Richard Sauer

May 28, 2023

Foundations of Programming: Python

Assignment 07

GitHub: [richsau/IntroToProg-Python-Mod07 (github.com)]

Pickling and Structured Error Handling

Introduction

The goal of this assignment was to write a program that demonstrates the use of pickling methods and exception handling. A ***Menu*** class is used to display to the user the various examples in the application and gets input from the user for their choice.

1) [Pickling] Append binary data to a file.
2) [Pickling] Read and display data from binary file.
3) [Error Handling] Basic try / except example.
4) [Error Handling] Using try / except to capture and print the exception.
5) [Error Handling] Using try / except to look for a specified exception.
6) [Error Handling] Using raise to cause a custom exception.
7) Exit program.

The two remaining classes, ***Pickling*** and ***ErrorHandling***, contain the code to the various methods described in the menu.

The Pickling Class

AppendBinaryData() Function

This function gets data in the form of an ***Item ID*** and ***Item Name*** from the user, then opens a file in ***append binary*** mode, then writes the data to the file using the ***pickle.dump()*** function. (Figures 1 & 2)

```python
    @staticmethod
    def AppendBinaryData():
        """ Gets data from the user writes it to a binary file.
        :return: nothing
        """
        itemId = str(input("Enter item ID: "))
        itemName = str(input("Enter item name: "))
        lstItemData = [itemId, itemName]
        objFile = open("ItemDatabase.dat", "ab")
        pickle.dump(lstItemData, objFile)
        print("Data: " + str(lstItemData) + " saved to file.")
        objFile.close()

```

Figure 1. Using pickle.dump() to write data to a binary file.

```
            Menu of Options
            1) [Pickling] Append binary data to a file.
            2) [Pickling] Read and display data from binary file.
            3) [Error Handling] Basic try / except example.
            4) [Error Handling] Using try / except to capture and print the exception.
            5) [Error Handling] Using try / except to look for a specified exception.
            6) [Error Handling] Using raise to cause a custom exception.
            7) Exit program


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

Enter item ID: AW56927
Enter item name: Glass Lamp
Data: ['AW56927', 'Glass Lamp'] saved to file.
```

Figure 2. Running the AppendBinaryData() function.

ReadBinaryData() Function

This function opens the data file in **read binary** mode, reads the data from the file using the ***pickle.load()*** function, then prints out the data in a formatted table. It uses exception handling in two ways. The first checks for an exception when opening the data file. If it encounters one, then it informs the user to use menu item #1 to write out a data file first. The second one reads data from the file in a loop until it encounters the ***EOFError*** exception to exit the loop. (Figures 3 & 4)

```python
    @staticmethod
    def ReadBinaryData():
        """ Shows the current items in the database using try/except to find end of file.
        :return: nothing
        """
        try:
            objFile = open("ItemDatabase.dat", "rb")
        except Exception as e:
            print(e)
            print("Try option 1 to write data to file first.")
        else:
            seperatorText = "----------"
            print("******* Contents of Database *******")
            print("{:10} {:20}".format("Item ID", "Item Name"))
            print("{:10} {:20}".format(seperatorText, seperatorText))
            while True:
                try:
                    lstItemData = pickle.load(objFile)
                    print("{:10} {:20}".format(lstItemData[0], lstItemData[1]))
                except EOFError:
                    break
            objFile.close()
```

Figure 3. Opening and reading data from the file using try / except to catch exceptions.

```
            Menu of Options
            1) [Pickling] Append binary data to a file.
            2) [Pickling] Read and display data from binary file.
            3) [Error Handling] Basic try / except example.
            4) [Error Handling] Using try / except to capture and print the exception.
            5) [Error Handling] Using try / except to look for a specified exception.
            6) [Error Handling] Using raise to cause a custom exception.
            7) Exit program


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

******* Contents of Database *******
Item ID    Item Name
---------- ----------
AZ4297A    Oak Table
AF83429    6' Bookcase
AW56927    Glass Lamp
```

Figure 4. Running the ReadBinaryData() Function.

The ErrorHandling Class

While (Figure 3) above shows some common ways to use exceptions in an application, the functions in this class demonstrate some other ways to use exceptions.

The BasicTryExcept() Function

This function shows the basic use of ***try*** and ***except***. This is useful when you just want to protect the application against crashing. It's especially useful when the application could encounter an error due to user input. This function also shows how you *could* continue a function after an exception using the ***finally:*** keyword. Although this works, after researching this subject, it's generally considered bad form to use the ***finally:*** keyword like this as you would usually want to just exit the function. (Figures 5 & 6)

```
78      @staticmethod
79      def BasicTryExcept():
80          """ Shows use of try / except.
81          :return: nothing
82          """
83          try:
84              print("30 / 0", end=" = ")
85              print(str(30 / 0))
86              # myValue = 30 / 0
87          except:
88              print("Custom message: An exception occurred.")
89          finally:
90              print("30 * 0", end=" = ")
91              print(str(30 * 0))
```

Figure 5. Using the basic try and except keywords to catch an exception.

```
            Menu of Options
            1) [Pickling] Append binary data to a file.
            2) [Pickling] Read and display data from binary file.
            3) [Error Handling] Basic try / except example.
            4) [Error Handling] Using try / except to capture and print the exception.
            5) [Error Handling] Using try / except to look for a specified exception.
            6) [Error Handling] Using raise to cause a custom exception.
            7) Exit program


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

30 / 0 = Custom message: An exception occurred.
30 * 0 = 0
```

Figure 6. Running the The BasicTryExcept() function.

