User

MANUAL

Lev6B Flight Planning v1.1

Christos Levy 2019

USER MANUAL

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION

A. GENERAL INFORMATION

1.1 System Overview

Extensive e6B program used for various flight planning calculations and information

- Written in Python 3.8 for macOS

1.2 Project References

References that were used in preparation of this document in order of importance to the end user.

Clancy, L.J.(1975), *Aerodynamics*, Chapter 3. Pitman Publishing Limited, London. ISBN 0-273-01120-0
https://www.aopa.org/training-and-safety/active-pilots/safety-and-technique/weather/density-altitude
https://www.weather.gov/media/epz/wxcalc/pressureAltitude.pdf
http://www.flight-mechanic.com/mean-aerodynamic-chord/

1.5 Organization of the Manual

User's Manual v1.0.

1.6 Acronyms and Abbreviations

Provide a list of the acronyms and abbreviations used in this document and the meaning of each.

App:	Application
Dalt:	Density Altitude
Palt:	Pressure Altitude
gs:	Groundspeed
AWOS/ASOS:	Automated Weather Observation System/Automated Surface Observation System
TAS:	True airspeed
AGL:	Above Ground Level

2.0 SYSTEM SUMMARY

B. SYSTEM SUMMARY

Using the software allows users to compute various aviation preflight calculations with ease. The program is divided into 7 Groups of functions with 27 individual functions. The program runs with a numeric system meaning to operate the system it will only require a numeric keypad. To quit, the user can type “quit” in the separate function group menus, however this task I possible with just numbers.

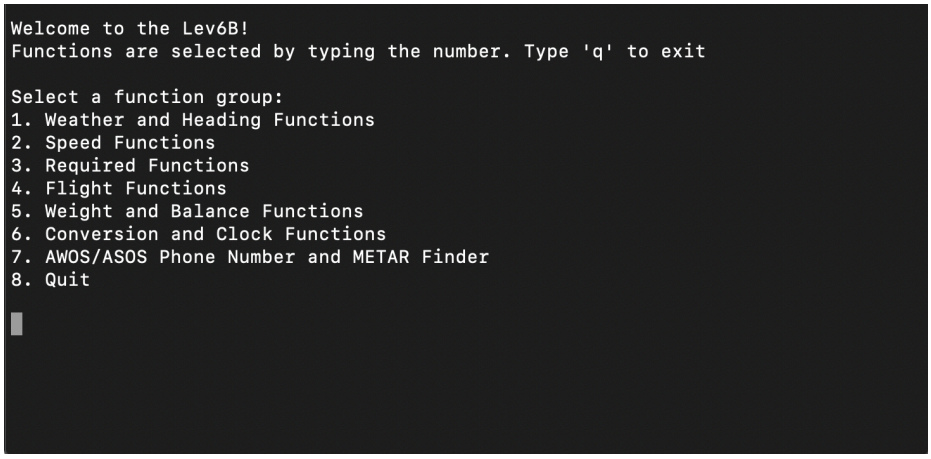
3.0 GETTING STARTED

C. GETTING STARTED

To launch the program, Open the exe file.

3.1 Launch

After the launching the application, the main menu will appear

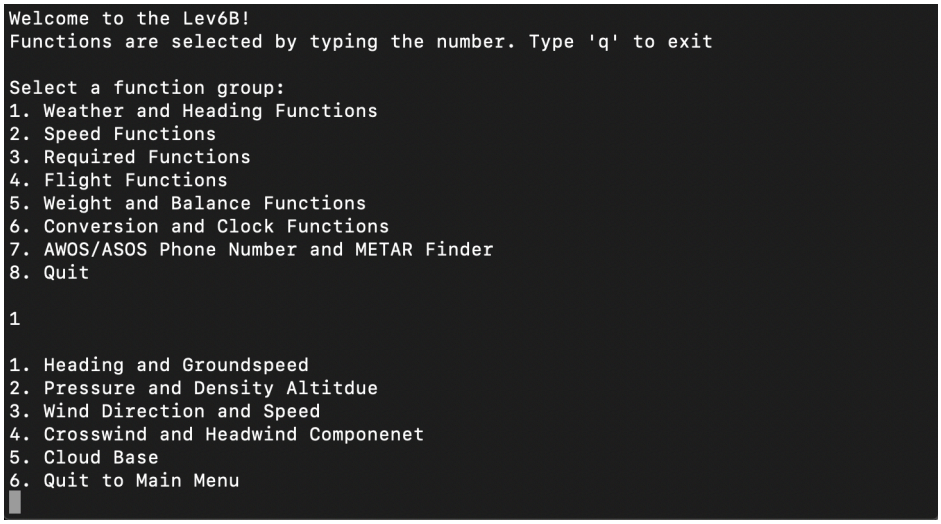


3.2 System Menu

The various function groups are now present and all selections on and after this screen need a numeric selection. Entering anything other than a number on screen will prompt you to try again.

3.2.1 Function Group Menus

Typing a number will bring the user to the corresponding function group. Below is an example in the weather and heading functions group.



3.2.2 Calculating Results

Enter values when asked. Units are not required and will require you to re-enter all values.

```
1. Heading and Groundspeed
2. Pressure and Density Altitdue
3. Wind Direction and Speed
4. Crosswind and Headwind Componenet
5. Cloud Base
6. Quit to Main Menu
1
Wind Direction: 270
Enter Windspeed: 10
Enter Course: 320
Enter True Airspeed: 110
GROUNDSPEED: 103.9
HEADING: 316
WCA: -4.2
```

3.3 Exit System

After each function runs once (with the exception of weight and balance functions) they will return to the function group menu.

4.0 Function Screenshots

4.1 Weather and Heading Functions

Summary
The weather and heading functions are functions related to how the weather affects aircraft speed and its heading. It also has functions that take various weather constants and output values like the cloud base or the Pressure or Density altitudes.

4.1.1 Heading and Groundspeed

Summary
Function takes wind direction, windspeed, course and true airspeed and outputs the aircraft groundspeed, the aircrafts heading and the wind correction angle necessary for this heading. Negative wind correction angles indicate turns to the left and positive indicate turns to the right.

Screenshot

```
1. Heading and Groundspeed
2. Pressure and Density Altitude
3. Wind Direction and Speed
4. Crosswind and Headwind Componenet
5. Cloud Base
6. Quit to Main Menu
1
Wind Direction: 270
Enter Windspeed: 10
Enter Course: 320
Enter True Airspeed: 110
GROUNDSPEED: 103.9
HEADING: 316
WCA: -4.2
```

4.1.2 Pressure and Density Altitude

Summary
Function calculates the pressure and density altitude when given the indicated altitude, the barometric pressure and the ambient air temperature in degrees Celsius.

Screenshot

```
1. Heading and Groundspeed
2. Pressure and Density Altitude
3. Wind Direction and Speed
4. Crosswind and Headwind Componenet
5. Cloud Base
6. Quit to Main Menu
2
Enter Indicated Altitude: 3500
Enter Barometric Pressure: 30.12
Enter Outside Temperature (Celsius): 17
PRESSURE ALTITUDE: 3300
DENSITY ALTITUDE: 2700
```

4.1.3 Wind Direction and Speed

Summary
This function intends to return the wind direction and the windspeed given the Course, TAS, groundspeed and the heading.

Screenshot

```
1. Heading and Groundspeed
2. Pressure and Density Altitude
3. Wind Direction and Speed
4. Crosswind and Headwind Componenet
5. Cloud Base
6. Quit to Main Menu
3
Enter Course: 320
Enter True Airspeed: 110
Enter Groundspeed: 103.9
Enter Heading: 316

WIND DIRECTION: 271
WIND SPEED: 9.6
```

4.1.4 Crosswind and Headwind Component

Summary
This function is used to determine the headwind and crosswind component during departure or landing. It takes the windspeed and direction and the runway number and will tell you the crosswind and headwind component as well its direction (left right, Head, tail)

Screenshot

```
1. Heading and Groundspeed
2. Pressure and Density Altitude
3. Wind Direction and Speed
4. Crosswind and Headwind Componenet
5. Cloud Base
6. Quit to Main Menu
4
Enter Wind direction: 220
Enter Wind speed: 10
Enter Runway number (Runway number and not heading): 27

Crosswind Left: 7.7
Headwind: 6.4
```

4.1.5 Cloud Base

Summary

Calculates the cloud base in feet AGL given the outside temperature and the dew point. Calculation can be done with Fahrenheit or Celsius and user must indicate which selection.

Screenshot

```
1. Heading and Groundspeed
2. Pressure and Density Altitude
3. Wind Direction and Speed
4. Crosswind and Headwind Componenet
5. Cloud Base
6. Quit to Main Menu
5

1. Fahrenheit
2. Celsius
1
Enter the outside temperature: 15
Enter the dew point: 1

Cloud Base: 3181.8 feet AGL
```

4.2 Speed Functions

Summary
The Speed functions contain calculations of how to find the aircraft groundspeed and the TAS. In the previous functions it was apparent the TAS was needed for multiple calculations and if a user was unsure of this value, they would come to these functions.

