MATLAB

Text Strings

Greg Reese, Ph.D
Research Computing Support Group
Academic Technology Services
Miami University
MATLAB
Text Strings

© 2010-2013 Greg Reese. All rights reserved
Content

Major sections

• Character arrays
• Formatted text
• String arrays
• Miscellaneous
Strings

MATLAB has two different types of text strings – *character arrays* and *cell arrays*

- Main internal difference is how stored in memory
- User manipulates two types slightly differently

Character arrays - best when considering individual letters of text

Cell arrays - best when considering words
Character Arrays

Text stored in two-dimensional array

Key point – All rows must have same number of columns

• If not enough text in a row, row is padded on right with blanks, i.e., MATLAB adds enough space characters to end of text to make row correct length
### Character Arrays

<table>
<thead>
<tr>
<th>I a m</th>
<th>S a m</th>
</tr>
</thead>
<tbody>
<tr>
<td>S a m</td>
<td>I a m</td>
</tr>
<tr>
<td>D o y o u l i k e</td>
<td></td>
</tr>
<tr>
<td>G r e e n e g g s a n d h a m</td>
<td></td>
</tr>
</tbody>
</table>

Four lines of text stored in a 4x18 array

MATLAB makes all rows as long as the longest row

- First three rows above have enough space characters added on ends to make each row 18 characters long
ASIDE
Each character actually occupies two bytes of memory because MATLAB accepts Unicode

- Unicode is common standard for working with non-English languages
- For more information, search for "Unicode" in MATLAB help system
Character Arrays

Pros

• Internally simple

• Can easily use with MATLAB functions that operate on arrays, e.g.,

```matlab
>> num_a = sum(seuss(:) == 'a')
ans = 6
```
Character Arrays

To make a character-array variable with text in it, set variable equal to text in single quote marks:

```python
>>> s = 'Hello world'
s = Hello World
```

**Warning** to C/C++ programmers:
Use a single quote mark ('), not a double quote mark (").
Character Arrays

Multiple lines

```python
>> subjects = [ 'math'; 'physics' ]
```

Gives error. Reason is two rows don't have same number of columns (letters)

There are two ways to fix problem
Fix One – pad by hand

```python
>> subjects=['math';...
              'physics']
```

```python
subjects =
math
physics
```
Character Arrays

Fix Two – use `char()`

```python
>> subjects = char('math', 'physics')

subjects = math
            physics

>> whos subjects
Name       Size  Bytes  Class     Attributes
subjects   2x7    28    char
```

"Attributes" column always empty for these slides so will omit from now on.

"Attributes" column always empty for these slides so will omit from now on.
Often want to stick two text strings together

Example (pseudocode)

```python
name = "edges"
if user wants JPEG output
    file = name + .JPG
else
    file = name + .TIF
```
Character Arrays

Sticking one text string to the end of another is called *concatenation* or *appending*.

To concatenate character array constants and/or variables, put all between square brackets [], separating each by a space or comma.
Character Arrays

Example

```matlab
name = 'edges';
if userEntered == 1
    file = [ name '.jpg' ];
else
    file = [ name '.tif' ];
end
```
Character Arrays

Try It

Make variables with the names "Harold" and "Maude", then use concatenation to store "Harold and Maude" in the variable "film"

```python
>>> young = 'Harold';
>>> old = 'Maude';
>>> film = [ young ' and ' old ]
  film = 'Harold and Maude'
```
Comparing Character Arrays

\texttt{strcmp(\ s1, \ s2\ )} returns 1 if the two strings (character arrays) are identical, returns 0 otherwise

- Strings may be different lengths
- Function is \textit{case-sensitive}, i.e., letters must be in same case to be equal
  - For case \texttt{insensitive} comparison, use \texttt{strcmipi(\ s1, \ s2\ )}

\[i = \text{insensitive}\]
Comparing Character Arrays

Try It

```matlab
>> s1 = 'Matlab';
>> s2 = 'matlab'
>> strcmp(s1,s2)
ans = 0
>> strcmpi(s1,s2)
ans = 1
>> strcmp( s1(2:end), s2(2:end) )
ans = 1
```
Comparing Character Arrays

