MATLAB

Strings II

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

Formatted Text

If just have a few pieces of text or numbers, it's easiest to put them together by concatenation

This is how you put a single quote in a string

```
>> 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
```

Formatted Text

```
>> 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

Formatted Text

Solution – convert number to character array

`c = int2str( n )` - converts number `n` to character array `c` representing integer, rounding if `n` is not an integer

`c = num2str( n )` – converts number `n` to character array `c`

- Can specify precision and format (type `help num2str`)

Formatted Text

Try It

```
>> 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

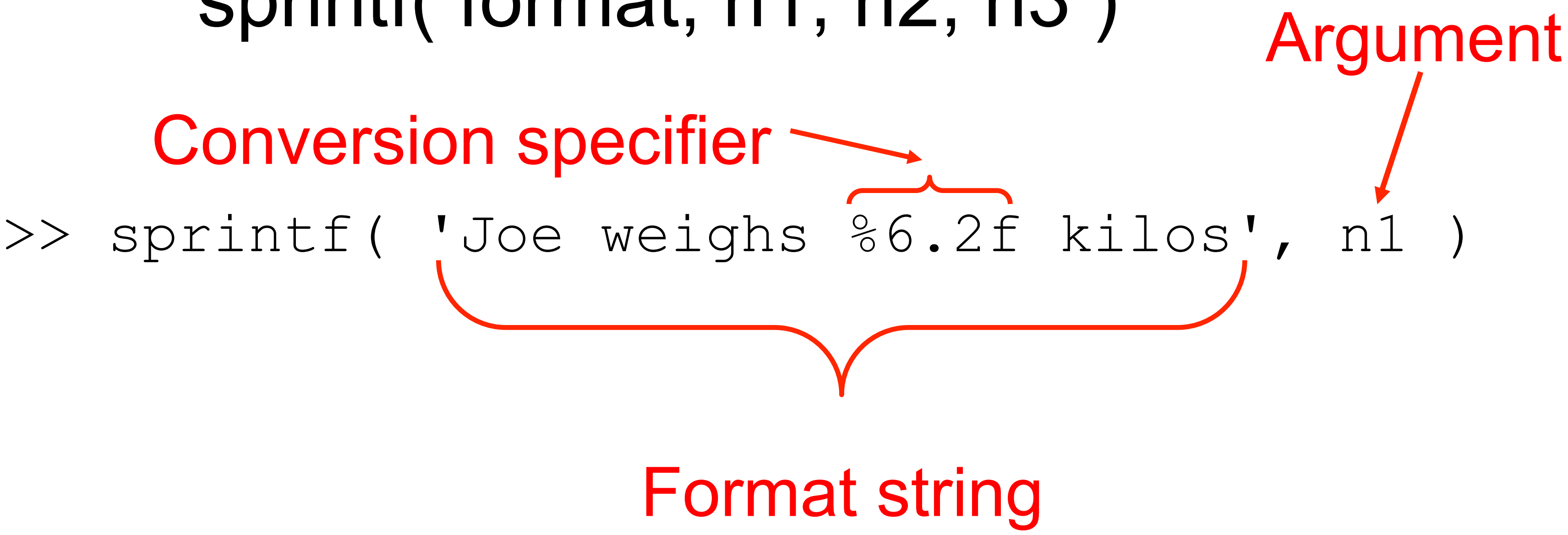
If have many elements to put together or format, concatenation gets clumsy. Instead, use `sprintf()`

sprintf means print formatted text to string

Formatted Text

Also, for full control of displayed number of digits, use `sprintf` command

sprintf(format, n1, n2, n3)



Argument

Conversion specifier

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

Format string

Formatted Text

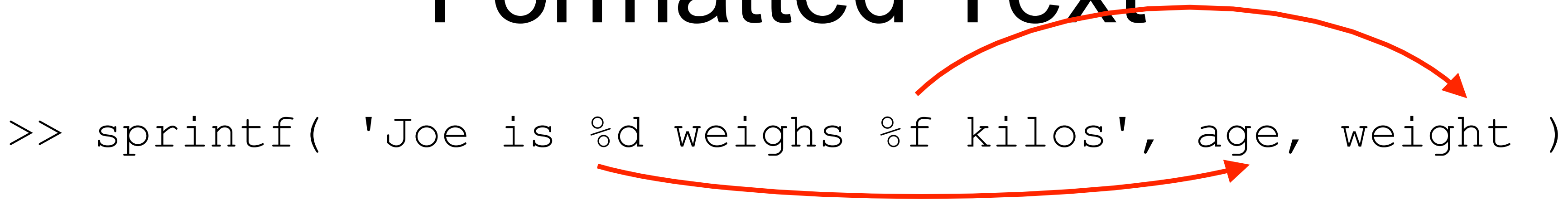
```
>> sprintf( 'Joe weighs %6.2f kilos', n1 )
```

