Overview

This project will combine all of your previous projects. You will manually display data using a raw drawing module called *tkinter*. Visualizing data is an important tool to data science. This project is challenging, but you will see that essentially you can do anything you want provided you know how to draw it!

Assignment

You will be parsing a data file and drawing a visualization of the data. The data file is a list of Meterological Aerodrome Reports (METARs for short). These give pilots the current conditions over an airport, such as McGhee-Tyson Airport in Knoxville (KTYS).

METAR and Data File Format

KMOR 311435Z AUTO 27007G14KT 10SM 23/17 A2995 RMK A01

KTYS 311353Z 28009KT 10SM 23/17 A2995 RMK AO2 SLP132 T02330167

KLSV 311355Z 29002KT 10SM 21/07 A2992 RMK AO2A SLP093 WND DATA ESTMD T02140071

KLAS 311356Z 01006KT 4SM 22/08 A2990 RMK AO2 SLP102 T02220078

KVGT 311353Z 22003KT 1SM 21/08 A2991 RMK AO2 SLP125 T02060083

PHNL 310153Z 07007KT 3SM 26/19 A3001 RMK AO2 SLP162 T02560189

KBNA POS 312353Z 00000KT 1/2SM 08/01 A2997 RMK AO2 SLP140 T02500144

You will notice that the METAR always begins with a four letter identifier. This identifies the airport (KMOR is Morristown, KTYS is McGhee-Tyson, KLSV is Nellis AFB, Nevada, etc). For purposes of simplifing this project, I removed all ceilings (such as SCT040, BKN100, etc.) in case you're familiar with METARs.

A 6 digit timecode followed by a 'Z' (for Zulu time) comes next. The first two digits are the day of the month. In the examples below, 31  is the 31st day. Notice that the month and year are not given. The rest of the four digits are the 24-hour zulu time of the report. 1435Z means 2:35pm in Zulu time zone. Currently, our time zone is -4 from Zulu, so this would be a report from 10:35am.

The field that ends in KT (knots for wind speed) is the wind direction and speed. Notice that KMOR has AUTO  listed, you must be able to detect the wind speed (notice that the wind speed always ends in KT [knots]). The first three digits are the compass direction (clockwise from 000 (Magnetic North) to 090 (Magnetic East) to 180 (Magnetic South) to 270 (Magnetic West) and back to 000). The last two digits are the speed in knots. So 09 is 9 knots of wind speed. The entire field 01006KT would be wind speed of 6 knots from the north-north-east direction. A value of 00000KT would mean "calm" winds. You will notice that one field (for KMOR) has a G14. This means that the winds are steady at 7 knots but gust up to 14 knots. Your program must be able to decode the G (gust) field if one exists.

The field that ends in SM is the visibility. SM means "statute miles" (rather than nautical miles). So 10SM means 10 miles of visibility. No visibility can go above 10; however, some may be fractional, such as 1/2SM, which means 0.5 miles of visibility. Visibility can be modified with a P or M for "more than" or "less than", respectively. For example, P5SM would mean more than 5 miles of runway visibility, but not enough for 6 SM of visibility.

The next field contains the temperature over the dew point in celsius. For example, 21/07 means 21 degrees with a dew point of 7 degrees (all in celsius). The field can be modified with an M which means minus for negative values. For example M01/M10 would mean -1C temperature with a -10C dewpoint.

The next field starts with an A, which stands for "altimeter" for altitude meter. This is the value that pilots must set their altimeter. This value is in inches of mercury (inHg). There is an implied decimal after the first two digits. For example, A2984 means that the altimeter must be set to 29.84 inHg.

Everything starting with and after *RMK*  (remark) must be ignored. Any other field not mentioned must also be ignored, such as AUTO.

You must NOT detect the fields based on their position, except the airport identifier. The only field guaranteed to be in position is the airport identifier (the first 4 letters). You will notice that each field either ends with a suffix, such as KT, SM, or Z or a prefix, such as A, or it has a well defined format, such as TT/DD for temperature over dewpoint.

METARs can contain several special symbols, such as VRB  for variable wind directions. Using judgment we can see that winds of variable direction means we cannot put an azimuth indicator on the wind direction, but we can put down wind speed.

I will not test your code with erroneous data, however you should have error checking in your code! By no means should your program report incorrect data or crash if it cannot understand a given symbol.

