Roslyn Melookaran
5/31/20
Foundations of Programming Python
Assignment 7
https://github.com/roslynm/IntroToProg-Python-Mod07

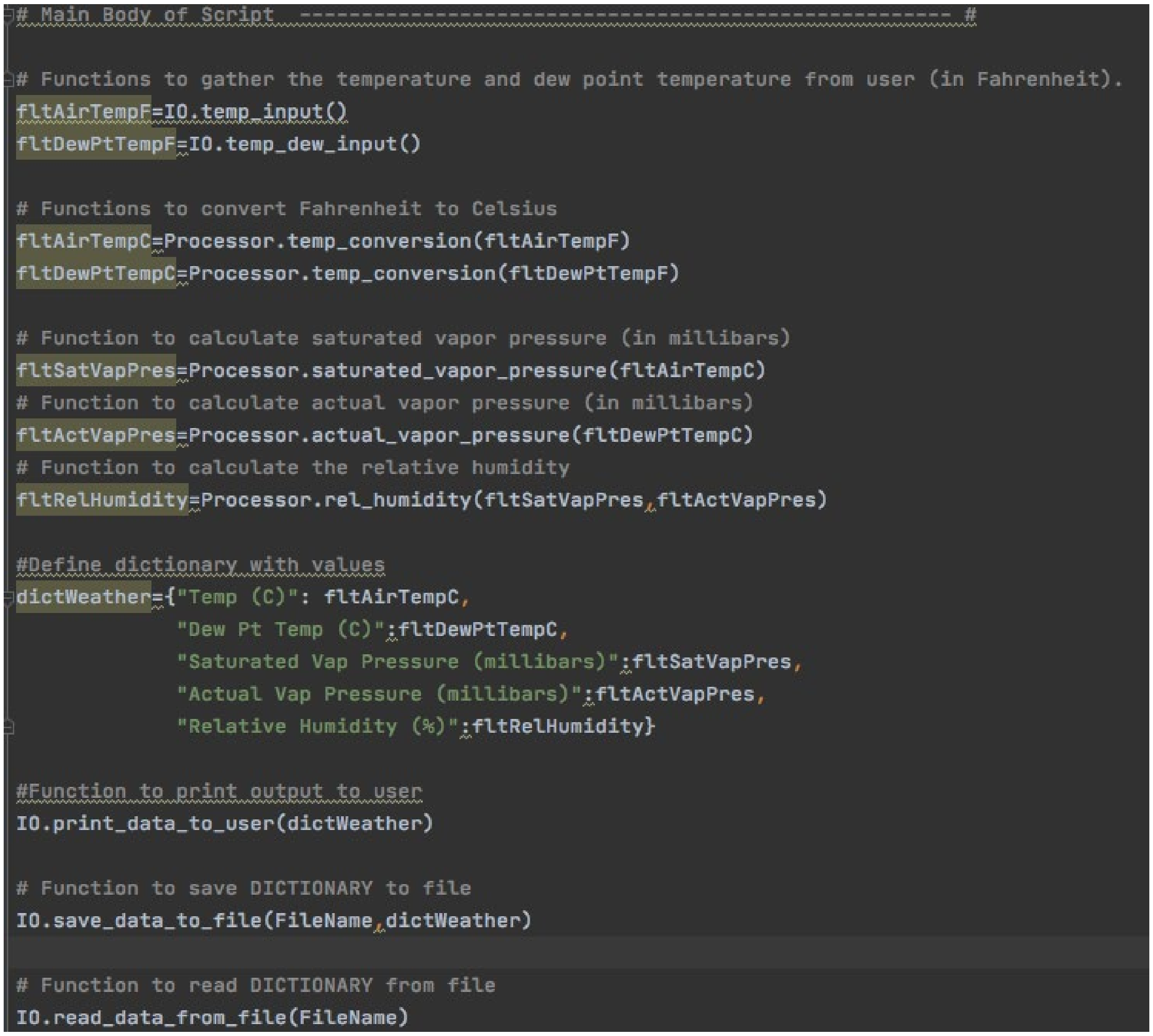
Introduction

In this module, we continue to work with functions and continue to work with reading and writing to files. In this module, however, we learn how to pickle data into a file. This allows us to store more complex data stores, and access them easier. In this assignment, I have created a script that demonstrates pickling and shelving. As well as demonstrates error handling and how to use exception classes in order to error handle. We also learn more about the capabilities of GitHub, and how we can use markdown language to create a webpage.

