Name: Aimee Richardson
Date: 02/28/2023
Course: IT FDN 110 - Foundation of Programming: Python
Assignment: Assignment 7
GitHub Repository: https://github.com/amrich1111/IntroToProg-Python-Mod07
GitHub Site: https://amrich1111.github.io/IntroToProg-Python-Mod07/

Assignment 07 - Pickling & Error Handling in Python

Introduction

In the following documentation we explore the pickling and exception handling features in Python. When handling binary files, the use of both pickling and exception handling together is important to ensure your program runs as intended. Pickling in Python allows programs to serialize and store complex objects to a file. In the following sections, we will walk through pickling and exception handling in Python while building a Mood Tracker. This program will allow users to input data to add new entries, access existing entries, and view all entries from a binary file.

Binary & Text Files in Python

Python can handle loads of different variables and objects and save them to files. So far, we have become more familiar with text (.txt) files and saving user input to them. A limitation of text files is their inability to store complex objects such as entire lists or dictionaries. In order to save lists or dictionaries to text files, you must iterate over them, and save individual parts of the larger object to the file.

An alternative to using text files, is to store these complex objects in binary files. In order to effectively store data to binary files, an additional Python module needs to be utilized which will be explored further in the following section.

Pickling in Python

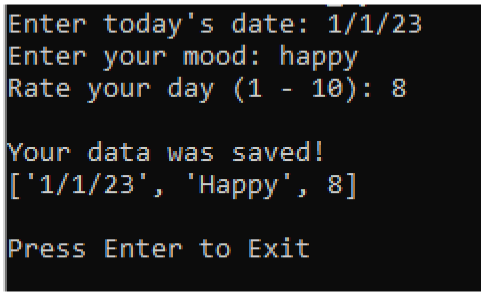
The Pickle module in Python allows programmers to access multiple methods which facilitate the task of storing data in binary files. Pickle is a unique package that can be imported into any Python program when developers need to store complex objects to files. The following sections will go a bit more in depth regarding the pickle.dump() and pickle.load() methods and how those differ from how we might work with storing data in text files.

Dumping with Pickle (Writing Data to a Binary File)

Listing 1

```
import pickle

# Define the data
pickle_file = "moodTracker.abt"
pickled_list = None

# Get user input
```

```
user_date = str(input("Enter today's date: ")).strip()
user_mood = str(input("Enter your mood: ")).strip()
user_rate = int(input("Rate your day (1 - 10): "))

# Process the data
pickled_list = [user_date, user_mood, user_rate] # Add user inputted data to
list object

# Save data to file
binary_file = open(pickle_file, "wb") # Open/create new binary file to write
new data to
pickle.dump(pickled_list, binary_file) # "Dump" data to binary
binary_file.close() # Close file

# Presentation
print(f"Your data was saved!\n{pickled_list}\n")
input("Press Enter to Exit")
```



```
Enter today's date: 1/1/23
Enter your mood: happy
Rate your day (1 - 10): 8

Your data was saved!
['1/1/23', 'Happy', 8]

Press Enter to Exit
```

Figure 7.1 - Output from Listing 1

In Listing 1, there is an example of the dump method being used to save user input to a binary file. First, we need to import the module so we can access the pickling functions. Next, a binary file must be defined to write our objects to, in this case our binary file uses the extension '.abt' versus '.txt' which is used for text files. The program then receives user input, and stores it as a list to a variable. Then, we get to pickling.

First, a file is opened, using the .open() method. A difference to note here is the mode used to tell the program how to interact with the file. Since we are working with binary files, the modes will be different compared to saving data to text files. When handling text files, we used modes such as 'w', 'r', and 'a' – when handling binary files we use 'wb', 'rb', and 'ab'. In this case, since we are writing new data to our binary file, we use the 'wb' as our mode. If the file already exists, its contents will be overwritten, if it doesn't exist, the file will be created.

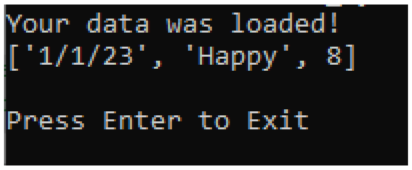
Next, the list object that stores the user's data is 'dumped' to the binary file using the .dump() method. This method takes the object to save and the file name as arguments. Once completed, the data has successfully been serialized in binary code and saved to the file (Figure 7.1).

Loading with Pickle (Reading Data from a Binary File)

Listing 2

```
import pickle
```

```
# Define the data
pickle_file = "moodTracker.abt"

# Load data from file
binary_file = open(pickle_file, "rb") # Open existing binary file to read data
from
unpickled_list = pickle.load(binary_file) # "Load" data from binary file
binary_file.close() # Close file

# Presentation
print(f"Your data was loaded!\n{unpickled_list}\n")
input("Press Enter to Exit")
```



```
Your data was loaded!
['1/1/23', 'Happy', 8]

Press Enter to Exit
```

Figure 7.2 - Output from Listing 2

Data can also be unpickled from binary files using the ‘rb’ mode and pickle.load() method. Unpickling a binary object, reads the binary code and translates it back into its original object form in Python. In Listing 2, we see a similar setup as Listing 1, however we now have changed the mode in the .open() method to ‘rb’ – telling Python to read data from the binary file. In order to translate the binary back into our original object, we use the .load() method which takes the file name as an argument. Once Python has loaded and stored the binary data back into a variable, we are able to access its original form (Figure 7.2).

