Tsedey Tadesse
IT FND 110 A AU 22
Assignment07
GitHub link: https://github.com/tsedeytadesse/Module07

Pickling and Exception Handling in Python

Introduction

In this module, I explored pickling data and exception handling of scripts in Python. Pickling allows us to save data in a binary mode while keeping the complexity of the data intact. I have learned how to write data to and read data from binary files by importing the pickle module in my scripts. Exception handing is also a very essential method used in python to prevent our program to crash or end unexpectedly when it run into an error. Therefore creating exception handling by utilizing the *try-except* statement and clause to display error messages without ending the program before it finishes running. I have also worked on a simple script that takes user data and calculate the body mass index of a person in order to demonstrate the use of pickling and exception handling.

Body Mass Index (BMI) Calculator Script

In order to demonstrate the use of pickling and exception handling in Python, I created a simple script that takes the weight and height of a person and calculate the BMI and outputs the result along with the meaning of the result. The program allows the user to choose either the metric system or the imperial system as their using for their input. My script uses picking to save and load data, and it uses default and custom exceptions to handle error messages. I have used classes and functions by utilizing separation of concerns to organize my script.

Pickling in Python

Picking allows us to save data to a file and read data from a file without losing the complexity of the data, such as formatting and data type. Pickling is used in Python by first importing the *pickle* module. "The pickle module implements binary protocols for serializing and de-serializing a Python object structure" (https://docs.python.org/3/library/pickle.html). Meaning that, the data we save by using the *pickle* module is stored only in binary mode and it can't be read by a normal text reader. ***Figure 1*** below shows the beginning of my script, where the first thing I did was importing the *pickle* module and defining the variables that I want to use in my script. Also note that the file uses the file extension .dat. The file can be opened by a text editor, but it cannot be read.

```
# ------------------------------------------------------------------------ #
# Title: Assignment 07
# Description: Simple calculator of body mass index (BMI)
#              by demonstrating use of pickling and exception
#              Handling in Python
# ChangeLog (Who,When,What):
# TTadesse,11.28.2022, Created script
# TTadesse,11.29.2022, Added Exception Handling
# ------------------------------------------------------------------------ #

import pickle

# Data ------------------------------------------------------------------- #

thefile = 'bmi.dat'
list_BMI = []
menu_choice = ""
```

Figure 1: importing pickle module

Dump in Pickling

The pickle.dump(*x, y)* function is used it save (dump) data in a binary file, where *x* is the data to be saved, and *y* is where the data is being saved. ***Figure 2*** below demonstrate how the function is used in my BMI script. The file that contains the data is opened in append mode, the new data is added, and then the file is closed. The new data is added as is without any formatting. The '*ab*' mode refers to the appending of the data in binary mode.

```
# Processing  ------------------------------------------------------------ #
class Pickling:
    """  Performs Processing tasks """

    @staticmethod
    def save_data_to_file(file_name, list_of_data):
        """ Writes data from a list of rows to a file in binary mode

        :param file_name: (string) name of file:
        :param list_of_data: (list) data in dictionary to be saved in file:
        :return: nothing
        """
        file = open(file_name, 'ab')
        pickle.dump(list_of_data, file)
        file.close()
```

Figure 2: use of pickle.dump() function

Load in Pickling

As data is saved using picking, it can also be read by using the pickle.load(x) function, where x is the file where the data is contained. To load data in pickle mode, first the file is opened in read mode, the file is read using the pickle.load() function, and then the file is closed. The '*rb*' mode refers to reading of the data from a binary file. ***Figure 3*** demonstrate the lines of code I used in my script to load data from a file.

```
    @staticmethod
    def read_data_from_file(file_name):
        """ Reads data from a binary file into rows of lists

        :param file_name: (string) with name of file:
        :return: lists of data
        """
        file = open(file_name, 'rb')
        data = pickle.load(file)
        file.close()
        return data
```

Figure 3: use of pickle.load() function

Presentation (IO Class of My Script)

The IO class of my script outputs the menu of options for my script and the explanation of the range where the BMI of a user might fall into, and it takes input to choose what the user would like to perform as well as the input of their weight and height data. ***Figure 4*** shows the IO class of the script.

```
class IO:
    """ Performs Input and Output tasks """

    @staticmethod
    def output_menu_options():
        """  Display a menu of choices to the user

        :return: nothing
        """
        print("\n Let's Calculate Your Body Mass Index (BMI) \n")
        print('''
        1) Calculate in Metric Unit
        2) Calculate in Imperial Unit
        3) Display BMI and Explanation
        4) Exit Program
        ''')
        print()
```