Pickling

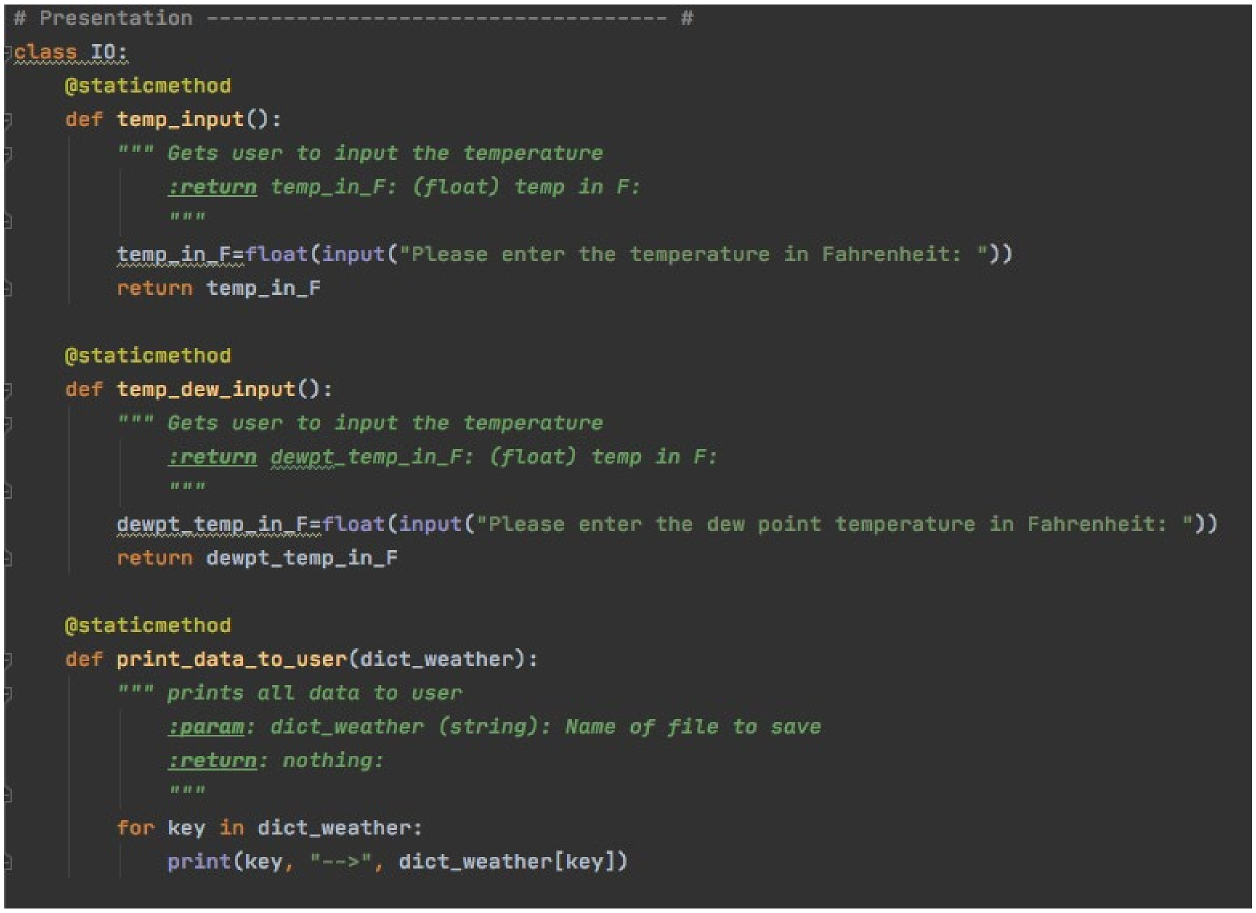
Pickling allows you to save files in binary format instead of plain text. Storing in binary obscures the files content and can reduce file size. In order to use pickling functionality, you must first import the pickle module. The pickle module allows you to pickle and store more complex data in a file. When writing to a file, you simply write the pickled object to the file. Pickled objects *have* to be stored as a binary file. They cannot be stored in a text file.

For this assignment, I have created a script that asks the user to input today's temperature and dew point temperature in Fahrenheit. The script then runs through some functions that convert the temperature to Celsius, then calculates the saturated vapor pressure, actual vapor pressure, and relativity humidity. In **Figure 1**, you will see the main script of the function. **Figure 2** and **3** show the functions that gather input from the user, then process that data.



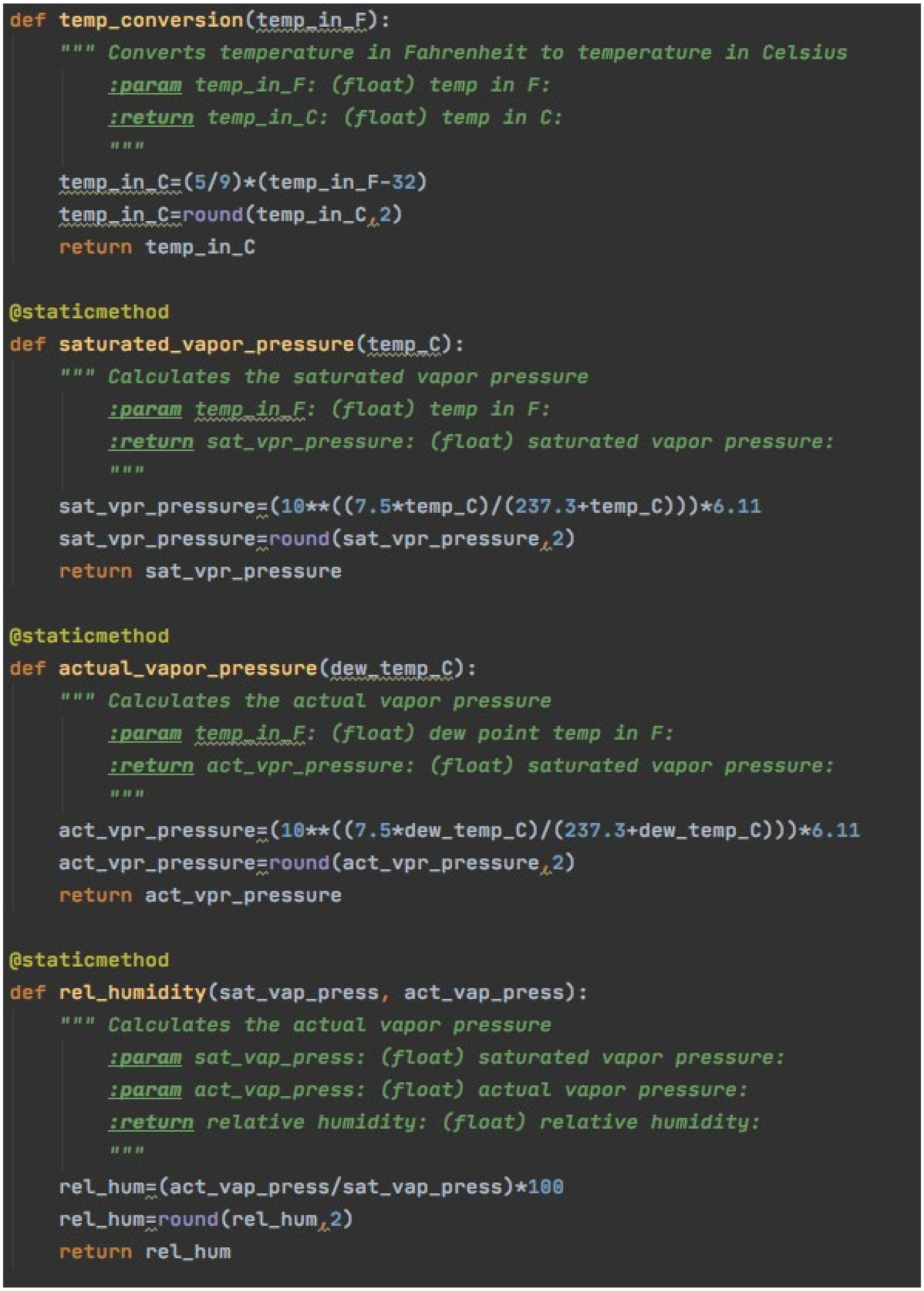
```
# Main Body of Script  --------------------------------------------------------- #

# Functions to gather the temperature and dew point temperature from user (in Fahrenheit).
fltAirTempF=IO.temp_input()
fltDewPtTempF=IO.temp_dew_input()

# Functions to convert Fahrenheit to Celsius
fltAirTempC=Processor.temp_conversion(fltAirTempF)
fltDewPtTempC=Processor.temp_conversion(fltDewPtTempF)

# Function to calculate saturated vapor pressure (in millibars)
fltSatVapPres=Processor.saturated_vapor_pressure(fltAirTempC)
# Function to calculate actual vapor pressure (in millibars)
fltActVapPres=Processor.actual_vapor_pressure(fltDewPtTempC)
# Function to calculate the relative humidity
fltRelHumidity=Processor.rel_humidity(fltSatVapPres,fltActVapPres)

#Define dictionary with values
dictWeather={"Temp (C)": fltAirTempC,
             "Dew Pt Temp (C)":fltDewPtTempC,
             "Saturated Vap Pressure (millibars)":fltSatVapPres,
             "Actual Vap Pressure (millibars)":fltActVapPres,
             "Relative Humidity (%)":fltRelHumidity}

#Function to print output to user
IO.print_data_to_user(dictWeather)

# Function to save DICTIONARY to file
IO.save_data_to_file(FileName,dictWeather)


# Function to read DICTIONARY from file
IO.read_data_from_file(FileName)
```