An important note: If multiple objects are expected to be outputted from a binary file, the .load() method must be called multiple times. We will get into this further in the following section.

Using ‘with’ when Opening Files

Listing 3

```
import pickle

# Define the data
pickle_file = "moodTracker.abt"
pickled_list = None

# Get user input
user_date = str(input("Enter today's date: ")).strip()
user_mood = str(input("Enter your mood: ")).strip().capitalize()
user_rate = int(input("Rate your day (1 - 10): "))

# Process the data
pickled_list = [user_date, user_mood, user_rate] # Add user inputted data to
list object

# Save data to file
with open(pickle_file, "wb") as binary_file: # Open/create new binary file to
write new data to
  pickle.dump(pickled_list, binary_file) # "Dump" data to binary file
```

```
# Presentation
print(f"\nYour data was saved!\n{pickled_list}\n")
input("Press Enter to Exit")
```

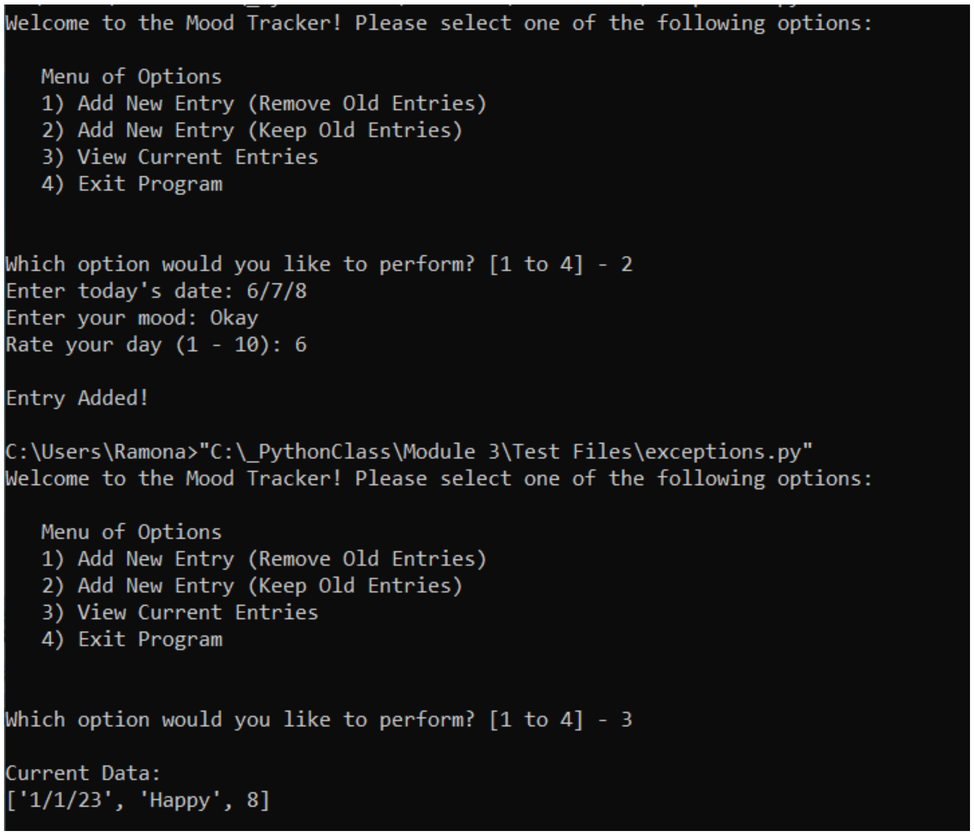
When opening and closing files in Python, the 'with' statement can be utilized to clean up your code a bit and make it more readable. Listing 3 executes the same function as Listing 1, however the 'with' statement is used alongside the .open() method. The 'with' statement will close the file for you since it is managing the file stream (Manthanchauhan, 2022) . Using 'with' can help manage exceptions that may arise when handling files, in this scenario either 'with' or .close() can be used effectively to achieve the same output as seen in Listing 1 and Listing 3.

