Programming 8 Pre-Lab - MATLAB Functions

In your MATLAB assignments, we have covered the following:

- Access MATLAB, create an M-File, and submit that M-File and its output to Blackboard
- Develop a basic code from a pseudo-code or flowchart
- Comment your code properly
- Basic Code Debugging
- Understand some basic Built-in Math functions
- Scalars, Arrays, and Matrices
  - Creating them as Variables
  - array & matrix math
  - Managing and indexing
- Creating arrays and use array math
- Input data from the command window or files
- Output data to the command window or a figure
- Relational and Logical Operators
- Conditional Statements (IFTT)
- For Loops

Some Useful Commands:

As a reminder, here are some useful commands:

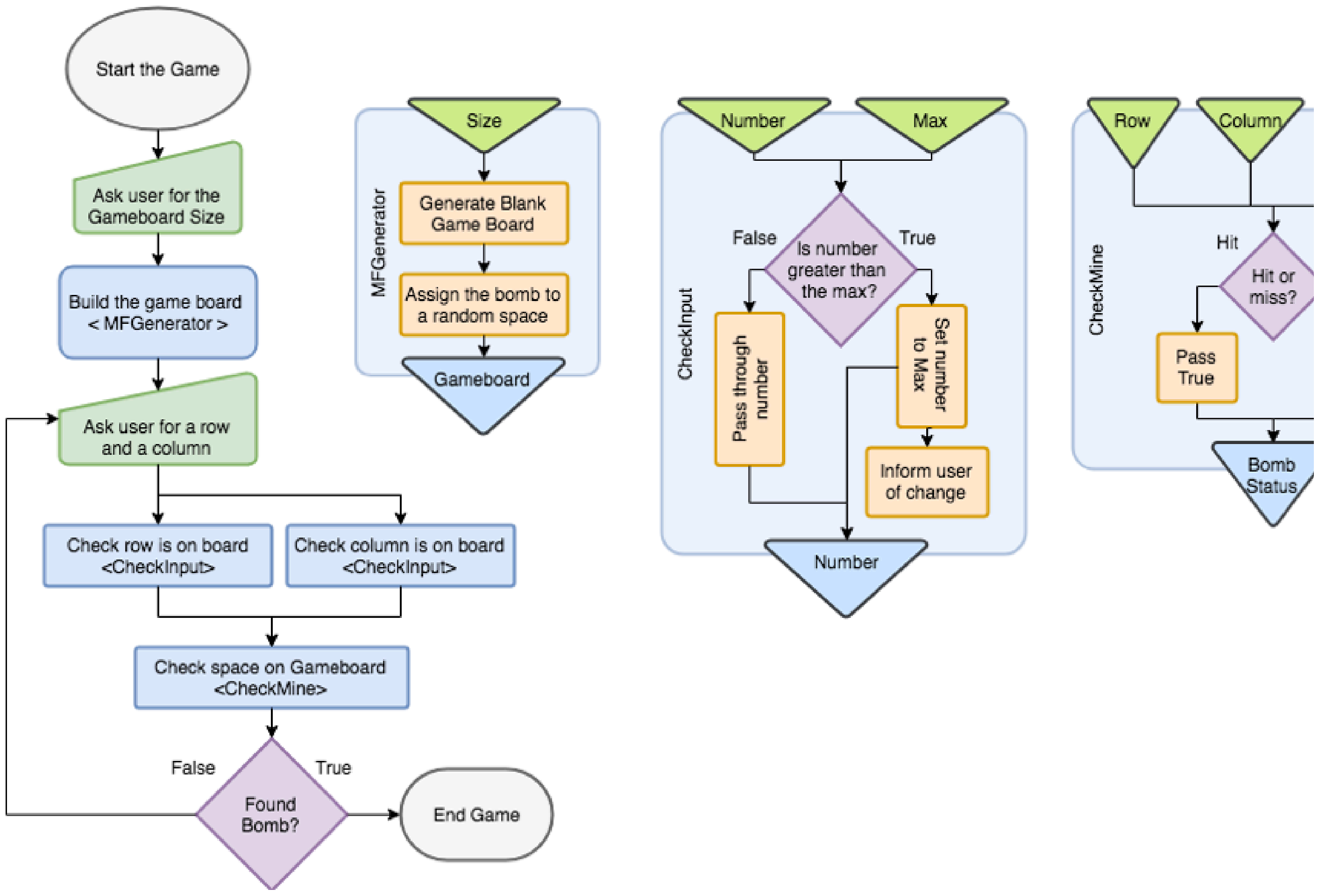
- To comment out a line or a selection of multiple lines
  - PC: Ctrl + r
  - Mac: Cmd + /
- To force a script to end:
  - PC: Ctrl + c or Ctrl + Break
  - Mac: Ctrl + . (period)

Representing Functions

Functions are isolated segments of code that are typically called with an input and return an output. They are particularly useful for when you have a segment of code repeated throughout your program. So instead of calling the same 10-15 lines of code over and over again with different inputs and outputs, you can set those 10-15 lines of code aside and set it up so that each instance is replaced by 1 line of code.