To get a character-by-character comparison use `==`

- Strings must be same length
- Comparison is *case-sensitive*
  - For case-insensitive comparison, use `upper()` or `lower()` (to be discussed soon) on both strings first
- Can use logical and relational operators to analyze text
Comparing Character Arrays

```matlab
>> s1 = 'Matlab';
>> s2 = 'Maltab';
>> s1 == s2
ans = 1 1 0 0 1 1 1
% number of matching letters
>> sum( s1==s2 )
ans = 4
% index of first mismatch
>> find( s1~=s2, 1 )
ans = 3
```
Categorizing Characters

`isletter()` determines which characters in an array are letters. `isspace()` determines which are whitespace (blank, tab, newline)

```plaintext
>> bond = 'Agent 007';
>> isletter( bond )
an = 1 1 1 1 1 0 0 0 0
>> isspace( bond )
an = 0 0 0 0 0 1 0 0 0
```
Often use `isletter()` or `isspace()` in conjunction with `any()` or `all()`

Example – get file name from user, but no spaces allowed (use MATLAB function `input()`)

```matlab
>> name = input( 'File name: ', 's' );
>> if any( isspace( name ) )
    disp( 'Illegal name – no spaces allowed' );
end
```
Categorizing Characters

Can check for lots of other types of characters by using `isstrprop(s, 'property')`, e.g.,

- `'alpha'` – letter
- `'alphanum'` – letter or number
- `'punct'` – punctuation

```python
>> isstrprop('www.muohio.edu', 'punct')
ans = 0 0 0 1 0 0 0 0 0 0 1 0 0 0
```

Type `help isstrprop` for all properties
Finding Characters

findstr(s1, s2) returns vector of indexes where shorter string is in longer

Example

```matlab
>> s1 = 'am';
>> s2 = 'Sam I am';
>> findstr(s1, s2)
ans = 2 7
>> findstr(s2, s1)
ans = 2 7
```
whitespace is any character for which `isspace()` returns true, i.e.,

- spaces
- newlines
- carriage returns
- tabs
- vertical tabs
- form feeds
Common functions

- \( s_2 = \text{deblank}( s_1 ) \) – returns string with trailing whitespace removed
- \( s_2 = \text{strtrim}( s_1 ) \) – returns string with leading and trailing whitespace removed
- \( s_2 = \text{lower}( s_1 ) \) – returns string with all letters in lower case
- \( s_2 = \text{upper}( s_1 ) \) – returns string with all letters in upper case
- \( s_2 = \text{strjust}( s_1 ) \) – returns string left, right, or center justified
Modifying Characters

Tip
When comparing strings make sure
• There is no leading or trailing space
• Both strings are all in the same case

This is especially useful if text is entered by user or comes from a file
Modifying Characters

**Tip Example**

```matlab
>> g1 = 'Green Eggs and Ham ';
>> g2 = 'Green eggs and Ham';

>> length(g1) == length(g2)
ans = 0

>> g1 = strtrim(g1), g2 = strtrim(g2);

>> length(g1) == length(g2)
ans = 1

>> g1 == g2
ans = 0

>> lower(g1) == lower(g2)
ans = 1
```
Modifying Characters

Tip
Can use text in switch statements but make sure to trim and convert case first

Try It
Download image_type.m
Modifying Characters

image_type.m

function image_type( extension )
switch extension
    case 'JPG'
        disp( 'JPEG file' );
    case 'TIF'
        disp( 'TIFF file' );
    otherwise
        disp( 'Unknown file' );
end
Modifying Characters

**Try It - image_type.m**

```matlab
>> image_type( 'jpg' )
'Unknown file'
>> image_type( 'TIF' )
'Unknown file'
>> image_type( 'TIFF' )
'Unknown file'
>> image_type( 'jpg' )
'Unknown file'
>> image_type( 'TIF' )
'Unknown file'
>> image_type( 'TIFF' )
'Unknown file'
```
Modifying Characters

function image_type(extension)
    extension = upper(strtrim(extension));
    switch extension
        case 'JPG'
            disp('JPEG file');
        case 'TIF'
            disp('TIFF file');
        otherwise
            disp('Unknown file');
    end
Modifying Characters

Try It

```python
>> image_type( 'jpg' )
'JPEG file'
>> image_type( 'TIF' )
'TIFF file'
>> image_type( 'TIFF' )
'Unknown file'
```
Replacing Characters

Use `strrep()` to find and replace characters in a string with other characters.

\[
\text{str} = \text{strrep}( \text{str1, str2, str3} )
\]

finds and replaces all occurrences of the string `str2` in `str1` with the string `str3`.

