Name: Jane Shin
Date: December 1, 2022
Course: IT FDN 110 A Au 22: Foundations of Programming: Python

https://github.com/ks180337/ITFnd100-Mod07

Assignment 07 – Files & Exceptions

Introduction

This document describes the steps taken in performing Assignment 07, where a Python script file from Assignment 06 is updated to demonstrate the concepts of exception handling and pickling module.

The script still uses a printed "menu" to guide the user through various options for the “To-do List”, including displaying current list, adding a new item, removing an existing item, saving data to the original file, and exiting the program. The scope of work is similar to that of Assignment 06, and uses functions and classes to organize Data, Processing, and Presentation sections. However, it uses pickling modules to read and write to files, and has a structured error handling feature.

The "To-do List" file contains two columns of data, "Task" and "Priority." The columns are loaded into a Python dictionary object. Each dictionary object represents one row of data, and these rows are added to a Python list object to create a table of data.

New tools and concepts introduced in Module 7, such as, pickling modules and a structured error handling feature, as well as previously addressed concepts such as list, dictionary, functions and classes will be used in accomplishing this task.

Instructions

Module 7 covered a detailed application of the function and class concepts, delivered through course videos, a book chapter, and web pages. The course material delivered detailed explanation and examples of the following topics:

- Working with text files – Read, Append modes
- Options for reading data
- Combining reading and writing
- Working with binary files, using pickling modules
- Structured error handling (Try-Except)
- Using the exception class
- Catching specific exceptions
- Raising custom errors
- Creating custom exception classes
- Creating GitHub pages and a markdown file, formatting the page

Research Exception Handling in Python

Helpful websites
https://www.learnpython.org/en/Exception_Handling
https://pythonbasics.org/try-except/

These websites had thorough explanations on the concept of error handling, the reason why it is needed, and practical examples of errors and potential handling suggestions for them.

Research Pickling in Python

Helpful website
https://docs.python.org/3/library/pickle.html

In addition to an overview with short examples, it provided a link to the GitHub page with a source code to review, which helped with a more holistic understanding of the concept and application.