Appending with Pickle

Listing 4

```
import pickle

# Define data
pickle_file = "moodTracker.abt"
pickled_list = None
unpickled_list = None

# Present menu to user
print("Welcome to the Mood Tracker! Please select one of the following
options:")
print('''
  Menu of Options
  1) Add New Entry (Remove Old Entries)
  2) Add New Entry (Keep Old Entries)
  3) View Current Entries
  4) Exit Program
  ''')
print()  # Add an extra line for looks

# Get user input
user_choice = str(input("Which option would you like to perform? [1 to 4] -
")).strip()

# Write new data to binary file
if user_choice == "1":
  user_date = str(input("Enter today's date: ")).strip()
  user_mood=str(input("Enter your mood: ")).strip()
  user_rate=int(input("Rate your day (1 - 10): "))
  pickled_list = [user_date, user_mood, user_rate]  # Add user inputted data to
list object

  with open(pickle_file, "wb") as binary_file:  # Open/create new binary file
to write new data to
      pickle.dump(pickled_list, binary_file)  # "Dump" data to binary file

  print("\nEntry Added!")

# Add new data to existing file
```

```
elif user_choice == "2":
  user_date = str(input("Enter today's date: ")).strip()
  user_mood = str(input("Enter your mood: ")).strip()
  user_rate = int(input("Rate your day (1 - 10): "))
  pickled_list = [user_date, user_mood, user_rate]  # Add user inputted data to
list object

  with open(pickle_file, "ab") as binary_file: # Open/create a new or existing
binary file to append new data to
      pickle.dump(pickled_list, binary_file) # "Dump" data to binary file

  print("\nEntry Added!")

# Read data from binary file
elif user_choice == '3':
  with open(pickle_file, "rb") as binary_file:  # Open existing binary file to
read data from
     unpickled_list = pickle.load(binary_file)  # "Load" data from binary
  print("\nCurrent Data:")
  print(unpickled_list)

# Exit program
elif user_choice == '4':
  input("\nPress Enter to Exit")
```



```
Welcome to the Mood Tracker! Please select one of the following options:

   Menu of Options
   1) Add New Entry (Remove Old Entries)
   2) Add New Entry (Keep Old Entries)
   3) View Current Entries
   4) Exit Program


Which option would you like to perform? [1 to 4] - 2
Enter today's date: 6/7/8
Enter your mood: Okay
Rate your day (1 - 10): 6

Entry Added!

C:\Users\Ramona>"C:\_PythonClass\Module 3\Test Files\exceptions.py"
Welcome to the Mood Tracker! Please select one of the following options:

   Menu of Options
   1) Add New Entry (Remove Old Entries)
   2) Add New Entry (Keep Old Entries)
   3) View Current Entries
   4) Exit Program


Which option would you like to perform? [1 to 4] - 3

Current Data:
['1/1/23', 'Happy', 8]
```

Figure 7.3 - Output from Listing 4

In Listing 4, we see a demonstration of how the append mode can be used when handling binary files. Write and append execute very similar tasks – both utilize the .dump() method to add new data to a binary file. The one key difference when you decide to write versus append is the existing data in the file. Defining 'ab' as the mode when opening a file tells the program you want to preserve all the file's existing data, but add to it. The 'wb' mode will overwrite all existing data in the file. Similar to write, the append mode will create a new file if one does not already exist.

In Figure 7.3, the user appends a new entry to their tracker. One thing to note – when the user tries to view all their entries, we don't see the appended one appear. This is due to how the .load() method is being called in our current program. In the following sections, we look further into handling this issue.

Handling Exception Errors

When a program cannot run as expected, Python will return an error and will not continue running. There are multiple different cases where errors can occur, some are due to syntax, data types, indexes, files, amongst others.

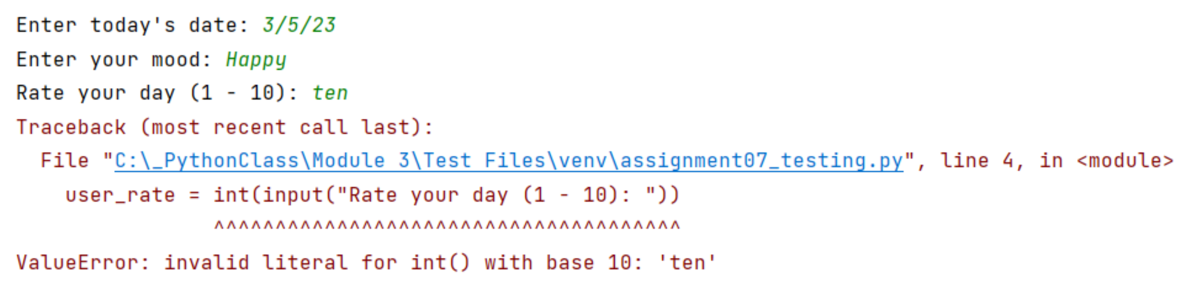
Errors can be detrimental to a program because once they occur, the program will not finish executing or result in an unexpected output. Some errors are inevitable when working with files and user input. In Python, to avoid this, there are strategies to handle errors to ensure the program runs smoothly for the user and as intended by the developer.

Try Except Blocks

Utilizing try and except in programs is a great way to handle errors in Python to avoid any problems that may occur.

Listing 5

```
# Get user input
user_date = str(input("Enter today's date: ")).strip()
user_mood = str(input("Enter your mood: ")).strip()
user_rate = int(input("Rate your day (1 - 10): "))

# Presentation
print(f"\nDate: {user_date}\nMood: {user_mood}\nDaily Rating: {user_rate}\n")
input("Press Enter to Exit")
```