- `str2` and `str3` can be different lengths.

```matlab
>> s = 'Brown is excellent; Brown is expensive';
>> s2 = strrep( s, 'Brown', 'Dartmouth' )
\s2 = Dartmouth is excellent; Dartmouth is expensive
```
Modifying Characters

Try It

In "Native of miami Valley" use string replacement to:

1 – Make first letter of last word lower case

```matlab
>> s = 'Native of miami Valley';
>> s = strrep( s, 'V', 'v' )
```

```matlab
s = Native of miami valley
```

2 – Capitalize the third word

```matlab
>> s = strrep( s, 'mia', 'Mia' )
```

```matlab
>> s = Native of Miami valley
```
Replacing Characters

For more on replacing characters, see
• `strtok()`
• `strmatch()`
• `textscan()`
• Regular expressions
Character Arrays

Questions?
**Formatted Text**

*Formatted text* is text made up from smaller pieces of text, numbers, etc. Characteristics of the text, such as the displayed precision of numbers, justification, and width of display can be set.
If just have a few pieces of text or numbers, it's easiest to put them together by concatenation.

```python
>> dog = 'Kitty', cat = 'Mittens';
>> s = [ 'My dog''s name is ' dog ]
  s = My dog's name is Kitty
>> [ 'My pets are ' dog ' and ' cat ]
  ans = My pets are Kitty and Mittens
```
>> weight = 65.2;
>> s = [ dog ' weighs ' weight ' lbs' ]
s = Kitty weighs A lbs

Q: What's going on?

A: Concatenation brackets [] expect every entry to be text (character or string cell array). But weight holds a number, not characters
Solution – convert number to character array

\[ c = \text{int2str}(n) \] - converts number \( n \) to character array \( c \) representing integer, rounding if \( n \) is not an integer

\[ c = \text{num2str}(n) \] – converts number \( n \) to character array \( c \)

• Can specify precision and format (type \texttt{help num2str})
>> dog = 'Kitty', cat = 'Mittens';
>> weight = 65.2;

>> s = [ dog ' weighs '...
       int2str(weight) ' lbs' ]

s = Kitty weighs 65 lbs

>> s = [ dog ' weighs '...
       num2str(weight) ' lbs' ]

s = Kitty weighs 65.2 lbs
Formatted Text

If have many elements to put together or format, concatenation gets clumsy. Instead, use `sprintf()`

`sprintf` means print formatted text to string
Also, for full control of displayed number of digits, use `sprintf` command:

```
sprintf( format, n1, n2, n3 )
```

For example:

```
>> sprintf( 'Joe weighs %6.2f kilos', n1 )
```
Formatted Text

>> sprintf( 'Joe weighs %6.2f kilos', n1 )

Format string
• May contain text and/or conversion specifiers
• Must be enclosed in SINGLE quotes, not double quotes, aka quotation marks (" ")
FormData Text

>> sprintf( 'Joe is %d weighs %f kilos', age, weight )

Arguments

• Number of arguments and conversion specifiers must be the same

• Leftmost conversion specifier formats leftmost argument, 2nd to left specifier formats 2nd to left argument, etc.
Formatted Text

Conversion specifier

>> sprintf( 'Joe weighs %f kilos', n1 )

Common conversion specifiers

- `%f` fixed point (decimal always between 1's and 0.1's place, e.g., 3.14, 56.8
- `%e` scientific notation, e.g., 2.99e+008
- `%d` integers (no decimal point shown)
- `%s` string of characters
Formatted Text

Conversion specifier

\[ \text{printf( 'Joe weighs } \%6.2f \text{ kilos', n1 )} \]

To control display in fixed or scientific, use \%w.pf or \%w.pe

\begin{itemize}
  \item \textbf{w} = width: the minimum number of characters to be displayed
  \item \textbf{p} = “precision”: the number of digits to the right of the decimal point
\end{itemize}

Handy: if omit "w", MATLAB will display correct precision and just the right length
Example

```matlab
>> e = exp( 1 );
>> sprintf( 'e is about %4.1f', e )
ans = e is about 2.7
>> sprintf( 'e is about %10.8f', e )
ans = e is about 2.71828183
>> sprintf( 'e is about %10.8e', e )
ans = e is about 2.71828183e+000
>> sprintf( 'e is about %10.2e', e )
ans = e is about 2.72e+000
>> sprintf( 'e is about %f', e )
ans = e is about 2.718282
```
Formatted Text

