Appendix A: File Algorithms

evanpark_Cam_Main.py

```
#evanpark_Cam_Main.py

#Goal: Serve as the primarly function for the Cam analysis program. This file should probably sort the users input so that cam profiles can
#be manually created or imported form a .txt file. Once the cam profile is successfully created or imported, the logic within this program uses function
#from other programs to either completely preform the user selected function or return the nessicary data that is then prepared within this file

#Input: User Inputs for the menus, the cam profile (imported and/or manually inputted). If manually inputted, angle intervals, type of motion, overall displacement
#angular velocity

#Output: General Information about the Cam, .txt Profile, .cvs of kinematics of the follower, visual display of the needed cam profile, animation of the cam motion

#libaries needed: tkinter, evanpark_Cam_Properties, evanpark_Utilities, time, datetime, os
#Spefic classes/functions needed: Tk, Toplevel, askopenfilename, asksaveasfile, CamProperties, sleep, datetime


#Main function (Goals, Inputs, Outputs described above)
   #keepGoing gets True
   #while loop with sentry keepGoing, changes via quit option
      #display first menu via menu1()
      #get the user's input, store to userResp1
      #if userResp is "1", execute the following
         #create new profile via createProfile()

      #elif userResp is "2", execute the following
         #import profile via importProfile()

      #elif userResp is 3, execute the following
         #quit by setting keepGoing to False

      #else, execute the following
         #invalid input, print error message


#define function createProfile(Goal: Create a cam profile from scratch. No Inputs passed in. Outputs a Cam profile in the correct format for a later function)
   #keepGoing2 gets True
   #while loop with sentry keepGoing2, exits via valid input
      #keepGoing2 gets False, anticipating valid input
      #print the second menu via function menu2()
      #get user's input, store to userResp2
      #if userResp2 is "1", execute the following
         #units gets "cm"
      #if userResp2 is "2", execute the following
         #units gets "m"
      #if userResp3 is "3", execute the following
         #units gets "in"
      #if userResp3 is "4", execute the following
         #units gets "ft"
      #else, execute the following
         #invalid input, set keepGoing2 to True to loop back through
         #print error message

   #keepGoing2 gets True
   #prompt user for the starting radius of the cam, store to startRadius
   #while loop with sentry keepGoing2, changes via valid input
      #if startRaius is a number (from function isNumber), execute the following
         #startRadius is converted to a float
         #if startRadius is more than 0, execute the following
            #keepGoing2 gets False since the input is valid
         #else, execute the following
            #prompt the user for a valid input, store to startRadius
```

```
   #else, execute the following
      #prompt the user for a valid input, store to startRadius

#keepGoing2 gets True
#prompt the user for the angular velocity of the cam, store the input to angularW
#while loop with sentry keepGoing2
   #if the angularW is a number (from function isNumber), execute the following
      #convert angularW to float
      #if angularW is more than 0, execute the following
         #keepGoing2 gets False since this is a valid input
      #else, execute the following
         #prompt the user for a valid input, store to angularW
   #else, execute the following
      #prompt the user for a valid input, store to angularW

#keepGoing2 gets True
#camDetails gets empty list
#pastAngle gets initial value of 0
#loopNum gets initial value of 1
#currentPosition gets initial value of startRadius
#while loop with keepGoing2, exits after 360 is entered subsequent information is entered
   #[startAngle, endAngle, Type of Movement, Change in R] stored to currentLine

   #Print Label for the current interval
   #keepGoing3 gets True
   #prompt the user for the end angle of the current interval, store input to finalAngle
   #while loop with sentry keepGoing3, exits on valid input
      #if finalAnlge is an integer (via function isInteger), execute the following
         #if finalAngle is more than the past angle and less than or equal to 360
            #keepGoing3 gets false since input is valid
            #finalAngle is convert to an integer
            #the second item of currentLine gets the value of finalAngle
         #else, execute the following
            #print an error message
            #prompt the user again for a valid input, store to finalAngle
      #else, execute the following
         #print an error message
         #prompt the user again for a valid input, store to finalAngle

   #if finalAngle is equal to 360, execute the following
      #if the currentPosition is equal to the startRadius, execute the following
         #print message explaining that the motion type must be a dwell
         #change Radius gets value of 0
         #motionType gets "Dwell"
      #else, execute the following
         #changeRadius gets the value of the startRadius minus the currentPosition
         #print message explaining that the motion type must have a specific rise or run of a specific displacement
         #keepGoing gets value of True
         #while loop with sentry of keepGoing3
            #keepGoing gets False, with anticipation of valid input
            #print the paired down motion type menu via menu4() function
            #prompt user for input, store to userResp4