Figure 1. Main body of script



```python
# Presentation ------------------------------------ #
class IO:
    @staticmethod
    def temp_input():
        """ Gets user to input the temperature
            :return temp_in_F: (float) temp in F:
            """
        temp_in_F=float(input("Please enter the temperature in Fahrenheit: "))
        return temp_in_F

    @staticmethod
    def temp_dew_input():
        """ Gets user to input the temperature
            :return dewpt_temp_in_F: (float) temp in F:
            """
        dewpt_temp_in_F=float(input("Please enter the dew point temperature in Fahrenheit: "))
        return dewpt_temp_in_F

    @staticmethod
    def print_data_to_user(dict_weather):
        """ prints all data to user
            :param: dict_weather (string): Name of file to save
            :return: nothing:
            """
        for key in dict_weather:
            print(key, "-->", dict_weather[key])
```

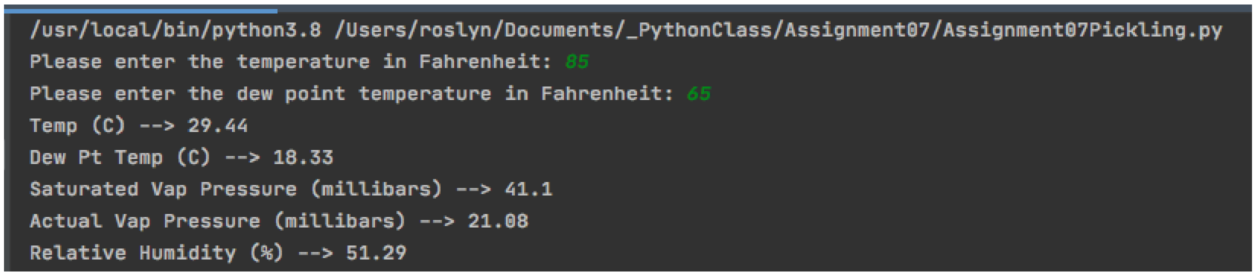
Figure 2. Functions that gather input from users and prints data back out



```python
def temp_conversion(temp_in_F):
    """ Converts temperature in Fahrenheit to temperature in Celsius
        :param temp_in_F: (float) temp in F:
        :return temp_in_C: (float) temp in C:
        """
    temp_in_C=(5/9)*(temp_in_F-32)
    temp_in_C=round(temp_in_C,2)
    return temp_in_C

@staticmethod
def saturated_vapor_pressure(temp_C):
    """ Calculates the saturated vapor pressure
        :param temp_in_F: (float) temp in F:
        :return sat_vpr_pressure: (float) saturated vapor pressure:
        """
    sat_vpr_pressure=(10**((7.5*temp_C)/(237.3+temp_C)))*6.11
    sat_vpr_pressure=round(sat_vpr_pressure,2)
    return sat_vpr_pressure

@staticmethod
def actual_vapor_pressure(dew_temp_C):
    """ Calculates the actual vapor pressure
        :param temp_in_F: (float) dew point temp in F:
        :return act_vpr_pressure: (float) saturated vapor pressure:
        """
    act_vpr_pressure=(10**((7.5*dew_temp_C)/(237.3+dew_temp_C)))*6.11
    act_vpr_pressure=round(act_vpr_pressure,2)
    return act_vpr_pressure

@staticmethod
def rel_humidity(sat_vap_press, act_vap_press):
    """ Calculates the actual vapor pressure
        :param sat_vap_press: (float) saturated vapor pressure:
        :param act_vap_press: (float) actual vapor pressure:
        :return relative humidity: (float) relative humidity:
        """
    rel_hum=(act_vap_press/sat_vap_press)*100
    rel_hum=round(rel_hum,2)
    return rel_hum
```

Figure 3. Functions that calculate temperature in Celsius, the saturated vapor pressure, the actual vapor pressure, and the relativity humidity