4.2.1 Groundspeed

Summary
This function calculates the estimated groundspeed of the aircraft given the distance covered and the time. The function accepts hours, minutes, and seconds as separate inputs however can take decimals in the hour portion and operate the same.

Screenshot

```
1. Groundspeed
2. Plan TAS
3. Actual TAS
4. Quit to Main Menu
1
Enter Distance covered: 100
Enter Hours: 0
Enter Minutes: 55
Enter Seconds: 10

Groundspeed: 108.8 MPH
```

4.2.2 Plan TAS

Summary
The Plan TAS is an estimate of the TAS using just the altitude and the airspeed. For a more accurate calculation of TAS the Actual TAS function is used. The plan TAS is used when only the altitude and airspeed are available.

Screenshot

```
1. Groundspeed
2. Plan TAS
3. Actual TAS
4. Quit to Main Menu
2
Enter Indicated Airspeed: 105
Enter Altitude: 3200

True Airspeed: 105.1
```

4.2.3 Actual TAS

Summary
As stated above, The Actual TAS function calculates the TAS with more accuracy. It takes pressure altitude, temperature and Indicated airspeed which gives more values to use for TAS calculations. It will also return the density altitude.

Screenshot
```
1. Groundspeed
2. Plan TAS
3. Actual TAS
4. Quit to Main Menu
3
Enter Pressure Altitdue: 3200
Enter Temperature in Celsius: 15
Enter indicated airpseed: 105

TAS: 111.3
Denstiy Altitude: 3968.0
```

4.3 Required Functions

Summary
Required functions are used to determine the quantity or magnitude of a certain calculation is needed given specifications. For example the required fuel to make a trip, the required rate of climb for successful climb out, the required rate of descent given descent rate and distance, required TAS for wind and the required funds to make a trip.

4.3.1 Required Fuel

Summary
Required fuel calculates how much fuel is needed for a trip given the time the trip will take and the fuel burn in gallons per hour.

Screenshot
```
1. Required Fuel
2. Required Rate of Climb
3. Required Rate of Descent
4. Required TAS
5. Required Money
6. Quit to Main Menu
1
Enter hours: 1
Enter minutes: 20
Enter seconds: 13
Enter Fuel burn: 10

Required Fuel: 13.4 Gallons
```

4.3.2 Required Rate of Climb

Summary
This function determines the required rate of climb for instrument procedures. It takes the minimum climb in feet per mile and the groundspeed and outputs the required rate of climb in FPM

Screenshot.

```
1. Required Fuel
2. Required Rate of Climb
3. Required Rate of Descent
4. Required TAS
5. Required Money
6. Quit to Main Menu
2
Enter Minimum climb in feet per mile: 400
Enter groundspeed: 110

Required Rate of Climb: 733.3 Feet Per Minute
```

4.3.4 Required Rate of Descent

Summary
Required fuel calculates how much fuel is needed for a trip given the time the trip will take and the fuel burn in gallons per hour.

Screenshot

```
1. Required Fuel
2. Required Rate of Climb
3. Required Rate of Descent
4. Required TAS
5. Required Money
6. Quit to Main Menu
3
Enter Indicated Altitude: 3500
Enter Crossing Altitude: 1500
Enter Groundspeed: 110
Enter Fix Distance: 5

Required Rate of Descent: 730 Feet Per Minute
```

4.3.5 Required Money(Cost Calculator)

Summary
Required money function operates like a cost calculator. It computes the total cost of fuel for a flight given the average fuel cost, fuel burn, and trip info (time or distance & groundspeed)

Screenshot

```
1. Required Fuel
2. Required Rate of Climb
3. Required Rate of Descent
4. Required TAS
5. Required Money
6. Quit to Main Menu
5

1. Calculate by distance
2. Calculate by time
1
Enter average fuel cost: 5.00
Enter fuel burn: 10
Enter Trip Distance: 205
Enter Groundspeed: 110
Fuel Cost for 205.0 Miles: $93.18
```

4.4 Flight Functions

Summary

Flight functions menu provides various functions for in-flight computations or values regarding specific procedural questions. The functions include the distance flown, the top of descent, the endurance based on current fuel, leg time, range and FPH. All these function have a relationship to in-flight inquiries.

4.4.1 Distance Flown

Summary

This function calculates the distance flown when passed the groundspeed and then the time as hours, minutes and seconds. The all tabs can accept decimal answers and compute the correct distance.

Screenshot

```
1. Distance Flown
2. Top of Descent
3. Endurance
4. Leg Time
5. Specific Range
6. Fuel Per Hour
7. Quit to Main Menu
1
Enter Groundspeed: 110
Enter Hours: 1
Enter Minutes: 20
Enter Seconds: 15

147.1 Mile(s)
```

4.4.2 Top of Descent

Summary
This function calculates the top of the descent given the speed, indicated altitude and destination altitude and the rate of descent. The function will output a distance which is the distance from the bottom of descent given the destination altitude and the rate. If the bottom of descent needs to be the pattern of an airport, the top of descent will return the distance from the airport that the descent needs to begin.

Screenshot

```
1. Distance Flown
2. Top of Descent
3. Endurance
4. Leg Time
5. Specific Range
6. Fuel Per Hour
7. Quit to Main Menu
2
Enter Groundspeed: 110
Enter Indicated Altitude: 3500
Enter Destination Altitude: 1700
Enter Rate (FPM): 400

8.2 Mile(s)
```

4.4.3 Endurance

Summary
This function calculates the endurance given the total fuel on board and the fuel per hour consumption. It will return the time in an hour, minute and second format.

Screenshot

```
1. Distance Flown
2. Top of Descent
3. Endurance
4. Leg Time
5. Specific Range
6. Fuel Per Hour
7. Quit to Main Menu
3
Enter total Fuel: 40
Enter fuel burn (FPH): 10

4.0 Hour(s)
```

4.4.4 Leg Time

Summary
This function calculates the time a distance will take given that distance and the aircraft groundspeed.

Screenshot

```
1. Distance Flown
2. Top of Descent
3. Endurance
4. Leg Time
5. Specific Range
6. Fuel Per Hour
7. Quit to Main Menu
4
Enter Distance: 100
Enter groundspeed: 110

0 Hour(s) 54 Minute(s) 32 Second(s)
```

4.4.5 Specific Range

Summary
Specific range is a planning function used to determine the most desirable altitude for a long range flight. Range is calculated in miles given the total fuel, groundspeed and fuel burn.

Screenshot

```
1. Distance Flown
2. Top of Descent
3. Endurance
4. Leg Time
5. Specific Range
6. Fuel Per Hour
7. Quit to Main Menu
5
Enter Fuel: 40
Enter fuel burn (FPH): 10
Enter groundspeed: 110

440.0 Mile(s)
```

4.4.6 Fuel Per Hour

Summary
The fuel per hour function computes an accurate fuel burn given the total fuel and the time. The time input takes hours, minutes, and seconds. All inputs can accept decimals.

Screenshot

```
1. Distance Flown
2. Top of Descent
3. Endurance
4. Leg Time
5. Specific Range
6. Fuel Per Hour
7. Quit to Main Menu
6
Enter total Fuel: 40
Enter hours: 4
Enter minutes: 10
Enter seconds: 1

9.6 Gallons per Hour
```

4.5 Weight and Balance Functions

Summary
The weight and balance functions are necessary to compute the total weight, the Center of Gravity and the total moment to determine whether the current weight and moment are within the aircrafts envelopes. The weight and balance function has to variances, one taking the weight and arm and the other taking the weight and the moment.

4.5.1 Weight and Arm

Summary
The weight and arm function is used to calculate the moment, total weight and Center of gravity. It takes the weight of an element on the aircraft and the arm (distance from the datum). The function will continue to add the moments and weights as the user chooses to continue. When the user quits, the final weight, moment and CG gets outputted.

Screenshot

```
1. Weight and Arm
2. Weight and Moment
3. Percent Mean Aerodynamic Chord
4. Quit to Main Menu
1
Enter Weight: 100
Enter Arm: 35

MOMENT: 3500.0
GROSS WEIGHT: 100.0
CG: 35.0
Press Enter to enter another value or type '/'+ enter to quit:

Enter Weight: 200
Enter Arm: 36

MOMENT: 10700.0
GROSS WEIGHT: 300.0
CG: 35.67
Press Enter to enter another value or type '/'+ enter to quit:
/
MOMENT: 10700.0
GROSS WEIGHT: 300.0
CG: 35.67
```

4.5.2 Weight and Moment

Summary

Similar to the weight and arm function, the weight and moment is used to calculate the total weight and Center of gravity. It takes the weight of an element on the aircraft and the moment. The function will continue to add the moments and weights as the user chooses to continue. When the user quits, the final weight, moment and CG gets outputted. The difference between this and the previous function is the user does not need to input the arm and skips over the calculation of the moment. With the total weight and the total moment, the CG is calculated.

