Name: Kimberly Harms
Date: 8/16/2023
Course: IT FDN 110 A Su 23: Foundations of Programming: Python
Assignment 07
https://github.com/kimtara/IntroToProg-Python-Mod07

Files & Exceptions

Introduction

This module focused on the use of files to save data and ways to handle errors. I learned to use the pickling module to pass complex data objects into and out of files, preserving the structure of the data rather than converting it to and from string data. In this way, I can save lists, dictionaries, and other objects for use in other Python scripts without losing any of the original information. In the other part of the module, I learned some ways to handle exceptions, or errors that occur during the execution of my code. I can handle exceptions with one or more try/except statements that allow the program to bypass problematic code, while returning to the user some valuable information about what went wrong.

Pickling in Python

Pickling is a process in Python that allows you to save data objects in binary language rather than as string data. When an object is pickled, it is serialized to a binary byte stream. The byte stream contains all the information needed to reconstruct the object in another script. It can be saved to a file, shared, or be unpickled or deserialized back to an object.

Pickling is useful because it can save complex data, such as dictionaries, tuples, lists, and more. The structure of the object data is preserved without losing any relevant information. In this way you can save and recover your data as it was, and it is intact in different Python sessions. Pickling is fast, easy to use and requires little code.

Pickling does have some drawbacks. It is specific to Python, so the objects you wish to preserve may not be readily understood by other languages. And although a pickled object is not easily read by a human, it is not secure and therefore vulnerable to malicious code if getting data from unknown/untrustworthy sources.

How to Pickle

To pickle an object in Python first import the pickle module. Then use the pickle.dump() method to serialize your data object and save it in a file. In the example shown in figure 1, a dictionary object with three keys/value pairs is assigned to dic_obj. We open a file in ‘wb’ or write binary mode because we are working with binary rather than string data. Many types of files can receive the byte stream data, including text, data, or pickle files, here we use one with .dat extension. We pass in two arguments when using the pickle.dump() method, the first indicating which object to save to the file, and the second indicating the file receiving the data. We then “dump” the object data into the specified file and close the file. To check our work, the data object and data type are printed out before saving the object to file (figure 3).

```
# ---------------------------------------------- #
# Title: pickletest.py
# Description: trying to figure pickling out
# ChangeLog: (Who, When, What)
# KHarms,8.20.2023,created script
# ---------------------------------------------- #
import pickle

# create data object -- this is a dictionary
dic_obj = {1: "The Lovecats", 2: "Boys Don't Cry", 3: "A Night Like This"}
print(dic_obj)
print(type(dic_obj))  # data type

# save data to file using pickle.dump method
test_file = open('test_data.dat', 'wb')  # open file in write binary mode
pickle.dump(dic_obj, test_file)  # put dic_obj in file
print('\n*Pickled*\n')
test_file.close()  # close the file
```

Figure 1. Pickling a dictionary in Python. Import the pickle module, open a file in 'wb' or write binary mode, and use the pickle.dump method to save the dictionary object in the file.

To retrieve an object from a file, we use the pickle.load() method. As above, we start by importing the pickle module (if not previously loaded) and open the file that contains the data object with which we wish to work, this time in 'rb' or read binary mode. We use the pickle.load() method with an argument indicating which file contains the object of interest, and pass the resulting object into the data variable. The results shown in figure 3 show a dictionary matching our original object.

```
# read data from file using pickle.load method
test_file = open('test_data.dat', 'rb')  # open file in read binary mode
data = pickle.load(test_file)  # get the pickled object from the file

# here is our object again!
print('Un-pickled!')
print(data)
print(type(data))  # data type of original object is preserved
```

Figure 2. Unpickling the preserved dictionary object from a file. The pickle module was imported previously. The file containing the object was opened in 'rb' or 'read binary' mode and the pickle.load() method used to retrieve the object from the specified file.

```
C:\Users\kimbe\Python311\python.exe "C:\Users\kimbe\OneDrive\Documents\_PythonClass\Module07
{1: 'The Lovecats', 2: "Boys Don't Cry", 3: 'A Night Like This'}
<class 'dict'>

*Pickled*

Un-pickled!
{1: 'The Lovecats', 2: "Boys Don't Cry", 3: 'A Night Like This'}
<class 'dict'>

Process finished with exit code 0
```

Figure 3. Running the pickling script. A dictionary object is printed, pickled, then unpickled from the file and printed again. Note that the data starts and ends as a dictionary type.

Exception Handling

Several types of errors can cause a script to crash or halt. Some of these errors are introduced by users giving input that is not expected or using the program in unanticipated ways. Errors that are detected during the execution of a statement are called exceptions.

Exception handling can catch errors and keep the program from crashing. Handling exceptions properly can allow the program to run to completion. It can also improve the communication of those exceptions back to the user so that the user can understand and perhaps even correct what went wrong, improving the user experience. Exception handling can also help the developer to identify the kinds of problems that can occur with the code.