Format string

Format string

- May contain text and/or conversion specifiers
- Must be enclosed in SINGLE quotes, not double quotes, aka quotation marks (" ")

Formatted 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

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

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

- w = width: the minimum number of characters to be displayed
- p = “precision”: the number of digits to the right of the decimal point

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

Formatted Text

Example

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

Formatted Text

Use e*scape 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

Formatted Text

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

Formatted Text

Example

```
>> 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
```

Formatted Text

Try It

```
>> 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
```

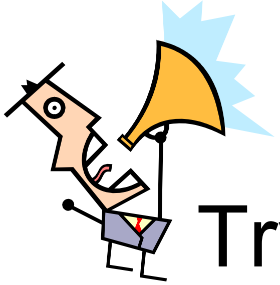
String 2

```
See Jane hop 100 meters in 26.4 seconds
```

String 3

```
Dick can run 2.0 times as fast as Jane can hop
```

Formatted Text

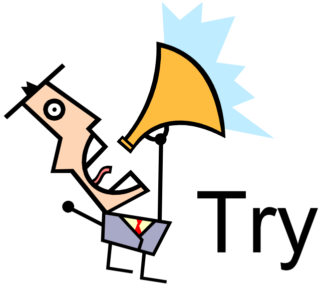


Try It

```
>> 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
```

Formatted Text



Try It

```
>> 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
```

Formatted Text

Tip

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

Tip: Put \n at end of format specifier

```
>> fprintf(...
  '%d score and %d years is %d years
\n',...
    4, 7, 4*20+7 )
4 score and 7 years is 87 years
```

Tip

Formatted Text

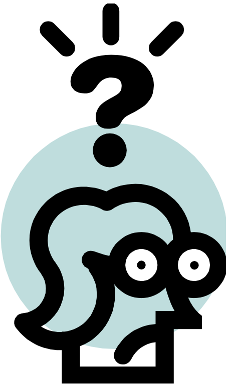
`error(s)` is a MATLAB function that prints the string s and then stops the MATLAB function in which it is called. However, it can also make a formatted string, display it, and then stop. Call it just as you call `sprintf()`

```
>> 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

```
>> a={'Greg' 'Reese'; 'Jimmy Bob' 'Bovedeaux'}
a = 'Greg'         'Reese'
    'Jimmy Bob'    'Bovedeaux'
>> size(a)
ans = 2     2
```

String Arrays - creation

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

```
>> a={'Greg' 'Reese'; 'Jimmy Bob' 'Bovedeaux'};
>> celldisp(a)
a{1,1} = Greg
a{2,1} = Jimmy Bob
a{1,2} = Reese
a{2,2} = 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 {}

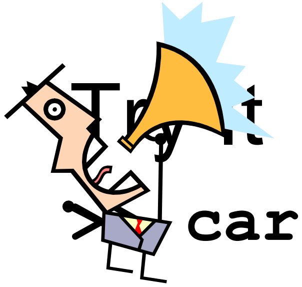
- *a(m,n)* – returns cell at row *m* and column *n* of array *a*
  - Returned value is always a cell
- *a{m,n}* – returns content of cell at row *m* and column *n* of cell array *a*

String Arrays - access

Example

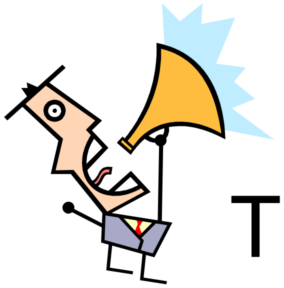
```
>> cars = { 'Toyota'; 'Chevy'; 'Ford' }
>> disp( [ 'My car is a ' cars{2,1} ] )
>> disp( [ 'My car is a ' cars(2,1) ] )
```

String Arrays - access



Try it

```
>> 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

String Arrays - conversion

Try It

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

character array

3 rows with 12 characters (columns) in every row

padded with trailing spaces (blanks on right)

String Arrays - conversion

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

- Each cell of vertical cell vector converted to one row of character array
- `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
```