Use *escape characters* to display characters used in conversion specifiers

- To display a percent sign, use `%%` in the text
- To display a single quote, use `' ' in the text (two sequential single quotes)
- To display a backslash, use `\` in the text (two sequential backslashes)
Formatted Text

Try It

Make the following strings
• Mom's apple 3.14
• Mom's apple 3.1415926
• Mom's apple 3.1e+000

Hint 1: "pi" is a built-in variable

Hint 2: after you enter the first command, use the up arrow key
Try It

>> sprintf( 'Mom''s apple %.2f', pi )
ans = Mom's apple 3.14

>> sprintf( 'Mom''s apple %.7f', pi )
ans = Mom's apple 3.1415927

>> sprintf( 'Mom''s apple %.1e', pi )
ans = Mom's apple 3.1e+000
Formatted Text

Format strings are often long. Can break a string by

1. Put an open square bracket ([ ] ) in front of first single quote
2. Put a second single quote where you want to stop the line
3. Follow that quote with an ellipsis (three periods)
4. Press ENTER, which moves cursor to next line
5. Type in remaining text in single quotes
6. Put a close square bracket ( ] )
7. Put the rest of the `sprintf` command
Example

```matlab
>> weight = 178.3;
>> age = 17;
>> s= sprintf( ['Tim weighs %.1f lbs'... ' and is %d years old'], weight, age )

s = Tim weighs 178.3 lbs and is 17 years old
Try It

```matlab
names = ['Dick'; 'Jane'];
actions = ['run'; 'hop'];
times = [13.2 26.4];

Use sprintf() to make the following:

String 1
See Dick run 100 meters in 13.20 seconds

String 2
See Jane hop 100 meters in 26.4 seconds

String 3
Dick can run 2.0 times as fast as Jane can hop
Try It

```matlab
>> s=sprintf(...
' See %s %s 100 meters in %.2f seconds', ...
names(1,:), actions(1,:), times(1) )
s = See Dick run 100 meters in 13.20 seconds

>> s=sprintf(...
' See %s %s 100 meters in %.1f seconds', ...
names(2,:), actions(2,:), times(2) )
s = See Jane hop 100 meters in 26.4 seconds
```
Try It

```matlab
s = sprintf( [ '%s can %s '...
    '%.1f times as fast as %s can %s' ],...
    names(1,:), actions(1,:),...
    times(2)/times(1), names(2,:), actions(2,:) )
```

```
s = Dick can run 2.0 times as fast as Jane can hop
```
Tip

To print a formatted message on the screen use `fprintf()`. Its arguments are the same as those of `sprintf()`

```c
>> fprintf(...
    '%d score and %d years is %d years\n',...
    4, 7, 4*20+7 )
```

four score and 7 years is 87 years
**Tip**

`error(s)` is a MATLAB function that prints the string `s` and then stops the MATLAB function in which it is called. However, it can also make a formatted string, display it, and then stop. Call it just as you call `sprintf()`

```matlab
>> badLine=16; inputFile='data.txt';
>> error( 'Couldn''t read line %d of %s', ...
          badLine, inputFile );
??? Couldn't read line 16 of data.txt
```


Formatted Text

`sprintf` has many more capabilities. To find out about them you can ask MATLAB for help on `sprintf`. 
Formatted Text

Questions?
String Arrays

*Cell arrays of strings* (also *string arrays* or *strings* or *text strings*) are MATLAB’s way of storing text. Use to:

- Get data from user or file
- Write data to file or display to user
- Dynamically (while program is running) create and execute MATLAB commands
- Represent certain types of data, e.g., genomic (DNA, RNA, proteins)
String Arrays - definition

A string array or cell array of strings is a cell array in which every element is a character array. The character arrays can be different sizes.
String Arrays - definition

This is a 4x1 string array
- 4 rows, 1 column
- Each array element has a character array
  - Character arrays can be different lengths
  - No padding necessary!
String Arrays - benefits

Benefits (versus character arrays)

• Easier to use when have different lengths of text
• Take up less memory if have many pieces of text and they have different lengths
String Arrays - creation

Initialize a string array same way as a character array but use curly braces {}:

- Use commas or spaces to separate elements in a row
- Use semicolon to mark end of row
String Arrays - creation

```matlab
>> a={'Greg' 'Reese'; 'Jimmy Bob' 'Bovedeaux'}
a = 'Greg'
    'Reese'
    'Jimmy Bob'
    'Bovedeaux'

>> size(a)
ans = 2 2
```

Note different length strings
String Arrays - creation

Can use `celldisp()` to display all elements of a cell array

```matlab
>> a={'Greg' 'Reese'; 'Jimmy Bob' 'Bovedeaux'};
>> celldisp(a)
a{1,1} = Greg
da{1,2} = Reese
>> a{1,1} = Jimmy Bob
>> a{1,2} = Bovedeaux
```
String Arrays - access

Remember, every element of a cell array is a cell. The content of a cell need not be, and is usually not, a cell. In fact, the content of every cell of a string array is a character array.
String Arrays - access

Tricky part – indexing a cell array.

