PSYCH 20A (PROF. FRANE) – LECTURE 2 PART A: Basic Functions & Logical Operations

Some basic computational functions

Functions take inputs in the parentheses, and produce outputs. Imagine `y` is either a single value or a vector:

`min(y)`	outputs the minimum value in vector `y`
`max(y)`	outputs the maximum value in vector `y`
`sum(y)`	outputs the sum of values in vector `y`
`mean(y)`	outputs the mean (the average) of the values in vector `y`
`numel(y)`	outputs the number of elements (number of values) in `y`
`sqrt(y)`	outputs the square root of each value in `y`
`abs(y)`	outputs the absolute value of each value in `y`
`flip(y)`	outputs the values in vector `y` in reverse order
`sort(y)`	outputs the values in vector `y` in ascending order
`round(y)`	outputs the values in `y` rounded to the nearest whole number
`floor(y)`	like the `round` function, but never rounds upwards, e.g., `floor(8.9)` outputs 8
`ceil(y)`	like the `round` function, but never rounds down, e.g., `ceil(8.1)` outputs 9
`fix(y)`	like the `round` function, but always rounds toward zero; e.g., `fix(1.9)` outputs 1, and `fix(-1.9)` outputs –1

Example: Define a vector called `transitMin` and then define `meanTransitMin` as the mean of that vector.

```
transitMin = [15, 17, 5, 7, 8, 22, 40, 2, 18, 6, 23, 11, 15, 16, 35]
meanTransitMin = mean(transitMin)
```

Basic logical operations: A logical 1 means true, and a logical 0 means false.

`x = 4`	before demonstrating logical operations, let's set x equal to 4
`x > 3`	we ask if the condition "x>3" is true or false (this outputs 1 because it's true that x > 3)
`x < 3`	we ask if the condition "x<3" is true or false (this outputs 0 because it's false that x < 3)
`x >= 3`	we ask if the condition "x is greater than or equal to 3" is true or false (this outputs 1)
`x <= 3`	we ask if the condition "x is less than or equal to 3" is true or false (this outputs 0)
`x ~= 3`	we ask if the condition "`x` is not equal to 3" is true or false (this outputs 1)
`x == 2+2`	we ask if the condition "x equals 2+2" is true or false (this outputs 1); we can't use the single equals sign here, because "x=2+2" means we're *setting* `x` equal to 2+2
`x > 3 & x < 5`	we ask if `x` is greater than 3 AND less than 5 (this outputs 1)
`x < 3 | x > 5`	we ask if `x` is less than 3 OR greater than 5 (this outputs 0)
`g = [8, 50, 25, 80]`	now let's define a vector called `g`
`g(2) > 40`	outputs 1 because the 2nd value in `g` is greater than 40

Applying logical operations to each value in a vector:

`g > 30`	outputs the logical vector [0 1 0 1] (meaning false true false true) because the 1st value in `g` isn't greater than 30, the 2nd value is, the 3rd value isn't, and the 4th value is
`h = [4, 20, 30, 90]`	now let's define a vector called `h` that has the same length as `g`
`g > h`	outputs the logical vector [1 1 0 0] (meaning true true false false) because `g(1)` is greater than `h(1)`, and `g(2)` is greater than `h(2)`, and `g(3)` isn't greater `h(3)`, and `g(4)` isn't greater than `h(4)`; notice we're comparing *corresponding values* (value with the same index) in the two vectors

We can use logical indexing instead of numeric indexing:
`g(g > 30)` outputs the vector [50 80] (the values in `g` that are greater than 30)

Remember, "g > 30" is the logical vector [0 1 0 1]. When we say `g(g > 30)` we're saying "give me the values of `g` at indexes 2 and 4" (i.e., "give me the values of g from the indexes where the 1s are in [0 1 0 1]").

Using the `sum` and `mean` functions with logical values:
Because Matlab treats true as 1 and false as 0, we can use the `sum` function to compute *how many* values in a vector a condition is true for (we're just adding up the 1s in the logical vector):

`sum(g > 30)` outputs 2 (the number of values in `g` that are greater than 30)

Similarly, we can use the `mean` function to get the *proportion* of values in a vector a statement is true for (we're just adding up the 1s in the logical vector and dividing by the number of values):

`mean(g > 30)` outputs 0.5000 (the proportion of values in `g` that are greater than 30)

The `find` function (gives the indexes of values for which a statement is true)

`find(g > 30)` outputs [2 4] because the 2nd and 4th values in `g` are greater than 30

Some functions that output logical values

`ismember(25, g)` asks if at least one value in `g` is 25 (this outputs 1)
`isequal(g, h)` asks if the entire vector `g` is identical to the entire vector `h` (this outputs 0); don't confuse this with "g == h", which compares the *corresponding values* in vectors `g` and `h`, and thus will output a *vector* of logical values rather than a single 1 or 0

`any(g > 30)` asks if any value in `g` is greater than 30 (this outputs 1)
`all(g > 30)` asks if all values in `g` are greater than 30 (this outputs 0)
`any(g > h)` asks if any value in `g` is greater than the corresponding value in `h` (this outputs 1)
`all(g > h)` asks if all values in `g` are greater than the corresponding values in `h` (this outputs 0)

Remember, "g > h" is the logical vector [1 1 0 0]. So saying `any(g > h)` asks if there are any 1s in the vector [1 1 0 0]. And saying `all(g > h)` asks if all the values in the vector [1 1 0 0] are 1s.