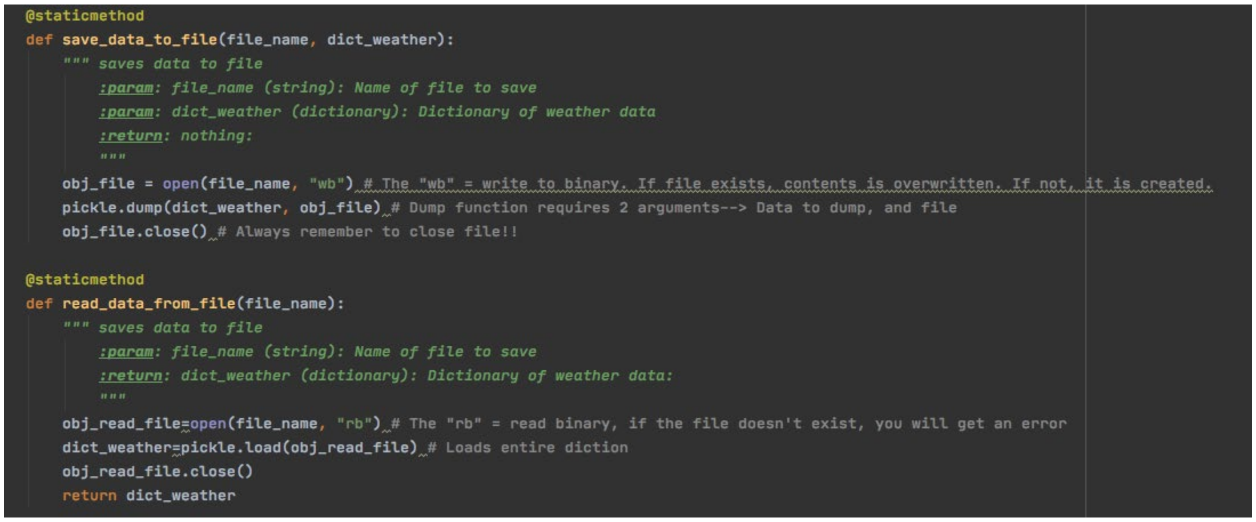
In **Figure 4**, below, you can see the script running in PyCharm.



```
/usr/local/bin/python3.8 /Users/roslyn/Documents/_PythonClass/Assignment07/Assignment07Pickling.py
Please enter the temperature in Fahrenheit: 85
Please enter the dew point temperature in Fahrenheit: 65
Temp (C) --> 29.44
Dew Pt Temp (C) --> 18.33
Saturated Vap Pressure (millibars) --> 41.1
Actual Vap Pressure (millibars) --> 21.08
Relative Humidity (%) --> 51.29
```

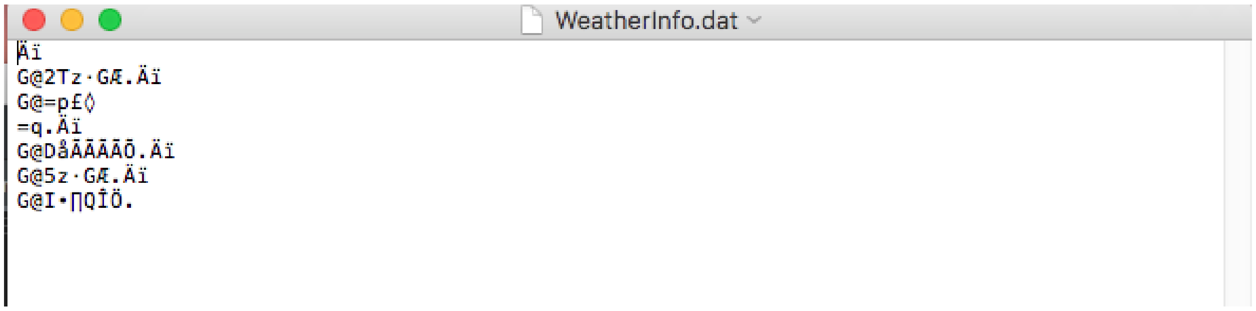
Figure 4. Running script in PyCharm

In the first run of this script, I will pickle the dictionary of weather data, *dictWeather* into the file. I will then load the dictionary object back from the file. You can see the functions that are executing in order to dump and load a dictionary into the file in **Figure 5**. In **Figure 6**, you can see that the data is dumped into a file.



```
@staticmethod
def save_data_to_file(file_name, dict_weather):
    """ saves data to file
        :param: file_name (string): Name of file to save
        :param: dict_weather (dictionary): Dictionary of weather data
        :return: nothing:
        """
    obj_file = open(file_name, "wb") # The "wb" = write to binary. If file exists, contents is overwritten. If not, it is created.
    pickle.dump(dict_weather, obj_file) # Dump function requires 2 arguments--> Data to dump, and file
    obj_file.close() # Always remember to close file!!

@staticmethod
def read_data_from_file(file_name):
    """ saves data to file
        :param: file_name (string): Name of file to save
        :return: dict_weather (dictionary): Dictionary of weather data:
        """
    obj_read_file=open(file_name, "rb") # The "rb" = read binary, if the file doesn't exist, you will get an error
    dict_weather=pickle.load(obj_read_file) # Loads entire diction
    obj_read_file.close()
    return dict_weather
```