Screenshot

```
1. Weight and Arm
2. Weight and Moment
3. Percent Mean Aerodynamic Chord
4. Quit to Main Menu
2
Enter Weight: 100
Enter Moment: 3500

MOMENT: 3500.0
GROSS WEIGHT: 100.0
CG: 35.0
Press Enter to enter another value or type '/'+ enter to quit:
/
MOMENT: 3500.0
GROSS WEIGHT: 100.0
CG: 35.0
```

4.5.3 Percent Mean Aerodynamic Chord

Summary
The weight and arm function is used to calculate the moment, total weight and Center of gravity. It takes the weight of an element on the aircraft and the arm (distance from the datum). The function will continue to add the moments and weights as the user chooses to continue. When the user quits, the final weight, moment and CG gets outputted.

Screenshot

```
1. Weight and Arm
2. Weight and Moment
3. Percent Mean Aerodynamic Chord
4. Quit to Main Menu
3
Enter CG: 37.27
Enter Leading Edge Mean Aerodynamic Chord: 22.29
Enter mean aerodynamic chord: 61.4
Percent MAC: 24.4%
```

4.6 Conversion and Clock Functions

Summary
The conversion and clock function group include 3 useful functions. The first two are a timer and a stopwatch that can be used to keep track of time in an efficient way. The third function contains a temperature conversion both from Fahrenheit to Celsius and Celsius to Fahrenheit.

4.6.1 Timer

Summary
This function creates a timer for a specified hours, minutes and seconds. When the timer finishes, the display will blink “Timer Done!” until Control-C is entered to quit.

Screenshot

```
1. Timer
2. Stopwatch
3. Convert Temperature
4. Quit to Main Menu

1
Timer Function
Type Control-C to end
Enter Hours: 0
Enter Minutes: 5
Enter Seconds: 15
4 Minute(s) 53 Second(s)
```

4.6.2 Stopwatch

Summary

This function creates a timer for a specified hours, minutes and seconds. When the timer finishes, the display will blink “Timer Done!” until Control-C is entered to quit.

Screenshot

```
1. Timer
2. Stopwatch
3. Convert Temperature
4. Quit to Main Menu

2
STOPWATCH FUNCTION
Type Control-C to end
Would you like to begin Timing? (y/n): y
0 Minute(s) 12 Second(s)
```

4.6.3 Convert Temperature

Summary

The convert temperature function converts temperature from Fahrenheit to celsius or vice versa. It will ask the user to input which direction and then will take the temperature to convert.

Screenshot

```
1. Timer
2. Stopwatch
3. Convert Temperature
4. Quit to Main Menu

3

1. Convert Fahrenheit to Celsius
2. Convert Celsius to Fahrenheit

1
Enter Degrees to Convert: 32
0.0 Degrees Celsius
```

4.7 AWOS/ASOS and METAR Functions

Summary

These two functions are web connected functions that will access the weather information of the input error. This can be in the form of the AWOS phone number or frequency. The other weather information is in the form of a METAR and will translate it into plain english.

4.7.1 AWOS or ASOS Finder

Summary

The AWOS/ASOS Finder will accept an airport identifier and will return the phone number for AWOS or ASOS information. If the airport does not have an AWOS on field it will return the closest one and the distance.

Screenshot

```
1. AWOS or ASOS Finder
2. Get Airport METAR
3. Quit to Main Menu
1
AWOS DETECTOR
(type 'quit' to quit to Main Menu)

Enter Airport Identifier (Starting with K): KLAF
WX ASOS: PHONE 765-743-9687
```

4.7.2 Get Airport METAR

Summary

Airport METARs include weather information and entering an airport identifier into this function, will return a translated METAR in real time.

Screenshot

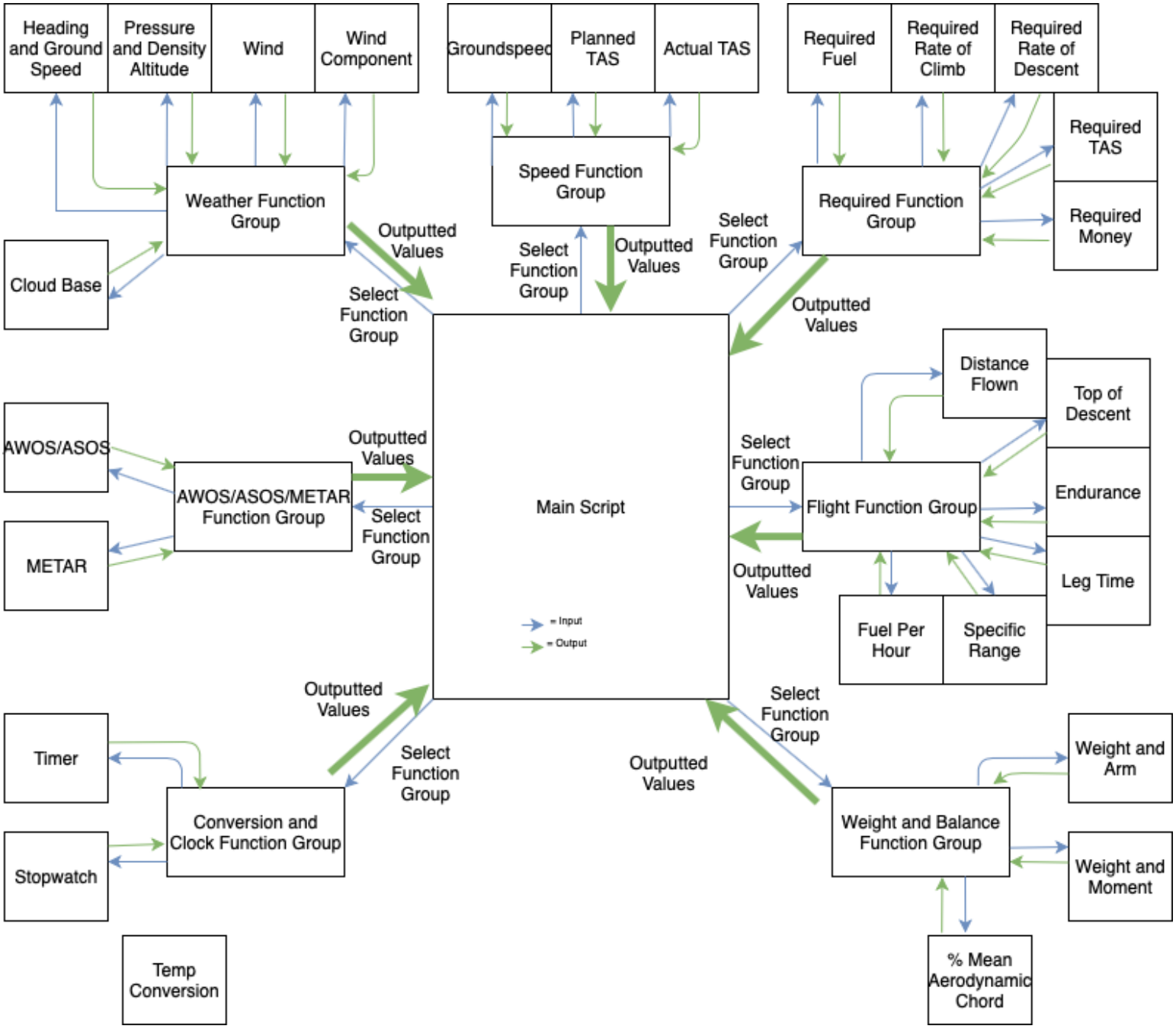
```
1. AWOS or ASOS Finder
2. Get Airport METAR
3. Quit to Main Menu
2
AIRPORT METARS
(type 'quit' to quit to Main Menu)

Enter Airport Identifier (Starting with K): KLAF

Full Metar: KLAF 051654Z 19006KT 10SM CLR 06/01 A3007 RMK AO2 SLP188 T00560006
KLAF - Purdue University Airport
12/5/2019
1654 Zulu
Wind: 190 Degrees at 06 knots
Visibility: 10.0 Statute Mile(s)
[]
[]
Temperature: 6 Degrees Celsius
Dew Point: 1 Degrees Celsius
Altimeter: 30.07
```

10.0 APPENDIX

A. APPENDIX


Heading and Ground Speed
Pressure and Density Altitude
Wind
Wind Component
Groundspeed
Planned TAS
Actual TAS
Required Fuel
Required Rate of Climb
Required Rate of Descent
Required TAS
Required Money
Weather Function Group
Speed Function Group
Required Function Group
Cloud Base
Outputted Values
Select Function Group
Main Script
Distance Flown
Top of Descent
Endurance
Leg Time
Fuel Per Hour
Specific Range
Flight Function Group
AWOS/ASOS
METAR
AWOS/ASOS/METAR Function Group
= Input
= Output
Timer
Stopwatch
Temp Conversion
Conversion and Clock Function Group
Weight and Arm
Weight and Moment
Weight and Balance Function Group
% Mean Aerodynamic Chord