            #if userResp4 is "1", execute the following
               #motionType gets "Uniform"
            #elif userResp4 is "2", execute the following
               #motionType gets "Parabolic"
            #elif userResp4 is "3", execute the following
               #motionType gets "Harmonic"
            #elif userResp4 is "4", execute the following
               #motionType gets "Cycloidal"
            #else, execute the following
               #keepGoing3 gets True so that the input loop runs again
               #print error message
      #keepGoing2 gets false since the values inputted into currentLine later are now all valid

   #else (finalAngle is not 360), execute the following
      #keepGoing3 gets True
      #while loop with sentry keepGoing3, exits via valid input
         #keepGoing3 gets False, anticipates valid input
```

```
            #print the full motion type menu via function menu3()
            #prompt user for input, store response to userResp3
            #if userResp is "1", execute the following
               #motionType gets "Dwell"
            #elif userResp is "2", execute the following
               #motionType gets "Uniform"
            #elif userResp3 is "3", execute the following
               #motionType gets "Parabolic"
            #elif userResp3 is "4", execute the following
               #motionType gets "Harmonic"
            #elif userResp3 is 5, execute the following
               #motionType gets "Cycloidal"
            #else, execute the following
               #keepGoing3 gets True, since input is not valid
               #print error message

         #if the motionType is "Dwell", execute the following
            #changeRadius gets 0

         #else (motionType is not "Dwell"), execute the following
            #keepGoing3 gets True
            #propmt the user for the change in radius, store to changeRadius
            #while loop with sentry keepGoing3, exits via valid input
               #if changeRadius is a number (via function isNumber), execute the following
                  #if the currentPosition plus the changeRadius is more than 0, execute the following
                     #keepGoing3 gets False since the input is valid
                     #convert changeRadius to a float
                  #else, execute the following
                     #print an error message
                     #prompt the user again for a valid input
               #else, execute the following
                  #print an error message
                  #prompt the user again for a valid input

      #third item of currentLine gets the value of the motionType
      #fifth (last) item of currentLine gets the value of the changeRadius plus the currentPosition

      #convert currentLine to tuple called currentLineTup
      #append currentLineTup to camDetails

      #currentPosition now gets the value of the final position from the previous interval
      #pastAngle now gets the value of the finalAngle from the previous interval
      #inc loopNum

   #camInformation gets list of the units and the angularW

   #run function camAnalysis with the inputs camInformation and camDetails


#define function importProfile(Goal: Create a cam profile from using the information from the file. No Inputs passed in. Outputs a Cam profile in the correct format for a later function)
   #dialogBox gets the class Tk()
   #dialogBox is the main window/focus via method focus_force()
   #dialgoBox is sent to the front, on top of any other windows
   #Tk box is hidden from the user via method withdraw()
   #the inputFilename gets the result of askopfilename (This prompts the user to pick a .txt file to load in)

   #try the following
      #open inputFilename in read mode, set to inputFile
         #read all of the lines from the inputFile, store to inputData as list
         #print messge saying the inputFilename has succesfully been loaded
      #strippedData gets empty list
      #lineNum gets 0
      #close the inputFile
      #for every line in the inputData list, loop through
         #strip the lines of any newlines
         #replace any spaces with nothing
         #add the line to the list strippedData
         #uppercase the entire current line
```

```
            #if UNITS appears in the line, execute the following
               #replace any colons with nothing
               #replace the UNITS with nothing, then lowercase the line. Value gets set to units
            #elif ROTATIONALSPEED appears in the line, execute the following
               #replace any colons with nothing
               #replace RAD/S with nothing
               #replace the ROTATIONALSPEED with nothing, then convert what is left to float and set to angularW
            #elif CAMDETAILS appears in the line, execute the following
               #lineCamDetails gets the value of lineNum plus 1
            #inc lineNum

         #for loop from 0 to lineCamDetails (sentry is ival)
            #delete the first item in list strippedData

         #camDetails gets empty list

         #for each line in strippedData, loop through
            #lineValues gets list of the line after all of the values seperated by commas have been extracted
            #currentLinTup gets the values of the lineValues with the first two being converted to integers, the third being in title format, and the last two
being made floats
            #add the currentLineTup to the list camDetails

         #camInformation gets list of units and angularW

         #destroy the dialogBox
         #run function camAnalysis with inputs of camInformation and camDetails