Figure 4-1: Presentation – menu of options

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

        :return: string of a digit
        """
        menu_choice = str(input("What would you like to perform? Please enter [1 to 4] "))
        print()
        return menu_choice

    @staticmethod
    def input_weight_height_in_metric():
        """  Gets weight and height of a user to calculate BMI in metric unit

        :return: (float, float) weight and height
        """
        weight = float(input("What is your weight in kG? "))
        height = float(input("What is your height in meters? "))

        return weight, height

    @staticmethod
    def input_weight_height_in_imperial():
        """  Gets weight and height of a user to calculate BMI in imperial unit

        :return: (float, float) weight and height
        """

        weight = float(input("What is your weight in pounds? "))
        height = float(input("What is your height in inches? "))

        return weight, height
@staticmethod
def output_description_BMI(BMI):
    """ Outputs descriptions of BMI ranges that fits the user BMI

    :return:nothing
    """
    if BMI <= 18.5:
        print("Your BMI falls within the underweight range. \n")
    elif BMI >= 24.9:
        print("Your BMI falls within the overweight range. \n")
    else:
        print("Your BMI falls within healthy weight range. \n")
```

Figure 4-2: Presentation – user input and BMI range output

Note that in the functions *input_weight_height_metric and input_weight_height_imperial,* the user input to be taken are floats. The user intentionally or unintentionally might input a string, which leads to a type error message and a sudden ending of the program. To combat unexpected ending of the program, we introduce exception handling to the script.

Exception Handling

Exception handing allows our program from crashing or ending unexpectedly when the program runs into an error. If the exception is not handled, the program displays the error message and ends the program and the program cannot be run until the error is resolved. Exceptions that demand the incorporation of handling are usually introduced by the user and not the programmer. Therefore, thinking ahead about what could go wrong as well as what seems right, but is logically wrong, is an important programming practice.

To solve this, the *try-except* statement and clause can be utilized in Python to handle the exception and run the program until completion. The *try* statement allows us to "section off some code that could potentially raise an exception." Then the *except* clause can be written "with a block of statements that are executed only if an exception is raised." Dawson, Michael, *Python Programming for the complete beginner, Third Edition*, Course Technology, 2010.

As demonstrated in my BMI script in ***Figure 5***, multiple exceptions could be used for the same section of code as multiple errors could arise. This can be done by identifying and specifying the exception type in the *except* clause block of the script. This required knowing the names Python uses to identify the errors. For example, ZeroDivisionError, as the name implies, is the type of error that would display when we try to divide a number by a zero.

```
while (True):
    # Step 1 Display a menu of choices to the user and ask for a menu of choice
    IO.output_menu_options()  # Shows menu
    menu_choice = IO.input_menu_choice()  # Get menu option

    # Step 2 - Process user's menu choice
    if menu_choice.strip() == '1':  # use metric method to take data from user
        try:
            weight, height = IO.input_weight_height_in_metric()
            BMI = weight/(height*height)
            list_BMI = [weight, height, BMI]
            Pickling.save_data_to_file(file_name=thefile, list_of_data=list_BMI)
```

Figure 5-1: main – menu options and menu choice

```
# Step 2 - Process user's menu choice
if menu_choice.strip() == '1':  # use metric method to take data from user
    try:
        weight, height = IO.input_weight_height_in_metric()
        BMI = weight/(height*height)
        list_BMI = [weight, height, BMI]
        Pickling.save_data_to_file(file_name=thefile, list_of_data=list_BMI)
    except ValueError as v:
        print("Your weight and height should only be a number \n")
        print("Built-In Python error info:")
        print(v.__doc__, type(v), sep='\n')
    except ZeroDivisionError as z:
        print("your height cannot be zero \n")
        print("Built-In Python error info:")
        print(z.__doc__, type(z), sep='\n')
    except FileNotFoundError as f:
        print("Binary data must exist before running this script")
        print("Built-In Python error info:")
        print(f.__doc__, type(f), sep='\n')
    try:
        if height >= 2.8:
            raise Exception("Height should be in meters")
    except Exception as e:
        print("Are you taller than the tallest man in the world?")
        print("try again")
        print("Built-In Python error info:")
        print(e.__doc__, type(e), sep='\n')