B. APPENDIX

10.2.1 Main Script

```
#!/usr/bin/env python
'''
================================================================================
ENGR 133 Program Description
  Main script that runs all the e6B functions

Project Information
  Project Title:  Lev6B
  Author:         Christos Levy, levy30@purdue.edu
  Team ID:        002-10
================================================================================
'''

## IMPORTED FUNCTION CLASSES
import watch as w
import flight as f
import weightBalance as wb
import weather as wthr
import speed as s
import required as r
import awos as a



numError = "Error: Input(s) must be a number"

## Main Loop
while True:
    killed = False
    funKill = False
    print("\n\nWelcome to the Lev6B!")
    print("Functions are selected by typing the number. Type 'q' to exit\n")

    while True: ## Finds Function Group
        print("Select a function group: ")
        funcGroup = "1. Weather and Heading Functions\n2. Speed Functions\n3. Required Functions\n4. Flight Functions\n5. Weight and Balance Functions\n6. Conversion and Clock Functions\n7. AWOS/ASOS Phone Number and METAR Finder\n8. Quit\n\n"
        funcGSelect = input(funcGroup)
        if funcGSelect == "q" or funcGSelect == 'quit' or funcGSelect == '8':
            print("Thank you for using the Lev6B!\n\n")
            killed = True
            break
        try:
            funcGSelect = int(funcGSelect)
            if funcGSelect > 8 or funcGSelect < 1:
                print("Function Group Not Recognized\n")
```

```
                print("Function Group Not Recognized\n")
                continue
            else:
                break
        except:
            print("\nError: Enter the number of the list or type 'quit' to exit\n")
            continue

    if killed == True: ## Kills the entire Program
        break

    ## Function Groups
    while killed == False:
        while True:

            ## Weather and Heading Functions WORKING
            if funcGSelect == 1:
                while funKill == False:
                    funcSelect = input("\n1. Heading and Groundspeed\n2. Pressure and Density Altitude\n3. Wind Direction and Speed\n4. Crosswind and Headwind Component\n5. Cloud Base\n6. Quit to Main Menu\n").lower()
                    try:
                        if funcSelect == "quit" or funcSelect == "q":
                            funKill = True
                            killed = True
                            break
                        funcSelect = int(funcSelect)
                    except:
                        print(f"{numError}\n")
                        continue
                    if funcSelect == 1: ## Heading and Groundspeed
                        print(wthr.hdgGs())
                        break
                    elif funcSelect == 2: ## Pressure and Density altitude
                        print(wthr.pressureDensityAlt())
                        break
                    elif funcSelect == 3: ## Wind direction and speed
                        print(wthr.wind())
                        break
                    elif funcSelect == 4: ##Cross wind and headwind components
                        print(wthr.windComponent())
                        break
                    elif funcSelect == 5: ## Cloudbase
                        while True:
                            degreeSelect = input("\n1. Fahrenheit\n2. Celsius\n")
                            try:
                                degreeSelect = int(degreeSelect)
```

```
                                if degreeSelect == 1:
                                    print(wthr.cloudBase(1))
                                    break
                                elif degreeSelect == 2:
                                    print(wthr.cloudBase(2))
                                    break
                            except:
                                print("Error: Enter a number 1 or 2")
                                continue
                            break

                    elif funcSelect == 6:
                        funKill = True
                        killed = True
                        break
                    else:
                        print("Error: Function not identified")
                        continue
                if killed == True:
                    break

            ## Speed Functions WORKING
            elif funcGSelect == 2:
                while funKill == False:
                    funcSelect = input("\n1. Groundspeed\n2. Plan TAS\n3. Actual TAS\n4. Quit to Main Menu\n").lower()
                    try:
                        if funcSelect == 'quit' or funcSelect == 'q':
                            funKill = True
                            killed = True
                            break
                        funcSelect = int(funcSelect)
                    except:
                        print(f"{numError}\n")
                        continue
                    if funcSelect == 1:
                        print(s.gs())
                        break
                    elif funcSelect == 2:
                        print(s.planTAS())
                        break
                    elif funcSelect == 3:
                        print(s.actTAS())
                        break
                    elif funcSelect == 4:
                        funKill = True
                        killed = True
                        break
                    else:
                        print("Error: Function not identified")
```

```
                        continue
                if killed == True:
                    break

            ## Required Functions WORKING
            elif funcGSelect == 3:
                while funKill == False:
                    funcSelect = input("\n1. Required Fuel\n2. Required Rate of Climb\n3. Required Rate of Descent\n4. Required TAS\n5. Required Money\n6. Quit to Main Menu\n").lower()
                    try:
                        if funcSelect == 'quit' or funcSelect == 'q':
                            funKill = True
                            killed = True
                            break
                        funcSelect = int(funcSelect)
                    except:
                        print(f"{numError}\n")
                        continue
                    if funcSelect == 1:
                        print(r.fuel())
                        break
                    elif funcSelect == 2:
                        print(r.climb())
                        break
                    elif funcSelect == 3:
                        print(r.descent())
                        break
                    elif funcSelect == 4:
                        print(r.tas())
                        break
                    elif funcSelect == 5:
                        print(r.money())
                        break
                    elif funcSelect == 6:
                        funKill = True
                        killed = True
                        break
                    else:
                        print("Error: Function not identified")
                        continue
                if killed == True:
                    break

            ## Flight Functions WORKING
            elif funcGSelect == 4:
                while funKill == False:
                    funcSelect = input("\n1. Distance Flown\n2. Top of Descent\n3. Endurance\n4. Leg Time\n5. Specific Range\n6. Fuel Per Hour\n7. Quit to Main Menu\n").lower()
```

```
                    try:
                        if funcSelect == 'quit' or funcSelect == 'q':
                            funKill = True
                            killed = True
                            break
                        funcSelect = int(funcSelect)
                    except:
                        print(f"{numError}\n")
                        continue
                    if funcSelect == 1: ## Distance Flown
                        while True:
                            groundspeed = input("Enter Groundspeed: ")
                            hours = input("Enter Hours: ")
                            min = input("Enter Minutes: ")
                            seconds = input("Enter Seconds: ")
                            try:
                                groundspeed = float(groundspeed)
                                hours = float(hours)
                                min = float(min)
                                seconds = float(seconds)

print(f"\n{f.distFln(groundspeed,hours,min,seconds)}")
                                break
                            except:
                                print(numError)
                                continue
                    elif funcSelect == 2: ## Top of Descent function
                        while True:
                            groundspeed = input("Enter Groundspeed: ")
                            iAlt = input("Enter Indicated Altitude: ")
                            dAlt = input("Enter Destination Altitude: ")
                            rate = input("Enter Rate (FPM): ")
                            try:
                                groundspeed = float(groundspeed)
                                iAlt = float(iAlt)
                                dAlt = float(dAlt)
                                rate = float(rate)

print(f"\n{f.topDscn(groundspeed,iAlt,dAlt,rate)}")
                                break
                            except:
                                print(numError)
                                continue
                    elif funcSelect == 3: ##Endurance Function
                        while True:
                            fuel = input("Enter total Fuel: ")
                            fph = input("Enter fuel burn (FPH): ")
                            try:
                                fuel = float(fuel)
                                fph = float(fph)
```

```
                                fph = float(fph)
                                print(f"\n{f.endur(fuel,fph)}")
                                break
                            except:
                                print(numError)
                    elif funcSelect == 4: ## Leg time
                        while True:
                            distance = input("Enter Distance: ")
                            groundspeed = input("Enter groundspeed: ")
                            try:
                                distance = float(distance)
                                groundspeed = float(groundspeed)
                                print(f"\n{f.legTime(distance,groundspeed)}")
                                break
                            except:
                                print(numError)
                                continue
                    elif funcSelect == 5: ## Specific Range
                        while True:
                            fuel = input("Enter Fuel: ")
                            fph = input("Enter fuel burn (FPH): ")
                            gs = input("Enter groundspeed: ")
                            try:
                                fuel = float(fuel)
                                fph = float(fph)
                                gs = float(gs)
                                print(f"\n{f.spcRange(fuel,fph,gs)}")
                                break
                            except:
                                print(numError)
                                continue
                    elif funcSelect == 6: ## Fuel per Hour
                        while True:
                            fuel = input("Enter total Fuel: ")
                            hour = input("Enter hours: ")
                            min = input("Enter minutes: ")
                            sec = input("Enter seconds: ")
                            try:
                                fuel = float(fuel)
                                hour = float(hour)
                                min = float(min)
                                sec = float(sec)

print(f"\n{f.fuelPerHour(fuel,hour,min,sec)}")
                                break
                            except:
                                print(numError)
                                continue
                    elif funcSelect == 7:
                        funKill = True
```

```
281                         killed = True
282                         break
283                 if killed == True:
284                     break
285 
286             # Weight and Balance Functions WORKING
287             elif funcGSelect == 5:
288                 while funKill == False:
289                     funcSelect = input("\n1. Weight and Arm\n2. Weight and
    Moment\n3. Percent Mean Aerodynamic Chord\n4. Quit to Main Menu\n").lower()
290                     try:
291                         if funcSelect == 'quit' or funcSelect == 'q':
292                             funKill = True
293                             killed = True
294                             break
295                         funcSelect = int(funcSelect)
296                     except:
297                         print(numError)
298                         continue
299                     if funcSelect == 1:
300                         wb.weightArm()
301                     elif funcSelect == 2:
302                         wb.weightMom()
303                     elif funcSelect == 3:
304                         wb.mac()
305                     elif funcSelect == 4:
306                         funKill = True
307                         killed = True
308                         break
309                     else:
310                         print("Function not Recognized")
311                 if funKill == True:
312                     break
313 
314             ## Conversion and Clock Functions WORKING
315             elif funcGSelect == 6:
316                 while funKill == False:
317                     funcSelect = input("\n1. Timer\n2. Stopwatch\n3. Convert
    Temperature\n4. Quit to Main Menu\n\n").lower()
318                     try:
319                         if funcSelect == 'quit' or funcSelect == 'q':
320                             funKill = True
321                             killed = True
322                             break
323                         funcSelect = int(funcSelect)
324                     except:
325                         print("Error: You must enter a number\n")
326                         continue
327                     if funcSelect == 1:
328                         print("Timer Function\nType Control-C to end")
```

```
329                         hour = input("Enter Hours: ")
330                         minute = input("Enter Minutes: ")
331                         seconds = input("Enter Seconds: ")
332                         try:
333                             hour = int(hour)
334                             minute = int(minute)
335                             seconds = int(seconds)
336                             w.timer(hour,minute,seconds)
337                             break
338                         except:
339                             print("Error: Not all the values were integers.
    Enter Numbers only\n")
340                             continue
341                     elif funcSelect == 2:
342                         w.stopwatch()
343                         break
344                     elif funcSelect == 3:
345                         type = input("\n1. Convert Fahrenheit to Celsius\n2.
    Convert Celsius to Fahrenheit\n\n")
346                         while True:
347                             degrees = input("Enter Degrees to Convert: ")
348                             try:
349                                 degrees = float(degrees)
350                                 conversion = w.tempConvert(type,degrees)
351                                 print(conversion)
352                                 break
353                             except:
354                                 print("Error: Enter a number to convert")
355                                 continue
356                     elif funcSelect == 4:
357                         funKill = True
358                         killed = True
359                         break
360                     else:
361                         print("Error: Enter a number 1-4")
362                 if killed == True:
363                     break
364 
365             ## AWOS/ASOS Function WORKING
366             elif funcGSelect == 7:
367                 while funKill == False:
368                     funcSelect = input("\n1. AWOS or ASOS Finder\n2. Get
    Airport METAR\n3. Quit to Main Menu\n").lower()
369                     try:
370                         if funcSelect == "quit" or funcSelect == 'q':
371                             killed=True
372                             break
373                         funcSelect = int(funcSelect)
374                         if funcSelect>3 or funcSelect < 1:
375                             print("Fuction not found. Enter a number from the
```

```
                            print("Function not found. Enter a number from the
list\n")
376                             continue
377                     except:
378                         print("Error: Value must be a number\n")
379                         continue
380                     if funcSelect == 1:
381                         print("AWOS DETECTOR\n(type 'quit' to quit to Main
Menu)\n")
382                         while True:
383                             airport = input("Enter Airport Identifier
(Starting with K): ").upper()
384                             if airport == "QUIT":
385                                 killed = True
386                                 break
387                             try:
388                                 awos = a.awos(airport)
389                             except:
390                                 print("Error: Enter the 4 letter
identifier.\n")
391                                 continue
392                             if awos == None:
393                                 print("Error: Airport/AWOS not found.\n")
394                                 continue
395                             else:
396                                 print(awos)
397                                 break
398                     elif funcSelect == 2:
399                         print("AIRPORT METARS\n(type 'quit' to quit to Main
Menu)\n")
400                         while True:
401                             airport = input("Enter Airport Identifier
(Starting with K): ").lower()
402                             if airport == "quit":
403                                 killed = True
404                                 break
405                             try:
406                                 metar = a.metar(airport)
407                             except:
408                                 print("Airport not Found")
409                                 continue
410                             print(metar)
411                             break
412                     elif funcSelect == 3:
413                         funKill = True
414                         killed = True
415                         break
416
417                 if killed == True:
418                     break
```