Tips for using Matlab's command window

- For brief help with a Matlab function, type `help` followed by the name of the function (e.g., `help flip`). Or for more detailed documentation (including examples), type `doc` followed by the name of the function. And of course, there's always Google.

- To re-enter the previous line you entered, press the up-arrow. This is useful when you want to repeat what you just did, perhaps with a slight modification (e.g., if you made a small typo the first time). You can also press the up-arrow multiple times to re-enter a line you typed several lines ago.

- To clear the screen of the command window, type `clc`

- To list all the variables that are currently defined and their characteristics, type `whos`

- To clear all the variables that are currently defined, type `clear`

- To keep the output of a command from printing to the command window, end the command with a semicolon: `x = 4 ;`

- If Matlab stops responding while performing a task, try CTRL-C to abort the current operation.

PSYCH 20A – LECTURE 2 TOPICS PART B: Writing Programs ("Scripts")

So far, we've mainly been entering code in the "command window," which lets us type a single line of code and run it instantly. But sometimes we want to write a whole sequence of lines of code and then keep that sequence around so we can edit it or run it again later. That sequence is called a "script" or "program," which is created in the "editor window."

To start writing a program, click on the "New" drop-down menu in the editor window and select "Script." Or type `edit` followed by the name of the program you want to use, e.g., `edit MyDemoProgram`

To save a program, click on the "Save" drop-down menu in the editor window and select "Save." Matlab will add a `.m` extension to the filename. Don't use spaces in the filename, or it won't run!

To open a program that you created earlier, click the "Open" drop-down menu in the editor window. Or go to the directory where the program is, and then type `edit` followed by the name of the program (don't include the `.m` extension).

Unlike in the command window, the code you type in the editor window doesn't automatically run when you finish a line. To run your code, first save it, then click the "Run" button in the editor window. Or from the command window, type the name of the program (without the `.m` extension); to do this, the current directory must be the one where the program is saved or must be added to the search path (see below).

Navigating directories

To run a program, you must set the current working directory ("folder") to the directory where the program file is (or you can add the program's directory to the search path; to do that, click on "Set Path" from the Home tab and locate the directory you need). You can change the current directory using the address bar at the top of the command window, or by using the `cd` function (`cd` stands for "change directory").

`cd`	outputs the name of the current working directory
`cd MyPrograms`	changes the current working directory to the folder `MyPrograms`
`cd /Users/YourName/Downloads`	changes the current working directory to the given pathname; the exact path depends on your system, e.g., on a PC the path may be something like `c:/Users/UserName/Downloads`

`cd ..`	changes the current working directory to one folder up
`cd ../..`	changes the current working directory to two folders up; e.g.,if the current working directory is `/Users/UserName/Downloads` then typing `cd ../..` will change the current working directory to `/Users`

`dir`	lists the files and folders in the current working directory
`dir *.m`	lists the files and folders in the current working directory that end in `.m`
`dir af*`	lists the files and folders in the current working directory that start with `af`
`dir /Users/YourName/af*`	lists the files and folders in folder `/Users/YourName` that start with `af`

Using comments

Comments are little notes you write in the program to explain what the code is doing. The comments themselves don't actually "do" anything, because Matlab ignores them. But they can help someone else (or help you when you look at your code months later) to follow what's going on. To write a comment in your program, just type the % symbol, followed by your comment.

Typically, we use an "in-line" comment, which goes at the end of the line it refers to:

```
timeInSeconds = timeInMinutes * 60 ; % convert minutes to seconds
```

But we can also use an "above-line" comment. This is useful when the comment wouldn't fit on the screen when placed in-line:

```
% vector of transit times in minutes
transitMin = [15, 17, 5, 7, 8, 22, 40, 2, 18, 6, 23, 11, 15, 16, 35] ;
```

If you use an above-line comment, typically you should skip a line before the comment for clarity.

Tips for writing programs

- Begin the program with opening comments that clearly describe what the program does.
- Use comments throughout the program to explain what each part of the program does. Typically, almost every command should have a comment.
- Make the first non-comment line in the program `clear` (to clear any previously defined variables).
- Use spaces to help readability; for example, `x = y^2 + 3*z – 1` is clearer than `x=y^2+3*z-1` (though both will work). But **when indexing, don't put a space between the variable name and the open parenthesis**; that can cause problems.
- Insert blank lines between different sections of code to help clarity
  (Matlab ignores any extra spaces and line-breaks, except in certain cases, such as within quotes).
- Use descriptive variable names that are easy to understand (e.g. `depressionScore` rather than `ds`)
- Avoid giving variables names that are also the names of functions (e.g., `mean`).
- To keep the command window from getting cluttered when defining variables, you typically want to **suppress output to the command window by ending lines with a semicolon**: `x = 4 ;`
- When you open the editor window, the command window gets really small; you can expand or contract the command window by double-clicking on the top bar of the command window.
- Check your work, e.g., for each object you define, "look" at it (by typing the name of the object into the command window) after you run the program; verify that the object's size and values make sense.
- Be careful with your syntax; one misplaced comma or parenthesis can have devastating effects!
- Pay attention to red and orange markups that Matlab puts in your code in the editor window. Sometimes Matlab will underline part of your code in red and will place a corresponding red mark in the right-hand margin. That indicates a syntax error in your code. Hover over the underline or mark to see the problem, or click on the mark to jump to the problematic part of the code. Orange underlines and marks indicate that something, though not an outright syntax error, looks unorthodox in some way and thus might be a mistake.
- Programs often don't work at first, due to typos or other bugs; don't be discouraged when you get error messages! Debugging is just part of the process.