Scripts

MATLAB Scripts

- a script is text file containing a *sequence* of MATLAB commands
  - each command usually occurs on a separate line of the file
- MATLAB can run the commands in a script by reading the file and interpreting the text as MATLAB commands
  - commands are run in order that they appear in the script file

MATLAB Scripts

- the filename of a MATLAB script always has the following form:

`yourScriptName.m`

where **`yourScriptName`** must be a valid MATLAB variable name

- i.e., must begin with a letter and may only contain letters and spaces and underscores
  - no spaces or symbols!

MATLAB scripts

- scripts are useful for keeping a permanent record of a sequence of commands
- typically, you will want to output the result of the script
  - e.g., as a plot of some kind
  - or as a message of some kind
- also, you may want to save the results of the computations that the script has performed

Text output

- a MATLAB string is a row vector of characters
- any text enclosed by single quotes is considered a MATLAB string

```
s = 'Any characters'
```

- the string is actually a vector that contains the numeric codes for the characters (codes 0 to 127 are ASCII)
- the length of S is the number of characters
- a quotation within the string is indicated by two quotation marks

Text output

- the command **`disp`** will display a MATLAB string to the screen without displaying the name of the variable

```
>> s = 'Any characters'
s =
Any characters

>> disp(s)
Any characters
```

Formatted output

- it is often very useful to generate strings programmatically
  - display output more elaborate than just a variable value
  - display a table of values
  - export data to a non-native MATLAB format
  - generate the string for a MATLAB command
- the function **`sprintf`** is used to generate formatted strings
  - use **`doc sprintf`** to get help for **`sprintf`**
  - **`help sprintf`** is less useful

Formatted output

- the syntax of the **`sprintf`** command is:

```
str = sprintf(formatSpec, A1, ..., An)
```

- **`formatSpec`** is a formatting string with a large number of options
  - probably inherited from the C programming language (or C's predecessor languages)
- **`A1, ..., An`** are the arrays containing the information to format

Formatted output

- the formatting string describes how MATLAB should convert the information stored in the arrays into text
- the formatting string can include:
  - text
  - escape characters
  - zero or more percent signs each followed by:
    - optional operator characters, and a conversion character

Formatted output

- examples using only plain text and escape characters
  - see textbook or **`doc sprintf`** for a table of escape characters

```
>> sprintf('hi')
>> sprintf('''hi''')
>> sprintf('hi\tbye')
>> sprintf('hi\nbye')
>> sprintf('50%%')
```

Formatted output

- usually you will need to include a conversion character and an array containing data
  - the conversion character tells MATLAB what conversion it should use when converting the data in the array to text
  - most commonly used conversions are:

`%d`	base 10 integer
`%i`	base 10 integer
`%f`	fixed-point floating-point
`%e`	exponential notation
`%s`	string

Formatted output

- examples using simple conversions

```
>> n = 10;
>> sprintf('There are %d items', n)

>> degc = 100;
>> degf = (9 / 5) * degc + 32;
>> sprintf('%f deg C = %f deg F', degc, degf)
>> sprintf('%e deg C = %e deg F', degc, degf)

>> name = 'Jessica';
>> sprintf('Her name was %s', name)
```

Formatted output

- for floating-point conversions, you can specify the number of digits after the decimal place
  - called the *precision*
  - default value is 6

```
>> degc = 100;
>> degf = (9 / 5) * degc + 32;
>> sprintf('%.2f deg C = %.2f deg F', degc, degf)
>> sprintf('%.1e deg C = %.1e deg F', degc, degf)
>> sprintf('%.0f deg C = %.15f deg F', degc, degf)
```

Formatted output

- for all conversions, you can specify the minimum number of characters to output

```
>> n = 10;
>> sprintf('There are %5d items', n)

>> degc = 100;
>> degf = (9 / 5) * degc + 32;
>> sprintf('%8.1f deg C = %8.1f deg F', degc, degf)

>> name = 'Jessica';
>> sprintf('Her name was %15s', name)
```

Formatted output

- there are many other options in **`sprintf`**
  - see **`doc sprintf`** for details
  - experiment the options

Formatted output

- when using **`sprintf`** with an array, the formatting string is recycled columnwise through the elements of the array

```
>> A = [1 2 3 4; 5 6 7 8];
>> s = sprintf('%d %d\n', A)
s =

1 5
2 6
3 7
4 8
```

Formatted output

- when using **`sprintf`** with an array, the formatting string is recycled columnwise through the elements of the array

```
>> A = [1 2 3 4; 5 6 7 8; 9 10 11 12];
>> s = sprintf('%d %d\n', A)
s =

1 5
9 2
6 10
3 7
11 4
8 12
```

Formatted output

- when using **`sprintf`** with an array, the formatting string is recycled columnwise through the elements of the array

```
>> A = [1 2 3 4; 5 6 7 8; 9 10 11 12];
>> >> s = sprintf('%d %d %f %f\n', A)

s =

1 5 9.000000 2.000000
6 10 3.000000 7.000000
11 4 8.000000 12.000000
```

Saving results to a file

- you can save some or all of the variables in your workspace to a file that MATLAB can reload
- the command

```
>> save
```

  saves all of the workspace variables to a file named **`matlab.mat`** in the current working folder
- the file is not human readable; it is saved as a MATLAB readable binary format

Saving results to a file

- the commands

```
>> save('myfile.mat')
>> save myfile.mat
```

saves all of the workspace variables to a file named **`myfile.mat`** in the current working folder

Saving results to a file

- the commands

```
>> save('myfile.mat', 'degc', 'degf', 'name')
>> save myfile.mat degc degf name
```

saves only the specified workspace variables to a file named **`myfile.mat`** in the current working folder

Saving results to a file

- the commands

```
>> save('myfile.mat', 'degc', 'degf', 'name')
>> save myfile.mat degc degf name
```

saves only the specified workspace variables to a file named **`myfile.mat`** in the current working folder

Saving results to a file

- if you want a human readable file specify the option **`'-ascii'`** and possibly **`'-double'`**

```
>> save('myfile.mat', 'degc', 'degf', 'name', '-ascii')
>> save myfile.mat degc degf name -ascii
```

- this has severe limitations; I don't recommend using the **`'-ascii'`** option unless you are sure you know what you are doing

Loading variables from a file

- you can load a binary .mat file using the **`load`** command
  - you get back the saved variables with their original names

```
>> load('myfile.mat')
>> load myfile.mat
```

Loading variables from a file

- **`load`** will also load a plain text file if it contains a rectangular table of numbers with an equal number of elements in each row
  - for example, suppose you have a file **`mydata.dat`** that contains the following:

```
 0.5377   -1.3077   -1.3499
 1.8339   -0.4336    3.0349
-2.2588    0.3426    0.7254
 0.8622    3.5784   -0.0631
 0.3188    2.7694    0.7147
```

Loading variables from a file

- the command:

```
>> x = load('mydata.dat')
```

will load the contents of the file into the variable **x**