10.2.2 Weather Script

```
'''
================================================================================
ENGR 133 Program Description
  Weather script containing functions related to weather affecting performance

Project Information
  Project Title:  Lev6B
  Author:         Christos Levy, levy30@purdue.edu
  Team ID:        002-10
================================================================================
'''
import math as m

## Redfine Sin and cos functions for easy usage
def sin(x):
    return m.sin(m.radians(x))
def asin(x):
    return m.degrees(m.asin(x))
def cos(x):
    return m.cos(m.radians(x))


## Calculates the groundspeed, the wind correction angle and the heading
def hdgGs():
    while True:
        wdir = input("Wind Direction: ")
        wspd = input("Enter Windspeed: ")
        crs = input("Enter Course: ")
        tas = input("Enter True Airspeed: ")
        try:
            wdir = float(wdir)
            wspd = float(wspd)
            crs = float(crs)
            tas = float(tas)
            break
        except:
            print("Error: All values must be numbers\n")
            continue
    angleC = abs(wdir-crs)
    gs = round(m.sqrt(wspd**2+tas**2-(2*wspd*tas*cos(angleC))),1)
    if gs == 0:
        return 0
    else:
        wca = ( asin( (wspd*sin(wdir-crs)) /gs ) )
        heading = round(crs + wca)
        return (f"GROUNDSPEED: {gs}\nHEADING: {heading}\nWCA:
{round(wca,1)}")
```

```
{round(wca,1)}")
47
48 ## Calulates the Pressure and density altitude
49 def pressureDensityAlt():
50     while True:
51         indicatedAlt = input("Enter Indicated Altitude: ")
52         baro = input("Enter Barometric Pressure: ")
53         temp = input("Enter Outside Temperature (Celsius): ")
54         try:
55             indicatedAlt = float(indicatedAlt)
56             baro = float(baro)
57             temp = float(temp)
58             break
59         except:
60             print("Error: Values must be numbers\n")
61     pAlt = round((29.92-baro)*1000+indicatedAlt)
62     degreeChange = 15 - (-2*(indicatedAlt/1000))
63     dAlt = round((temp-degreeChange)*120+pAlt)
64     return (f"PRESSURE ALTITUDE: {pAlt}\nDENSITY ALTITUDE: {dAlt}")
65
66 ## Returns windspeed and direction
67 def wind():
68     while True:
69         course = input("Enter Course: ")
70         tas = input("Enter True Airspeed: ")
71         groundspeed = input("Enter Groundspeed: ")
72         heading = input("Enter Heading: ")
73         try:
74             course = float(course)
75             tas = float(tas)
76             groundspeed = float(groundspeed)
77             heading = float(heading)
78             break
79         except:
80             print("Error: All values must be numbers")
81             continue
82     windCorrAngle = heading-course
83     wspd = (m.sqrt(groundspeed**2+tas**2-(2*(groundspeed)*
(tas)*cos(windCorrAngle))))
84     if wspd == 0:
85         return 0
86     else:
87         angleC =asin(((groundspeed*sin(heading-course))/wspd))
88         wdir = (abs(angleC+course))
89         return f"\nWIND DIRECTION: {int(wdir)}\nWIND SPEED: {round(wspd,1)}"
90
91 ## Returns Components of headwind and crosswind given the runway numbers and
the wind
92 def windComponent():
93     while True:
```

```
        wdir = input("Enter Wind direction: ")
        wspd = input("Enter Wind speed: ")
        runway = input("Enter Runway number (Runway number and not heading):
")
        try:
            wdir = int(wdir)
            wspd = float(wspd)
            runway = int(runway)
            if wdir > 360:
                print("Error: Wind Direction must be less than 360")
                continue
            if runway > 36:
                print("Error: Runway must be less than 36")
                continue
            break
        except:
            print("Error: Values must be numbers")
            continue
    runwayH = runway*10
    diff = runwayH - wdir
    ang = 90-diff
    xwind = round(wspd*cos(ang),1)
    hwind = round(wspd*sin(ang),1)

    ## Multiple nested If statements determine direction
    if xwind < 0:
        xwindSide = 'Crosswind Right'
        if hwind < 0:
            hwindSide = "Tailwind"
        elif hwind > 0:
            hwindSide = "Headwind"
        else:
            hwindSide = "No Headwind Component"
    elif xwind > 0:
        xwindSide = 'Crosswind Left'
        if hwind < 0:
            hwindSide = "Tailwind"
        elif hwind > 0:
            hwindSide = "Headwind"
        else:
            hwindSide = "No Headwind Component"
    else:
        xwindSide = "No Crosswind Component"
        if hwind < 0:
            hwindSide = "Tailwind"
        elif hwind > 0:
            hwindSide = "Headwind"
        else:
            hwindSide = "No Headwind Component"