Let's use a simple minefinder game to illustrate this and showcase Function representation.

Functions as a Flowchart


Start the Game
Ask user for the Gameboard Size
Build the game board < MFGenerator >
Ask user for a row and a column
Check row is on board <CheckInput>
Check column is on board <CheckInput>
Check space on Gameboard <CheckMine>
False
True
Found Bomb?
End Game
Size
MFGenerator
Generate Blank Game Board
Assign the bomb to a random space
Gameboard
Number
Max
CheckInput
False
True
Is number greater than the max?
Pass through number
Set number to Max
Inform user of change
Number
Row
Column
CheckMine
Hit
Hit or miss?
Pass True
Bomb Status

Flowchart Representation

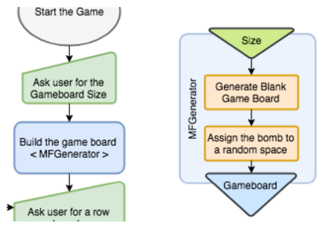
The above is an example of of a minefinder game that has the main code and 3 functions; MFGenerator, CheckInput, and CheckMine. Note that each of the functions is labelled in some way. They also have a border to visually indicate that they are isolated. An arrow shape in and out of that border indicates the inputs and outputs. The blocks that use the functions also indicate that as well.

Take particular not of *CheckInput*. It's used in 2 locations in the main code using different inputs. That's the primary benefit of functions. If you're reusing code, you can replace it with a function. In both cases there's a number, a limit for that number, and a need to check to make sure that value is within the limit. In the first case, we're checking it against the available rows and in the second case, we're checking it against the available columns. It's just using different variables but the same actions. The pseudocode section will really illustrate this.

Functions as a Pseudocode

Let's lay out the logic in more detail to illustrate how to create functions and when.

Let's go over how you should represent functions in your logical representations. Now you usually don't want to include functions at first, only when you start recognizing patterns or areas where you can offload some complexity to an isolated function. It is clear cut here because of the assignment requirements but I'll walk you through the reasoning.


Start the Game
Ask user for the Gameboard Size
Build the game board < MFGenerator >
Ask user for a row
Size
MFGenerator
Generate Blank Game Board
Assign the bomb to a random space
Gameboard

First, let's talk about how we represent functions. Some of the language will become clear once we go over them. You can see how they're represented in a flowchart from above. You give them their own flowchart with inputs at the top and outputs at the bottom. In the main flowchart, include a process block with an indicator of what function is being called, *MFGenerator* in this case. So essentially you're just making a flowchart to further detail a process in your main program.

What about in pseudocode though? Well you have your main program and then you have functions that your program can call. So we denote our main program with `PROGRAM` and we denote the functions with `FUNCTION`. When we refer to them in the text, we say `CALL <function name>`. You can also use `GET <function output> FROM <function name>`. And if you need to include inputs for your function, add `WITH <the inputs>` when you're capturing an expected output. In the function pseudo code, you should use `PASS IN <inputs>` and `PASS OUT <outputs>` for handling the information being taken in or sent out.

Let's talk walk through determining when to use a function while showing some pseudo-code examples. First of all, there's never a hard need for a function. You can repeat code through copy-paste and slight edits. You do end up with a long and unwieldy code that's all about the brute force. I am fine with brute force but you can't rely purely on that. Let's take a look at the beginning of the problem, skipping to where you check the user inputs to see if they're on the game-board:

```
PROGRAM ML3-demo:
...
Skipping to the point where ONE function is used in TWO circumstances
...
Ask user to guess a row and a column value
IF the row guessed is above the maximum number of rows
    Inform user that the program will use the maximum value of rows instead of their guess
    Set row used to the maximum value of rows
ELSE
    Use the row guessed
ENDIF
IF the column guessed is above the maximum number of columns
    Inform user that the program will use the maximum value of columns instead of their guess
    Set column used to the maximum value of columns
ELSE
    Use the column guessed
ENDIF
...
END
```