Creating the Script

Header and Initial Comments

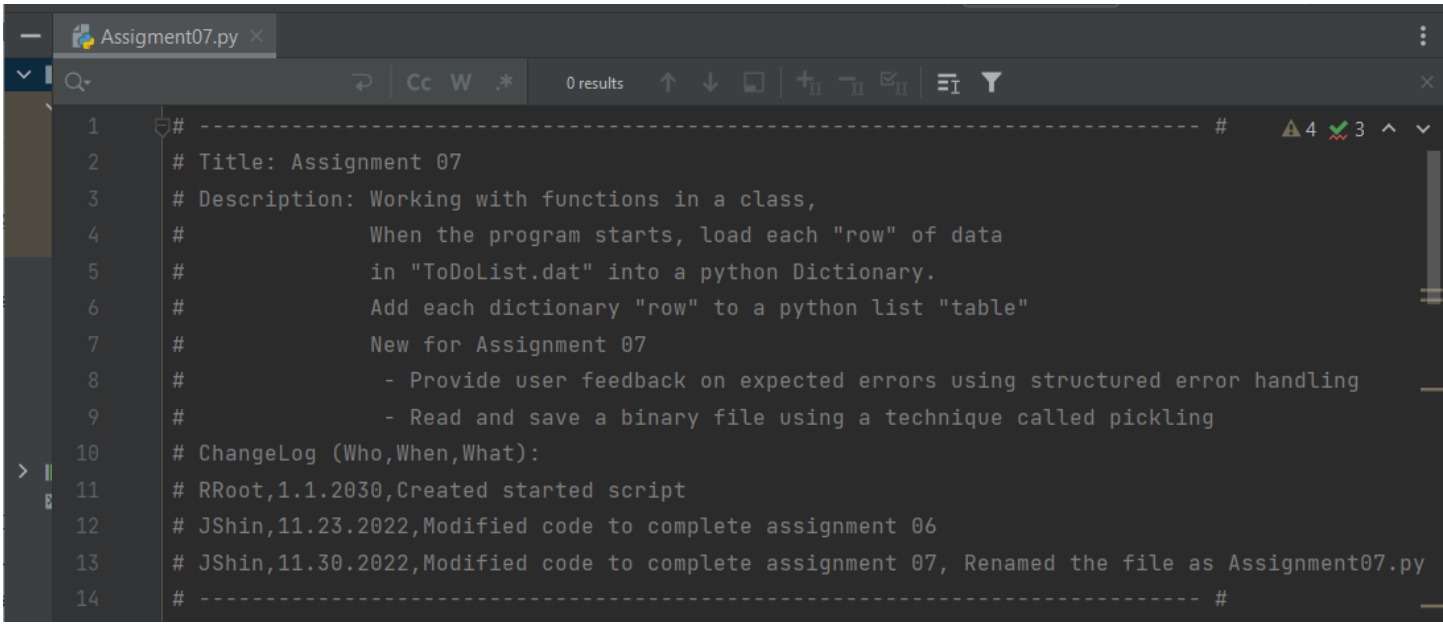

Figure 1. Header and initial comments

- PyCharm was used to create and run the script. The assignment was saved as a .py file named "Assignment07.py" at the location according to a direction given in the instruction.
- Script header and comments – A script header was created per the examples given in the instruction to communicate the function and document changes of the script.
- The Change Log was updated to reflect the codes added to the original starter script to complete the assignment.

Declaration of Variables

```
16  # Data --------------------------------------------------------------------- #
17  # Declare variables and constants
18  file_name_str = "ToDoList.dat"  # The name of the data file
19  file_obj = None  # An object that represents a file
20  row_dic = {}  # A row of data separated into elements of a dictionary {Task,Priority}
21  table_lst = []  # A list that acts as a 'table' of rows
22  choice_str = ""  # Captures the user option selection
23  task_str = ""  # Captures the task input from functions to be processed in main body
24  priority_str = ""   # Captures the priority input from functions to be processed in main body
```

Figure 2. Declaration of variables

- Using the “Separation of Concerns” concept, variables and constants are declared within the “Data” section along with comments explaining the purposes of each variable.
- Variables are organized in order of appearance.
- Variable from the original script were retained, and several new variables were added.
- The value of *file_name_str* was updated to “ToDoList.dat” to be used with pickling module.

Overall Script Structure

```
23  # Processing  ------------------------------------------------------------ #
24  class Processor:...
83
84
85  # Presentation (Input/Output)  ---------------------------------------- #
86  class IO:...
145
146 # Main Body of Script  ------------------------------------------------- #
147
148 # Step 1 - When the program starts, Load data from ToDoFile.txt.
149 Processor.read_data_from_file( file_name=file_name_str, list_of_rows=table_lst)  # read file data
150
151 # Step 2 - Display a menu of choices to the user
152 while (True):
153     # Step 3 Show current data
154     IO.output_current_tasks_in_list(list_of_rows=table_lst)  # Show current data in the list/table
155     IO.output_menu_tasks()  # Shows menu
156     choice_str = IO.input_menu_choice()  # Get menu option
157
158     # Step 4 - Process user's menu choice
159     if choice_str.strip() == '1':...
163
164     elif choice_str == '2':...
168
169     elif choice_str == '3':...
173
174     elif choice_str == '4':...
```

Figure 3. Overall Script Structure

- The overall structure of Assignment06.py was retained. This report will only review the specific sections updated to show:
    - Pickling module
        - Function, read_data_from_file
        - Function, write_data_to_file
    - Exception handling
        - Function, read_data_from_file
        - Main body, code block to obtain a new task
- Details for individual functions were reviewed in Assignment 06.
- Since the assignment was to update a pre-existing script, it was necessary to understand the overall structure and intent of the original starter script. Figure 3. shows a high-level flow of the starter script.
- The “Processing” section contains a class named **Processor**. This class consists of several functions that processes data such as reading from or writing to a file or a list.
- The “Presentation” section contains a class named **IO**. This class consists of several functions that handles inputs and outputs of data such as presenting a menu and obtaining a menu choice from the user.
- The details of each class and the functions belonging to the class are explained in later sections.
- The main body of the script will call upon the class and the functions to carry out the intent of the script.