For more information about reading a METAR, please see: [How to Decode METAR, TAF, and pilot reports (nps.edu)Links to an external site.](https://met.nps.edu/~bcreasey/mr3222/files/helpful/DecodeMETAR-TAF.html)

This site contains much more information about a METAR than we are going to test, but since you will be retrieving real-world weather reports, you might encounter information we haven't covered. So, the website above might help you discern what the METAR actually means.

Getting Data

You will be getting data from aviationweather.gov.

[AWC - METeorological Aerodrome Reports (METARs) (aviationweather.gov) Links to an external site.](https://aviationweather.gov/cgi-bin/data/metar.php?ids=KTYS&hours=0)

import requests as rq  
data = rq.get("https://aviationweather.gov/cgi-bin/data/metar.php?ids=KTYS&hours=0”)

After all of this above, we get the METAR data, which when I wrote this looks as follows:

KTYS 130153Z 00000KT 9SM CLR 01/01 A3021 RMK AO2 SLP236 T00110006

So, you can use this code to download data using the requests module.

Graphical User Interface

For the graphical user interface, you will be using the tkinter  package, which is built into most Python builds (including repl.it).

See tkinter  tutorial from the menu on the left.

Starter Code

You may use the following code to get started. This code creates the dropdown box for you and a canvas so you can draw:

import tkinter as tk

def run(metar):

# Create the root Tk()

root = tk.Tk()

# Set the title

root.title("COSC505 - Weather")

# Create two frames, the list is on top of the Canvas

list\_frame = tk.Frame(root)

draw\_frame = tk.Frame(root)

# Set the list grid in c,r = 0,0

list\_frame.grid(column=0, row=0)

# Set the draw grid in c,r = 0,1

draw\_frame.grid(column=0,row=1)

# Create the canvas on the draw frame, set the width to 800 and height to 600

canvas = tk.Canvas(draw\_frame, width=800, height=600)

# Reset the size of the grid

canvas.pack()

# THESE ARE EXAMPLES! You need to populate this list with the available airports in the METAR

# which is given by metar parameter passed into this function.

choices = ["KONE", "KTWO", "KTHRE"]

# Create a variable that will store the currently selected choice.

listvar = tk.StringVar(root)

# Immediately set the choice to the first element. Double check to make sure choices[0] is valid!

listvar.set(choices[0])

# Create the dropdown menu with the given choices and the update variable. This is stored on the

# list frame. You must make sure that choices is already fully populated.

dropdown = tk.OptionMenu(list\_frame, listvar, \*choices)

# The dropdown menu is on the top of the screen. This will make sure it is in the middle.

dropdown.grid(row=0,column=1)

# This function is called whenever the user selects another. Change this as you see fit.

def drop\_changed(\*args):

canvas.delete("airport\_text")

canvas.create\_text(100, 100, text=listvar.get(), fill="black", tags="airport\_text")

# Listen for the dropdown to change. When it does, the function drop\_changed is called.

listvar.trace('w', drop\_changed)

# You need to draw the text manually with the first choice.

drop\_changed()

# mainloop() is necessary for handling events

tk.mainloop()

# Entry point for running programs

if \_\_name\_\_ == "\_\_main\_\_":  
 data = requests.get("https://aviationweather.gov/cgi-bin/data/metar.php??ids=KTYS&format=raw&date=&hours=0")  
 # Parse data here

run(data)

Notice that no file handling has been added. This is your job. When you run the program, you should provide it with a metar file that contains the list of airports. The choices needs to be populated with the available airports. Whenever the user selects an airport, the canvas should reset and display the values of the selected airport.

Do not change the location of the canvas and dropdown box, but for everything else, you are free to change it.

Data Structures

Using a dictionary to store each airport and a nested dictionary for the information is probably the best way to store the information for easy recall. For example, the data file given above would be parsed into the following:

{

'KMOR': {'date': 31, 'utc': 1435, 'wind\_dir': 270, 'wind\_speed': 7, 'wind\_gust': 14, 'vis': 10, 'degrees': 23, 'dewpoint': 17, 'altimeter': 29.95},

'KTYS': {'date': 31, 'utc': 1353, 'wind\_dir': 280, 'wind\_speed': 9, 'wind\_gust': 0, 'vis': 10, 'degrees': 23, 'dewpoint': 17, 'altimeter': 29.95},

'KLSV': {'date': 31, 'utc': 1355, 'wind\_dir': 290, 'wind\_speed': 2, 'wind\_gust': 0, 'vis': 10, 'degrees': 21, 'dewpoint': 7, 'altimeter': 29.92},