Two ways to index – () and {}

- \( a(m,n) \) – returns cell at row \( m \) and column \( n \) of array \( a \)
  - Returned value is always a cell

- \( a\{m,n\} \) – returns content of cell at row \( m \) and column \( n \) of cell array \( a \)
String Arrays - access

Example

```
>> cars = { 'Toyota'; 'Chevy'; 'Ford' }
>> disp( [ 'My car is a ' cars{2,1} ] )
>> disp( [ 'My car is a ' cars(2,1) ] )
```
String Arrays - access

Try It

```python
>> cars={ 'Toyota'; 'Chevy'; 'Ford' }
cars =
'Toyota'
'Chevy'
'Ford'
```
String Arrays

Try It

```>> q1 = [ 'My car is a ' cars{2,1} ];
>> whos q1
```
```
Name   Size  Bytes  Class
q1     1x17   34    char
```
```>> disp( q1 )
My car is a Chevy
```
```>> q2 = [ 'My car is a ' cars(2,1) ];
>> whos q2
```
```
Name   Size    Bytes  Class
q2     1x2     154    cell
```
```>> disp( q2 )
'My car is a '    'Chevy'
```
String Arrays - conversion

Use `cellstr()` to convert a character array into a string array

• Each row of character array is stored in one cell of a vertical cell vector

• `cellstr()` removes trailing blanks
### Try It

```matlab
>> names1 = ['Joe Blow'; 'Sally Mae'; 'Jenny Hudson'];
>> whos names1
Name       Size            Bytes  Class       character array
names      3x12               72  char

>> names1
names1 =
Joe Blow
Sally Mae
Jenny Hudson
```

- **3 rows with 12 characters (columns) in every row**
- **Padded with trailing spaces (blanks on right)**

**Character Array**
String Arrays - conversion

Use \texttt{char()} to convert a string array into a character array

- Each cell of vertical cell vector converted to one row of character array
- \texttt{char()} adds trailing blanks to each row so that all rows have same number of columns
String Arrays - conversion

Try It

>> whos names2

Name          Size         Bytes  Class
names2        3x1          238    cell

>> names3 = char( names2 );

>> whos names3

Name          Size         Bytes  Class
names3        3x12         72     char

>> names3

names3 =

Joe Blow
Sally Mae
Jenny Hudson

padded with trailing spaces (blanks on right)
String Arrays - conversion

Try It

>> names1 == names3

ans =
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Back exactly to what we started with
String Arrays - comparison

Use `strcmp(a,b)` to compare two string arrays

- Arrays must have same dimensions
- Comparison is case sensitive
- Returns logical array of same dimension with 1 (same) or 0 (different) at each cell

`strcmpi()` works same way but does a case-insensitive comparison
Try It

>> cars1 = { 'Audi' 'AUDI'; 'Toyota' 'Chevy' }
cars1 = 'Audi'    'AUDI'
      'Toyota'    'Chevy'

>> cars2 = { 'Audi' 'Audi'; 'Toyota' 'Chevy' }
cars2 = 'Audi'    'Audi'
      'Toyota'    'Chevy'

>> strcmp( cars1, cars2 )
ans =
    1  0
    1  1

>> strcmpi( cars1, cars2 )
ans =
    1  1
    1  1
String Arrays - comparison

Tip
Don't use == to compare two strings because if the strings are different lengths, you'll get an error

```matlab
>> a = 'Jack';
>> b = 'jack';
>> c = 'Jacques';
>> a == b
ans = 0 1 1 1 1
>> a == c
??? Error using ==> eq
Matrix dimensions must agree.
```
String Arrays - sorting

Use `sort()` to sort a cell array of strings

- Sorts into ascending, alphabetical order
- Comparison is case sensitive
- Always returns a vector with same number of elements as input
- If input is a 2D or higher array, converted to 1D and then sorted
String Arrays - sorting