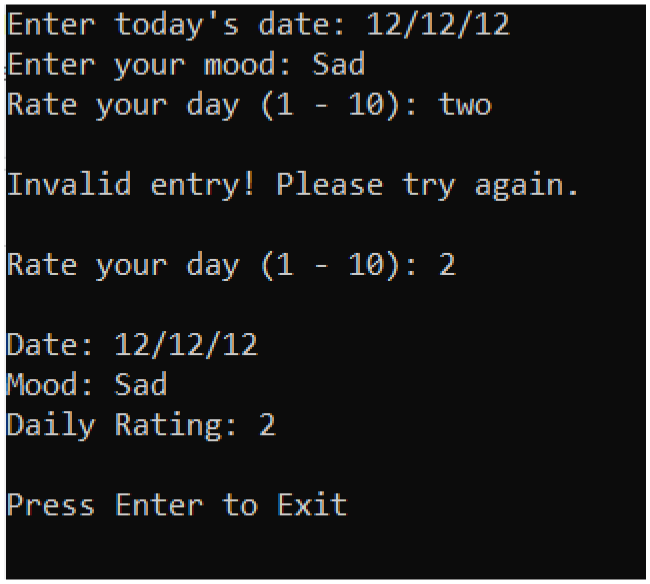
```
Enter today's date: 3/5/23
Enter your mood: Happy
Rate your day (1 - 10): ten
Traceback (most recent call last):
  File "C:\_PythonClass\Module 3\Test Files\venv\assignment07_testing.py", line 4, in <module>
    user_rate = int(input("Rate your day (1 - 10): "))
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'ten'
```

Figure 7.4 - Error outputted from Listing 5

Listing 5 demonstrates a potential error that our Mood Tracker may face when handling user input. When working with user input, errors could arise if the inputted data is not the intended type. For example, in Figure 7.4, we see an error has occurred because the user has inputted text where the program expected an integer.

Listing 6

```
# Get user input
user_date = str(input("Enter today's date: ")).strip()
user_mood = str(input("Enter your mood: ")).strip()

while True:
  try:
     user_rate=int(input("Rate your day (1 - 10): "))
     break
  except:
     print("\nInvalid entry! Please try again.\n")

# Presentation
print(f"\nDate: {user_date}\nMood: {user_mood}\nDaily Rating: {user_rate}\n")
input("Press Enter to Exit")
```

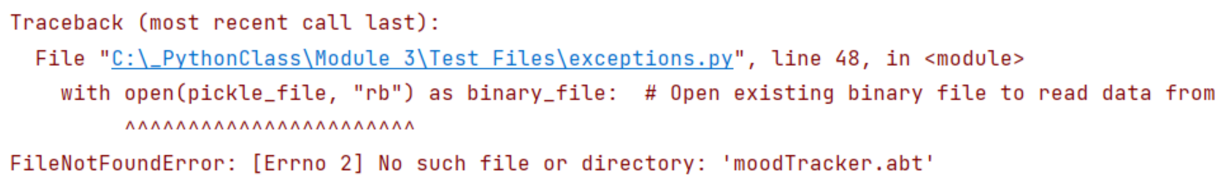


```
Enter today's date: 12/12/12
Enter your mood: Sad
Rate your day (1 - 10): two

Invalid entry! Please try again.

Rate your day (1 - 10): 2

Date: 12/12/12
Mood: Sad
Daily Rating: 2

Press Enter to Exit
```

Figure 7.5 - Output from Listing 6

To account for this potential input and avoid a failed program – we can use a try-except block. In Listing 6, a try-except block was implemented to catch this potential error. First, the program will try to get the user's input. For any reason, if the input causes the program to throw an error, it will move onto the except code, where developers can tell the user that something has gone wrong. As seen in Figure 7.5, the user, again, has inputted text where the program expected an integer. Instead of raising an error, the program moves onto the code in the except statement which tells the user that something was wrong with their input and they need to try again.

Handling Exceptions by Error Type



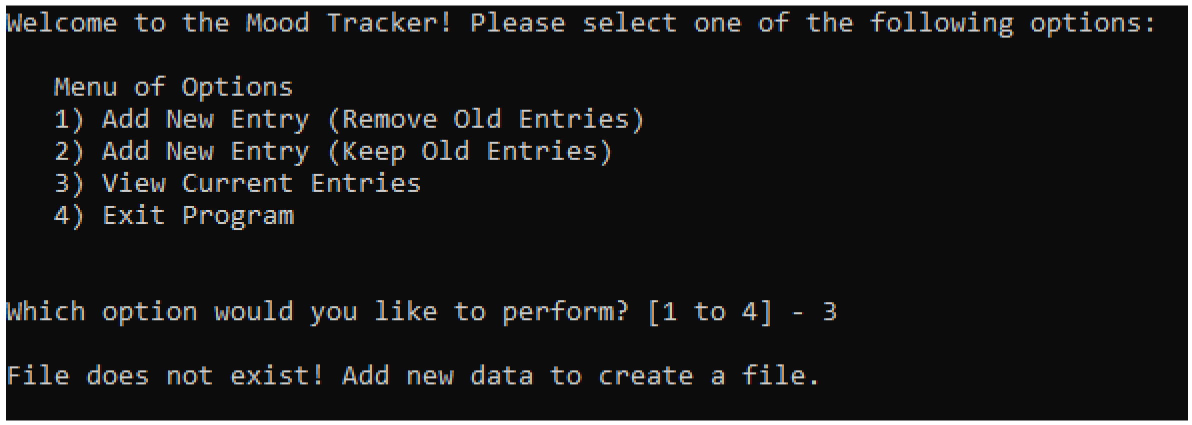
```
Traceback (most recent call last):
  File "C:\_PythonClass\Module 3\Test Files\exceptions.py", line 48, in <module>
    with open(pickle_file, "rb") as binary_file:  # Open existing binary file to read data from
         ^^^^^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'moodTracker.abt'
```

Figure 7.6 - Error outputted from Listing 4

Errors are also common when handling files. One of the main exceptions to consider is when a program attempts to read from a file, but the file does not already exist. Unlike the write or append modes, the read mode will not create a new file if one doesn't already exist. As seen in Figure 7.6, if a user attempts to read data from a file before it is created, the program will result in a 'FileNotFoundError'. We could implement a simple try-except block to account for this as we did above, however this is not as useful to the user. When handling errors, Python allows you to define exceptions for different error types, which can help users determine what has gone wrong.