```

```
143     return f"\n{xwindSide}: {abs(xwind)}\n{hwindSide}: {abs(hwind)}"
144 
145 ## Returns the cloud base in AGL and takes 1 parameter that selects which
    scale
146 def cloudBase(degreeSelect):
147     while True:
148         ## Fahrenheit Calculations
149         if degreeSelect == 1:
150             oat = input("Enter the outside temperature: ")
151             dewPoint = input("Enter the dew point: ")
152             try:
153                 oat = float(oat)
154                 dewPoint = float(dewPoint)
155                 CloudBase = ((oat-dewPoint)/4.4)*1000
156                 break
157             except:
158                 print("Error: Temperatures must be numbers")
159                 continue
160 
161         ## Celsius Calculations
162         elif degreeSelect == 2:
163             oat = input("Enter the outside temperature: ")
164             dewPoint = input("Enter the dew point: ")
165             try:
166                 oat = float(oat)
167                 dewPoint = float(dewPoint)
168                 CloudBase = ((oat-dewPoint)/2.5)*1000
169                 break
170             except:
171                 print("Error: Temperatures must be numbers")
172                 continue
173         else:
174             print("Error: Selection not recognized.")
175             continue
176 
177     return f"\nCloud Base: {round(CloudBase,1)} feet AGL"
178 
179 
180 ### ALL FUNCTIONS TESTED AND COMPLETED FOR WEATHER CLASS
181 
```

10.2.3 Speed Script

```
'''
================================================================================
ENGR 133 Program Description
  Speed script containing speed related functions

Project Information
  Project Title:  Lev6B
  Author:         Christos Levy, levy30@purdue.edu
  Team ID:        002-10
================================================================================
'''

import math as m

## Returns the ground speed given the time and the distance
def gs():
    while True:
        distance = input("Enter Distance covered: ")
        hours = input("Enter Hours: ")
        minutes = input("Enter Minutes: ")
        seconds = input("Enter Seconds: ")
        try:
            distance = float(distance)
            hours = float(hours)
            minutes = float(minutes)
            seconds = float(seconds)
            break
        except:
            print("Error: Values must be numbers\n")
            continue
    secToH = seconds/3600
    minToH = minutes/60
    hours += secToH + minToH
    if hours == 0:
        return 0
    else:
        groundspeed = round(distance/hours,1)
        return f"\nGroundspeed: {groundspeed} MPH"

## Returns True airpseed given the pressure altitude and indicated airspeed
def planTAS():
    while True:
        indicatedAS = input("Enter Indicated Airspeed: ")
        altitdue = input("Enter Altitude: ")
        try:
            indicatedAS,altitdue = float(indicatedAS),float(altitdue)
            break
        except:
```

```
        except:
            print("Error: All values must be numbers")
            continue
    TrueAS = round(indicatedAS + (.02*altitdue/1000),1)
    return f"\nTrue Airspeed: {TrueAS}\n"

## Returns the true airpseed based on the pressure altitude and the outside temperature. Uses the NOAA equation for pressure and various constants
def actTAS():
    while True:
        pressureAlt = input("Enter Pressure Altitdue: ")
        temp = input("Enter Temperature in Celsius: ")
        ias = input("Enter indicated airpseed: ")
        try:
            pressureAlt,temp,ias = float(pressureAlt),float(temp),float(ias)
            break
        except:
            print("Error: Numbers must be numeric.\n")
            continue
    tempK = 273.15+temp
    if tempK == 0:
        return 0
    dalt = round(pressureAlt + 120*(temp-(15-(pressureAlt/1000)*2)),1)
    pressureinPascals = 100*(m.exp(m.log(1-(pressureAlt/145366.45))/0.190284)*1013.25)
    density = pressureinPascals/(287.058*tempK)
    if density == 0:
        return 0
    tas = round(ias*m.sqrt(1.225/density),1)
    return f"\nTAS: {tas}\nDenstiy Altitude: {dalt}\n"



### ALL FUNCTIONS TESTED AND COMPLETED FOR SPEED CLASS


```

10.2.4 Required Functions Script

```
'''
================================================================================
ENGR 133 Program Description
  Required script that runs various functions calculating 'necessary' values

Project Information
  Project Title:  Lev6B
  Author:         Christos Levy, levy30@purdue.edu
  Team ID:        002-10
================================================================================
'''



import math as m

## REDEFINE TRIG FUNCTIONS
def cos(x):
    return m.cos(m.radians(x))
def sin(x):
    return m.sin(m.radians(x))
def asin(x):
    return m.degrees(m.asin(x))

## Returns the required amount of fuel necessary for a given time and fuel burn
def fuel():
    while True:
        hours = input("Enter hours: ")
        minutes = input("Enter minutes: ")
        seconds = input("Enter seconds: ")
        fph = input("Enter Fuel burn: ")
        try:
            hours = float(hours)
            minutes = float(minutes)
            seconds = float(seconds)
            fph = float(fph)
            break
        except:
            print("Error: All values must be numbers\n")
            continue
    mintoH = minutes/60
    sectoH = seconds/3600
    hours += mintoH + sectoH
    fuel = round(fph*hours,1)
    return f"\nRequired Fuel: {fuel} Gallons"

## Returns the required rate of climb given the minimum climb and the groundspeed
```

```
groundspeed
def climb():
    while True:
        minClimb = input("Enter Minimum climb in feet per mile: ")
        groundspeed = input("Enter groundspeed: ")
        try:
            minClimb = float(minClimb)
            groundspeed = float(groundspeed)
            break
        except:
            print("Error: All values must be numbers\n")
            continue
    requiredClimb = round((minClimb*groundspeed)/60,1)
    return f"\nRequired Rate of Climb: {requiredClimb} Feet Per Minute"

## Returns the rate of descent
def descent():
    while True:
        ialt = input("Enter Indicated Altitude: ")
        destAlt = input("Enter Crossing Altitude: ")
        groundspeed = input("Enter Groundspeed: ")
        fixDist = input("Enter Fix Distance: ")
        try:
            ialt = float(ialt)
            destAlt = float(destAlt)
            groundspeed = float(groundspeed)
            if groundspeed == 0:
                return 0
            fixDist = float(fixDist)
            break
        except:
            print("Error: All values must be numbers\n")
            continue
    diffAlt = ialt-destAlt
    if groundspeed == 0:
        return 0
    time = fixDist/groundspeed
    minutes = time*60
    if minutes == 0:
        return 0
    descentRate = int(round(diffAlt/minutes,-1))
    return f"\nRequired Rate of Descent: {descentRate} Feet Per Minute"

## Returns true airspeed given wind direction, wind speed, course and
grounspeed
def tas():
    while True:
        wdir = input("Enter Wind Direction: ")
        wspd = input("Enter Wind Speed: ")
        course = input("Enter Course: ")
```

```
        groundspeed = input("Enter Groundspeed: ")
        try:
            wdir,wspd,course,groundspeed = float(wdir),float(wspd),float(course),float(groundspeed)
            break
        except:
            print("Error: Values must be numbers\n")
            continue
    angleB = abs(wdir-course)
    if groundspeed == 0:
        return 0
    angleC = asin((sin(angleB)*wspd)/(groundspeed))
    angleA = 180-angleB-angleC
    tas = round(m.sqrt(groundspeed**2+wspd**2-(2*(groundspeed)*(wspd)*cos(angleA))),1)
    hdg = int(round(course+angleC,0))
    return f"TAS: {tas}\nHeading: {hdg}\n"

## Returns the amount of money needed to make a trip given the cost of fuel and the distance/time
def money():
    while True:
        selector = input("\n1. Calculate by distance\n2. Calculate by time\n")
        try:
            selector = int(selector)
        except:
            print("Error: Enter 1 for distance calculation or 2 for time calculations")
            continue

        if selector == 1:
            while True:
                fuelCost = input("Enter average fuel cost: ")
                fuelBurn = input("Enter fuel burn: ")
                distance = input("Enter Trip Distance: ")
                groundspeed = input("Enter Groundspeed: ")
                try:
                    fuelCost = float(fuelCost)
                    distance = float(distance)
                    fuelBurn = float(fuelBurn)
                    groundspeed = float(groundspeed)
                except:
                    print("Error: Values must be numbers\n")
                    continue
                time = distance/groundspeed
                totalCost = round(fuelCost*fuelBurn*time,2)
                break
            return f"Fuel Cost for {distance} Miles: ${totalCost}"
            break
```

```
140
141         elif selector == 2:
142             while True:
143                 fuelCost = input("Enter average fuel cost: ")
144                 fuelBurn = input("Enter fuel burn: ")
145                 hours = input("Enter Hours: ")
146                 minutes = input("Enter Minutes: ")
147                 seconds = input("Enter Seconds: ")
148                 try:
149                     fuelCost = float(fuelCost)
150                     fuelBurn = float(fuelBurn)
151                     hours = float(hours)
152                     minutes = float(minutes)
153                     seconds = float(seconds)
154                 except:
155                     print("Error: Values must be numbers\n")
156                     continue
157                 hours += (seconds/3600) + (minutes/60)
158                 totalCost = round(fuelCost*fuelBurn*hours,2)
159                 break
160             return f"Fuel Cost for {round(hours,1)} hours: ${totalCost}"
161             break
162         else:
163             print("Error: Function not recognized.")
164             continue
165
166
167
168 ### ALL FUNCTIONS TESTED AND COMPLETED FOR REQUIRED CLASS
169
170
171
172
173
174
175
176
```