B = sort( A )

• A is vector of strings
• B is sorted vector with same size as A

Example

>> cars = { 'Toyota' 'Chevy' 'Ford' }
cars =
   'Toyota'    'Chevy'    'Ford'
>> sorted_cars = sort( cars )
sorted_cars =
   'Chevy'    'Ford'    'Toyota'
String Arrays - sorting

Can also get original indexes of sorted string. This is useful if original strings had other data associated with them.

\[
[ B \ IX ] = \text{sort}( A )
\]

- \( A \) is vector of strings
- \( B \) is sorted vector with same size as \( A \)
- \( IX \) is corresponding index in original array, i.e., \( IX(1) \) is the index of \( B(1) \) in \( A \), \( IX(2) \) is the index of \( B(2) \) in \( A \), etc.
String Arrays - sorting

Example

Jason, Jack, Amber, and Bill are 44, 20, 9, and 80 years old. Make a string vector with their names and a numerical vector with their ages. Sort the names into alphabetical order and print the name and age of the first and last person on the sorted list.
String Arrays - sorting

**Example**

```matlab
>> names = {'Jason' 'Jack' 'Amber' 'Bill'};
>> ages = [44 20 9 80];
>> [sortedNames ix] = sort( names )
sortedNames = 'Amber' 'Bill' 'Jack' 'Jason'
ix = 3 4 2 1
```

```matlab
>> fprintf( 'First on list is %s, who is %d\n', ... 
            sortedNames{1}, ages( ix(1) ) );
First on list is Amber, who is 9
```

```matlab
>> fprintf( 'Last on list is %s, who is %d\n', ... 
            sortedNames{end}, ages( ix(end) ) );
Last on list is Jason, who is 44
```

Must access with {}, not ()
String Arrays - search

Use `strfind()` to find where one string occurs as a substring in members of a string array

\[ k = \text{strfind}( \text{array}, \text{string} ) \]

- `array` is a cell array of strings
- `string` is a character array
- `k` is cell array of same dimension as `array` with `k{p}` being a vector of indexes in `array{p}` in which `string` occurs
String Arrays - search

Try It

>> seuss = { 'Sam I am'; 'I am Sam'; ...
    'Do you like'; 'Green eggs and ham' };

seuss = 'Sam I am'
    'I am Sam'
    'Do you like'
    'Green eggs and ham'

>> indexes = strfind( seuss, 'am' );

>> whos indexes

    Name      Size            Bytes  Class      Attributes
    indexes   4x1               280    cell

>> celldisp( indexes )

indexes{1} = 2    7
indexes{2} = 3    7
indexes{3} = [] % no "am" in "Do you like"
indexes{4} = 17
Try It

```matlab
>> indexes = strfind( seuss, 'Sam' );
>> celldisp( indexes )
 indexes{1} = 1
 indexes{2} = 6
 indexes{3} = []
 indexes{4} = []
>> indexes = strfind( seuss, 'sam' );
>> celldisp( indexes )
 indexes{1} = []
 indexes{2} = []
 indexes{3} = []
 indexes{4} = []
```

Why?
String Arrays - search

`ismember()` determines if a string is in a group of strings

\[
\text{yesNo} = \text{ismember}( A, S )
\]

- \(A\) is cell array of strings
- \(S\) cell array of strings
- \(\text{yesNo}\) is logical array of same dimension as \(A\) with \texttt{true} (1) meaning that element is in \(S\) and \texttt{false} (0) meaning it is not in \(S\)
- \(A\) and/or \(S\) can also be character arrays.

See MATLAB help
Try It

Make these arrays for this and following slides

```matlab
>> fratBoys = { 'Terrence' 'Wilfred' 'Jacques' 'Harry' 'Joe' };
>> fratStates = { 'Indiana' 'Ohio' 'Indiana' 'Ohio' 'Ohio' };
>> randomBoys = { 'Tom' 'Dick' 'Harry' };
```

Determine whether each random boy is or is not a frat boy

```matlab
>> ismember( randomBoys, fratBoys )
ans =     0     0     1
```
String Arrays - search

Example
Without making a new variable, determine if Bubba is a frat boy

```matlab
>> ismember( 'Bubba', fratBoys )
ans =     0
```

Note:
- Comparing character array to string array
- `ismember()` removes trailing (but not leading) blanks before comparing
String Arrays - intersection