Listing 7

```
# Read data from binary file
elif user_choice == '3':
  try:
     with open(pickle_file, "rb") as binary_file:  # Open existing binary file 
to read data from
        unpickled_list = pickle.load(binary_file)  # "Load" data from binary
     print("\nCurrent Data:")
     print(unpickled_list)
  except FileNotFoundError:
     print("\nFile does not exist! Add new data to create a file.\n")
  except Exception as e:
     print(f"A {e.__class__.__name__} error has occurred!")
```



```
Welcome to the Mood Tracker! Please select one of the following options:

   Menu of Options
   1) Add New Entry (Remove Old Entries)
   2) Add New Entry (Keep Old Entries)
   3) View Current Entries
   4) Exit Program


Which option would you like to perform? [1 to 4] - 3

File does not exist! Add new data to create a file.
```

Figure 7.7 - Output from Listing 7

In Listing 7, the read portion of the Listing 4 code has been updated with a try-except block. In the except statement, the 'FileNotFoundError' is specified which then tells the program to do something specific when this error arises. In this example, if the user attempts to read from a file that does not exist – the program will tell the user the file does not exist and to add data to create a file (Figure 7.7). This is much more useful to the user rather than displaying a simple error message. When developers handle errors by type, their program can run smoothly while also informing the user exactly what has gone wrong and how to change their input.

When handling errors by type, it can also be beneficial to have a final ‘catch all’ exception in case unexpected errors come up. In the final two lines of Listing 7, we define an exception and then print the name of the error back to the user using the .__class__.__name.__ attribute of the exception raised by Python. Although this is not extremely helpful, it will give the user a better understanding of what has happened in the program.

Handling Pickles (Putting it all Together)

Listing 8

```
import pickle

# Define data
pickle_file = "moodTracker.abt"
pickled_list = None
unpickled_list = []

# Present menu to user
print("Welcome to the Mood Tracker! Please select one of the following
options:")
print('''
  Menu of Options
  1) Add New Entry (Remove Old Entries)
  2) Add New Entry (Keep Old Entries)
  3) View Current Entries
  4) Exit Program
  ''')
print()  # Add an extra line for looks

# Get user input
user_choice = str(input("Which option would you like to perform? [1 to 4] -
")).strip()

# Write new data to binary file
if user_choice == "1":
  user_date = str(input("Enter today's date: ")).strip()
  user_mood=str(input("Enter your mood: ")).strip()
  while True:
     try:
        user_rate=int(input("Rate your day (1 - 10): "))
        break
     except ValueError:
        print("\nInvalid entry! Please only enter numbers.\n")
     except Exception as e:
        print(f"A {e.__class__.__name__} error has occurred!")
  pickled_list = [user_date, user_mood, user_rate]  # Add user inputted data to
list object

  with open(pickle_file, "wb") as binary_file:  # Open/create new binary file
to write new data to
```

```
    pickle.dump(pickled_list, binary_file)  # "Dump" data to binary file

  print("\nEntry Added!")

# Add new data to existing file
elif user_choice == "2":
  user_date = str(input("Enter today's date: ")).strip()
  user_mood = str(input("Enter your mood: ")).strip()
  while True:
    try:
      user_rate=int(input("Rate your day (1 - 10): "))
      break
    except ValueError:
      print("\nInvalid entry! Please only enter numbers.\n")
    except Exception as e:
      print(f"A {e.__class__.__name__} error has occurred!")
  pickled_list = [user_date, user_mood, user_rate]  # Add user inputted data to
list object

  with open(pickle_file, "ab") as binary_file: # Open/create a new or existing
binary file to append new data to
    pickle.dump(pickled_list, binary_file) # "Dump" data to binary file

  print("\nEntry Added!")

# Read data from binary file
elif user_choice == '3':
  try:
    with open(pickle_file, "rb") as binary_file:  # Open existing binary file
to read data from
      while True: # Continue to load all objects file until all objects are
loaded
        try:
          unpickled_list.append(pickle.load(binary_file))  # "Load" data
from binary
        except EOFError:  # Once end of file is reached, break out of loop
          break
    print("\nCurrent Data:")
    print(unpickled_list)
  except FileNotFoundError:
    print("\nFile does not exist! Add new data to create a file.\n")
  except Exception as e:
    print(f"A {e.__class__.__name__} error has occurred!")

# Exit program
elif user_choice == '4':
  input("\nPress Enter to Exit")

input("\nPress Enter to Exit")
```

Now, putting all the pieces together – Listing 8 shows a single run program that allows users to track their daily mood by adding new entries or view all of their current entries. The program starts off by welcoming the user and displaying a menu of options for the user to choose from. Based on the user's selection, they are able to add new data by writing or appending entries to the binary file and they are able to view all the entries currently in the binary file by loading and unpickling the data. Throughout each section, try-except blocks are utilized to catch potential errors caused by user input or the current file states. The program goes further to define exceptions for specific errors to display more useful information back to the user.