Notice the similarity between the two If statements. If you replace the words `row` or `column` with a more generic term `VALUE INPUT` and `maximum number of rows` or `maximum number of columns` with `MAX VALUE`, you end up with the same exact logic.

```
IF VALUE INPUT guessed is above the MAXIMUM VALUE
    Set INPUT used to the MAXIMUM VALUE
    Inform user that the program will use the MAX VALUE instead of their guess
ELSE
```

Since we're repeating steps with just different inputs, we can isolate that into its own function. I use `GET` because I want an output from the function and not just `CALL`ing it.

```
PROGRAM ML3-demo:
...
You know the maximum possible value of rows and of columns from the generated game field;
```

```
Ask user to guess a row and a column value;
GET row value FROM CheckInput WITH row guess and max rows;
GET column value FROM CheckInput WITH column guess and max columns;
...
END

FUNCTION CheckInput:
PASS IN: VALUE INPUT and MAX VALUE;
IF VALUE INPUT is greater than MAX VALUE
    Let user know the input is changing;
    PASS OUT: MAX VALUE;
ELSE
    PASS OUT: VALUE INPUT;
ENDIF
ENDFN
```

Now in the case of 2 instances of 6 lines of pseudocode, I've reduced it to 2 lines in the main pseudocode and added 11 lines of a function. Yes, this is only a reduction of 1 line total but what if it was a larger block of logic and more instances, like 20 lines of pseuodocode in 4 places that got put into a 28 line function. Then you can save yourself even more lines , 80 lines to 32 lines. And that's just the pseudocode. Image how much actual programming you saved yourself by recognizing a pattern then accounting for it in your logic.

Functions are not always created for a repeated task though. Sometimes they're done to remove some complexity from the main program and set it aside. That's the reasoning behind including MFGenerator and CheckMine. These are isolated actions of the program. You can easily test them on their own and then set them aside once they are working allowing you to continue working on a less cluttered program. This takes time to recognize when these might be better set aside in a function. It may even be a decision you make after you've started programming. In this lesson, we will be taking a familiar complex block of code and setting aside to simplify our main program.

Question 1 Show Correct Answer Show Responses

Multiple answers: Multiple answers are accepted for this question

When representing a function use in your main script's logic, what do you need to clarify?

A The inputs

B	The outputs
C	Its name
D	How it works

Functions in MATLAB

We've already dealt with functions in MATLAB. You've used `fprintf()` and other built-in functions already. We're going to create our own now. Let's look to the example provided with this assignment

https://raw.githubusercontent.com/boconn7782/CourseCode/main/loan.m

You can either download this or open a new m-file and save that content into it. Make sure you name that file **loan.m** and save it in your *Current Folder* though. You now have a user-generated function at your disposal. Run the following command

Parts of a MATLAB Function:

Let’s break this down line-by-line to illustrate how a user-defined function is created:

Function definition line:

The first line defines that it is a function, the outputs, the name of the function, and the inputs. It takes the form `function [y1,...,yN] = myfun(x1,...,xM)`. The first word(`function`) of the M-File will always be **function**. The array that follows(`[y1,...,yN]`) are the outputs. On the other side of the equal sign (`=`), the single word is the name of the function(`myfun`) and the name by which you would call the function. The second array(`(x1,...,xM)`) indicates the expected inputs to the function.

For `loan.m`, the first line is `function [mpay,tpay] = loan(amount, rate, years)`. This indicates that it has 2 outputs, `mpay` and `tpay`. The name of the function is `loan` which matches the name of the file. It also has 3 inputs - `amount`, `rate`, and `years`. To call the function, you must use the name and include the required inputs. MATLAB will only recognize a user-defined function if it's in the current folder or if it's in your custom path. We won't get into custom paths but it allows you to call your user-defined function no matter what folder you're working in.

NOTE: When you are saving an M-File as a function, you must name the file the same as the function name you want to use.

Help Text:

The next series of commented lines is the help text. If you run the command `help loan`, it'll print out in the command window. If you run the command `doc loan`, it'll open up the MATLAB File Help window with that text in it. You should include some description and instruction for any function you create. Run the following to see how it works:

1. `help loan`

Question 2 Show Correct Answer Show Responses

MATLAB recognizes an m file as a function file when the first line of code start with:

Function Body(program)

This is lines 12-15 of **loan.m**. This is where you put the inputs to use as part of the actual program that makes up the function. For instance, when we called `loan(40000,.03,7)` the function assigns `amount` equal to `40000`, `rate` equal to `.03`, and `years` equal to `7`, based on their order as defined in the function file and the order they are called when the function is used. If you had variables `X`, `Y`, and `Z` in your program, you could call `loan(X,Y,Z)` and the function would run after assigning the values of those variable to the values of the corresponding input variables in the function. **It's the value that's passed**, not the variable itself.