Pickling module – function read_data_from_file

```python
import pickle   # This imports code from another code file

class Processor:
    """  Performs Processing tasks """

    @staticmethod
    def read_data_from_file(file_name, list_of_rows):
        """ Reads data from a file into a list of dictionary rows

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        list_of_rows.clear()  # clear current data
        try:
            file = open(file_name, "rb+")   # open file with rb+ mode; r for read,
            # b for binary, + to generate an error if file does not exist.
            list_of_rows = pickle.load(file)    # read/unpickle the data with pickle.load method
            file.close()
        except FileNotFoundError as e:  # Using more specific exception classes
            print("ToDoList.dat file is not found.")
            print("Select Option 4 to exit the program.")
            print("Built-In Python error info: ")
            print(e, e.__doc__, type(e), sep='\n')
        return list_of_rows
```

Figure 4. Overview of function **read_data_from_file**

- Import pickle is added to import code needed for pickling module.
- The **read_data_from_file** function was defined with two parameters, *file_name* and *list_of_rows.* It will read data from a file into a list of dictionary rows.
- It is a common practice to include a helpful header at the beginning of a function, which is known as *docstring*. PyCharm can display tooltips to show you a developer's notes (ctrl + q). This is common throughout the functions defined in the rest of the script.
- The **open()** function was used to open the file that contains initial data, "ToDoList.txt".
- "rb+" mode allows to read from a binary file. If the file does not exist, Python will generate an error.
- The file is then assigned to the variable *file*.
- **pickle.load** method was used to read (or unpickle) the data from the binary file, then saved into a list object named *list_of_rows*.
- The Try-Except portion of the function will be review in a later section.
- The **close()** function was used to close the file.
- The **read_data_from_file** function then returns a list object named *list_of_rows* as its return value.

Pickling module – function write_data_from_file

```
80      @staticmethod
81      def write_data_to_file(file_name, list_of_rows):
82          """ Writes data from a list of dictionary rows to a File
83
84          :param file_name: (string) with name of file:
85          :param list_of_rows: (list) you want filled with file data:
86          :return: (list) of dictionary rows
87          """
88          file = open(file_name, "wb")    # open file with wb mode; w for write, b for binary.
89          pickle.dump(list_of_rows,file)  # write/pickle the data with pickle.dump method
90          file.close()
91          return list_of_rows
```

Figure 5. Overview of unction **write_data_to_file**

- The **write_data_to_file** function was defined with two parameters, *file_name* and *list_of_rows.* It will write data from a list of dictionary rows to a file.
- The **open()** function was used to open a file.
- The "wb" mode allows to write to a binary file. If the file already exists, its content is overwritten. If the file does not exist, it is newly created.
- The file is then assigned to the variable *file*.
- **pickle.dump** method was used to write (or pickle) the data stored in a list variable *list_of_rows* into the binary file.
- The **close()** method was used to close the file.
- The **write_data_to_file** function then returns a list object named *list_of_rows* as its return value.

Exception handling – function read_data_from_file

```
28  import pickle   # This imports code from another code file
29
30  class Processor:
31      """  Performs Processing tasks """
32
33      @staticmethod
34      def read_data_from_file(file_name, list_of_rows):
35          """ Reads data from a file into a list of dictionary rows
36
37          :param file_name: (string) with name of file:
38          :param list_of_rows: (list) you want filled with file data:
39          :return: (list) of dictionary rows
40          """
41          list_of_rows.clear()  # clear current data
42          try:
43              file = open(file_name, "rb+")   # open file with rb+ mode; r for read,
44              # b for binary, + to generate an error if file does not exist.
45              list_of_rows = pickle.load(file)    # read/unpickle the data with pickle.load method
46              file.close()
47          except FileNotFoundError as e:  # Using more specific exception classes
48              print("ToDoList.dat file is not found.")
49              print("Select Option 4 to exit the program.")
50              print("Built-In Python error info: ")
51              print(e, e.__doc__, type(e), sep='\n')
52          return list_of_rows
```