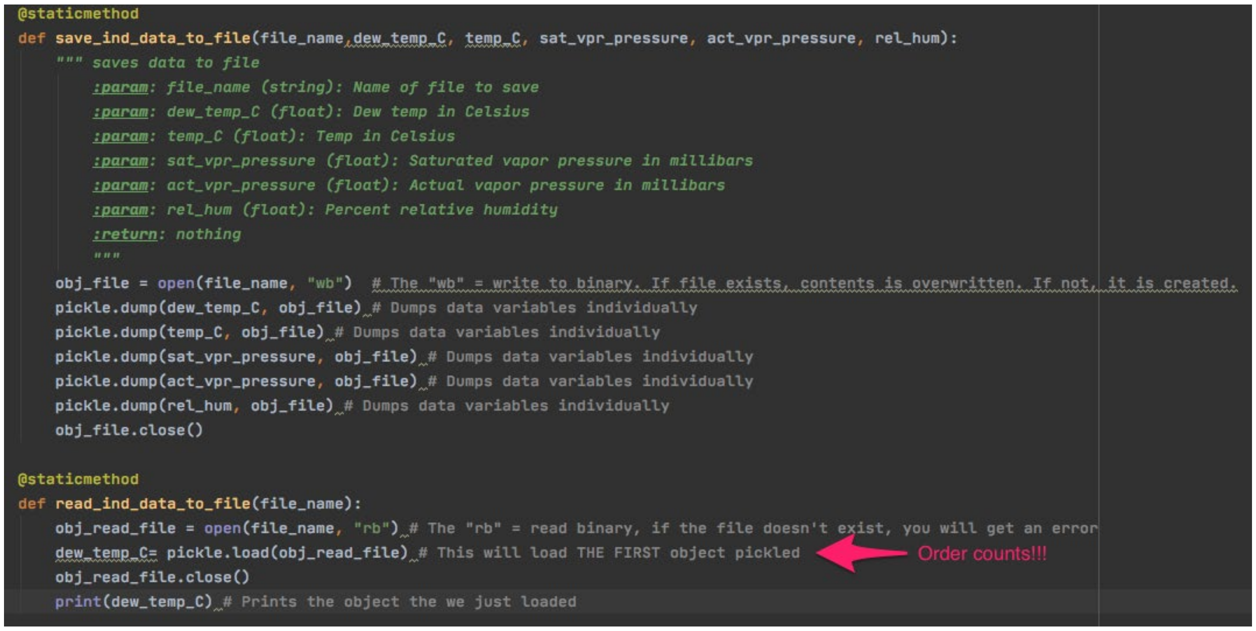
Figure 5. Pickling a dictionary



```
Äï
G@2Tz·GÆ.Äï
G@=p£◊
=q.Äï
G@DåĀĀĀĀŌ.Äï
G@5z·GÆ.Äï
G@I•∏QÎÖ.
```

Figure 6. Data pickled into file

The dump() function requires two arguments, the data to pickle, and the file. Note, when pickling, you load the entire object from the file. You can pickle a variety of objects, including lists, strings, numbers, touples. Since I am loading the data into the file in a dictionary, it is hard to access certain components of the dictionary from the file. Instead I have to load the object as a whole back into my script. Why does this matter? One of the main reasons pickling can be useful is being able to work with huge amounts of data. If I had a huge dictionary, then I would have to load the entire dictionary back into my memory, instead of being able to access key parts. In the script below, you can see examples of me dumping the weather data into a file as individual variables, instead of as a dictionary. In this way, I can also load the data piece by piece from the file. See functions in **Figure 7**.

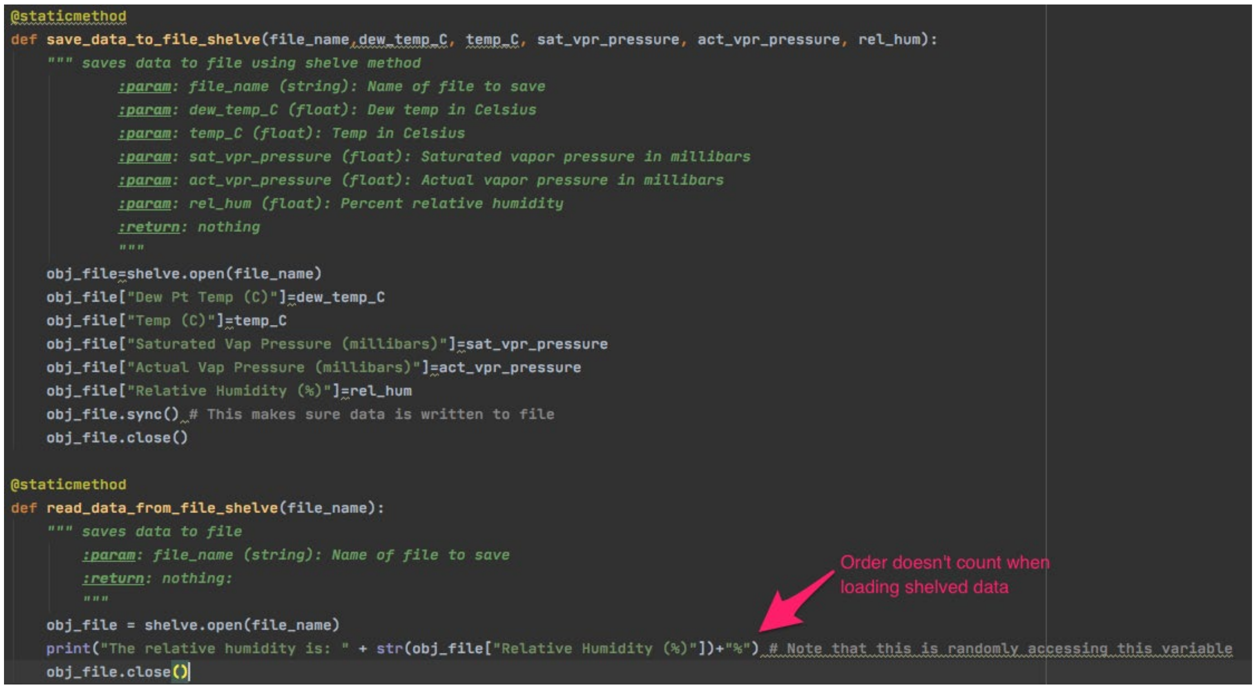


```
@staticmethod
def save_ind_data_to_file(file_name,dew_temp_C, temp_C, sat_vpr_pressure, act_vpr_pressure, rel_hum):
    """ saves data to file
        :param: file_name (string): Name of file to save
        :param: dew_temp_C (float): Dew temp in Celsius
        :param: temp_C (float): Temp in Celsius
        :param: sat_vpr_pressure (float): Saturated vapor pressure in millibars
        :param: act_vpr_pressure (float): Actual vapor pressure in millibars
        :param: rel_hum (float): Percent relative humidity
        :return: nothing
        """
    obj_file = open(file_name, "wb")  # The "wb" = write to binary. If file exists, contents is overwritten. If not, it is created.
    pickle.dump(dew_temp_C, obj_file) # Dumps data variables individually
    pickle.dump(temp_C, obj_file) # Dumps data variables individually
    pickle.dump(sat_vpr_pressure, obj_file) # Dumps data variables individually
    pickle.dump(act_vpr_pressure, obj_file) # Dumps data variables individually
    pickle.dump(rel_hum, obj_file) # Dumps data variables individually
    obj_file.close()

@staticmethod
def read_ind_data_to_file(file_name):
    obj_read_file = open(file_name, "rb") # The "rb" = read binary, if the file doesn't exist, you will get an error
    dew_temp_C= pickle.load(obj_read_file) # This will load THE FIRST object pickled
    obj_read_file.close()
    print(dew_temp_C) # Prints the object the we just loaded
```

Figure 7. Pickling data variable by variable

It is very important to note that when pickling multiple objects, the order counts! That is, the first object that you pickle, will be the first object that is loaded back up.