Question 3

If I run a function command using the variable X as one of the inputs, that function now has access to my variable X and can make changes to it.

A	True
B	False

HINT

EXPLANATION

Show Responses   Show Answer

Assignment of Values

For the function to be complete, you need to assign a value to all outputs. This is the value that gets passed out of the function to whatever program called it. If you do not assign a value, it will give you an error.

Function definition examples

Function Definition	Outputs	Inputs
`function [wkly, yrly] = payroll(emp,rate)`	2	2
`function [V, S] = SphereVolArea(r)`	2	1
`function [A] = RectArea(a,b)`	1	2
`function A = RectArea(a,b)`	1	2

`function trajectory(v, h, g)`	0	3

NOTE: Notice how when there is only a single output, you don't need the square brackets. You can still use them but they are not necessary in that case. You also don't need to have an output.

Question 4

Show Correct Answer Show Responses

Loan.m has ________ inputs and ________ outputs.

Variables in Functions

This is something lots of novice programmers have issue with so it's being mentioned again. When you call functions in MATLAB, you're entering a different domain. You can pass information in through variables being used as inputs but even then, you're only passing along their values. The function and the program that calls do not share variables. If you set a variable in one, you cannot just call it in the other.

- Variables are inherently local. They are only known to the script or function where they are created.
- You can declare a variable as global but that is bad practice at this stage. Don't do it.
- You can use the same variable names in your script and in the function but they won't be the same thing.

Example:

```
Function [dee, dum] = tweedle(x,k,c,d)
```

Called in program:

```
[M,N] = tweedle(f,g,h,i);
```

Question 5

Show Correct Answer Show Responses

What is the only acceptable name for the m-file defined by this line:
function [dee, dum] = tweedle(x,k,c,d)

CODING REQUIREMENTS FOR FUNCTIONS:

- **DO NOT** use the same variables in your program and functions.
  - There is nothing stopping you from doing so other than my policy that you do not do this. I require this so that I'm sure you understand that there is a difference between the variables in the function and the main script.
- **DO** include your name and the assignment in the help text.
  - Treat the first few lines of the function help text as your title block for that m-file.

Now onto the activity:

Part 1 - WeatherData Function

We are going to take your weather data plotter from the first MATLAB lab and pull the support code out and make it it's own function. We'll start with your weather plotting script. Just so we're all on the same page, use the following:

https://raw.githubusercontent.com/boconn7782/CourseCode/main/ML3L1_Base

The naming convention is from when I delineated things by application not general programming or graphics. You're going to rename it anyways, so I've just kept it for now.

For space purposes, I won't be copying all of the code into this document as I go over creating a function from this, so expect some discrepancies.

```
...
close; clc; clear;

% Introduction ...
disp( ... );

```

```
%% Ask user for Time Period
% User inputs for the date and number of days of data
D = input('Enter a date: ','s');
N = input('Enter number of days: ');



%% Load Temperature and Wind Velocity data
% Connect with GHCN and download raw data
% Prepare and format all information as required to use
% the GHCN (Global Historical Climatology Network)
% https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/readme.txt
dataset =  'daily-summaries';
station = 'USW00014755'; % Mt. Washington station ID
% For more information on this weather station:
% https://www.ncdc.noaa.gov/cdo-
web/datasets/GHCND/stations/GHCND:USW00014755/detail
startDate = char(datetime(D,'InputFormat','yyyy-MM-dd','Format','y-
MM-dd'));
endDate =   char(datetime(D,'InputFormat','yyyy-MM-dd','Format','y-
MM-dd')+(N-1));
dataTypes = 'TMAX,TMIN,AWND';
format =    'csv';
base = 'https://www.ncei.noaa.gov/access/services/data/v1?dataset=';
call = strcat(base,dataset,'&dataTypes=',dataTypes,...
    '&stations=',station,'&startDate=',startDate,...
    '&endDate=',endDate,'&format=',format);
% Make a request of the GHCN and store the returned data
Data = webread(call);

% Break out the data needed:
% Dates(DT), wind speeds(WD), low temperatures(LT), and high
temperatures(HT)
DT = Data.('DATE');
% Wind provided in 10ths of meters per second
WD = (Data.('AWND')*.1)*(1000/(60*60)); % Converted to kph
% Temperature provided 10ths of degrees C
LT = (Data.('TMIN')*.1); % Converted to deg C
HT = (Data.('TMAX')*.1); % Converted to deg C
```

```
42.
43.
44.
45. %% Calculate daily Avg. Temp and Wind Chill Factor
46. % Calculate Daily Avg. Temp
47. AT = (HT + LT)/2;
48.
49. % Calculate Wind Chill Factor
50. WCF=35.74 + 0.6215.*AT - 35.75.*WD.^0.16 + 0.4275.*AT.*WD.^0.16;
51.
52. %% Plot Temperatures and wind chill factor
53. % Plot Avg Temp and WCF vs dates
54. plot(DT,AT,'b-.o',DT,WCF,'r--d')
55. % Update figure with title and labels
56. legend('Average Temperatures','Wind Chill Factors');
57. xlabel('Day');
58. ylabel('Temperature(°F)');
59. title('Temperatures and Wind Chill Factors Mt. Washington');
```

