Yesterday

- Vectors, Matrices
- Scripts
- Flow control and logic.

- Let’s fire up Matlab.

TODAY

Functions.

Interacting with memory.

cells.

Functions

-can take input (passing variables)

-can return outputs.

- variables are **internal**.

- NAME OF M-FILE AND FUNCTION HAVE TO BE THE SAME.

Functions

Example:

- First thing: we have to identify the script as a function, with the keyword *function*

  At the top of the m file we type:

function *output* = nameoffunction(*passvar1*,*passvar2*…);

Then you can write your program, like a normal script.

```
function output = 
nameoffunction(passvar1,passvar2,…);
```

- *output* will hold the value that will be returned by the function (if any). Outputs can be of any type. You can give it any name you want.
- *passvar1, passvar2…* are the passing variables (aka **arguments)** for the function. You can pass as many **arguments** as you'd like.

Passing variables

These are values that constrain the execution of the function. For example:

```
function DrawSquare(x_left,y_top,size);
```

{x_left,y_top, size} will be the values you’ll probably need to draw a square.

Passing variables

In your code, you will use the passing variables by referring to them by the name you specified on the function statement.

When you call the function in your program, you simply specify values, or use other variables to PASS the values.

Example:

Take the function:

```
function rdow = scramble(word);
%these are your help lines
%that you use to explain your code
name_sz = length(word);
neworder = randperm(name_sz);

rdow=word(neworder);


%% That’s it!! SAVE IT AS scramble.m
```

Example:

Now go to the command prompt and use your new function:

```
> myname = ‘alejo’;
> newname = scramble(myname);
```

here, myname is passing the value 'alejo' to the function 'scramble'.

Note, we did NOT use 'word'.

word is *internal* to the function.

Example:

Now try:

```
> scramble(myname);  %don't have to
  assign the result to any variables.

> scramble('alejo'); %don't need a
  variable to pass the value.
```

Example:

Now try:

```
> scramble alejo
> scramble myname        (surprised?)
```

> WHEN INPUT IS STRING, Matlab allows you to call the function without quotes and without parenthesis (as a shortcut)

OTHER types of variables

GLOBAL variables:

If you want more than one function to share a single copy of a variable, declare it as global. You need to do so on every function using the variable.

```
> function h = falling(t);
> global GRAVITY
```

```
>function headache = hangover(num_drinks,num_heartbreaks);
>global GRAVITY
```

OTHER types of variables

GLOBAL variables:

- If you want the workspace to see this variable, also declare it on the command line (or script that calls the functions).
- Declaration has to happen BEFORE you use the variable.
- CAPITALIZE global variables…

OTHER types of variables

GLOBAL variables:

- ARE NOT ERASED from memory when MATLAB exits the function… so values are retained from one use to the next.
- AYE!!! memory!!

AYE!!! memory!!

Modify scramble as such:

```
function rdow = scramble(word);
global GRAVITY
GRAVITY = 9.81;
t=GRAVITY.*3;
rdow = shuffle(word);
```

RUN IT. WORKSPACE???

AYE!!! memory!!

Now type:

```
> GRAVITY
```

WHAT HAPPENS???

```
> global GRAVITY
```

WHAT's on your workspace?

```
> GRAVITY
> clear GRAVITY   %is it there?
> global GRAVITY %where dit it come from?
```

AYE!!! MEMORY!!!

SO BEWARE OF MEMORY USE with GLOBAL variables.

GRAVITY exists! Even if you have no evidence that it does!

And it can weigh on you memory! (hah!)

Persistent Variables

If you want a function to keep accessing a variable, every time you run it, declare the variable as *persistent.*

In other words, Matlab will NOT erase it upon exiting the function, so it will be able to access it again.

Persistent Variables

Only the function in which you declare a persistent variable can use it (unlike GLOBAL variables).

Only work for functions.

To use:

```
> persistent variable_name
> clear functionname  %to erase
  clear all
  editing (and saving) the function
```

Persistent Variables

Initializing a persistent variable:

```
> function runSum(inc)
> persistent TOTSUM
>
> if isempty(TOTSUM)
       TOTSUM = 0;
> end
> TOTSUM = TOTSUM + inc;
```

let's explore

Call the function a few times.

```
> runSum(1)
> runSum(34)
> runSum(100)

> WHAT's going on in your workspace?
```

let's explore

Declare on your workspace:

```
> global TOTSUM


> runSum(1009)


> TOTSUM
```

let's explore

Allow the function to show TOTSUM's value. (this clears TOTSUM)

Call it again:

```
> runSum(1)
> runSum(34)
> runSum(100)
> TOTSUM        %call from workspace!
```

let's explore

Now clear TOTSUM (from workspace)

> clear TOTSUM

> runSum(4444)

> That's how well protected your function is.