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

String Arrays - 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

```
>> a = 'Jack';
>> b = 'jack';
>> c = 'Jacques';
>> a == b
ans = 0      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 ] = 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

```
>> names = { 'Jason' 'Jack' 'Amber' 'Bill' };
>> ages = [ 44 20 9 80 ];
>> [ sortedNames ix ] = sort( names )
sortedNames = 'Amber'     'Bill'     'Jack'     'Jason'
ix = 3     4     2     1
>> fprintf( 'First on list is %s, who is %d\n', ...
      sortedNames{1}, ages( ix(1) ) );
First on list is Amber, who is 9
>> 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 = strfind( array, 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
  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
```

String Arrays - search

Try It

```
>> 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

```
yesNo = ismember( A, S )
```

- `A` is cell array of strings
- `S` cell array of strings
- `yesNo` is logical array of same dimension as `A` with `true` (1) meaning that element is in `S` and `false` (0) meaning it is not in `S`
- `A` and/or `S` can also be character arrays. See MATLAB help

String Arrays - search

Try It

Make these arrays for this and following slides

```
>> 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

```
>> ismember( randomBoys, fratBoys )
ans =      0      0      1
```

String Arrays - search

Example

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

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

Note:

- Comparing character array to string array
- `ismember()` removes trailing (but not leading) blanks before comparing

String Arrays - intersection

`intersect()` finds all strings that are in each of two groups

```
both= intersect( A1, A2 )
```

- `A1` is cell array of strings
- `A2` cell array of strings
- `both` is cell array of strings, each of which is in `A1` and `A2`
  - `both` sorted in alphabetical order
- `A1` and/or `A2` can also be character arrays. See MATLAB help

String Arrays - intersection

Try It

Find the names of the random boys who are also frat boys

```
>> intersect( randomBoys, fratBoys )
ans = 'Harry'
```

String Arrays - difference

`setdiff()` finds all strings that are in one group but not in another

```
    diff = setdiff( A1, A2 )
```

- `A1` is cell array of strings
- `A2` is cell array of strings
- `diff` has the strings that are in `A1` but not in `A2`
  - `diff` sorted in alphabetical order
- `A1` and `A2` can also be character array. See MATLAB help

String Arrays - difference

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

```
>> 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 = 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

String Arrays - unique

Try It

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

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

>> length( uniqueStates )
ans = 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 - case insensitive

All string array functions discussed (except `strcmpi()`) 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

```
>> 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

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

String Arrays - case insensitive

Example

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

```
>> commonSports = intersect( upper(school1Sports), ...
      upper(school2Sports) )

commonSports =

    'BASKETBALL'    'FENCING'
```

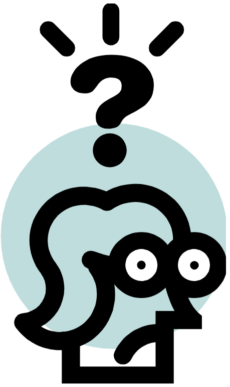
String Arrays - case insensitive

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

```
>> [ 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

`eval( s )` evaluates (executes) a MATLAB command in the text string `s`

Handy MATLAB function to get input

```
str = input( prompt, 's' )
```

- `prompt` is text displayed to user
- `'s'` forces function to just return user's input as a character array
- `str` is character array with what user typed

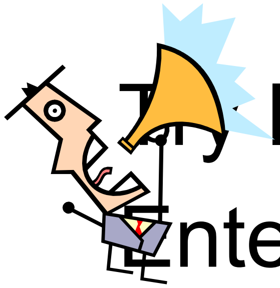
Evaluate String

Try It

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

```
>> 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

```
>> 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"

File Name Parts

MATLAB function `fileparts()` pulls file name apart. `fullfile()` puts name together*

[path name extension version] = fileparts(filename)

filename = 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

File Name Parts

Example

Good to make all output names in one function?

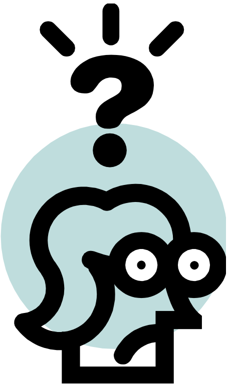
```
function name = makeOutputName( fileType )
switch fileType
    case 'anovaOutput'
        name = 'anova.txt';
    case 'anovaInput'
        name = 'anova_inputs.txt';
    % file type not needed in this name
    case 'powerGraph'
         name = 'power_output;
end
```

Programming problem?

Good style?

Programming problem?

Misc. String Topics



Questions?