The TryExceptCapture() Funtion

In addition to wrapping the bad code in the ***try except*** block, it also puts the exception in a variable and prints out additional information about the exception by printing the exception, followed by the ***type()*** of exception, followed by more details on the exception. (Figures 7 & 8)

```
1 usage
93      @staticmethod
94      def TryExceptCapture():
95          """ Captures exception and shows details.
96          :return: nothing
97          """
98          try:
99              print("30 / 0", end=" = ")
100             print(str(30 / 0))
101         except Exception as e:
102             print(e)
103             print(type(e))
104             print(e.__doc__)
105
```

Figure 7. Capturing the exception and printing out additional information about it.

```
            Menu of Options
            1) [Pickling] Append binary data to a file.
            2) [Pickling] Read and display data from binary file.
            3) [Error Handling] Basic try / except example.
            4) [Error Handling] Using try / except to capture and print the exception.
            5) [Error Handling] Using try / except to look for a specified exception.
            6) [Error Handling] Using raise to cause a custom exception.
            7) Exit program


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

30 / 0 = division by zero
<class 'ZeroDivisionError'>
Second argument to a division or modulo operation was zero.
```

Figure 8. Running the TryExceptCapture() function.

The TrySpecifiedException() Funtion

This function shows how to look for a specific exception and then an additional handler for any other exception. (Figure 9 & 10)

```
106     @staticmethod
107     def TrySpecifiedException():
108         """ Looks for specific exception.
109         :return: nothing
110         """
111         try:
112             print("30 / 0", end=" = ")
113             print(str(30 / 0))
114         except ZeroDivisionError:
115             print("Custom message: Division by zero exception occurred.")
116         except Exception as e:
117             print(e)
118
```

Figure 9. Catching a specific exception.

```
            Menu of Options
            1) [Pickling] Append binary data to a file.
            2) [Pickling] Read and display data from binary file.
            3) [Error Handling] Basic try / except example.
            4) [Error Handling] Using try / except to capture and print the exception.
            5) [Error Handling] Using try / except to look for a specified exception.
            6) [Error Handling] Using raise to cause a custom exception.
            7) Exit program


Which option would you like to perform? [1 to 4] - 5

30 / 0 = Custom message: Division by zero exception occurred.
```

Figure 10. Running the TrySpecifiedException() function.

The RaiseExample() Funtion

This function shows how to create a custom exception using the ***raise*** keyword. In this example, it looks for negative numbers. (Figures 11 & 12)

```
119     @staticmethod
120     def RaiseExample():
121         """ Shows use of raising a custom exception.
122         :return: nothing
123         """
124         try:
125             print("x = -21")
126             x = -21
127             if (x < 0):
128                 raise Exception("Custom message: Sorry, a negative number was not expected.")
129         except Exception as e:
130             print(e)
```

Figure 11. Creating a custom exception using the raise keyword.

```
            Menu of Options
            1) [Pickling] Append binary data to a file.
            2) [Pickling] Read and display data from binary file.
            3) [Error Handling] Basic try / except example.
            4) [Error Handling] Using try / except to capture and print the exception.
            5) [Error Handling] Using try / except to look for a specified exception.
            6) [Error Handling] Using raise to cause a custom exception.
            7) Exit program


Which option would you like to perform? [1 to 4] - 6

x = -21
Custom message: Sorry, a negative number was not expected.
```

Figure 12. Running the RaiseExample() function.

Running the Application from the Command Line

Finally, running one of the application functions from the command line. (Figure 13)

```
Command Prompt - python /

C:\_PythonClass\Module07 - Files and Exceptions\Assignment07>python Assignment07.py

            Menu of Options
            1) [Pickling] Append binary data to a file.
            2) [Pickling] Read and display data from binary file.
            3) [Error Handling] Basic try / except example.
            4) [Error Handling] Using try / except to capture and print the exception.
            5) [Error Handling] Using try / except to look for a specified exception.
            6) [Error Handling] Using raise to cause a custom exception.
            7) Exit program


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

******* Contents of Database *******
Item ID    Item Name
---------- ----------
AZ4297A    Oak Table
AF83429    6' Bookcase
AW56927    Glass Lamp
```

Figure 13. Running the application from the command line.

Summary

The goal of this assignment was to explore the use of ***Pickling*** functions as well as using exception handlers in various ways. The Pickling functions were used to write and read data to and from a binary file. The ***ErrorHandling*** class showed different ways to use the ***try*** and ***except*** keywords in code to capture exception instead of letting the application crash.