Note that there is still a lot of code, primarily the code that handles `%% Load Temperature and Wind Velocity data` by communicating with the GHCN. We are going to push that into it's own function. Let's go through the steps.

1. **Identify the code you want to set aside into a function**
   1. Already done, I've told you what to take out. The section `%% Load Temperature and Wind Velocity data`. Just this section, we'll leave `%% Calculate daily Avg. Temp and Wind Chill Factor` and beyond in the main script
2. **Identify the inputs necessary**
   1. That script uses 2 user inputs so we'll use those as our inputs (Lines 8 & 9)
   2. So we'll want those commands to stay in our main code since we'll use the values from `D` and `N` as inputs to our function call.
3. **Identify the outputs**
   1. We need the array for the winds and temperatures for calculations and the dates for plotting so those have to make their way out of our function.
   2. So we'll need to make sure we capture `DT`, `WD`, `LT`, and `HT` in the function and return their values as outputs
4. **Identify where you use them**
   1. We start using them in line 47 so that's the latest line we have to keep
   2. We'll keep the comments in the main script before it as well so we'll cut before that

Now that we've identified that lines 14 - 41 above can be transferred to a function as long as we keep track of the inputs and outputs.

Start a new function

Time to create a place to put that code

- Start a new m-file
- Declare it as a function - `function [DT,LT,HT,WD] =  weatherData(D,N)`
  - Proclamation: `function`
  - Outputs: `[DT,LT,HT,WD]`
  - Name: `weatherData`
  - Inputs: `( D, N )`
- Save the file as `weatherData.m`
  - This is the only allowable filename since it must match the function name

You should now have an m-file that looks like this (You'll have to manually add the `end`):

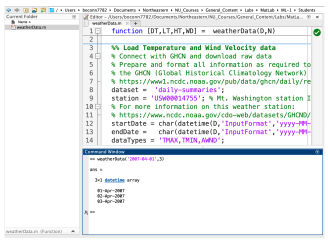
```
function [DT,LT,HT,WD] =  weatherData(D,N)

end
```

Take **JUST** the code you want to replace with this function from your main script and put it into the function. **DO NOT** copy over all of the code, we just want the stuff that handles the connection to the NOAA database and gets you the desire arrays of `DT,LT,HT,WD`. We don't have to worry about the actual variable names for the inputs and outputs right now because we used the original ones for creating our initial function file.

```
function [DT,LT,HT,WD] =  weatherData(D,N)

%% Load Temperature and Wind Velocity data
% Connect with GHCN and download raw data
% Prepare and format all information as required to use
% the GHCN (Global Historical Climatology Network)
% https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/readme.txt
dataset =  'daily-summaries';
...
% Dates(DT), wind speeds(WD), low temperatures(LT), and high temperatures(HT)
DT = Data.('DATE');
```

```
% Wind provided in 10ths of meters per second
WD = (Data.('AWND')*.1)*(1000/(60*60)); % Converted to kph
% Temperature provided 10ths of degrees C
LT = (Data.('TMIN')*.1); % Converted to deg C
HT = (Data.('TMAX')*.1); % Converted to deg C

end
```