      #if the error FileNotFoundError occurs, execute the following
         #print a message telling the user that the dialog box had been closed
         #destroy the dialogBox
      #if any other errors occur, execute the following
         #print a general error message
         #destroy the dialgoBox

#define function camAnalysis (Goal: Provide the need logic to act as a hub for launching functions/methods to preform the task that the user has
selected.
#Inputs passed in: camMotionInfo (Intervals for each motion, type of motion, start and end locations) and camBasicInformation (units and angular
velocity)
#Outputs: Changing information such as the units/angular velocity, creating a .txt file for the profle, .cvs file for the follower kinematics, viusal
output
#of the cam's shpae, and an animation of the cam rotating

      #keepGoing4 gets True
      #CamP gets class CamProperties with inputs camMotionInfo (list), camBasicInformation[0] (units), and camBasicInformation[1] (angular
velocity)

      #while loop with sentry keepGoing4, changes with quit option
         #prints menu of options via function menu5()
         #get input from the user, save input to userResp5

         #if userResp is "1", execute the following
            #Display Cam Data and Information via method displayInformation in class CamP
            #wait for the user to hit enter to continue

         #elif userResp5 is "2", execute the following
            #Update Rotational Speed
            #keepGoing5 gets True
            #print newline
            #while loop with sentry keepGoing5, change via valid input
               #prompt user for a rotational velocity, store input to rotationalInput
               #if rotationalInput is a number (via funciton isNumber), execute the following
                  #set CamP.rotatioalSpeed to float of rotatioalInput
                  #keepGoing5 gets False since the user's input is valid
               #else, execute the following
                  #print an error message

         #elif userResp5 is "3", execute the following
            #Update Units
            #keepGoing5 gets True
```

```
    #while loop with sentry keepGoing5, exits with valid input
        #keepGoing5 gets False for anticipation for valid input
        #print menu for units via function menu2()
        #prompt the user for their input, store to unitsInput
        #if unitsInput is "1", execute the following
            #CamP.unitsType gets "cm"
        #elif unitsInput is "2", execute the following
            #CamP.unitsType get "m"
        #elif unitsInput is "3", execute the following
            #CamP.unitsType gets "in"
        #elif unitsInput is "4", execute the following
            #CamP.unitsType gets "ft"
        #else, execute the following
            #keepGoing gets True since the user input was not valid
            #print an error message to the user

#elif userResp5 is "4", execute the following
    #Generate and Save Cam Profile File
    #rawDate gets the current date and time
    #fileDefault gets the following string containing the date and time
    #inFileData gets the following string containing the date and time
    #textFileBody gets the strings of inFileDate plus CamP.getCamInformation()
    #dialogBox gets class Tk()
    #dialogBox becomes the main window via method focus_force()
    #dialogBox is sent to the front of any other windows
    #hide the Tk box via method withdraw()
    #try the following
        #outputFile gets the output of asksaveasfile in write mode, with the intialfile of fileDefault and defaultextension of .txt
        #write the textFileBody to outputFile
        #close the outputFile
        #print message of succesfully writing the filename to the directory's path
    #if the error AttributeError occurs, execute the following
        #print message explaining that the user closed the Dialog Box
    #if another error occurs, execute the following
        #print message explaining that the file failed to write
    #destroy the dialogBox

#elif userResp5 is "5", execute the following
    #Generate and Save Cam Analysis File
    #rawDate gets the current time and date
    #fileDefault gets the string with the date and time in the following format
    #dialogBox gets class Tk()
    #dialogBox becomes the main window via method focus_force()
    #send dialogBox to front of any other windows
    #hide the Tk box from the user
    #try the following code
        #outputFile puts the output of asksaveasfile in mode write with initialfile of fileDefault and defaultextension of .csv
        #write return of CamP.getAngularDataPointsCSV to outputFile
        #close the outputFile
        #print message that the filename was written the to the file directory
    #if error AttributeError occurs, execute the following
        #print message explaining that the user closed the DialogBox
    #if any other error occurs, execute the following
        #print an error message that the file failed to write
    #destroy the dialogBox

#elif userResp5 is "6", execute the following
    #Display Mock Cam Profile via method CamP.displayCamVisual()

#elif userResp5 is "7", execute the following
    #Display Mock Cam animation via method CamP.displayCamVisual()

#elif userResp5 is "8", execute the following
    #quit
    #print message that the user is quitting to the main menu
    #keepGoing4 gets False so that the loop is exitted

#else, execute the following
    #print invalid input message
```

```
        #wait for 0.25 seconds

#if the file is the main file being run
    #run the main function
```