        continue  # to show the menu
```

Figure 5-2: main – try statement - except clause (metric)

For option '1' of the menu, where the user decides to enter their weight and height in metric unit, I came up with four scenarios where there could be errors in the program. The first one is to handle an exception for a ValueError. This error could happen if the user to decide to input anything other than a number. The program would let the user the that there is an error, tells the user what the error is in custom form as well as by displaying the built-in error message of Python, and carries on with running the program. Adding custom notes to why the error occurred is very important. (https://docs.python.org/3/tutorial/errors.html#enriching-exceptions-with-notes) The second exception type is ZeroDivisionError, where an error occurs if the user inputs zero for their height. This exception is handled by outputting a custom and built-in error messages as shown in ***Figure 5-2.*** The third exception is FileNotFoundError, where the file to dump to and load from might not exist. The last exception is a custom exception that I

created if the user decides to input illogical inputs. If the user decided to use the metric unit and input 2.8 meters for their height, the program creates an error because they probably are not taller than the tallest person that ever existed. Therefore I raise an exception to it. (https://realpython.com/python-exceptions/) Similar to the other exceptions, this one also prints the custom message for the error as well as the built-in Python error message.

The exception handling works the same way if the user decides to use the imperial unit. ***Figure 5.3*** shows the script for that section. The only difference is that the custom exception now is in units on inches.

```
if menu_choice.strip() == '2':  # use imperial method to take data from user
    try:
        weight, height = IO.input_weight_height_in_imperial()
        BMI = weight/(height*height)  # calculate the BMI
        list_BMI = [weight, height, BMI]
        Pickling.save_data_to_file(file_name=thefile, list_of_data=list_BMI)

    except ValueError as v:
        print("Your weight and height should only be a number \n")
        print("Built-In Python error info:")
        print(v.__doc__, type(v), sep='\n')
    except ZeroDivisionError as z:
        print("your height cannot be zero \n")
        print("Built-In Python error info:")
        print(z.__doc__, type(z), sep='\n')

    except FileNotFoundError as f:
        print("Binary data must exist before running this script")
        print("Built-In Python error info:")
        print(f.__doc__, type(f), sep='\n')
    try:
        if height <= 10:
            raise Exception("Enter height in units of inches, not feet")
    except Exception as e:
        print("Did you mean to put your height in inches, not feet?")
        print("Built-In Python error info:")
        print(e.__doc__, type(e), sep='\n')
        continue  # to show the menu
```

Figure 5-3: main – try statement - except clause (imperial)

The rest of the script is shown in ***Figure 5-4,*** and it is for the user if they choose option 3 or 4 on the menu.

```
elif menu_choice == '3':  # Explains what the BMI mean
    list_BMI = Pickling.read_data_from_file(file_name=thefile)  # read file data
    current_BMI = round(float(list_BMI[2]), 1)
    print("Your Current BMI is " + str(current_BMI))
    IO.output_description_BMI(BMI=current_BMI)
    continue  # to show the menu

elif menu_choice == '4':  # Exit Program
    print("Thank you! Check the CDC Website for More Information!")
    break  # by exiting loop
```

Figure 5-4: main – the rest of the code

Figure 6 shows the script in ***Figure 5*** run in PyCharm. And ***Figure 7*** shows the script run in the Terminal.

```
/Users/yilikalademe/Documents/_PythonClass/Assignment07/bin/python /Users/yilikalademe/Do

 Let's Calculate Your Body Mass Index (BMI)


       1) Calculate in Metric Unit
       2) Calculate in Imperial Unit
       3) Display BMI and Explanation
       4) Exit Program


What would you like to perform? Please enter [1 to 4] 2

What is your weight in pounds? 130
What is your height in inches? 65

 Let's Calculate Your Body Mass Index (BMI)


       1) Calculate in Metric Unit
       2) Calculate in Imperial Unit
       3) Display BMI and Explanation
       4) Exit Program
```

Figure 6-1: BMI script run in PyCharm

```
What would you like to perform? Please enter [1 to 4] 3

Your Current BMI is 0.0
Your BMI falls within the underweight range.


 Let's Calculate Your Body Mass Index (BMI)


        1) Calculate in Metric Unit
        2) Calculate in Imperial Unit
        3) Display BMI and Explanation
        4) Exit Program


What would you like to perform? Please enter [1 to 4] 4

Thank you! Check the CDC Website for More Information!
```

Figure 6-2: BMI script run in PyCharm (2)

Figure 6-3 demonstrates when an error is handled and the program continues to run.