As highlighted in Listing 8, with pickling and unpickling objects, it is important to effectively handle exceptions to ensure the program runs smoothly. One notable change which was made in this final code, was implementing an additional try-except within the ‘read’ block. As mentioned in previous sections, if there are multiple objects in a binary file, the .load() method must be called multiple times to load all the data. In previous listings, only the first object from the file would be outputted when trying to display the current data. To ensure all the data is loaded and the program does not throw an error – a try-except block was implemented within a while loop to continuously load data until the program reached the end of the binary file.

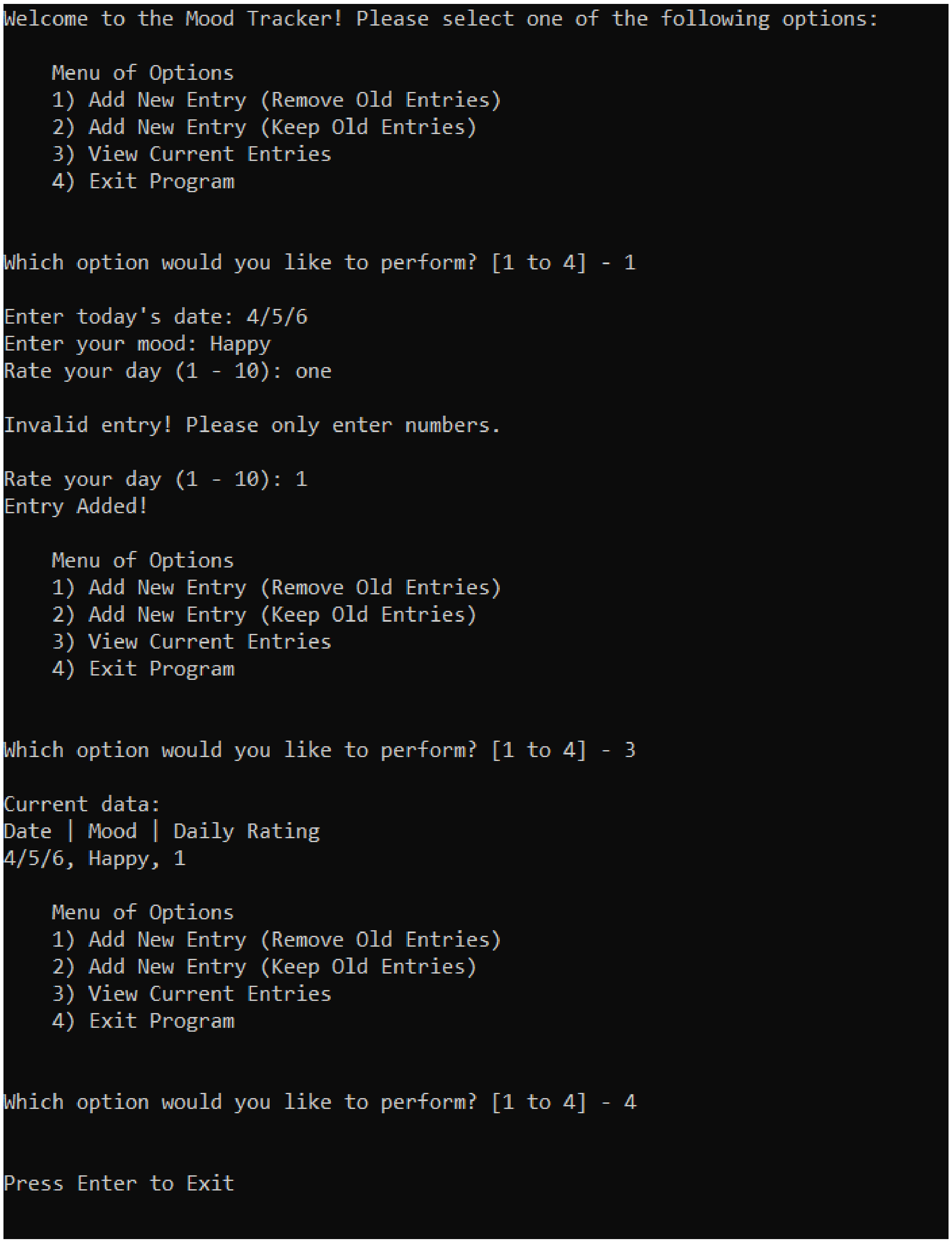
Listing 9

```
import pickle

# Define the data
pickle_file = "moodTracker.abt"

# Functions
def output_menu_options():
  """  Display a menu of choices to the user

  :return: nothing
  """
  print('''
  Menu of Options
  1) Add New Entry (Remove Old Entries)
  2) Add New Entry (Keep Old Entries)
  3) View Current Entries
  4) Exit Program
  ''')
  print()  # Add an extra line for looks


def input_menu_choice():
      """ Gets the menu choice from a user

      :return: string
      """
      choice = str(input("Which option would you like to perform? [1 to 4] -
")).strip()
      print()  # Add an extra line for looks
      return choice


def add_entry():
  """  Gets user input and adds entry to pickled_list

         :return: nothing
         """
  user_date = str(input("Enter today's date: ")).strip()
  user_mood = str(input("Enter your mood: ")).strip().capitalize()
  while True:
      try:
          user_rate = int(input("Rate your day (1 - 10): "))
          break
      except ValueError:
          print("\nInvalid entry! Please only enter numbers.\n")
      except Exception as e:
          print(f"\nA {e.__class__.__name__} error has occurred!")
  pickled_list.append([user_date, user_mood, user_rate])  # Add user inputted
data to list object
  print("\nEntry Added!")


def format_data():
  """  Formats and displays data from binary file to user

         :return: nothing
         """
  print("Current data:")
  print("Date | Mood | Daily Rating")