Let's test your function by calling it in the command window, trying



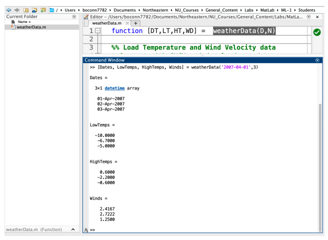
```
function [DT,LT,HT,WD] =  weatherData(D,N)

%% Load Temperature and Wind Velocity data
% Connect with GHCN and download raw data
% Prepare and format all information as required to
% the GHCN (Global Historical Climatology Network)
% https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/re
dataset =  'daily-summaries';
station = 'USW00014755'; % Mt. Washington station I
% For more information on this weather station:
% https://www.ncdc.noaa.gov/cdo-web/datasets/GHCND/
startDate = char(datetime(D,'InputFormat','yyyy-MM-
endDate =   char(datetime(D,'InputFormat','yyyy-MM-
dataTypes = 'TMAX,TMIN,AWND';
```

```
>> weatherData('2007-04-01',3)

ans =

  3×1 datetime array

   01-Apr-2007
   02-Apr-2007
   03-Apr-2007

>>
```

just calling `weatherData('2007-04-01',3)`. You should end up just seeing 1 array of information as the output. Only the first output variable is automatically exported when you call a function.

MAKE SURE `weatherData.m` is in your Current Folder when you try to run it as a command.

To capture all outputs, you need to set up an matching array of variables to



```
function [DT,LT,HT,WD] =  weatherData(D,N)

%% Load Temperature and Wind Velocity data
```

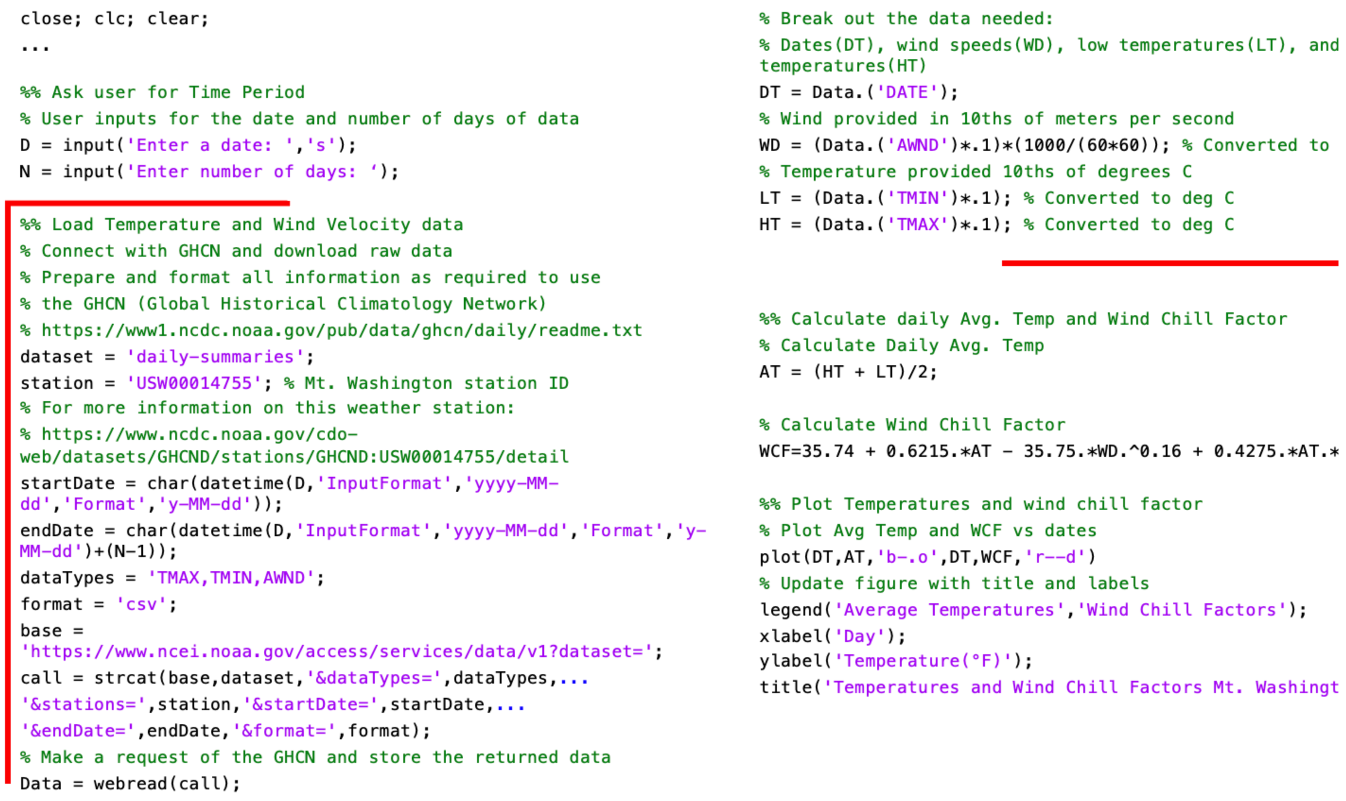
```
>> [Dates, LowTemps, HighTemps, Winds] = weatherData('2007-04-01',3)

Dates =

  3×1 datetime array

   01-Apr-2007
   02-Apr-2007
   03-Apr-2007


LowTemps =

  -10.0000
   -6.7000
   -5.0000


HighTemps =

    0.6000
   -2.2000
   -0.6000


Winds =

    2.4167
    2.7222
    1.2500

>>
```