```
What would you like to perform? Please enter [1 to 4] 1

What is your weight in kG? 60
What is your height in meters? 3
Are you taller than the tallest man in the world?
try again
Built-In Python error info:
Common base class for all non-exit exceptions.
<class 'Exception'>

 Let's Calculate Your Body Mass Index (BMI)


        1) Calculate in Metric Unit
        2) Calculate in Imperial Unit
        3) Display BMI and Explanation
        4) Exit Program
```

Figure 6-3: exception handled

```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(Assignment07) yilikals-MBP:Assignment07 tsedeytadesse$ python3 Assignment07.py

 Let's Calculate Your Body Mass Index (BMI)


        1) Calculate in Metric Unit
        2) Calculate in Imperial Unit
        3) Display BMI and Explanation
        4) Exit Program


What would you like to perform? Please enter [1 to 4] 1

What is your weight in kG? 60
What is your height in meters? 1.67

 Let's Calculate Your Body Mass Index (BMI)
        1) Calculate in Metric Unit
        2) Calculate in Imperial Unit
        3) Display BMI and Explanation
        4) Exit Program


What would you like to perform? Please enter [1 to 4] 3

Your Current BMI is 21.5
Your BMI falls within healthy weight range.


 Let's Calculate Your Body Mass Index (BMI)


        1) Calculate in Metric Unit
        2) Calculate in Imperial Unit
        3) Display BMI and Explanation
        4) Exit Program


What would you like to perform? Please enter [1 to 4] 4

Thank you! Check the CDC Website for More Information!
(Assignment07) yilikals-MBP:Assignment07 tsedeytadesse$
```

Figure 7: BMI script run in Terminal

Though the file is saved in the same folder as my script, (See ***Figure 8),*** it cannot be opened with a text editor. ***Figure 9*** shows the binary data opened in PyCharm. Though, the file could be opened, it cannot be read as plain text.

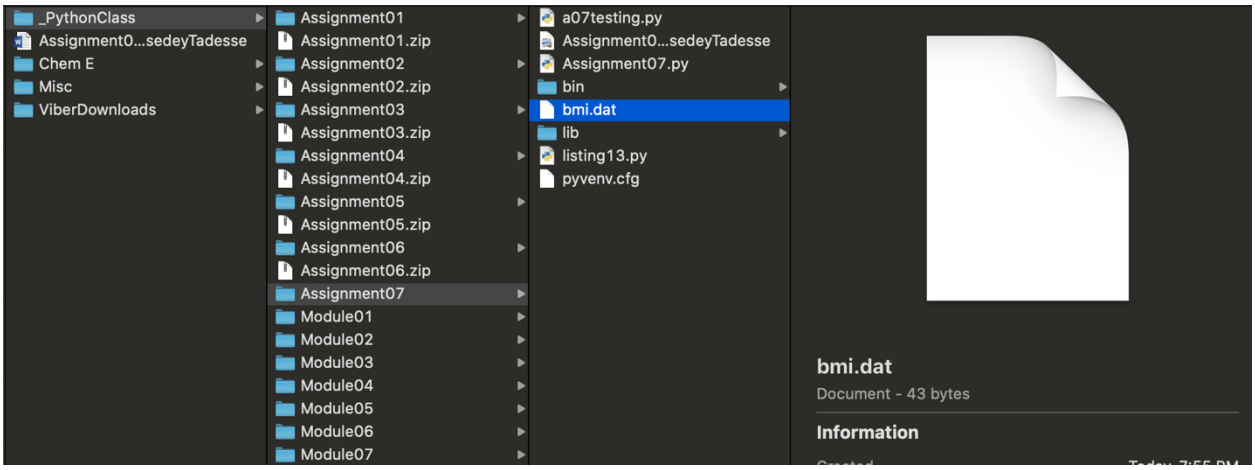

Figure 8: Binary file in the same folder as script

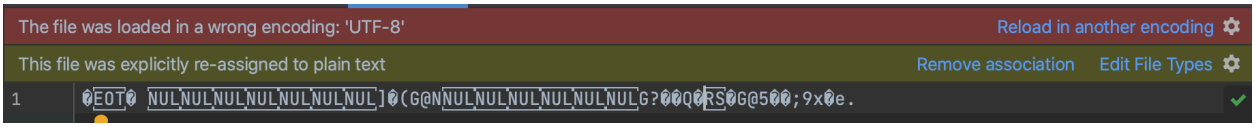

Figure 9: Binary file opened in PyCharm

Summary

Through the class notes, the text book, and the research I have done on my own, I learned about pickling and exception handling in Python. I also wrote simple math script that demonstrate how to add and load data by using the pickle module and exception error messages by handling them in various ways. Pickle.dump() and pickle.load() are the two main functions used to add data to and the read data from a file, respectively. The try statement along with the except clause are used to customize and handle error messages so that the program runs to completion even though errors occur in the program.