COGS 119/219
MATLAB for Experimental Research

Winter 2016 – Week 2
Built-in array functions, Data types
.m files, begin Flow Control
# Built-in array functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| mean(A)   | If A is a vector, returns the mean value of the elements of the vector | `>> A = [5 9 2 4];
            >> mean(A)
            ans =
            5` |
| std(A)    | If A is a vector, returns the standard deviation of the elements of the vector | `>> A = [5 9 2 4];
            >> std(A)
            ans =
            2.9439` |
| sum(A)    | If A is vector, returns the sum of the elements of the vector | `>> A = [5 9 2 4];
            >> sum(A)
            ans =
            20` |
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C = \text{max}(A)$</td>
<td>If $A$ is a vector, $C$ is the largest element in $A$.</td>
<td>$\gg C = [5 \ 9 \ 2 \ 4 \ 1 \ 1 \ 6]$; $\gg C = \text{max}(A)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ans = 11</td>
</tr>
<tr>
<td></td>
<td>If $A$ is a matrix, $C$ is a row vector containing the largest element of each column of $A$.</td>
<td>$\gg [d,n] = \text{max}(A)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$d = 11$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$n = 5$</td>
</tr>
</tbody>
</table>
## Built-in array functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>min(A)</td>
<td>The same as max(A), but for the smallest element.</td>
<td><code>&gt;&gt; A = [5 9 2 4];</code>&lt;br&gt;<code>&gt;&gt; min(A)</code>&lt;br&gt;<code>ans = 2</code></td>
</tr>
<tr>
<td>[d,n] = min(A)</td>
<td>The same as [d,n] = max(A), but for the smallest element.</td>
<td><code>&gt;&gt; [d,n] = min(A)</code>&lt;br&gt;<code>d = 2</code>&lt;br&gt;<code>n = 3</code></td>
</tr>
</tbody>
</table>
Built-in array functions

See help of these functions for various useful ways of using them.

size          flipud
all            fliplr
prod           rot90
any            repmat

Try:
>> mymat = eye(3) * diag([4 5 7]) + 2;
>> mybigmat = repmat(mymat, [3 6]);
.m files

- We can write the MATLAB commands that we type at the command window in a file: extension .m
.m files

```matlab
myarray = zeros(3) + 2;
myarray2 = round(rand(3)*10);
myarray3 = myarray + myarray2;
myarray4 = [1, 2, 3];
newarray = myarray .* myarray3;
newarray2 = myarray .* myarray3;
newarray2 = myarray4 .* myarray;
newarray3 = myarray .* myarray3;
newarray3 = myarray .* myarray3;
```
.m files
% This is a comment
% Explanation of the script goes here...

myarray = zeros(3)+2;
myarray2 = round(rand(3)*10);
myarray3 = myarray + myarray2;
myarray4 = [1, 2, 3];
newarray = myarray .* myarray3;
newarray2 = myarray .* myarray3;
newarray2 = myarray4 * myarray;
newarray3 = myarray .* myarray3;
newarray3 = myarray .* myarray3;
Data types

>> help datatypes
Data types and structures.

Data types (classes)

double
  - Convert to double precision.
logical
  - Convert numeric values to logical.
cell
  - Create cell array.
struct
  - Create or convert to structure array.
single
  - Convert to single precision.
uint8
  - Convert to unsigned 8-bit integer.
uint16
  - Convert to unsigned 16-bit integer.
uint32
  - Convert to unsigned 32-bit integer.
uint64
  - Convert to unsigned 64-bit integer.
int8
  - Convert to signed 8-bit integer.
int16
  - Convert to signed 16-bit integer.
int32
  - Convert to signed 32-bit integer.
int64
  - Convert to signed 64-bit integer.
inline
  - Construct INLINE object.
function handle
  - Function handle array.
javaArray
  - Construct a Java Array object.
javaMethod
  - Invoke a Java method.
javaObject
  - Invoke a Java object constructor.
javaMethodEDT
  - Invoke a Java method on the Swing Event Dispatch Thread.
javaObjectEDT
  - Invoke a Java object constructor on the Swing Event Dispatch
Data types: Examples

```matlab
>> a = 1;
>> b = 1.0;
>> whos
Name      Size      Bytes  Class       Attributes
  a      1x1        8  double
  b      1x1        8  double

>> c = round(b);
>> d = uint8(b);
>> whos
Name      Size      Bytes  Class       Attributes
  a      1x1        8  double
  b      1x1        8  double
  c      1x1        8  double
  d      1x1        1  uint8
```
Data types: Examples

```plaintext
>> e = true;
>> whos
    Name      Size Bytes Class Attributes
    a       1x1    8  double          
    b       1x1    8  double          
    c       1x1    8  double          
    d       1x1    1  uint8         
    e       1x1    1  logical        

>> f = 'true';
>> f2 = 'cool';
>> g = 'l';
>> h = 'this is easy!';
>> whos
    Name      Size   Bytes Class Attributes
    a       1x1      8  double          
    b       1x1      8  double          
    c       1x1      8  double          
    d       1x1      1  uint8         
    e       1x1      1  logical        
    f       1x4      8  char           
    f2      1x4      8  char           
    g       1x1      2  char           
    h       1x13     26 char         
```
Data types: Examples