Figure 6. Overview of function **read_data_from_file**

- Try-Except structure was used to control the flow of the program execution, and to provide the user more information when an error occurs.
- “rb+” mode allows to read from a binary file. If the file does not exist, Python will generate an error.
- Line 47 shows more specific exception classes, where it will give custom error messages if there is no “ToDoList.dat” file in the directory.
- It prints a specific file name that is missing, then instruct the user to the next steps.
- It then prints the built-in Python error messages.

Exception handling – Adding a New Task

```
168     # Step 4 - Process user's menu choice
169     if choice_str.strip() == '1':  # Add a new Task
170         # A structured error handling code added to flag users of empty inputs and returns to the
171         while (True):
172             try:
173                 task_str, priority_str = IO.input_new_task_and_priority()
174                 if not task_str or not priority_str:
175                     raise ValueError
176             except ValueError as e:
177                 print("Task/priority cannot be empty")
178                 print("Built-In Python error info: ")
179                 print(e, e.__doc__, type(e), sep='\n')
180             else:
181                 break
182         table_lst = Processor.add_data_to_list(task=task_str, priority=priority_str, list_of_rows=
183         continue  # to show the menu
```

Figure 7. Overview of the "Adding a New Task" section

- *choice_str*, the variable that contains user option choice is evaluated using the if…elif sequence. If *choice_str* equals 1, then Option 1 is chosen to add a new task
    - Functions **IO.input_new_task_and_priority** and **Processor.add_data_to_list** were called to accomplish this task.
    - The **elif** statement is bookended with continue to force the loop to jump back to the beginning and display the menu and asks the user to make the next choice.
- Try-Except structure was used to control the flow of the program execution, and to provide the user more information when an error occurs.
- If no values were entered for Task or Priority, it will generate ValueError.
- Line 176 shows more specific exception classes, where it will give custom error messages if no values were entered for Task or Priority.
- It prints the built-in Python error messages, then break out of the while loop to ask for input again.