evanpark_Cam_Poperties.py

```
#evanpark_Cam_Properties.py

#Goal: Create a class with methods that can set/get the angular velocity and units, get the values needed for the .csv file,
#update the values for each angle from 0 to 359 given changes in angular velcotiy, show the static cam profile, and show an
#animation of the cam rotating

#Input: Takes in camDataAnalysis (motion intervals), unitString (units), and rotatinalW (angular speed)

#Output/Methods: set/get Rotational Speed, set/get Units, get cam basic info, display basic info, get the points for each of the
#angle points in a format compatible to writing a .csv file, update the angular points value, display the static cam profile, and
#show an animation of the cam rotating

#libaries needed: math, evanpark_Cam_Visual
#Spefic classes/functions needed: pi, sin, cos, pow, CamVisual

#Define Class CamProperties, inherits CamVisual (Info for Class Above)
   #initialize the class with inputs self, camDataAnalysis, unitString, rotationalW
      #self.camData gets the list camDataAnalysis
      #use method self.setRotationalSpeed with input rotationalW
      #use method self.setUnits with input unitsString

   #define method setRotationalSpeed with inputs self, rotSpeed
      #if the rotSpeed is more than 0, execute the followig
         #self.__rotationalW gets rotSpeed
         #run method self.updateCamData()
      #elif rotSpeed is equal to 0
         #if return of method self.getRotationalSpeed is 0, execute the following
            #print message explaing that the angular velocity will be set to a different value
            #self.__rotationalW gets 1
            #run method self.updateCamData()
         #else, execute the following
            #print error message
      #elif rotSpeed is negative, execute the following
         #explain that negative rotational speeds cannot be used, value will be made positive
         #self.__rotationalW gets -1 times the rotSpeed
         #run method self.updateCamData()

   #define method getRotatinalSpeed with input self
      #return value of self.__rotationalW

   #define method setUnits with input self and units
      #self.__units gets units

   #define method getUnits with input self
      #return value of self.__units

   #define method getCamInformation with input self
      #camInfoLong gets the string with the labels for units, rotational speed, and CamDetails and units and rotatinal speed value
      #for 0 to length of self.camData, sentry mVal
         #camInfoLong gets camInfoLong plus each of the items in self.camData for line mVal
      #return the string camInfoLong

   #define method getAngularDataPointsCSV with input self
      #textCSV gets the string for each of the column labels
      #for every line in self.angleDataPoints, loop through
         #itemFirst gets defined as True
         #for each item in line, loop through
            #if itemFirst is True, execute the following
               #add value to textCSV
               #itemFirst gets False
            #else, execute the following
               #add value to textCSV with comma in front of it
         #add newline to textCSV

      #return textCSV string

   #define displayInformation with input self
```

```
   #print the output of method self.getCamInformation

#define updateCamData with input self
   #numberMotions gets length of self.camData
   #currentAngle gets value of zero
   #self.angleDataPoints gets empty list
   #for mValue in range of 0 to numberMotions (sentry mValue)
      #lVal gets value of self.camData[mValue][4] (interval end displacement) minus self.camData[mValue][3] (interval start displacement)
      #intervalValue gets self.camData[mValue][1] (interval end angle) minus self.camData[mValue][0] (interval start angle)
      #convert intervalValueDeg to radians, set to intervalValueRad
      #for currentAngle in range of start angle to final angle of interval (sentry: currentAngle)
         #thetaOverB gets the currentAngle minus the start Angle divided by the intervalValueDeg
         #multiple thetaOverB by Pi, set to thetaOverBPI
         #if self.camData[mValue][2] (motion type) in uppercase is "DWELL"
            #see documentation for formulas for position, velocity, acceleration, and jerk
         #elif self.camData[mValue][2] (motion type) in uppercase is "UNIFORM"
            #see documentation for formulas for position, velocity, acceleration, and jerk
         #elif self.camData[mValue][2] (motion type) in uppercase is "HARMONIC"
            #see documentation for formulas for position, velocity, acceleration, and jerk
         #elif self.camData[mValue][2] (motion type) in uppercase is "PARABOLIC"
            #if thetaOverB is less than 0.5, use this set of formulas found in documentation for position, velocity, and acceleration
            #elif thetaOverB is more than 0.5, use this set of formulas found in documentation for position, velocity, and acceleration
            #see documentation for formulas for position, velocity, acceleration, and jerk
         #add list of currentAngle, time (calculated in line), currentPosition, currentVelocity, currenAcceleration, and currentJerk

#define method displayCamVisual with input self
   #print message explaining how to exit mainloop in master
   #class camVisual with inputs of self.angleDataPoints, self.getUnits, self.getRotationalSpeed, and "Image"

#define method displayCamAnimation with input self
   #print message explaining how to exit mainloop in master
   #class camVisual with inputs of self.angleDataPoints, self.getUnits, self.getRotationalSpeed, and "Animation"

#define property rotationalSpeed with fget of getRotationalSpeed and fset of setRotationalSpeed
#define property unitsType with fget of getUints and fset of setUnits
```