10.2.5 Flight Script

```
'''
================================================================================
ENGR 133 Program Description
  Flight script containing functions related to inflight calculations

Project Information
  Project Title:  Lev6B
  Author:         Christos Levy, levy30@purdue.edu
  Team ID:        002-10
================================================================================
'''

## Takes time and ground speed and returns the distance flown
def distFln(gs,hr,min,sec):
    minToH = min/60
    secToH = sec/3600
    totalHours = minToH + secToH + hr
    dist = round(gs*totalHours,1)
    return f"{dist} Mile(s)"

## Returns the distance that is the top of the descent
def topDscn(gs,ialt,dalt,rate):
    alt = ialt-dalt ## Altitude to descend
    if rate == 0:
        return 0
    else:
        timeM = alt/rate ## Time in minutes
        timeH = timeM/60
        dist = round(timeH*gs,1)
        return f"{dist} Mile(s)"

## Returns the Endurance of the Aircraft based on the amount of fuel it contains and its fuel burn
def endur(fuel,fph):
    if fph == 0:
        return 0
    else:
        hours = fuel/fph
        if hours % 1 == 0:
            return f"{hours} Hour(s)"
        else:
            remainderM = hours % 1
            minutes = 60*remainderM
            remainderS = minutes%1
            seconds = int(remainderS*60)
            return f"{int(hours)} Hour(s) {int(minutes)} minute(s) {seconds} second(s)"
```

```
47 ## Returns the time the leg will take given the ground speed and the distance
48 def legTime(distance,gs):
49     if gs == 0:
50         return 0
51     else:
52         time = distance/gs
53         if time%1 == 0:
54             return f"{time} Hours"
55         else:
56             remM = time%1
57             hours = int(time)
58             min = (remM*60)
59             sec = (min%1)*60
60             sec = int(sec)
61             return f"{hours} Hour(s) {int(min)} Minute(s) {sec} Second(s)"
62 
63 ## Calulates the Range at altitude given the amount of fuel, the fuel burn,
   and the ground speed
64 def spcRange(fuel,fph,gs):
65     if fph == 0:
66         return 0
67     else:
68         time = fuel/fph
69         range = round(time*gs,1)
70         return f"{range} Mile(s)"
71 
72 ## Calculates the fuel burn given the time and the amount of fuel burned
73 def fuelPerHour(fuel,hour,min,sec):
74     minToH = min/60
75     secToH = sec/3600
76     time = hour+minToH+secToH
77     if time == 0:
78         return 0
79     else:
80         fph = fuel/time
81         return f"{round(fph,1)} Gallons per Hour"
82 
83 
84 
85 
86 ### ALL FUNCTIONS TESTED AND COMPLETED FOR FLIGHT CLASS
```

10.2.6 Weight and Balance Script

```
'''
================================================================================
ENGR 133 Program Description
  WeightBalance Script containing the Weight and balance calculator function

Project Information
  Project Title:  Lev6B
  Author:         Christos Levy, levy30@purdue.edu
  Team ID:        002-10
================================================================================
'''
def inf():
    while True:
        yield


## Calculates the CG, total moment and total weight
def weightArm():
    WeightList = []
    totalMoment = 0
    centerGravity = 0
    for i in inf():
        weight = input("Enter Weight: ")
        arm = input("Enter Arm: ")
        try:
            weight = float(weight)
            arm = float(arm)
            moment = weight*arm
            WeightList.append(weight)
            totalWeight = sum(WeightList)
            totalMoment += moment
            if totalWeight == 0:
                centerGravity += 0
            else:
                centerGravity = totalMoment/totalWeight
        except:
            print("Error: Values were not numbers")
            continue
        print(f"\nMOMENT: {totalMoment}\nGROSS WEIGHT: {totalWeight}\nCG: {round(centerGravity,2)}")
        goAgain = input("Press Enter to enter another value or type '/'+ enter to quit:\n")
        if goAgain == "":
            continue
        else:
            print(f"MOMENT: {totalMoment}\nGROSS WEIGHT: {totalWeight}\nCG: {round(centerGravity,2)}\n")
            break
```

```
            break


## Calculates the CG, total moment and total weight without asking for arm
def weightMom():
    WeightList = []
    totalMoment = 0
    centerGravity = 0
    for i in inf():
        weight = input("Enter Weight: ")
        moment = input("Enter Moment: ")
        try:
            weight = float(weight)
            moment = float(moment)
            WeightList.append(weight)
            totalWeight = sum(WeightList)
            totalMoment += moment
            centerGravity = totalMoment/totalWeight
        except:
            print("Error: Values were not numbers")
            continue
        print(f"\nMOMENT: {totalMoment}\nGROSS WEIGHT: {totalWeight}\nCG: {round(centerGravity,2)}")
        goAgain = input("Press Enter to enter another value or type '/'+ enter to quit:\n")
        if goAgain == "":
            continue
        else:
            print(f"MOMENT: {totalMoment}\nGROSS WEIGHT: {totalWeight}\nCG: {round(centerGravity,2)}\n")
            break

## Calculates the percent of the mean aerodynamic chord
def mac():
    for i in inf():
        cg = input("Enter CG: ")
        Lemac = input("Enter Leading Edge Mean Aerodynamic Chord: ")
        mac = input("Enter mean aerodynamic chord: ")
        try:
            cg = float(cg)
            Lemac = float(Lemac)
            mac = float(mac)
            if mac == 0:
                print("Error: MAC Cannot be 0")
                continue
            else:
                percentMac = ((cg - Lemac)/mac)*100
                print(f"Percent MAC: {round(percentMac,1)}%")
                break
        except:
```

```
            print("Error: Values were not numbers")
            continue



### ALL FUNCTIONS TESTED AND COMPLETED FOR WEIGHT AND BALANCE CLASS
```

10.2.7 Conversion and Clock Script

```
#/usr/bin/python3
'''
================================================================================
ENGR 133 Program Description
  Watch script containing stopwatch,timer and a conversion function

Project Information
  Project Title:  Lev6B
  Author:         Christos Levy, levy30@purdue.edu
  Team ID:        002-10
================================================================================
'''

import time


## This function prints a stopwatch on the screen
def stopwatch():
    print("STOPWATCH FUNCTION\nType Control-C to end")
    ## Get Start Input
    timeLoop = False
    while True:
        start = input("Would you like to begin Timing? (y/n): ").lower()
        if start == "y":
            timeLoop = True
            break
        elif start == "n":
            break
        else:
            print("Error: Enter (Y/N)\n")

    ## Initiate Variables at 0
    Sec = 0
    Min = 0
    Hour = 0

    ## Begin timeLoop that prints time
    while timeLoop:
        try:
            if Hour == 0:
                Sec+=1
                print(f"{Min} Minute(s) {Sec} Second(s)", end = '\r')
                time.sleep(1)

                if Sec == 60:
                    Sec = 0
                    Min += 1
                if Min > 60:
```

```
                if Min > 60:
                    Hour += 1
            elif Hour > 0:
                Sec += 1
                print(f"{Hour} Hour(s) {Min} Minute(s) {Sec} second(s)", end
= "\r")
                time.sleep(1)

                if Sec == 60:
                    Sec = 0
                    Min += 1
                if Min > 60:
                    Hour += 1
        except KeyboardInterrupt: ## Need a better way to kill the program.
            timeLoop = False
            break

## This function Counts down like a timer
def timer(hour,min,sec):
    timerDone = False

    ## Converts hours to minutes to make the code less complex
    min += hour*60
    while True:
        try:
            if sec > 0:
                sec -= 1
                print(f"{min} Minute(s) {sec} Second(s)" , end = "\r")
                time.sleep(1)

            elif sec == 0:
                min-=1
                sec = 60
                print(f"{min} Minute(s) {sec} Second(s)", end = "\r")
                time.sleep(1)
            if sec == 0 and min == 0:
                timerDone = True
                break
        except KeyboardInterrupt: ## Need a better way to kill the program.
            break

    while timerDone:
        try:
            print("Timer Done!")
            time.sleep(0.5)
        except KeyboardInterrupt: ## Need a better to kill program
            break

## This function converts degrees celsius to fahrenheit and vice versa
def tempConvert(type,degrees):
```

```
97      if type == "1": ##Convert to celsius
98          celsius = (5/9)*(degrees-32)
99          return f"{round(celsius,1)} Degrees Celsius"
100     elif type == "2": ## Convert to farenheit
101         faren = (9/5)*degrees + 32
102         return f"{round(faren,1)} Degrees Fahrenheit"
103     else:
104         return
105
106
107 ### ALL FUNCTIONS TESTED AND COMPLETED FOR WATCH CLASS
```

