical comparison
- a different **block** of code
- the keyword `else`
- a different **block** of code

`if`/`else` statement

We create an `if`/`else` statement as follows:

- the keyword `if`
- a logical comparison (more on these!)
- a **block** of code
- the keyword `elseif` (note this!)
- a new logical comparison
- a different **block** of code
- the keyword `else`
- a different **block** of code
- the keyword `end`

`if/else` example

```
if nargin < 3
    xOrder = 1:size(values,1);
elseif nargin < 6
    if isscalar(varargin{2}) || ischar(varargin{2})
        xOrder = 1:size(values,1);
        yOder = 'ok'
    else
        [tmp xOrder] = sort(varargin{1});
    end
else
    fprintf('Error')
end
```

Logical statements

MATLAB does *NOT* have a `bool` data type.

Logical statements

MATLAB does *NOT* have a `bool` data type.

Instead of `True`/`False`, MATLAB uses integers:

- `0` means `false`
- `1` means `true`
- recognises `false` and `true`. Does not give error but stores as `0` and `1`
- `logical` data type

Logical Operators

Available `logical` operators include:

`<`, `>`, `<=`, `>=`, `==`, `~=`

`&&` means ‘and’, `||` means ‘or’

`ismember` checks if in arrays.

```
B = [ 1 2 3 4 5 ];
ismember( 5,B )
```

Logical Operators

Available `logical` operators include:

`<`, `>`, `<=`, `>=`, `==`, `~=`

`&&` means ‘and’, `||` means ‘or’

`ismember` checks if in arrays.

```
B = [ 1 2 3 4 5 ];
ismember( 5,B )
ans = 1
```

Also, logical operators work as indices!

Logical Operators

Available `logical` operators include:

`<`, `>`, `<=`, `>=`, `==`, `~=`

`&&` means ‘and’, `||` means ‘or’

`ismember` checks if in arrays.

```
B = [ 1 2 3 4 5 ];
ismember( 5,B )
ans = 1
```

Also, logical operators work as indices!

```
A( A>2 )
```

Logical Operators

`A( A>2 )` what is the ans?

Logical Operators

`A( A>2 )` what is the ans?

» A = [1, 5, 1; 3, 6, 2]

```
A =

     1     5     1
     3     6     2
```

Logical Operators

`A( A>2 )` what is the ans?

» A = [1, 5, 1; 3, 6, 2]

```
A =
     1     5     1
     3     6     2
```

» A>2

```
ans =
  2×3 logical array
   0   1   0
   1   1   0
```

Logical Operators

`A( A>2 )` what is the ans?

» A = [1, 5, 1; 3, 6, 2]

```
A =
     1     5     1
     3     6     2
```

» A>2

```
ans =
  2×3 logical array
   0   1   0
   1   1   0
```

» A(A>2)

```
ans =
   3
   5
   6
```

Random Numbers

Random numbers

MA supports many varieties of Random Number Generator:

A. `rand`, uniform distribution $(0, 1)$
B. `randn`, normal distribution
C. `randi`, random integers $[1, n]$

Note: what is the starting value if not given? Remember if both ends are included?

Note that these commands are quite different from thsPython!

rand

```
rand( 5 );        % generate 5x5 matrix
rand( 5, 1 );      % generate 5x1 column vector
```

rand

```
rand( 5 );        % generate 5x5 matrix
rand( 5, 1 );      % generate 5x1 column vector

10 * rand( 3 ); % 3x3 matrix from (0,10)
```

randi

```
randi( 5 );        % generate number from [1,5]
randi( 5, 2 );      % generate 2x2 matrix
```

randi

```
randi( 5 );        % generate number from [1,5]
randi( 5, 2 );      % generate 2x2 matrix

randi( [ -5, 5 ], 10, 1 ); % from [-5,5] in 10 x 1
randi( [ -5, 5 ], [10, 1] ); % same as above
randi( [ -5 5 ], [10 1] ); % same as above
```

randn

```
randn();            % single normal number
randn( 5 );         % generate 5x5 matrix
randn( 5, 2 );       % generate 5x2 matrix
randn( 5, 2 )*10 + 3;     % generate 5x2 matrix
```

How is the last one different from the second last one?

Example: Seed

```
rng( 1 );
x = linspace( 0, 2*pi,101 )';
y = sin( x/50 ) ./ x + .002 * randn( 101,1 );
```

datetime type

Dates and times can usefully be stored as values:

```
» t = datetime( Y,M,D,H,MI,S );
%assume Y,M,D.. already defined

» t = datetime(
'now','TimeZone','local','Format','d-MMM-y
HH:mm:ss Z' );

» t = datetime(
'2017-12-01','InputFormat','yyyy-MM-dd' );
```

Summary

A. Like NumPy, but no `import`s (anywhere).
B. Remember to change: parentheses, indexing from 1, `end` keywords.
C. Hard to do `dict`-like things, easy to do `numpy`-like operations.