\text{intersect()} \text{ finds all strings that are in each of two groups}

\text{both} = \text{intersect( A1, A2 )}

- \text{A1} is cell array of strings
- \text{A2} cell array of strings
- \text{both} is cell array of strings, each of which is in \text{A1} and \text{A2}
  - \text{both} sorted in alphabetical order
- \text{A1} and/or \text{A2} can also be character arrays. See MATLAB help
Try It
Find the names of the random boys who are also frat boys

```python
>> intersect(randomBoys, fratBoys)
an = 'Harry'
```
String Arrays - difference

\texttt{setdiff()} finds all strings that are in one group but not in another

\[
\text{diff} = \text{setdiff}( \text{A1}, \text{A2} )
\]

- \text{A1} is cell array of strings
- \text{A2} is cell array of strings
- \text{diff} has the strings that are in \text{A1} but not in \text{A2}
  - \text{diff} sorted in alphabetical order
- \text{A1} and \text{A2} can also be character array. See MATLAB help
Try It
Find the names of the random boys that are not frat boys. Also, find the names of the frat boys that are not random boys

```python
>> randomNotFrat = setdiff( randomBoys, fratBoys )
randomNotFrat =
   'Dick'   'Tom'

>> fratNotRandom = setdiff( fratBoys, randomBoys )
fratNotRandom =
   'Jacques'   'Joe'   'Terrence'   'Wilfred'
```
String Arrays - unique

unique() removes all but one copy of duplicate strings

\[ b = \text{unique}(A) \]

- **A** is cell array of strings
- **b** has the same values as **A** but without repetitions
  - **b** sorted in alphabetical order
- **A** can also be character array. See MATLAB help
Try It

What are the different states that the frat boys come from and how many of those states are there?

```matlab
>> uniqueStates = unique(fratStates)
uniqueStates = 'Indiana'    'Ohio'

>> length(uniqueStates)
an = 2
```
String Arrays - more

`setxor(A, B)` finds all strings that are in A or B but not in both

`union(A, B)` finds all strings that are in A or B or both

See MATLAB help for details
String Arrays – number of occurrences

Common MATLAB trick

Logical comparisons in MATLAB return 1 if true and 0 if false. Use \texttt{sum} to count the number of true comparisons

\begin{verbatim}
>> v = [ 3 0 5 9 9 99 9 4 6 2 ];
>> v == 9
ans = 0 0 0 1 1 0 1 0 0 0
>> sum( [ 0 0 0 1 1 0 1 0 0 0 ] )
ans = 3
>> sum( v==9 ) \% how many 9’s in \( v \)?
ans = 3
\end{verbatim}
String Arrays – number of occurrences

>> v = [ 3 0 5 9 9 99 9 4 6 2 ];

How many even numbers?

>> sum( rem(v,2) == 0 )
ans = 4

How many numbers that are at least ten?

>> sum( v >= 10 )
ans = 1
String Arrays – number of occurrences

Can also use `strcmp(a, b)` to compare one string to all strings in a string array

- *a* is a char array or a 1x1 string array
- *b* is a string array

- Returns logical array of same dimension as *b* with 1 (same) or 0 (different) at each cell

- `strcmp(b, a)` gives same result

`strcmpi()` works same way but does a case-insensitive comparison
String Arrays – number of occurrences

How many times does one string occur in a string array?

\[
\text{sum( strcmp( aString, strings ) )}
\]

How many frat boys are named “Harry”?

\[
\text{>> sum( strcmp( 'Harry', fratBoys ) )}
\]
\[
\text{ans = 1}
\]
String Arrays – number of occurrences

Now suppose Harold Blankenship III (“Harry”, to his friends) from Bloomfield Hills, Michigan, joins the frat. Now how many members are named “Harry”?

```matlab
>> fratBoys{end+1} = 'Harry';
>> fratStates{end+1} = 'Michigan';
>> sum( strcmp( 'Harry', fratBoys ) )
ans = 2
```
String Arrays – number of occurrences

How many times does each unique string occur in a string array? Completely non-intuitive answer is

\[
[\text{us, } \sim, \text{ix}] = \text{unique}(\text{strings}); \\
c = \text{histc}(\text{ix},1:\text{max}(\text{ix}));
\]

• **strings** – (vector) string array
• **us** – (vector) string array of unique strings in **strings**, in alphabetical order
• **ix** – index in **us** of each string in **strings** (ix is same dimension as **strings**)
• **c** – number of times each string in **us** occurs in **strings**
String Arrays – number of occurrences