evanpark_Cam_Visual.py

```
#evanpark_Cam_Visual.py

#Goal: Dipslay a cam static image that can be saved or display an animation of the cam rotating

#Input: Takes in passedInDataPoints(all of the kinematic values of the follower for each angle), unitString (units),
#rotationalW (angular velocity), and the modeType ("Image" or "Animation")

#Output/Methods: windowSetUp, canvasCreate, staticImage, dynamicImage, playAgain, saveStatic, quitProgram, drawCam

#libaries needed: tkinter, datetime, math, PIL (Pillow - Must be PIP Installed), ghostscript (needed for Pillow - Must be PIP Installed)
#Spefic classes/functions needed: asksaveasfilename, datetime, cos, sin, pi, Image

#Additional Notes: In order to save the static image as a PNG file, the following requirements are needed:
   #Ghostscript (AGPL License) needs to be installed (DL Page: https://www.ghostscript.com/download/gsdnld.html)
   #The gs binary file needs to be added to the PATH/Path Folder in the system's environment variables
      #EXAMPLE DIRECTORY THAT NEEDS TO BE ADDED TO PATH FOLDER: C:\Program Files\gs\gs9.50\bin

#If Ghostscript is not installed and added to the PATH/Path Foler, the PNG file cannot be saved, even with the Pillow
#and Ghostscript libraries installed. The Cam can still be properly displayed without the 3rd Party Libraries

#Try to import Image from PIL
   #pillowImport gets True if successful
#If it fails to import, execute the following
   #print message that the import has failed
   #pillowImort gets False if it failed

#Try to import ghostscript
   #ghostImport gets True if successful
#If it fails to import, execute the following
   #print message that the import has failed
   #ghostImport gets False if it failed

#Define class CamVisual
   #initialize the class with input self, passedInDataPoints, unitString, rotationalW, and modeType
      #self.camProfileData gets empty list
      #self.units gets unitString
      #self.omega gets value of rotationalW
      #self.maxDisplace gets value of -1
      #for line in passedInDataPoints, loop through
         #angle is equal to the 1st item, displacement is equal to the 3rd item
         #if displacemnt is more than self.maxDisplace, execute the following
            #self.maxDisplace gets the value of displacement
         #add tuple of angle and displacement to the self.camProfileData List

      #run method self.windowSetUP

      #if modeType is "Image", execute the following
         #run method self.staticImage
      #elif modeType is "Animation", execute the following
         #self.playButton gets button on self.frame with text Play Again and commmand self.playAgain
         #self.playButton is placed on grid at row 0, column 0
         #run method self.dynamicImage

      #self.master.mainloop (Loop unitl self.master is destroyed)
      #print closing message

   #define method windowSetUp with input self
      #self.master gets Class Tk()
      #make the self.master the main window
      #send self.master in front of any other windows
      #define self.master size as 800x775
      #self.master cannot be resized or maximized
      #self.master gets the title "Cam Profile Image"

      #self.frame gets Frame with parent self.master
      #self.frame is a grid
      #place self.frame's botoon right corner at 750, 762.5
```

```
    #run method self.canvasCreate

    #self.pixelsOverDistanceRation gets the value of 200 divided by self.maxDisplace

    #self.quitButton gets Button on self.frame with text Quit and command quitProgram
    #place self.quitButton on the grid at row 0, column 1

#define method canvasCreate with Input self
    #self.canvas gets Canvas on parent self.master with width 750 and height 700
    #place the canvas's top left corner at 25,25
    #create a center dot of cam on self.canvas

    #create the shaft in which the follower moves up and down on self.canvas

    #create the scale of the cam at the top left of the self.canvas

#define staticImage with input self
    #run method camDraw with input 0
    #self.saveButton gets Button on self.frame with text Save and command self.saveStatic
    #place self.saveButton on grid at row 0, column 0
    #if pillowImport and ghostImport are both true, execute the following
        #do not execute anything
    #if either one failed, execute the following
        #disable the self.saveButton

#define method dynamicImage with input self
    #disable both the quitButton and playButton
    #for degVal in range of 0 to 360 (sentry: degVal)
        #run method camDraw with input degVal
        #update the self.canvas
        #delete the self.followingBar
        #for lineVal in range 0 to 360 (sentry: lineVal)