Running the Script

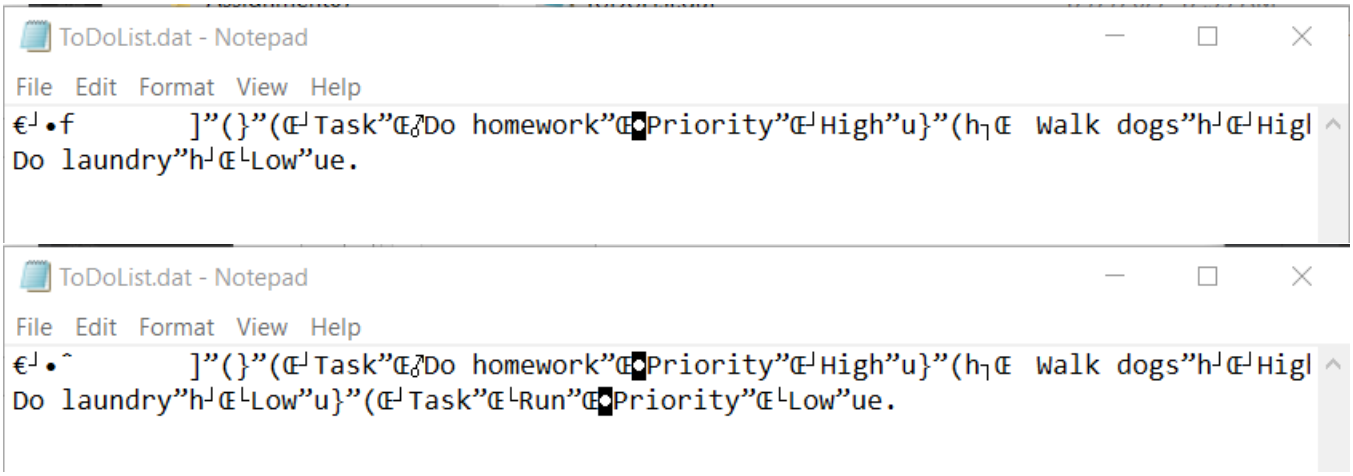

Figure 8. Updated binary files

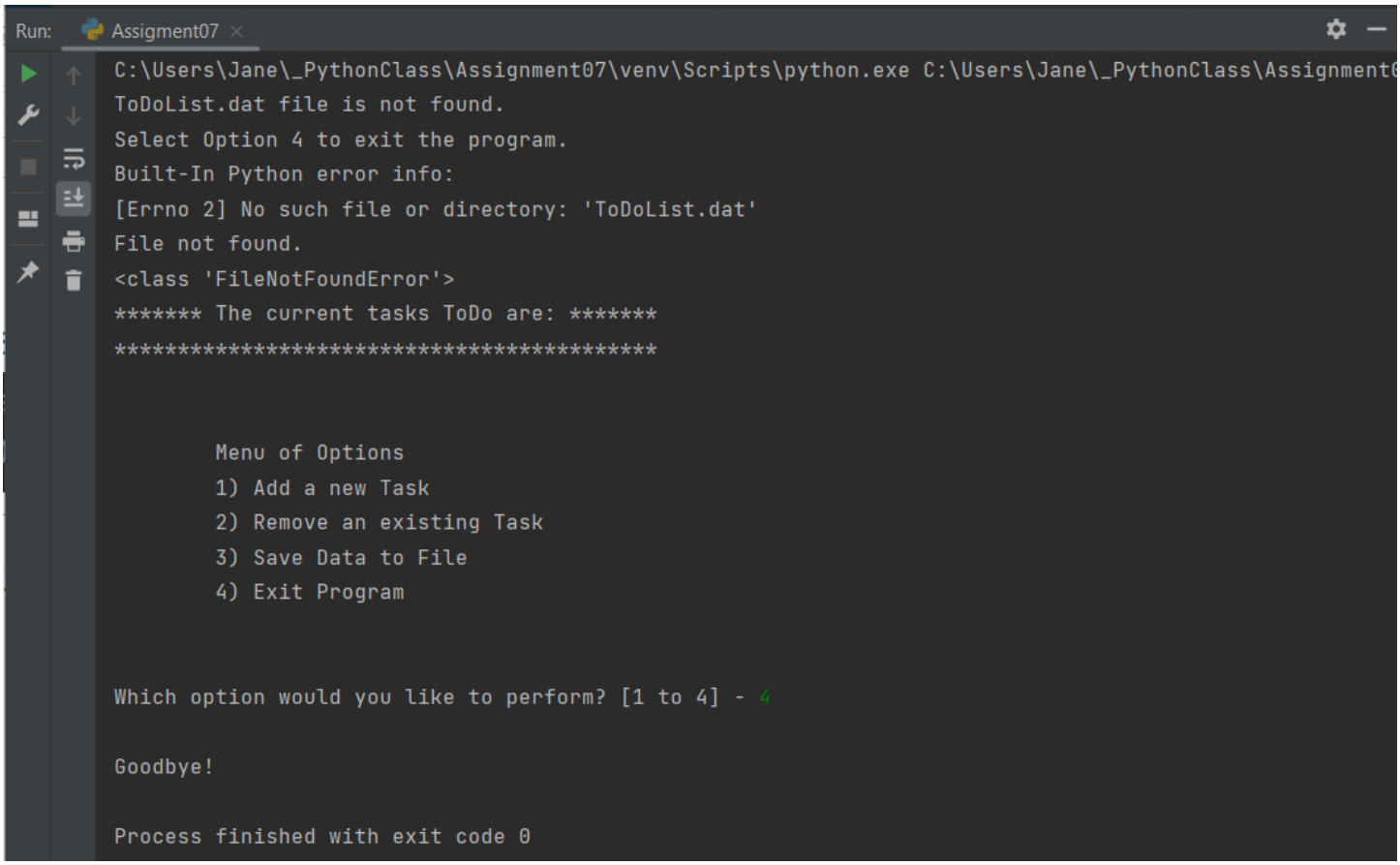

Figure 9. In PyCharm – error messages for no "ToDoList.dat" file present

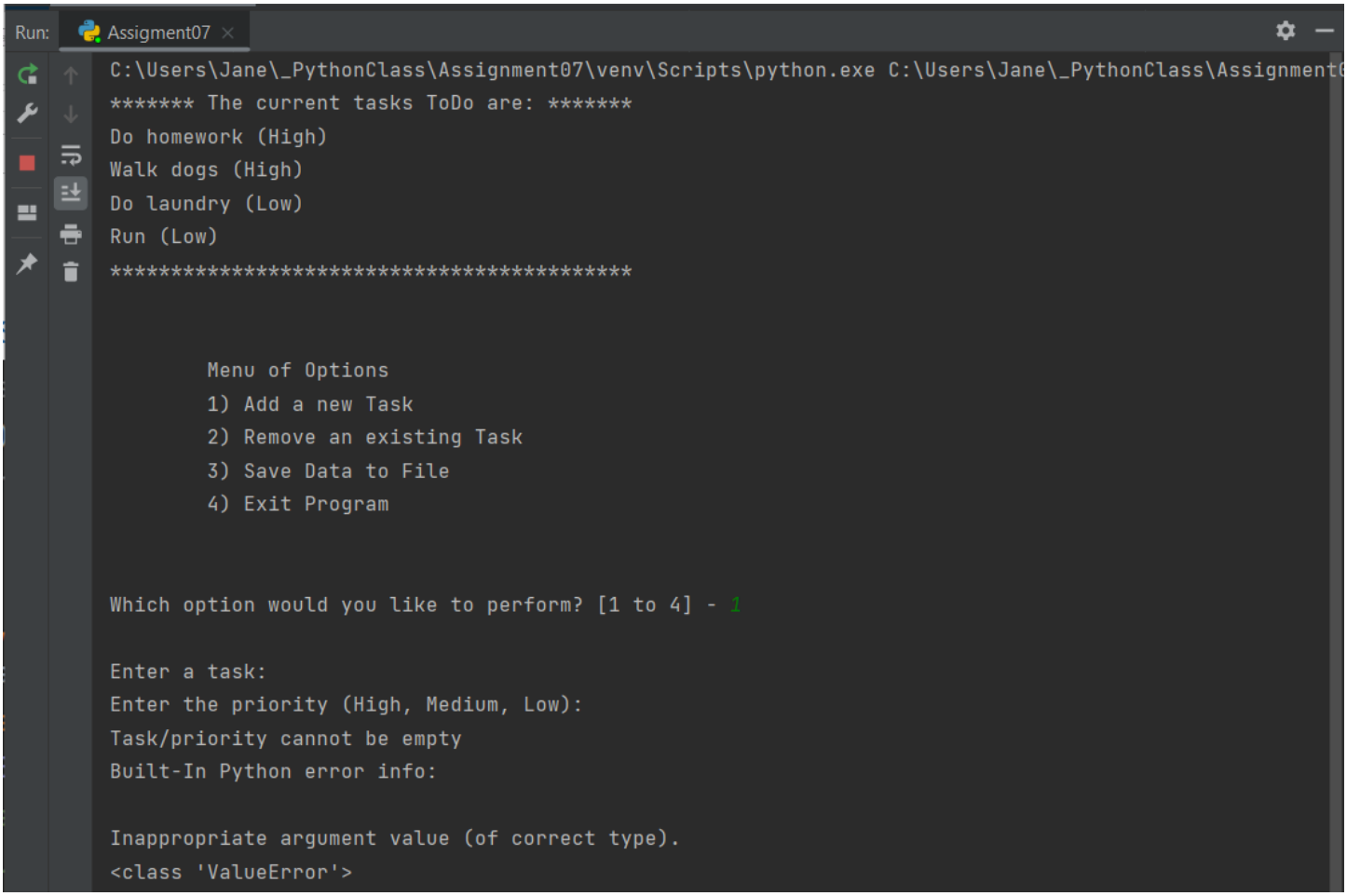

Figure 9. In PyCharm – error messages for empty task or priority

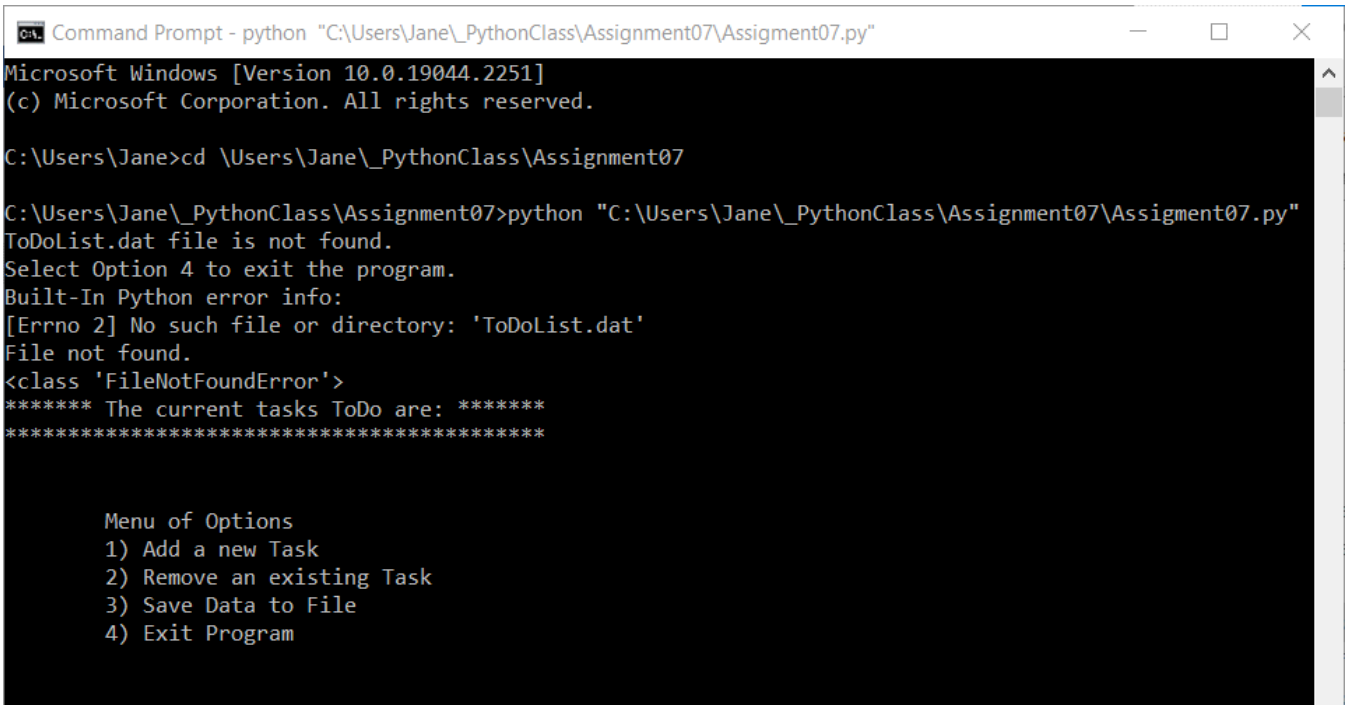

Figure 10. In command shell – error messages for no “ToDoList.dat” file present