> clear runSum

> runSum(1)

Summary

GLOBAL variables:

-> can be used ANYWHERE…

modified anywhere too!

PERSISTENT variables:

-> allow you to have some values persist in memory from one function use to the next.

-> UNTOUCHABLES!! outside of function.

-> beware of what's lurking under the surface.

If you'd like to know more

For further reading…

> *anonymous* functions (do not require an m-file)

> > fname = @(arglist)expression
>
> > sqr = @(x) x.^2;

TRY:

Wsize = @(name) length(name);

Call it: > Wsize(‘alejo’)

Multi = @(x,y) x.*y;

Call it: > Multi(9,3)

If you'd like to know more

For further reading…

> *subfunctions* are created within a function

> *private* functions: only visible to their parent directory scripts.

> *nested* functions: share variables.

What’s in memory?

- MATLAB matrices.
- FILES with DATA: RTs…
- FILES with TEXT: list of words

- later on: images and sounds.

SAVING AND READING MATRICES

- Use the command ‘save’ to save a matrix in matlab format (.mat), which is ONLY readable with Matlab.
- type:

```
> myworld = ‘is beautiful’;
> yourworld = 0;
> save Worlds;
```

SAVING AND READING MATRICES

- save NAME
  - saves ALL the variables in your workplace, regardless of differences in format, on file NAME.mat
  - Type:

```
> clear all      %CHECK WORKSPACE
                 %CHECK CURRENT DIR.
> load Worlds
```

SAVING AND READING MATRICES

- save NAME var1 var2
  - specifies which variables to save in file NAME.mat
- if you want to ADD stuff to a current mat file:
  - save NAME var1 –append

  %here, *var1* is being added to your
  % MAT file named NAME

SAVING AND READING MATRICES

- wilcard *
- save avariables a* %any guesses?
  - saves all variables starting with a in file avariables.mat

SAVING AND READING MATRICES

- load name

  loads all variables in name.mat.

- load name var1 var2
  - only loads var1 var2

- For further reading, do
  help load
  to learn how to load ascii files into variables!!

you might already know…

- reading text is different than reading numbers, but you can transform one into the other:
  - num2str: transforms numbers into strings
  - int2str: transforms integers into strings
  - mat2str: transforms 2D matrices into strings
  - char: convert to character array.

you might already know...

- reading text is different than reading numbers, but you can transform one into the other:
  - str2double: converts strings to double precision numbers.
  - str2num: converts a character array into a numeric matrix.

READING AND WRITING FILES p.18

- so, you might need to access a file and pull information from it, or print output/data from calculations you’ve performed.
  - print to string: sprintf
  - print to file: fprintf
  - read from string: sscanf
  - read from file: fscanf

let’s do ‘f’iles… (you can do ‘s’ on your own).

- files need to be given a file identifier (to keep track of them) and need to be OPENED.

```
> fid = fopen(filename)
```

fid is file identifier (1, 2, 3..)

-1 if file not found.

FILE has to be in working directory or in PATH.

Open goldilocks.txt

- fid = fopen(‘goldilocks.txt’)
  - what happens?
  - Note you can call fid whatever you want…
- Close goldilocks.txt:
  - fclose(fid)   %0 means success.

- OPEN IT AGAIN both in matlab and NOTEPAD.

What can you do with an open file?

- get first line of text:
- TRY:
  - firstline = fgetl(fid)
    - **%don’t suppress output**.

  - Run the command again.
  - Run the command again.
  - Run the command again. What’s going on?

What can you do with an open file?

- NOW TRY
  - `nextline = fgets(fid)`
  - ANY DIFFERENCES?

At the end…

- of lines, there is a new line symbol (\n) *invisible to mere humans*
  - fgetl reads a line but does NOT copy the end of line character to the string. **fgets** does.

At the end…

- of files, there is an end of file character, which we test with:
  - feof(fid) : 0 while not end of file

    1 once it is found.

Exercise:

- PRINT TO THE SCREEN THE ENTIRE STORY OF GOLDILOCKS.
  - 'while' might come in handy.

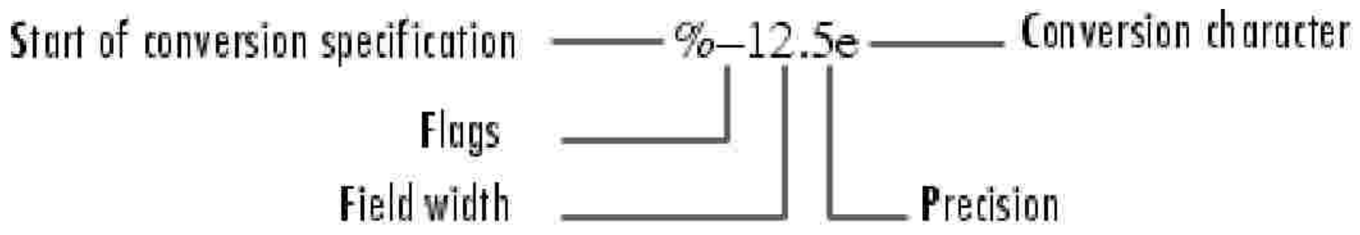
Exercise:

- fid = fopen(‘goldilocks.txt’);
- while feof(fid) ==0

```
   newline = fgetl(fid)
 end
 fclose(fid);
```

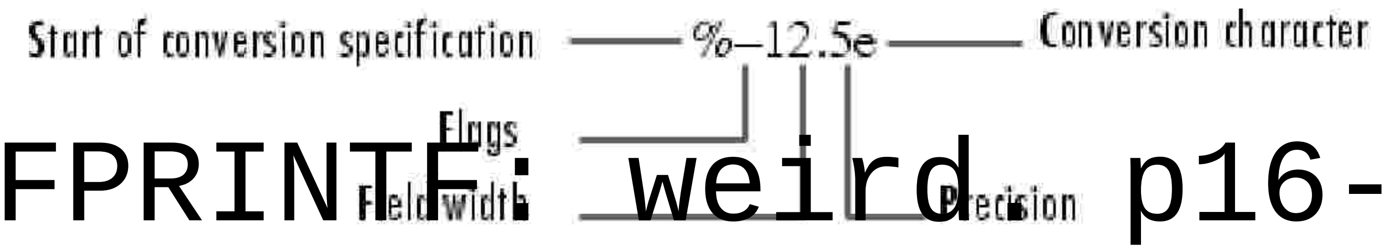
FPRINTF: weird. p16-17

- fprintf allows us to write to a file.
- fprintf(fileID, PRINTING FORMAT, variable).



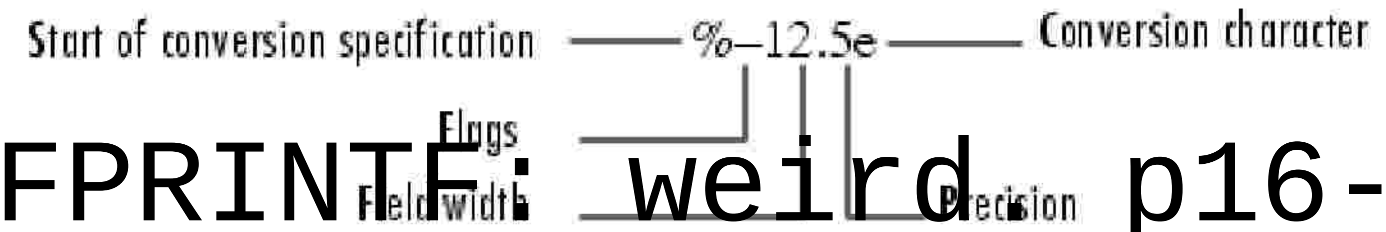
The 'printing format' is where you manage your CURSOR.

FPRINTF: weird. p16-17



- OPEN PAGE 17.
  - Field width: MINIMUM number of digits (spaces) to be printed
  - flags page 17
  - Precision: decimals after period.
  - CURSOR: RESERVES Leftward space
    Overruns *rightwards*.

(more on this later)



FPRINTF: weird. p16-17

- Most common conversion characters.
  - %c single character
  - %d decimal notation
  - %s string of characters
  - %f numbers with periods
  - INSIDE PRINTING area:

    \n new line

    \t horizontal tab

Exercise:

- Rather than printing the text to the screen, let’s transfer it to another file.

```
> newfid = fopen(‘newgoldi.txt’,’w’);
> fprintf(newfid, ‘%s \n’, newline);
```

  - ‘w’ means write to this new file
    %delete previous contents if there are any).
  - ‘r’ means open to read.
  - ‘rt’ reads as text.
  - ‘a’ means append (add at the end).

Exercise:

- fid = fopen(‘goldilocks.txt’);
- newfid = fopen(‘newgoldi.txt’,’w’);
- while feof(fid) ==0

```
    newline = fgets(fid);        %why s?
    fprintf(newfid, ‘%s’, newline);
  end
  fclose(fid);
  fclose(newfid);           %HOW TO with fgetl?
```

If you want to know more

About reading text files check out the functions:

textscan & textread

Very useful when you are reading txt files that mix DATA with strings (like words describing conditions)!!

More on this later...

Exercise

```
> fid = fopen('goldilocks.txt');
> firstline = fgetl(fid);
```

FIND SPACE CHARACTERS IN FIRST LINE.

`> spaces = find(firstline == ' ')`

```
%Double check with your array editor
```

BREAK!!

Be back at 11:00 a.m.