capture that information. Since `weatherData` has 4 outputs, you need to provide an array of 4 variables to capture all of them, as illustrated here. Notice that each of the variables used in that array to capture weatherData's output contains an array of information from the functions corresponding outputs.

Call the function in your main script

Now we can replace a large chunk of complicated code in your main script with a single function call.

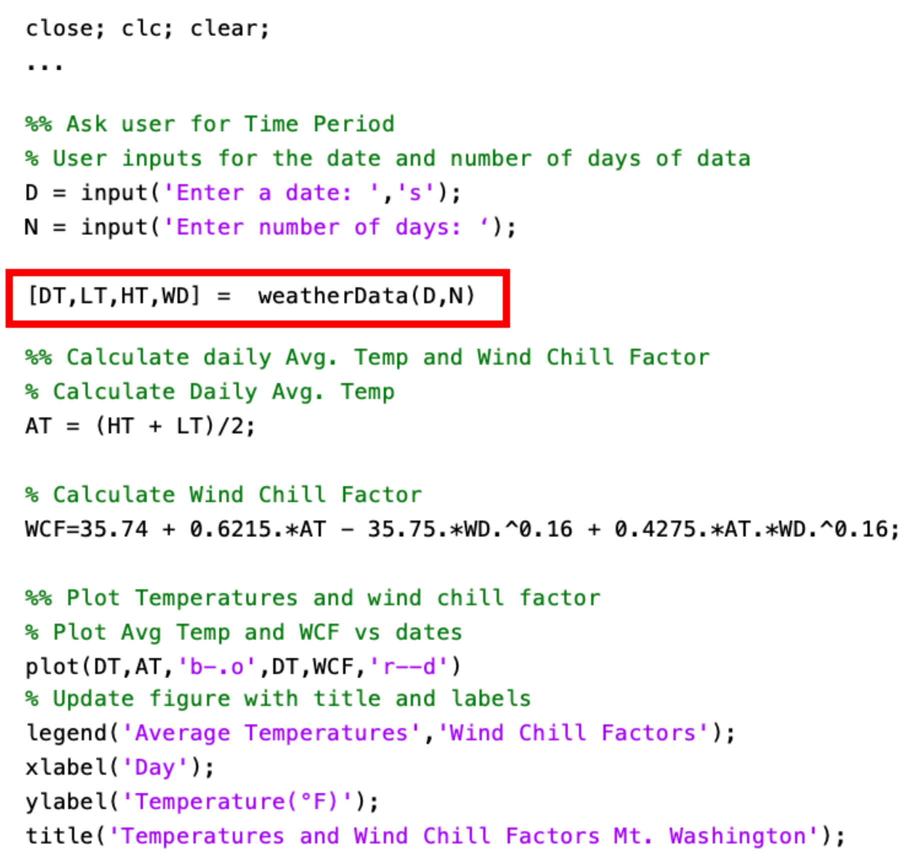
So this:



```
close; clc; clear;
...

%% Ask user for Time Period
% User inputs for the date and number of days of data
D = input('Enter a date: ','s');
N = input('Enter number of days: ‘);

%% Load Temperature and Wind Velocity data
% Connect with GHCN and download raw data
% Prepare and format all information as required to use
% the GHCN (Global Historical Climatology Network)
% https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/readme.txt
dataset = 'daily-summaries';
station = 'USW00014755'; % Mt. Washington station ID
% For more information on this weather station:
% https://www.ncdc.noaa.gov/cdo-
web/datasets/GHCND/stations/GHCND:USW00014755/detail
startDate = char(datetime(D,'InputFormat','yyyy-MM-
dd','Format','y-MM-dd'));
endDate = char(datetime(D,'InputFormat','yyyy-MM-dd','Format','y-
MM-dd')+(N-1));
dataTypes = 'TMAX,TMIN,AWND';
format = 'csv';
base =
'https://www.ncei.noaa.gov/access/services/data/v1?dataset=';
call = strcat(base,dataset,'&dataTypes=',dataTypes,...
'&stations=',station,'&startDate=',startDate,...
'&endDate=',endDate,'&format=',format);
% Make a request of the GHCN and store the returned data
Data = webread(call);
```

```
% Break out the data needed:
% Dates(DT), wind speeds(WD), low temperatures(LT), and
temperatures(HT)
DT = Data.('DATE');
% Wind provided in 10ths of meters per second
WD = (Data.('AWND')*.1)*(1000/(60*60)); % Converted to
% Temperature provided 10ths of degrees C
LT = (Data.('TMIN')*.1); % Converted to deg C
HT = (Data.('TMAX')*.1); % Converted to deg C


%% Calculate daily Avg. Temp and Wind Chill Factor
% Calculate Daily Avg. Temp
AT = (HT + LT)/2;

% Calculate Wind Chill Factor
WCF=35.74 + 0.6215.*AT - 35.75.*WD.^0.16 + 0.4275.*AT.*

%% Plot Temperatures and wind chill factor
% Plot Avg Temp and WCF vs dates
plot(DT,AT,'b-.o',DT,WCF,'r--d')
% Update figure with title and labels
legend('Average Temperatures','Wind Chill Factors');
xlabel('Day');
ylabel('Temperature(°F)');
title('Temperatures and Wind Chill Factors Mt. Washingt
```