How many times does each unique name in the fraternity occur?

```matlab
>> [ uFratBoys, ~, ix ] = unique( fratBoys )
uFratBoys =
'Harry' 'Jacques' 'Joe' 'Terrence' 'Wilfred'
ix = 4 5 2 1 3 1
>> c = histc( ix, 1:max(ix) )
c = 2 1 1 1 1 1 1
```
String Arrays - case insensitive

All string array functions discussed (except `strcmpli()`) do case-sensitive comparisons. To ignore case when comparing must convert all strings to upper case with `upper()` or to lower case with `lower()`.

If need original capitalization, get indexes from function output
String Arrays - case insensitive

Example

```python
>> school1Sports = { 'baseball' 'soccer' 'basketball'...
   'Fencing' }
>> school2Sports = { 'Diving' 'Fencing' 'Swimming' ...
   'Water polo' 'Broomball' 'Basketball' }

Find the sports the two schools have in common using case-sensitive comparisons

```python
>> commonSports = ... 
   intersect( school1Sports, school2Sports )
commonSports =
'Fencing'
```
**Example**

Find the sports the two schools have in common using case-insensitive comparisons

```plaintext
>> commonSports = intersect( upper(school1Sports), ... upper(school2Sports) )
commonSports =
 'BASKETBALL' 'FENCING'
```
String Arrays - case insensitive

Example

Find the sports the two schools have in common using case-insensitive comparisons and display the results with the capitalization they have in the list for school 2

```matlab
>> [ commonSports ix1 ix2 ] = intersect( upper(school1Sports), upper(school2Sports) );
>> school2Sports( ix2 )
ans =
    'Basketball'    'Fencing'
```
String Arrays

Questions?
Miscellaneous

Further string topics

– Evaluate a dynamically created MATLAB command
– Separate a file name into parts (drive, name, extension, etc.)
– Unicode (see MATLAB documentation)
– Regular expressions (see MATLAB documentation)
Evaluate String

\texttt{eval(s)} evaluates (executes) a MATLAB command in the text string \texttt{s}.

Handy MATLAB function to get input

\begin{verbatim}
str = input( prompt, 's' )
\end{verbatim}

- \texttt{prompt} is text displayed to user
- \texttt{'s'} forces function to just return user's input as a character array
- \texttt{str} is character array with what user typed
Evaluate String

Try It

Enter 'magic' for command and size of at least 3

```matlab
>> name=input( ... 'MATLAB matrix command: ', 's' )
>> size = input( 'Size: ', 's' )
>> command = [ name '([' size ']') ]
>> eval( command )
```
Evaluate String

Try It

Enter 'magic' for command and size of at least 3

```matlab
>> name=input( 'MATLAB matrix command: ', 's' )
MATLAB matrix command: magic
>> size = input( 'Size: ', 's' )
Size: 4
>> command = [ name ' (' size ')' ]
command = magic(4)
>> eval( command )
ans =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

What's this?
File Name Parts

Often want to get parts of file name

• Infer type of file from extension, e.g.,
  – .JPG is JPEG file, .TIF is TIFF file

• Make slight change to name and use for related file, e.g.,
  – If input file is "foo.txt", make output file be "foo_output.txt"
MATLAB function `fileparts()` pulls file name apart. `fullfile()` puts name together

\[
\text{[ path name extension version ]} = \text{fileparts( filename )}
\]

\[
\text{filename} = \text{fullfile( path, name, extension, version )}
\]

* Sort of. See example in documentation for `fileparts()`
File Name Parts

Example

>> inputFile = 'c:\projects\dog5.txt';
>> [ path name extension version ] =...
    fileparts( inputFile )
path = c:\projects    name = dog5
extension = .txt      version = '' xx

>> outputFile = [ path filesep name ...
    '_output' extension version ]
outputFile = c:\projects\dog5_output.txt

Note: filesep is a MATLAB function that returns the file-parts separator for
the operating system you're running on, e.g., "\" for Windows, "/" for Linux
Example

Good to make all output names in one function?

```matlab
function name = makeOutputName( fileType )
switch fileType
    case 'anovaOutput'
        name = 'anova.txt';
    case 'anovaInput'
        name = 'anova_inputs.txt';
% file type not needed in this name
    case 'powerGraph'
        name = 'power_output';
end
```
Misc. String Topics

Questions?
The End