            #delete the lineVal th element in list allCamLines
    #run method camDraw with input 0
    #Enable both the quitButton and playButton

#define method playAgain with input self
    #delete the self.followingBar
    #for lineVal in range 0 to 360 (sentry: lineVal)
        #delete the linVal th element in list allCamLines
    #run method self.dynamicImage

#define method quitProgram with input self
    #destroy self.master

#define method saveStatic with input self

    #try the following code
        #dialogBox gets the class Toplevel
        #hide the Toplevel box that appears

        #rawDate gets the current time and date
        #fileImageDefault gets the string with the date and time in the following format
        #filenamePhoto gets the output of asksaveasfilename with intialfile fileImageDefault and defaultextension .png

        #destroy the dialogBox

    #try the following code if an error occurs
        #print an error message
        #destroy the dialogBox

        #if filenamePhoto is empty, execute the following
            #print the message that the user closed the dialog box

        #else (filenamePhoto is not empty)
            #try the following code
                #update self.canvas
                #temp.eps is create via postscript method
```

```
            #staticImage gets the image opened with via class Image
            #staticImage is saved using the absolute path as a PNG file

            #print a message that the photo succesfully saved
        #if WindowsError occurs, execute the following
            #print message that the binary file needs to be added to the PATH enviornment variables
        #if another error occurs, execute the following
            #print an error message

#define method camDraw with input self and zeroAngleLoc
    #followerVal gets the value of zeroAngleLoc
    #zeroAngleLoc gets 360 minus the zeroAngleLoc
    #if zeroAngleLoc is 360, convert it to 0

    #currentAngle gets the value of
    #followerLoc gets the value of the pixel ratio (self.pixelsOverDistanceRatio) times the diplacement of the angle passed in

    #self.followingBar is drawn as a line from the point of the cam currently at angle 0 (90 for traditional coordinates) and drawn to be 225 pixels
    long

    #self.allCamLines gets empty list of 360 items
    #for currentAngle in range of 0 to length of self.camProfileData
        #prevAngle gets currentAngle minus 1
        #if prevAngle is -1, convert to 359

        #currentAngle Rad converts the currentAngle plus the zeroAngleLoc shift to radians
        #the currentRadius gets the pixel ratio (self.pixelsOverDistanceRatio) times the length value in self.camProfileData for the currentAngle
        #prevAngleRad converts the PrevAnlge plus the zeroAngleLoc shift to radians
        #the prevRadius gets the pixel ratio (self.pixelsOverDistanceRatio) times the length value in self.camProfileData for the prevAngle

        #calculates the x and y for the current point and the previous point (use cos for y and sin for x due to the 90 deg shift)

        #self.allCamLines[currentAngle] gets the line drawn from the previous point to the current point
```

evanpark_Utilities.py

#evanpark_Utilities.py
#This file is a collection of functions that are utilized throughout the main python file. The functions include printing menus
#and determining if a passed through value is a number or not and if a passed through value is an integer or not

#function menu1() prints the main menu for the program. It takes no input and outputs the menu via print

#function menu2() prints the units menu for the program. It takes no inputs and outputs the menu via print

#function menu3() prints the menu for selecting a motion types. It takes no inputs and outputs the menu via print

#function menu4() prints the menu for selecting a motinon type given that it cannot be a dwell. It takes no inputs and outputs the menu via print

#function menu5() prints the menu for once the cam has been created or successfully imported. It takes no inputs and outputs the menu via print

#funtion isNumber() retruns a T/F value for whether an inputted variable is a number or not. It takes a single input and outputs either True or False
    #try to convert the input to a float. If successful, return True. Otherwise, return False

#funtion isNumber() retruns a T/F value for whether an inputted variable is an integer or not. It takes a single input and outputs either True or False
    #Ensure the input is a number. If the float of the input is equal to the int of the input, the value is an integer and True is returned. Otherwise, return False