'KLAS': {'date': 31, 'utc': 1356, 'wind\_dir': 10, 'wind\_speed': 6, 'wind\_gust': 0, 'vis': 4, 'degrees': 22, 'dewpoint': 8, 'altimeter': 29.9},

'KVGT': {'date': 31, 'utc': 1353, 'wind\_dir': 220, 'wind\_speed': 3, 'wind\_gust': 0, 'vis': 1, 'degrees': 21, 'dewpoint': 8, 'altimeter': 29.91},

'PHNL': {'date': 31, 'utc': 153, 'wind\_dir': 70, 'wind\_speed': 7, 'wind\_gust': 0, 'vis': 3, 'degrees': 26, 'dewpoint': 19, 'altimeter': 30.01},

'KBNA': {'date': 31, 'utc': 2353, 'wind\_dir': 0, 'wind\_speed': 0, 'wind\_gust': 0, 'vis': 0.5, 'degrees': 8, 'dewpoint': 1, 'altimeter': 29.97}

}

Notice that when parsed, the date, utc, wind\_dir, etc are valid data types (integers or floats). You are free to store the information as you see fit as long as it can be recalled correctly.

Presentation

You will present the data by drawing "widgets" manually, including the altimeter, wind data, visibility, and temperature.

You will need to draw the following "widgets". Try to make these look close to the example shown below, but as long as they're close, you will receive credit.

| **Widget** | **x, y** | **width, height** | **Type** |
| --- | --- | --- | --- |
| Wind gauge | 300, 100 | 100, 100 | Oval |
| Temp gauge | 500, 100 | 100, 200 | Rectangle |
| Altimeter gauge | 300, 250 | 100, 100 | Oval |
| Visibility gauge | 300, 400 | 400, 100 | Rectangle |
| Airport name | 100, 100 | n/a | Text |
| Date | 100, 140 | n/a | Text |

The wind gauge has a small red dot in the center. The wind direction can be created using a line with an arrow, or using an arc. However, the magnetic north direction (straight up) is 0, whereas with graphs 0 would be the east direction (straight to the right).

You will have to convert the windspeed from knots to miles per hour. Do this by multiplying knots by 1.15. Remember, when you multiply, you'll get a float. However, for the windspeed, we want whole numbers only. If the winds are calm (0), then only print "CALM". Output the gust speed ONLY if a gust exists.

You will have to convert the time from a 24-hour clock to a 12-hour clock. Remember, 0000 is 12:00am, 1200 is 12:00pm. In other words, subtract 1200 from the time. If it's negative, you're in "am", otherwise, you're in "pm".

You will have to convert both the temperature and dewpoint into Fahrenheit. Remember, it is given in a METAR as Celsius. The temperature gauge has a range from 0 to 90 degrees. Notice that the dewpoint (blue part of gauge) is overlaid over the temperature (red part of the gauge).

The visibility gauge increases to the right from 0SM visibility to 10SM visibility. Therefore, if an airport has 10SM of visibility, the gauge should be fully orange. Otherwise, it should be some ratio of orange and gray depending on the amount of visibility.

When the user selects another airport, the entire screen will erase .delete("all")  and all widgets will be redrawn with the new data.

Finished Product

When completed, you should see the following:

A screenshot of a computer

Description automatically generated

The following shows KBNA, which I modified for illustration purposes:

A screenshot of a computer

Description automatically generated

Requirements

Your code must contain the following:

1. Functions for the useful systems, such as parsing a METAR line and returning a dictionary, reading the file, etc.
2. Comments of your code that documents the *logic*  of the code.
3. Mostly optimized code. You must use mutual exclusivity (else statements) where necessary. Do not make the computer calculate what is already known. Also, avoid duplicating code. If you have to run the same code more than once, use a function and / or a loop.

Submission

Submit your .py file. All of your code should be in one file and be well commented so I know what to look for!

Testing

All tests should be on the aviationweather.gov website, since this is the primary source of data. However, the weather constantly changes, so these files may contain additional things, such as AUTO, VRB (variable), and fractional visibility. You can test multiple airports by separating identifiers with commas, such as [https://aviationweather.gov/cgi-bin/data/metar.php?ids=KTYS%2CKBNA%2CKLAS%2CKLSV%2CKVGT&format=raw&date=&hours=0Links to an external site.](https://aviationweather.gov/cgi-bin/data/metar.php?ids=KTYS%2CKBNA%2CKLAS%2CKLSV%2CKVGT&format=raw&date=&hours=0Links%20to%20an%20external%20site.).

Plagiarism

This is an individual assignment.