Taking pickling one step further, we can add the shelf module. The shelve module allows you to store and randomly access pickled objects in a file. Using the shelve module, it acts like a dictionary that lets you access its components. In the example below, I show you the same program but instead of storing the weather data into a dictionary and pickling the dictionary, I shelve them as separate variables. In **Figure 8**, you can see that in the function where I am shelving all the variables, I am shelving them similar to how I created my original dictionary, *dictWeather*. In the function in which I am loading data from the file, you can see that I am randomly accessing just the relative humidity.



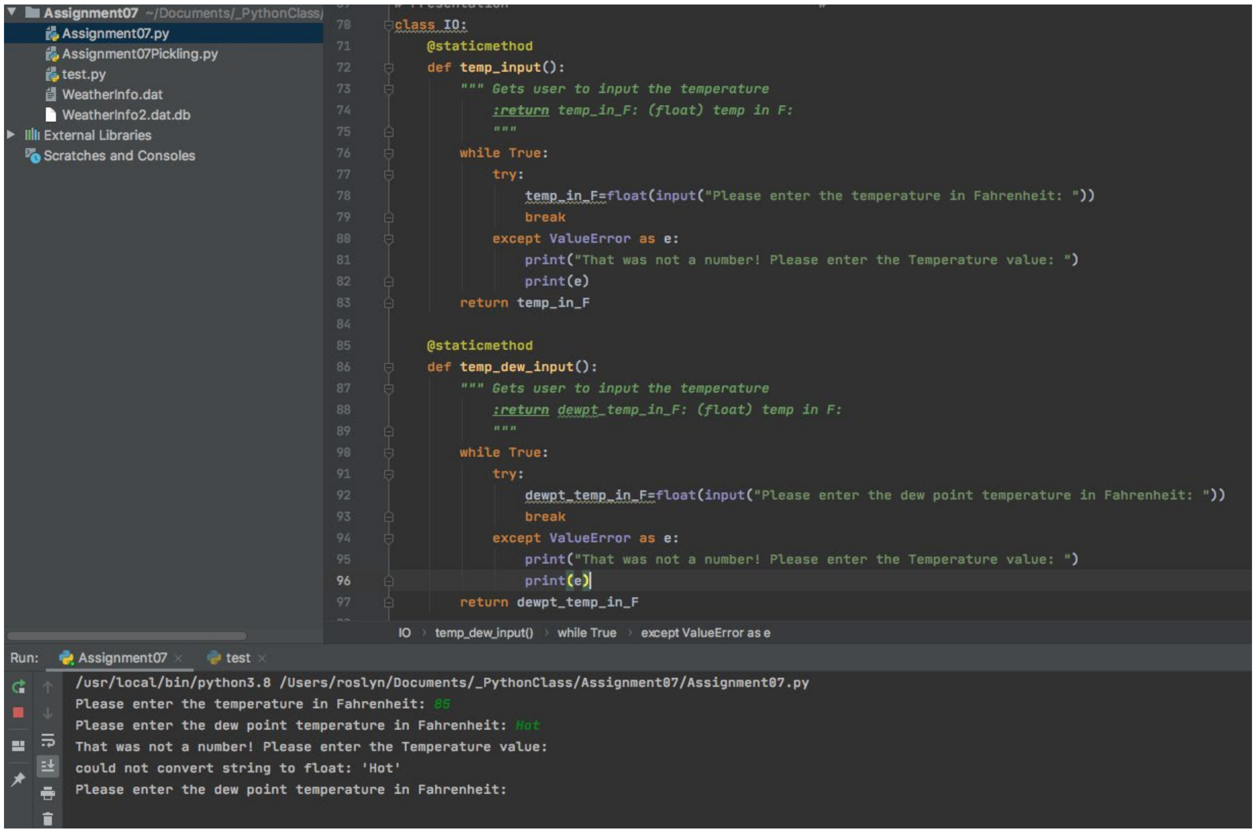
```
@staticmethod
def save_data_to_file_shelve(file_name,dew_temp_C, temp_C, sat_vpr_pressure, act_vpr_pressure, rel_hum):
    """ saves data to file using shelve method
            :param: file_name (string): Name of file to save
            :param: dew_temp_C (float): Dew temp in Celsius
            :param: temp_C (float): Temp in Celsius
            :param: sat_vpr_pressure (float): Saturated vapor pressure in millibars
            :param: act_vpr_pressure (float): Actual vapor pressure in millibars
            :param: rel_hum (float): Percent relative humidity
            :return: nothing
            """
    obj_file=shelve.open(file_name)
    obj_file["Dew Pt Temp (C)"]=dew_temp_C
    obj_file["Temp (C)"]=temp_C
    obj_file["Saturated Vap Pressure (millibars)"]=sat_vpr_pressure
    obj_file["Actual Vap Pressure (millibars)"]=act_vpr_pressure
    obj_file["Relative Humidity (%)"]=rel_hum
    obj_file.sync() # This makes sure data is written to file
    obj_file.close()

@staticmethod
def read_data_from_file_shelve(file_name):
    """ saves data to file
        :param: file_name (string): Name of file to save
        :return: nothing:
        """
    obj_file = shelve.open(file_name)
    print("The relative humidity is: " + str(obj_file["Relative Humidity (%)"])+"%") # Note that this is randomly accessing this variable
    obj_file.close()
```

Figure 8. Example of shelving data

I used this stack overflow page (https://stackoverflow.com/questions/4103430/what-is-the-difference-between-pickle-and-shelve) to better understand the difference between pickling and shelving. I think this website gave good examples of how the modules are used differently. I tried to show this difference in my assignment 7 example.