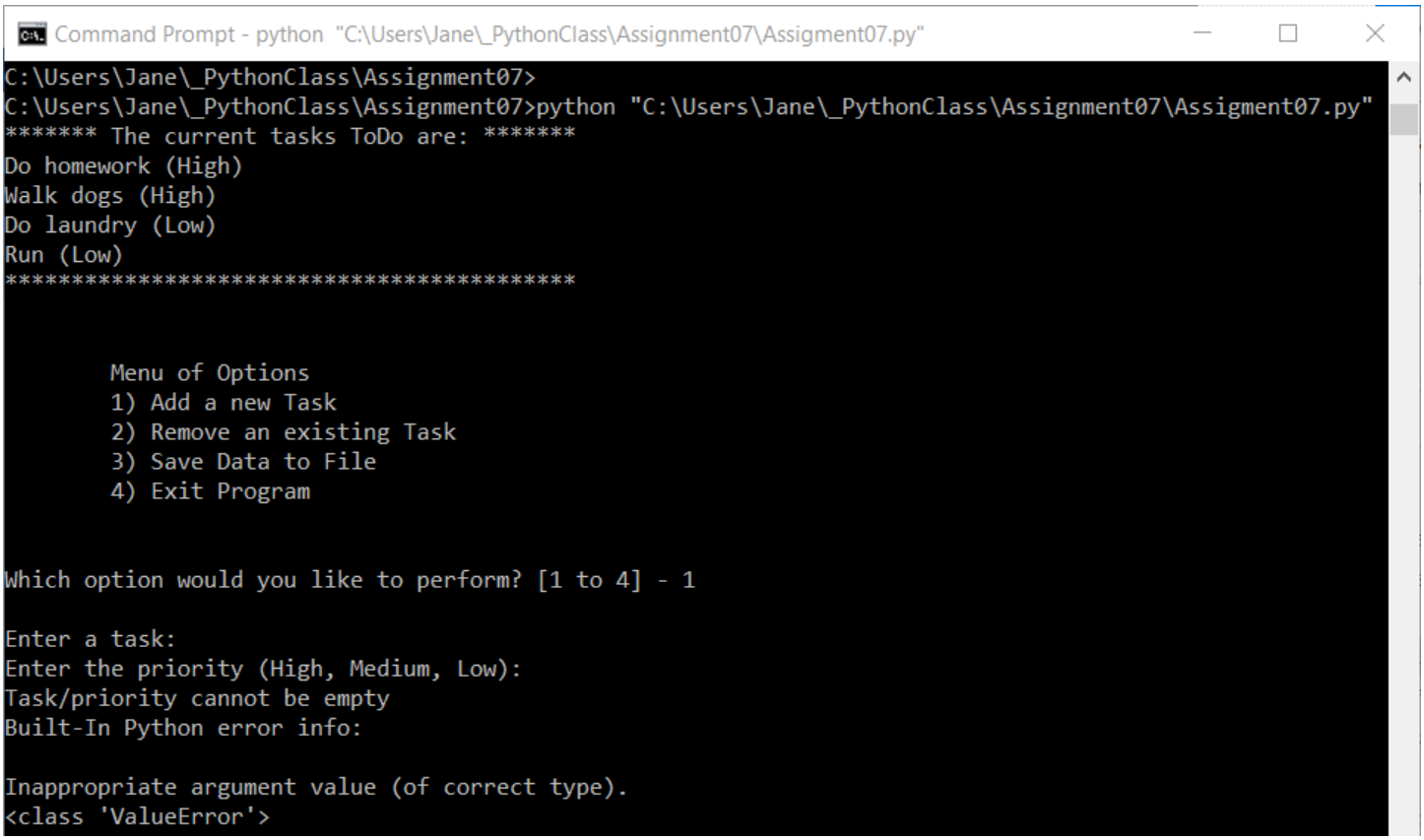

Figure 10. In command shell – error messages for empty task or priority

Summary

The script described above demonstrates several topics and guidelines covered in Module 7. Assignment 07 requirements outlined in Steps 3-6 were successfully carried out and documented in this report. A Python script file is created, using the script for Assignment 06 as base, to add codes that provide user feedback on expected errors using structured error handling, and read and save a binary file using a technique called pickling

New tools and concepts introduced in Module 7, such as, pickling modules and a structured error handling feature, as well as previously addressed concepts such as list, dictionary, functions and classes will be used in accomplishing this task. Throughout the script, comments were used to describe the functions and intents of the applicable set of codes.

The PyCharm debugging tool was used to troubleshoot and to test out various functions of the tool such as breakpoints and “walk” through the code.

The assignment artifacts were then uploaded to GitHub and Canvas to be submitted for evaluation.