>> f + f2
ans =
    215  225  228  209

>> h1 = int8(f2)
h1 =
    99  111  111  108

>> f2 = f + g
f2 =
    165  163  166  150
## Data types: Examples

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>ans</td>
<td>1x4</td>
<td>32</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>1x1</td>
<td>1</td>
<td>uint8</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>1x1</td>
<td>1</td>
<td>logical</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>1x4</td>
<td>8</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>f2</td>
<td>1x4</td>
<td>32</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>1x1</td>
<td>2</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>1x13</td>
<td>26</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>h1</td>
<td>1x4</td>
<td>4</td>
<td>int8</td>
<td></td>
</tr>
</tbody>
</table>

```
>> whos
Name      Size     Bytes   Class     Attributes
a        1x1       8      double    
an        1x4       32     double    
b        1x1       8      double    
c        1x1       8      double    
d        1x1       1      uint8    
e        1x1       1      logical  
f        1x4       8      char     
f2       1x4       32     double    
g        1x1       2      char     
h        1x13      26     char     
h1       1x4       4      int8     
```

```
>> f2(2)
an =
   163

>> f(2)
an =
r
```

```
>> f(1,1)
an =
t
>> f(1)
an =
t
```
Strings

>> first = 'Matlab is so';
>> last = 'cool';
>> full = first + last
Error using +
Matrix dimensions must agree.

>> full = [first last]
full =
Matlab is so cool

>> full2 = [first '_' last]
full2 =
Matlab is so_cool

>> full3 = [first '' last]
full3 =
Matlab is socool

>> full4 = [first ' ' last]
full4 =
Matlab is so cool
Strings

>> full5 = [first 'bloody' last]
full5 =
Matlab is sobloodycool

>> full6 = [first 'bloody' last]
full6 =
Matlab is so bloody cool

>> full7 = [first "bloody" last]
full7 =
Matlab is so "bloody" cool
Strings

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>1x12</td>
<td>24</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>full</td>
<td>1x16</td>
<td>32</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>full2</td>
<td>1x17</td>
<td>34</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>full3</td>
<td>1x16</td>
<td>32</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>full4</td>
<td>1x17</td>
<td>34</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>full5</td>
<td>1x22</td>
<td>44</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>full6</td>
<td>1x24</td>
<td>48</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>full7</td>
<td>1x26</td>
<td>52</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>full8</td>
<td>17x1</td>
<td>34</td>
<td>char</td>
<td></td>
</tr>
<tr>
<td>last</td>
<td>1x4</td>
<td>8</td>
<td>char</td>
<td></td>
</tr>
</tbody>
</table>
FLOW CONTROL

• In a simple program, the commands are executed one after the other in the order they are typed.

• Many situations require more sophisticated programs in which commands are not necessarily executed in the order they are typed.

• Matlab provides several tools:
  1. Conditional statements (if-else)
  2. Loops (for and while loops)
A conditional statement is a command that allows MATLAB to make a decision of whether to execute a group of commands that follow the conditional statement, or to skip these commands.

A conditional statement can be in three forms:

- if – end
- if – else – end
- if – elseif – else - end
If the conditional expression is true, the program continues to execute the commands that follow the if statement.

If the conditional expression is false, the program skips the commands between if and end, and continues to execute the commands that follow end.
Conditional expressions

- Conditional expressions consist of relational and/or logical operators.

<table>
<thead>
<tr>
<th>Relational Operators</th>
<th>Logical Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&amp; (AND)</td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;=</td>
<td>~ (NOT)</td>
</tr>
<tr>
<td>&gt;=</td>
<td></td>
</tr>
<tr>
<td>==</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td></td>
</tr>
<tr>
<td>&lt;=</td>
<td></td>
</tr>
</tbody>
</table>
if-end statement

clear all;
% x = 5;
% y = -3;
% disp ('enter an integer');
x = input ('enter an integer: ');
s = ''; 

if x > 0
    s = 'x is a positive number';
end

if x < 0 | x ==0
    s = 'x is not a positive number'
end

disp (s);
If the conditional expression is true, the program executes group 1 of the commands between `if` and the `else` statements and then skips to the `end`.

If the conditional expression is false, the program skips to the `else`, and then executes group 2 of commands between the `else` and the `end`.
if-else-end statement

clear all;
% x = 5;
% y = -3;
% disp ('enter an integer');
x = input ('enter an integer: ');
s = "; 

if x > 0
    s = 'x is a positive number';
else
    s = 'x is not a positive number';
end 

disp (s);
**if-elseif-else-end statement**

If the conditional expression is true, the program executes group 1 commands between the `if` and the `elseif` statement, and then skips to the `end`.

If the conditional expression in the `if` statement is false, the program skips to the `elseif` statement. If the conditional expression in the `elseif` statement is true, the program executes group 2 of commands between the `elseif` and the `else` and then skips to the `end`. 
If the conditional expression in the `elseif` statement is false, the program skips to the `else` and executes group 3 of commands between the `else` and the `end`. 
if-elseif-else-end statement

clear all;
\% x = 5;
\% y = -3;
\% disp ('enter an integer');
x = input ('enter an integer: ');
s = ";

if x > 0
    s = 'x is a positive number';
elseif x == 0
    s = 'x is neither positive nor negative';
else
    s = 'x is a negative number';
end

disp (s);
Comparing strings

- If the conditional expression compares string variables, then use `strcmp` instead of `==` or `!=

```plaintext
word = 'hello';
if strcmp(word, 'hello')
    display(word);
else
    display('The word is not hello.'
end
```

`>>help strcmp`  
`>>help display`