Error Handling

Up until this point, if there was an error in our program, the script would stop running and provide an error message. In python, you have capability to handle these errors in a more graceful way. What I mean by that is providing an error message that is easier to understand for the user. In **Figure 9**, you can see that I have added error handling to the temperature input functions. I have this set up such that it will keep prompting user to input a temperature if they try and enter a string instead of a number.

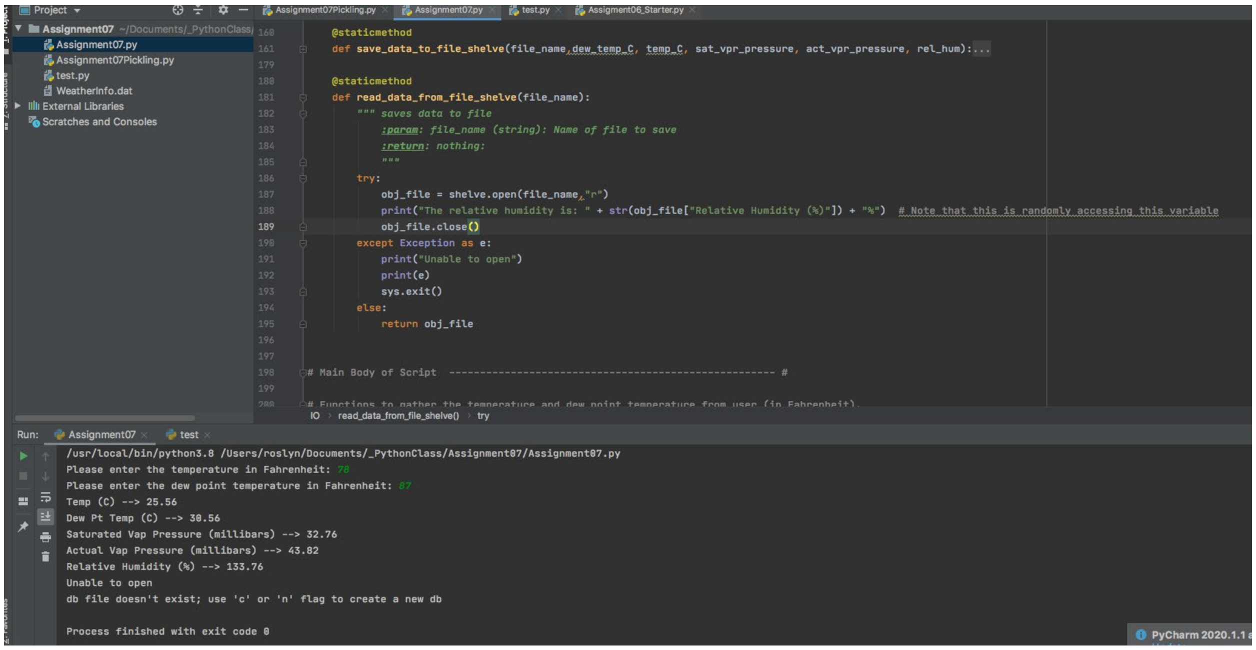

Assignment07 ~/Documents/_PythonClass/
Assignment07.py
Assignment07Pickling.py
test.py
WeatherInfo.dat
WeatherInfo2.dat.db
External Libraries
Scratches and Consoles
IO › temp_dew_input() › while True › except ValueError as e
Run: Assignment07 × test ×

```python
class IO:
    @staticmethod
    def temp_input():
        """ Gets user to input the temperature
            :return temp_in_F: (float) temp in F:
            """
        while True:
            try:
                temp_in_F=float(input("Please enter the temperature in Fahrenheit: "))
                break
            except ValueError as e:
                print("That was not a number! Please enter the Temperature value: ")
                print(e)
        return temp_in_F

    @staticmethod
    def temp_dew_input():
        """ Gets user to input the temperature
            :return dewpt_temp_in_F: (float) temp in F:
            """
        while True:
            try:
                dewpt_temp_in_F=float(input("Please enter the dew point temperature in Fahrenheit: "))
                break
            except ValueError as e:
                print("That was not a number! Please enter the Temperature value: ")
                print(e)
        return dewpt_temp_in_F
```

```
/usr/local/bin/python3.8 /Users/roslyn/Documents/_PythonClass/Assignment07/Assignment07.py
Please enter the temperature in Fahrenheit: 85
Please enter the dew point temperature in Fahrenheit: Hot
That was not a number! Please enter the Temperature value:
could not convert string to float: 'Hot'
Please enter the dew point temperature in Fahrenheit:
```

Figure 9. Error Handling

Note that in the example above, I printed the value “e”. Whenever an exception occurs, it is associated with a value (argument). You can capture this argument and print it to the user if you by using the word “as” after the exception. In **Figure 10**, you can see I used the general exception to catch the error. You can create multiple except statements to try and catch different types of errors.