  for entry in unpickled_list:
      for item in entry:
          print(f"{item[0]}, {item[1]}, {item[2]}")


def write_new_data():
  """  Adds new entry data to binary file and overwrites existing data

         :return: nothing
         """
  add_entry()
  with open(pickle_file, "wb") as binary_file:  # Open/create new binary file
to write new data to
      pickle.dump(pickled_list, binary_file)  # "Dump" data to binary file

def load_existing_data():
  """  Tries to read data from binary file and loads it into list object

         :return: nothing
         """
  try:
      with open(pickle_file, "rb") as binary_file:  # Open existing binary file
to read data from
          while True:
              try: # Continue to read all objects file until all objects are
loaded
                  unpickled_list.append(pickle.load(binary_file))  # "Load"
data from binary
              except EOFError: # Once end of file is reached, break out of
loop
                  break
      format_data()
  except FileNotFoundError:
        print("\nFile does not exist! Add data to the file to view.\n")
    except Exception as e:
        print(f"\nA {e.__class__.__name__} error has occurred!")

def append_data():
    """  Adds new entry and appends it to existing binary file

          :return: nothing
          """
    add_entry()
    with open(pickle_file, "ab") as binary_file: # Open/create a new or existing
binary file to append new data to
        pickle.dump(pickled_list, binary_file) # "Dump" data to binary file


# Show user menu
print("Welcome to the Mood Tracker! Please select one of the following
options:")
while True:
    # Define data
    pickled_list=[]
    unpickled_list=[]

    # Display menu
    output_menu_options()
    user_choice = input_menu_choice() # Stores user's menu selection

    # Write new data to binary file
    if user_choice == "1":
        write_new_data()
        continue
    # Add new data to existing file
    elif user_choice == "2":
        append_data()
        continue
    # Read data from binary file
    elif user_choice == '3':
        load_existing_data()
        continue
    # Exit program
    elif user_choice == '4':
        input("\nPress Enter to Exit")
        break
    else:
        print("Please only select 1 - 4!")
        continue
```