Try and Except Clauses

In Python, try and except clauses are used to handle exceptions. The try clause is the block of statements between the try and except keywords. The except clause is the block of statements following the except keyword (figure 4). In the except block, the developer gets to control how the program responds to the exception. The pair works together in this way:

- A try clause is executed until the first exception is encountered.
- In no exception occurs, the except clause is skipped and the try clause is executed completely.
- If an exception occurs, the try clause is skipped and the exception clause is executed.
- Then the program continues with statements that follow the try/except block.

```
try:
    # put the code that may cause exceptions in the try clause
    # the program will attempt to execute this clause first
except Exception as e:
    # when an exception is found the except clause will execute instead
    # tell the program how you want it to respond to an exception
```

Figure 4. The basic try/except structure

In one try/except block, there may be more than one except clause to handle different types of exceptions. This allows you to anticipate and handle multiple exceptions and the unique response to each. In this case, any exception statements after the first exception clause to be executed will be skipped, so it is important to put any general exceptions last. In the example shown in figure 5, two different exception types are handled with different messages: a value error for when non-numeric characters are entered instead of digits (figure 6), a zero-division error for when there is an attempt to divide by zero (figure 7). A third exemption clause is included that can capture any other exception type and return some basic information about the error. This type of statement can help a developer test and improve their code by handling each likely exception appropriately.

```
# ------------------------------------------------ #
# Title: exceptiontest.py
# Description: script for showing exception handling
# ChangeLog: (Who, When, What)
# KHarms,8.20.2023,created script
# ------------------------------------------------ #

first_num = None
second_num = None
```

```
try:
    # put the code that may cause exceptions in the try clause
    # the program will attempt to execute this clause first

    # get user input
    first_num = float(input('Please enter a number: '))
    second_num = float(input('Please enter another number: '))

    # Process and display sum, difference, product, quotient
    print(f'The product of  {first_num} and {second_num} is', first_num * second_num)
    print(f'The quotient of {first_num} and {second_num} is', first_num / second_num)

except ValueError as e:
    # when an exception is found the except clause will execute instead
    # tell the program how you want it to respond to a ValueError exception
    print('There was an error -- please enter numeric characters only')

except ZeroDivisionError as e:
    # tell the program how you want it to respond to a ZeroDivisionError exception
    print('There was an error -- cannot divide by 0')

except Exception as e:
    print(e, e.__doc__, type(e), sep='\n')

input('Hit enter to exit program')
```

Figure 5. Script to calculate product and quotient of two numbers entered by the user. Exceptions raised when text entered or attempt to divide by zero handled in first two exception clauses. The third clause would capture any other type of exception and return some information as to the error.

```
C:\Users\kimbe\Python311\python.exe "C:\Users\kimbe\OneDrive\Documents\_PythonClass\Module07
Please enter a number: 1
Please enter another number: two
There was an error -- please enter numeric characters only
Hit enter to exit program
```

Figure 6. User friendly message returned when user enters non-numerical characters.

```
C:\Users\kimbe\Python311\python.exe "C:\Users\kimbe\OneDrive\Documents\_PythonClass\Module07
Please enter a number: 1
Please enter another number: 0
The sum of 1.0 and 0.0 is  1.0
The difference between 1.0 and 0.0 is 1.0
The product of  1.0 and 0.0 is 0.0
There was an error -- cannot divide by 0
Hit enter to exit program

Process finished with exit code 0
```

Figure 7. User friendly message when user enters zero for the second number.

Summary

In this assignment I explored some basic techniques for working with file and handling exceptions. Exceptions are errors that occur while executing code. Many such errors can be handled with try/except blocks that capture the error and return some information about that error. Exception handling allows the developer to control what happens when an error occurs and provides the user with more user-friendly messages to understand what went wrong. I also learned how to save complex data, such as lists or dictionaries, using pickling. Pickling is a process of serialization that allows the data to be converted to

a byte stream and saved to a file, preserving its structure and allowing it to be used intact in other Python scripts.

References

General References

Randall Root, https://youtube.com/playlist?list=PLfycUyp06LG9fZlIIqBrxLcNV4CR50HEX, module 7.

Michael Dawson, Python Programming for the Absolute Beginner, 3rd Edition (2010).

Bing Chat AI. Used for comparison to simple Google search.

Pickling References

Tutorials Point, Python Pickling, https://www.tutorialspoint.com/python-pickling.
A simple explanation with pros, cons, and common error codes.

Data Camp, Python Pickle Tutorial: Object Serialization, https://www.datacamp.com/tutorial/pickle-python-tutorial.
Gives a good introduction to object serialization and why it is useful.

Python Programming, Python Pickle Module for saving Objects by serialization, https://pythonprogramming.net/python-pickle-module-save-objects-serialization/.
Has an easy-to-follow video with demonstration and example of why pickling used in more advanced applications.

Geeks for Geeks, Understanding Python Pickling with example, https://www.geeksforgeeks.org/understanding-python-pickling-example/.
This resource identified by Bing Chat AI

Exception Handling References

Python documentation, Errors and Exceptions, https://docs.python.org/3/tutorial/errors.html.

Said van de Klundert, RealPython.com, Python Exceptions: An Introduction, https://realpython.com/python-exceptions/.
A great summary of the try-except structure and how to use it. Written in plain language with well-designed graphics to support.

Tutorials Point, Python – Exceptions Handling, https://www.tutorialspoint.com/python/python_exceptions.htm.

Geeks for Geeks, Python Exception Handling, https://www.geeksforgeeks.org/python-exception-handling/.
Nice summary of different, common types of exceptions.