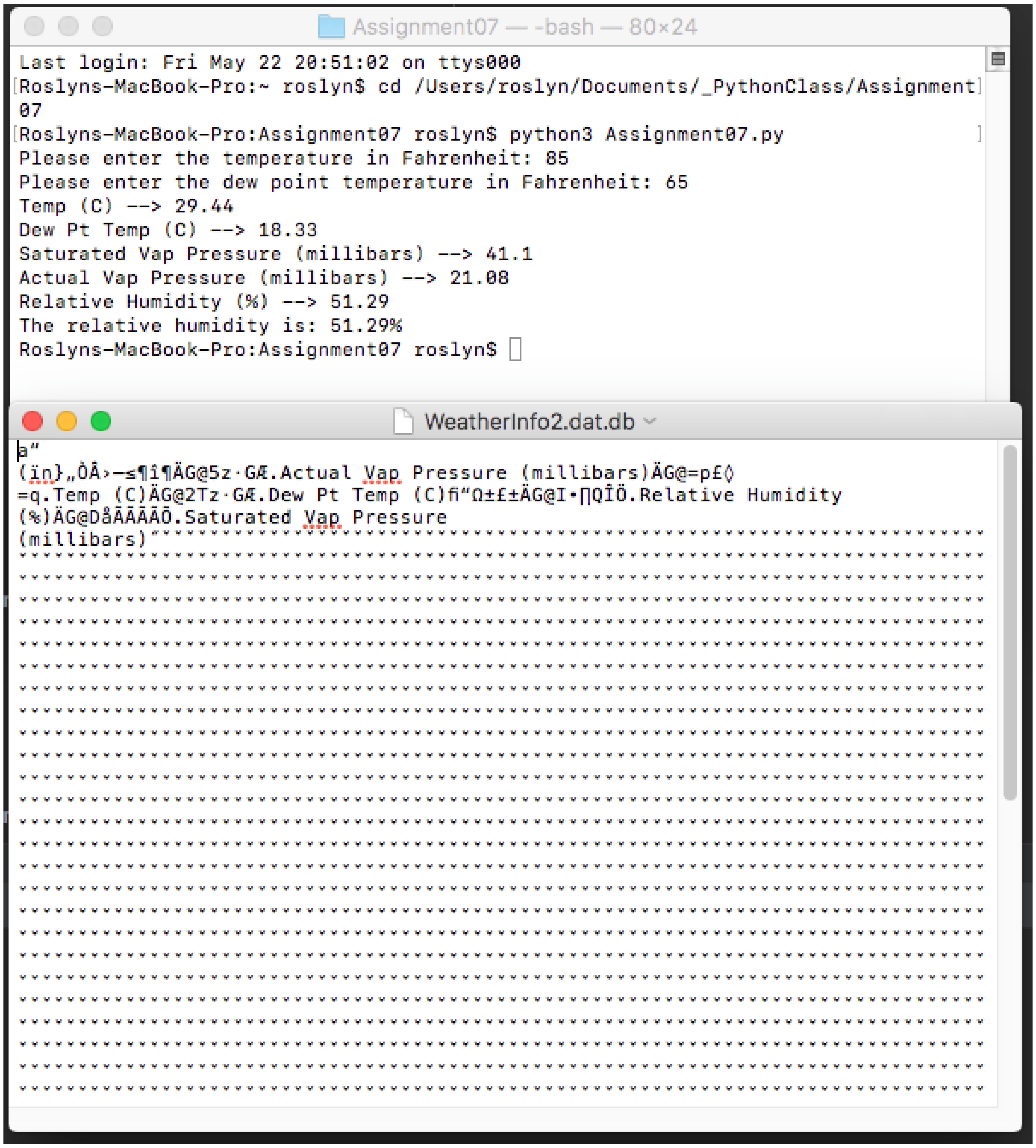
```python
    @staticmethod
    def save_data_to_file_shelve(file_name,dew_temp_C, temp_C, sat_vpr_pressure, act_vpr_pressure, rel_hum):...

    @staticmethod
    def read_data_from_file_shelve(file_name):
        """ saves data to file
            :param: file_name (string): Name of file to save
            :return: nothing:
            """
        try:
            obj_file = shelve.open(file_name,"r")
            print("The relative humidity is: " + str(obj_file["Relative Humidity (%)"]) + "%")  # Note that this is randomly accessing this variable
            obj_file.close()
        except Exception as e:
            print("Unable to open")
            print(e)
            sys.exit()
        else:
            return obj_file


# Main Body of Script  ---------------------------------------------------- #

# Functions to gather the temperature and dew point temperature from user (in Fahrenheit).
```

```
/usr/local/bin/python3.8 /Users/roslyn/Documents/_PythonClass/Assignment07/Assignment07.py
Please enter the temperature in Fahrenheit: 78
Please enter the dew point temperature in Fahrenheit: 87
Temp (C) --> 25.56
Dew Pt Temp (C) --> 30.56
Saturated Vap Pressure (millibars) --> 32.76
Actual Vap Pressure (millibars) --> 43.82
Relative Humidity (%) --> 133.76
Unable to open
db file doesn't exist; use 'c' or 'n' flag to create a new db

Process finished with exit code 0
```

Figure 10. General exception example

I used the following website (https://www.tutorialspoint.com/python/python_exceptions.htm) as a guide in error handling. What I liked about this website was that it had a list of common error types to easily reference. It also provided many examples of error handling.

In **Figure 11**, below, you can see the script running from the terminal and the binary file that was created.



```
Last login: Fri May 22 20:51:02 on ttys000
Roslyns-MacBook-Pro:~ roslyn$ cd /Users/roslyn/Documents/_PythonClass/Assignment
07
Roslyns-MacBook-Pro:Assignment07 roslyn$ python3 Assignment07.py
Please enter the temperature in Fahrenheit: 85
Please enter the dew point temperature in Fahrenheit: 65
Temp (C) --> 29.44
Dew Pt Temp (C) --> 18.33
Saturated Vap Pressure (millibars) --> 41.1
Actual Vap Pressure (millibars) --> 21.08
Relative Humidity (%) --> 51.29
The relative humidity is: 51.29%
Roslyns-MacBook-Pro:Assignment07 roslyn$
```

WeatherInfo2.dat.db

```
a“
(in}„ÒÂ›—≤¶î¶ÄG@5z·GÆ.Actual Vap Pressure (millibars)ÄG@=p£◊
=q.Temp (C)ÄG@2Tz·GÆ.Dew Pt Temp (C)ﬁ“Ω±£±ÄG@I•∏QÎÖ.Relative Humidity
(%)ÄG@DåÃÃÃÃÕ.Saturated Vap Pressure
(millibars)ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
ˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇˇ
```

Figure 9. Script running from terminal, and data that is saved to file.

Summary

In summary, there are many benefits to pickling and shelving data. When pickling, you can serialize objects in a single byte stream. The order in which you dump and load objects matters. Shelving builds onto this in that a serialized dictionary of pickled objects is created. Shelving allows you to associate each object with a key therefore allowing you to randomly access your data. We also added error handling to our scripts for the first time. This allows us to catch errors and provide a more helpful message about the error, rather than the canned python error response.