Listing 9 shows an enhanced version of Listing 8 - implementing a while loop to keep the program continuously running until the user chooses to exit and defining functions for some of the repetitive tasks in the program, such as adding new entries. The program also improves the display of data once it is read from the binary file, formatting to be more readable and user friendly.

Executing the Script

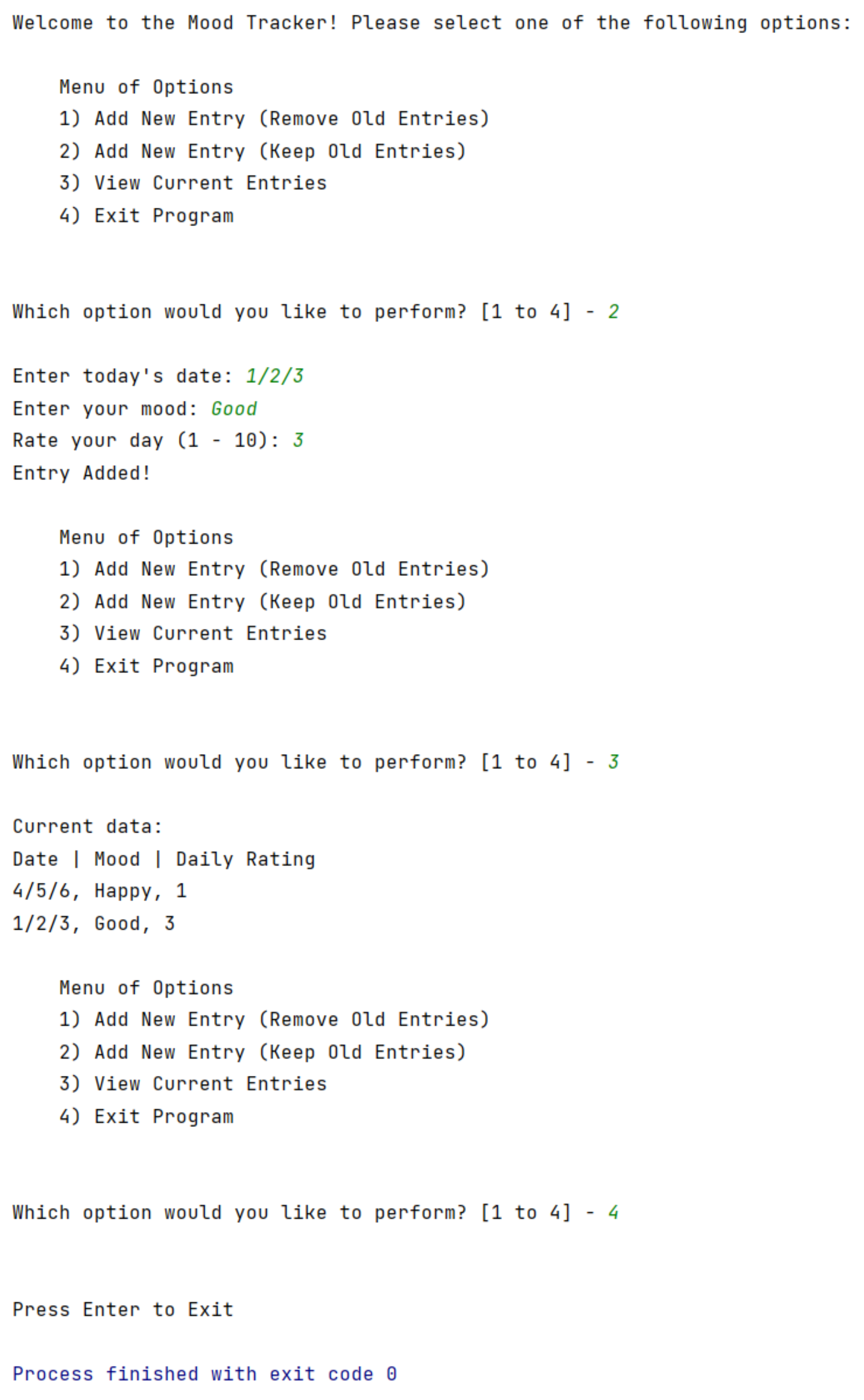
Executing in the Command Window



```
Welcome to the Mood Tracker! Please select one of the following options:

    Menu of Options
    1) Add New Entry (Remove Old Entries)
    2) Add New Entry (Keep Old Entries)
    3) View Current Entries
    4) Exit Program


Which option would you like to perform? [1 to 4] - 1

Enter today's date: 4/5/6
Enter your mood: Happy
Rate your day (1 - 10): one

Invalid entry! Please only enter numbers.

Rate your day (1 - 10): 1
Entry Added!

    Menu of Options
    1) Add New Entry (Remove Old Entries)
    2) Add New Entry (Keep Old Entries)
    3) View Current Entries
    4) Exit Program


Which option would you like to perform? [1 to 4] - 3

Current data:
Date | Mood | Daily Rating
4/5/6, Happy, 1

    Menu of Options
    1) Add New Entry (Remove Old Entries)
    2) Add New Entry (Keep Old Entries)
    3) View Current Entries
    4) Exit Program


Which option would you like to perform? [1 to 4] - 4


Press Enter to Exit
```

Figure 7.8 - Executing script in the Command Shell

Executing in the PyCharm



```
Welcome to the Mood Tracker! Please select one of the following options:

    Menu of Options
    1) Add New Entry (Remove Old Entries)
    2) Add New Entry (Keep Old Entries)
    3) View Current Entries
    4) Exit Program


Which option would you like to perform? [1 to 4] - 2

Enter today's date: 1/2/3
Enter your mood: Good
Rate your day (1 - 10): 3
Entry Added!

    Menu of Options
    1) Add New Entry (Remove Old Entries)
    2) Add New Entry (Keep Old Entries)
    3) View Current Entries
    4) Exit Program


Which option would you like to perform? [1 to 4] - 3

Current data:
Date | Mood | Daily Rating
4/5/6, Happy, 1
1/2/3, Good, 3

    Menu of Options
    1) Add New Entry (Remove Old Entries)
    2) Add New Entry (Keep Old Entries)
    3) View Current Entries
    4) Exit Program


Which option would you like to perform? [1 to 4] - 4


Press Enter to Exit

Process finished with exit code 0
```

Figure 7.9 - Executing script in PyCharm

The final Listing 9 script was successfully executed in both the Command Window (Figure 7.8) and PyCharm (Figure 7.9). Each feature of the program was tested (writing, reading, and appending) and outputted expected results in both environments.

Summary

The Mood Tracker program developed in this documentation showcases pickling and error handling in Python. In the final product, the user is able to write, read, and append data to their tracker – in Python this required the use of pickling and unpickling data from a binary file to securely store and retrieve data the user had entered. In addition, working with user input and files can often result in program errors. To mitigate any issues, try-except blocks were implemented throughout the program to effectively handle any potential problems the user may face when entering or interacting with their data.

Additional Resources

Pickling in Python

- Python Pickling - Good general resource for pickling methods with code examples and potential errors that developers could encounter when pickling.
- Using the Python pickle Module - Short and simple video that demonstrates both writing and loading data from binary files. Utilizes multiple data types and demonstrates pickling with a class object.
- Pickle.load() Method in Python - More information on the .load() pickle method which can be a bit more tricky to work with.

Exception Handling in Python

- Python Exceptions: An Introduction - Good introductory source that goes through try-except blocks in Python and some additional exception handling clauses.
- Exception & Error Handling in Python - Lots of good information on error handling with example code. Goes into how to utilize exceptions more in-depth.
- with Statement in Python - Goes more in-depth on the with statement and why it would be more advantageous to use in programs.