Becomes this:



```
close; clc; clear;
...

%% Ask user for Time Period
% User inputs for the date and number of days of data
D = input('Enter a date: ','s');
N = input('Enter number of days: ');

[DT,LT,HT,WD] =  weatherData(D,N)

%% Calculate daily Avg. Temp and Wind Chill Factor
% Calculate Daily Avg. Temp
AT = (HT + LT)/2;

% Calculate Wind Chill Factor
WCF=35.74 + 0.6215.*AT - 35.75.*WD.^0.16 + 0.4275.*AT.*WD.^0.16;

%% Plot Temperatures and wind chill factor
% Plot Avg Temp and WCF vs dates
plot(DT,AT,'b-.o',DT,WCF,'r--d')
% Update figure with title and labels
legend('Average Temperatures','Wind Chill Factors');
xlabel('Day');
ylabel('Temperature(°F)');
title('Temperatures and Wind Chill Factors Mt. Washington');
```

Your program should run the same as before, just with a much more streamlined main script.

Question 6 Show Correct Answer Show Responses

True or False:
You currently have the same variable names being used in your function and in your main script. They are therefore the same entities and they automatically share values.

A True

B False

The last thing I want to be sure you understand is that the variables in the main script are different than those in the function. We'll illustrate this by using different variable names in the main script and in the function.

Change the variable names in your main script to match the following. You'll need to do this everywhere these variables are used:

- Change `D` to `Date`
- Change `N` to `NDays`
- Change `LT` to `LowT`
- Change `HT` to `HighT`
- Change `WD` to `Winds`
- Change `DT` to `DTs`

These changes are somewhat arbitrary and you will have to change them **everywhere** in the main script (Pay attention to the MATLAB prompts because it will help you with this. There is also a find/replace tool in MATLAB as well). I want you to run your code with these changes though to fully understand that the values are transferred between the variables, not that the variables are actually shared between main script and function. A common misconception when first creating your own function is that the function call requires the main script to use the same variable name. The value stored in that variable gets written into the corresponding input variable.

You can also look at the workspace to see the difference. Try running your P5H1 (or the initial base script provided for this assignment) and the new one with an added function.

Question 7 Show Correct Answer Show Responses

Compare the workspace after you run the version of the weather plotting script with and without the function. What's missing?

A Information on the time period

B Information on the weather data

C Any information handled within the function

D	Nothing

Last requirement

Add help text to your function file to describe how it's used and what it does.

Question 8

What is the primary purpose of creating a separate function such as 'weatherData' in the main script?

?

A	To streamline the main script and avoid repetition of the same code.
B	To increase the length of the main script.
C	To complicate the process of data loading and processing.
D	To create a visual representation of the data.

HINT

EXPLANATION

Show Responses Show Answer

Question 9

What's the procedure of creating a function based on a certain section of code in a main script?

?

A	Identify the code, keep track of the inputs and outputs, start a new m-file, declare it as a function, save the file, put the code in the function.