10.2.8 AWOS/METAR Script

```
'''
================================================================================
ENGR 133 Program Description
  AWOS script that scrapes the internet for AWOS phone number and METAR data

Project Information
  Project Title:  Lev6B
  Author:         Christos Levy, levy30@purdue.edu
  Team ID:        002-10
================================================================================
'''

## Import Webscrape modules
from bs4 import BeautifulSoup as bs
from urllib.request import urlopen
import datetime
import os

## Contains all the different abbreviations for weather
def translator(stringOfAbbre):
    weatherStatements = []
    a = stringOfAbbre
    if "-" in a:
        weatherStatements.append("Light")
    if "+" in a:
        weatherStatements.append("Heavy")
    if "VC" in a:
        weatherStatements.append("In the Vicinity")
    if "MI" in a:
        weatherStatements.append("Shallow")
    if "PR" in a:
        weatherStatements.append("Partial")
    if "BC" in a:
        weatherStatements.append("Patches")
    if "DR" in a:
        weatherStatements.append("Low Drifting")
    if "BL" in a:
        weatherStatements.append("Blowing")
    if "SH" in a:
        weatherStatements.append("Showers")
    if "TS" in a:
        weatherStatements.append("Thunderstorm")
    if "FZ" in a:
        weatherStatements.append("Freezing")
    if "DZ" in a:
        weatherStatements.append("Drizzle")
    if "RA" in a:
        weatherStatements.append("Rain")
```

```
        weatherStatements.append("Rain")
    if "SN" in a:
        weatherStatements.append("Snow")
    if "SG" in a:
        weatherStatements.append("Snow Grains")
    if "IC" in a:
        weatherStatements.append("Ice Crystals")
    if "PL" in a:
        weatherStatements.append("Ice Pellets")
    if "GR" in a:
        weatherStatements.append("Hail")
    if "GS" in a:
        weatherStatements.append("Small Hail/Snow Pellets")
    if "UP" in a:
        weatherStatements.append("Unknown Precipitation")
    if "BR" in a:
        weatherStatements.append("Mist")
    if "FG" in a:
        weatherStatements.append("Fog")
    if "FU" in a:
        weatherStatements.append("Smoke")
    if "VA" in a:
        weatherStatements.append("Volcanic Ash")
    if "DU" in a:
        weatherStatements.append("Widespread Dust")
    if "SA" in a:
        weatherStatements.append("Sand")
    if "HZ" in a:
        weatherStatements.append("Haze")
    if "PY" in a:
        weatherStatements.append("Spray")
    if "PO" in a:
        weatherStatements.append("Sand Whirls")
    if "SQ" in a:
        weatherStatements.append("Squalls")
    if "FC" in a:
        weatherStatements.append("Funnel Cloud")
    if "SS" in a:
        weatherStatements.append("Sandstorm")
    if "DS" in a:
        weatherStatements.append("Duststorm")

    return weatherStatements

## Contains all the different abbreviations for sky conditions
def skyTranslator(stringofSkyCond):
    s = stringofSkyCond
    skyCond = []
    if "NCD" in s:
        skyCond.append("Nil Clouds Detected")
```

```
98          skyCond.append(int(s[3:len(s)])*100)
99      if "CLR" in s:
100         skyCond.append("Clear Skies")
101         skyCond.append(int(s[3:len(s)])*100)
102     if "FEW" in s:
103         skyCond.append("Few Clouds")
104         skyCond.append(int(s[3:len(s)])*100)
105     if "SCT" in s:
106         skyCond.append("Scattered Clouds")
107         skyCond.append(int(s[3:len(s)])*100)
108     if "BKN" in s:
109         skyCond.append("Broken Clouds")
110         skyCond.append(int(s[3:len(s)])*100)
111     if "OVC" in s:
112         skyCond.append("Overcast")
113         skyCond.append(int(s[3:len(s)])*100)
114     if "VV" in s:
115         skyCond.append("Vertical Visibilty")
116         skyCond.append(int(s[3:len(s)])*100)
117     return skyCond
118
119 ## Finds the AWOS of the entered airport and prints frequency and phone
    number
120 def awos(airportID):
121     html = urlopen("http://airnav.com/airport/"+airportID) #Opens URL and
    converts to HTML
122     airNav = bs(html,features="html.parser") # Converts to a soup file
123     f = open("Webdata.txt",'w+') # Opens a file to place webdata
124
125     ## Loop finds all table instances in HTML and writes them to the file
126     for strings in airNav.find_all('td'):
127         f.write(strings.get_text())
128     f.close()
129
130     ## Searches table HTML content for AWOS or ASOS
131     with open('Webdata.txt','r') as f:
132         for line in f:
133             line.rstrip('\n')
134             if "ASOS" in line or "AWOS" in line:
135                 return line
136                 directory = os.getcwd()+'/Webdata.txt'
137                 os.remove(directory)
138                 break
139
140 ## Finds the METAR of an airport and prints out a translated version using
    the translator functions
141 def metar(airportID):
142     ## Checks if airport has a metar
143     for a in list(airportID):
144         try:
```

```
            a = int(a)
            return("Airport does not have a METAR.\n")
            break
        except:
            continue

    ## Opens Webpage
    html = urlopen("http://aviationweather.gov/metar/data?ids="+airportID+"&format=raw&date=&hours=0")
    metarPage = bs(html,features="html.parser")

    ## Creates a list of every metar attribute
    for metarLine in metarPage.find_all('code'):
        fullMetar = metarLine.get_text()
    metar = fullMetar.split()

    ## Removes unnecessary info
    for i in metar:
        if i == 'RMK':
            for j in range(len(metar)-1,metar.index(i),-1):
                del metar[j]
            metar.remove(i)
        if i == "AUTO":
            metar.remove(i)

    ## Constants
    airport = metar[0]

    ## TIME VALUES (DATE AND ZULU TIME)
    dt = datetime.datetime.today()
    time = metar[1]
    date = f"{dt.month}/{int(time[0:2])}/{dt.year}"
    zuluTime = f"{time[2:6]} Zulu"

    ## WIND VALUES (direction,velocity,gust(try))
    wind = metar[2]
    if "VRB" in wind:
        visibility = metar[3]
        velocity = f"{wind[3:5]} knots"
        if "G" in wind:
            gust = f" Gusting {wind[6:8]} knots"
        direction = "Variable Direction"
    if "V" in metar[3]:
        variation = metar[3]
        visibility = metar[4]
        direction = f"Wind Variable between {variation[0:2]} and {variation[3:6]}"
    else:
        direction = f"{wind[0:3]} Degrees"
        velocity = f"{wind[3:5]} knots"
```

```
        visibility = metar[3]
        if "G" in wind:
            gust = f"Gusting {wind[6:8]} knots"


    # VISIBILITY VALUES(visibiltyNum)
    if "SM" not in visibility:
        addOn = float(visibility)
        visibility = metar[metar.index(visibility)+1]
    else:
        addOn = 0

    if "/" in visibility:
        frac = True
    else:
        frac = False
    visibilityNumList = visibility.split("SM")

    if frac == True:
        fracList = visibilityNumList[0].split("/")
        num = float(fracList[0])/float(fracList[1])
        visibilityNum = f"{num+addOn} Statute Miles"
    try:
        visibilityNum = f"{float(visibilityNumList[0])+addOn} Statute
Mile(s)"
    except:
        if "M" in visibility[0]:
            visibilityNum = f"Less than 1/4 Statute Miles"
        if "P" in visibility:
            visibilityNum = f"Greater than 10 Statute Miles"


    ## WEATHER CHECKER (weather and sky conditions)
    listofSkyConds = []
    if len(metar[metar.index(visibility)+1]) < 6 or
len(metar[metar.index(visibility)+1]) > 6:
        weatherL = metar[metar.index(visibility)+1] ## Statement is made for
both cases of visibility (being index 3 or 4)
        weather = translator(weatherL)
        counter = metar.index(weatherL)+1
    else:
        weather = "No Visible Weather Conditions"
        listofSkyConds.append(metar[metar.index(visibility)+1])
        counter = metar.index(visibility)+2



    while True:
        if not("/" in metar[counter]):
```

```
        if not("/" in metar[counter]):
            listofSkyConds.append(metar[counter])
            counter+=1
        else:
            break
    skyCond = []

    ## Translates SkyConds:
    for i in listofSkyConds:
        skyCond.append(skyTranslator(i))

    ## Temperature
    tempDew = metar[counter]
    tempDew = tempDew.split("/")
    for t in range(len(tempDew)):
        if "M" in tempDew[t]:
            tempDew[t] = -int(tempDew[t][1:len(tempDew[t])])
        else:
            tempDew[t] = tempDew[t]
    tempDew = f"Temperature: {int(tempDew[0])} Degrees Celsius\nDew Point:
{int(tempDew[1])} Degrees Celsius"

    ## Altimeter
    altimeter = metar[counter+1]
    altimeter = altimeter[1:len(altimeter)]
    altimeter = f"Altimeter: {altimeter[0:2]}.{altimeter[2:len(altimeter)]}"


    ## Format Variables into strings
    html = urlopen("http://airnav.com/airport/"+airportID) #Opens URL and
converts to HTML
    airNav = bs(html,features="html.parser") # Converts to a soup file
    airportName = airNav.title.string
    airportName = airportName[8:len(airportName)]

    try:
        return(f"\nFull Metar:
{fullMetar}\n{airportName}\n{date}\n{zuluTime}\nWind: {direction} at
{velocity}, {gust}\nVisibility:
{visibilityNum}\n{weather}\n{skyCond}\n{tempDew}\n{altimeter}")
    except:
        return(f"\nFull Metar:
{fullMetar}\n{airportName}\n{date}\n{zuluTime}\nWind: {direction} at
{velocity}\nVisibility:
{visibilityNum}\n{weather}\n{skyCond}\n{tempDew}\n{altimeter}")
```

```
281
282 ### ALL FUNCTIONS TESTED AND COMPLETED FOR AWOS CLASS
283
284
```