## Programming Assignments

## Introduction to Programming with MATLAB

## Lesson 5

- Unless otherwise indicated, you may assume that each function will be given the correct number of inputs and that those inputs have the correct dimensions. For example, if the input is stated to be three row vectors of four elements each, your function is not required to determine whether the input consists of three two-dimensional arrays, each with one row and four columns.
- Unless otherwise indicated, your function should not print anything to the Command Window, but your function will not be counted incorrect if it does.
- Note that you are not required to use the suggested names of input variables and output variables, but you must use the specified function names.
- Finally, read the instructions on the web page on how to test your functions with the auto-grader program provided, and what to submit to Coursera to get credit.
- Note that starred problems, marked by ***, are harder than usual, so do not get discouraged if you have difficulty solving them.

1. Write a function called **`generationXYZ`** that takes as its only input argument one positive integer specifying the year of birth of a person and returns as its only output argument the name of the generation that the person is part of ('X', 'Y', or 'Z ') according to the table below. For births before 1966, return 'O' for Old and for births after 2012, return 'K' for Kid. Remember that to assign a letter to a variable, you need to put it in single quotes, as in: **`gen = 'X'`**.

| Generation | Born |
|---|---|
| X | 1966-1980 |
| Y | 1981-1999 |
| Z | 2000-2012 |

2. Write a function called **`letter_grade`** that takes a positive integer called **`score`** as its input argument and returns a letter grade according to the following scale: A: 91 and above; B: 81-90; C: 71-80; D: 61-70; F: below 61. Remember that to assign a letter to a variable, you need to put it in single quotes, as in: **`grade = 'A'`**.

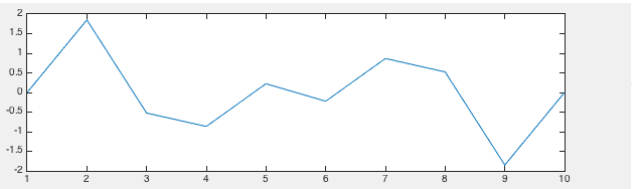
3. Write a function called **`sort3`** that takes three scalar arguments. It uses if-statements, possibly nested, to return the three values of these arguments in a single row-vector in increasing order (or more precisely, non-decreasing order), i.e., element one of the output vector equals the smallest input argument and element three of the output vector equals the largest input argument. NOTE: Your function may not use any built-in functions, e.g., **`sort`**.

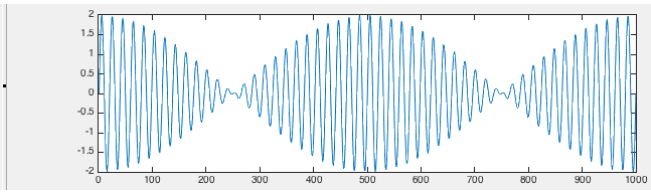
4. Write a function called **`classify`** that takes one input argument **`x`**. That argument will have no more than two dimensions. If **`x`** is an empty matrix, the function returns -1. If **`x`** is a scalar, it returns 0. If **`x`** is a vector, it returns 1. Finally, if **`x`** is none of these, it returns 2. Do not use the built-in functions **`isempty`**, **`isscalar`**, or **`isvector`**.

5. Write a function called **`older`** that takes as its input arguments six positive scalar integers: **`y1`**, **`m1`**, **`d1`**, **`y2`**, **`m2`**, **`d2`**, in that order, representing the birthdates of two persons. The variables that start with **`y`** stand for the year, **`m`** for the month and **`d`** for the day. The variables that end in **`1`** correspond to the first person, while those that end in **`2`** correspond to the second person. The function returns 1 if the first person is older, 0 if they have the same age, and -1 if the first person is younger. You do not need to check whether the inputs have appropriate values. For example, you may assume that both **`m1`** and **`m2`** are positive integers that are less than 13 and that the day numbers fit with their months.

6. Write a function called **`movies`** that takes the starting times of two movies, **`hr1`**, **`hr2`**, **`min1`**, **`min2`**, and their durations, **`durmin1`**, **`durmin2`**, as its input arguments and decides whether we can binge and watch both. The criteria are that they must not overlap and that we are not going to wait more than 30 minutes between the end of one and the beginning of the next. It returns **`true`** if the criteria are both met and returns **`false`** otherwise. You may assume that movie start times are always after 1 pm and before midnight. You may also assume that the first one starts earlier. The order of the input arguments is: **`hr1`**, **`min1`**, **`durmin1`**, **`hr2`**, **`min2`**, **`durmin2`**.

7. Consider the definition: **`function [s1, s2, sums] = sines(pts,amp,n1,n2)`**. The input, **`pts`**, is an integer, but **`amp`**, **`n1`**, and **`n2`** and are *not* necessarily integers. Output argument **`s1`** is a row vector whose length (number of elements) equals **`pts`**. The elements of **`s1`** are the values of the sine function when it is given equally spaced arguments that start at zero and extend through **`n1`** periods of the sine. (Note that we ask for full periods, so if **`n1`** is an integer, both the first and the last element of **`s1`** will be 0 other than a very small rounding error.) (Hint: if you have to make a 100m straight fence from 10m long segments, how many segments do you need? And how many poles?) The amplitude of the sine wave equals **`amp`**. The vector **`s2`** is the same as **`s1`** except that **`s2`** contains **`n2`** periods. The vector **`sums`** is the sum of **`s1`** and **`s2`**. If **`n2`** is omitted, then it should be set to a value that is 5% greater than **`n1`**. If **`n1`** is omitted also, then it should be set to 100. If **`amp`** is not provided, then it should default to 1. Finally, if **`pts`** is omitted as well, then it should be set to 1000. If you run **`sines`** as shown below and plot the third output argument, **`sums`**, the figures should look like these (if you stretch the plot windows horizontally):





```
[s1, s2, sums] = sines(10,1,2,3);
```

```
[s1,s2,sums] = sines(1000,1,50,52);
```

8. *** Write a function called **`moving_average`** that takes a scalar called **`x`** as an input argument and returns a scalar. The value it returns depends not only on the input but also on previous inputs to this same function when the function is called repeatedly. That returned value is a "moving average" of those inputs. The function uses a "buffer" to hold previous inputs, and the buffer can hold a maximum of 25 inputs. Specifically, the function must save the most recent 25 inputs in a vector (the buffer) that is a persistent variable inside the function. Each time the function is called, it copies the input argument into an element of the buffer. If there are already 25 inputs stored in the buffer, it discards the oldest element and saves the current one in the buffer. After it has stored the input in the buffer, it returns the mean of all the elements in the buffer. Thus, for each of the first 24 calls to the function, the function uses only the inputs it has received so far to determine the average (e.g., the first call simply returns **`x`**, the second call averages **`x`** and the input from the first call, etc.), and after that, it returns the average of the most recent 25 inputs.