Name: Joseph Géigel

Date: May 31, 2023

Course: Foundations of Programming: Python

Assignment: 07

GitHub URL: https://jgeigeluw.github.io/IntroToProg-Python-Mod07/

Working with Binary Files in Python: Pickling

Introduction

This paper introduces two Python concepts: error-handling using try-except statements, and the concept of pickling in the context of working with binary files. Previously we have discussed working with text files in Python by using a variety of methods, but this time we work with binary data. Text files store data, like strings or characters, while binary files can contain more complex data, such as executable files, images, audio, and more. The process of pickling and unpickling refers to the transformation of objects into byte streams, and vice versa. This paper will demonstrate that process by pickling and unpickling the data from the To-Do List created in the previous assignment.

Pickling and Unpickling

Pickling is the process in Python where object hierarchy is converted into a serialized byte stream, that is, a process to convert an object structure into a character stream that contains all the information necessary to reconstruct the object in another Python script. (GeeksForGeeks, 2018) The inverse operation of this process involves converting from a serialized byte stream into an object hierarchy, and it is called *unpickling.*

The Pickle Module

In essence, a module is a file containing Python code, and they can be imported into your script by using the keyword `import` followed by the name of the module. (Arne Hjelle, (n.d.)) One example of a module in Python is called `pickle`. The protocols required to perform the pickling and unpickling operations in Python are implemented by the `pickle` module. (Python.org, 2023) This module serializes objects in a binary format that is not readable by humans, but that can be stored or transmitted over a network.

Inside the Python `pickle` module there are four main methods: `dump, dumps, load` and `loads.` The first two are used for pickling operations, while the other two are used for unpickling. (Mastromatteo, n.d.) The `dump` method writes the pickled object to a file object. This method takes an object as its first parameter, and a file object as its second parameter, as in this example:

```
file = open("ExampleFile.dat", "wb")
pickle.dump(lstData,file)
file.close()
```

In this example, a binary file is opened in "wb" mode, which is a mode used to write data to a binary file. (Dawson, 2010) Then the dump method of the pickle module serializes the data in a list that is assigned to the lstData variable, and the data is written to the binary file, which is then closed. The data in lstData has been pickled.

With the list data pickled, we could use the load method to unpickle it and convert it to a readable format. The load method reads the pickled representation of an object from a file object and returns the object hierarchy specified. (Python.org, 2023) This method takes a file object as a parameter, like in this example. In the example, a binary file is opened in "rb" mode, which is a mode that reads from a binary file, and the contents of the binary file are unpickled by the load module. The unpickled data would be the list that we pickled with the dump method.

```
file = open("ExampleFile.dat", "rb")
pickle.load(file)
file.close()
```

Binary files

Serialized byte streams generated by pickling operations can be stored in binary files. These are files that are made up of characters that are not readable by humans, but that contain data that can be read by a computer. Binary files are designed to be compact and efficient files, and they can contain a variety of objects, such as: numbers, strings, tuples, lists, dictionaries, and more. There are many types of binary files, but in this paper, we will work with binary files that use the .dat extension.

Try-except: Error Handling

So far, we have written and discussed code that runs without any error handling strategies. When code is written in this way, and Python runs into an error, the program halts what it is doing and displays an error message. In Python, these are called exceptions, and they indicate that something exceptional occurred. (Dawson, 2010) However, it is possible to handle exceptions in a way that allow the program to continue to run, or at least in a way that provide more information about the type of error that occurred. One way of doing this is why try statements with except clauses. By using the try statement the programmer can isolate some code that has the potential of raising an exception, and then the except clause executes a block of code only if the exception is raised. (Dawson, 2010) If no exception occurs, the statements in the except clause code are simply skipped. This example shows the implementation of a try-except error handling approach:

```
try:
    number = int(input("Provide a number: "))
    break
except ValueError:
    print("Invalid input. Please provide a valid number.")
```

In this example, the code in the `try` statement prompts the user to enter a number, and if the user provides any other value that is not a valid number, an exception is raised. When the exception is raised, the program informs the user that the entered value is invalid, and that the user should try again. The `ValueError` exception is one of many built-in exceptions in Python, this particular one raises when an operation or function receives an argument with an inappropriate value. (Python.org, 2023)

Built-in exceptions

There are many built-in exceptions in Python, in the previous example we saw the `ValueError` exception, but many others can be raised under different circumstances. Table 1 provides a list of common Python built-in exceptions and their descriptions.

Exception	Description
ArithmeticError	Raised when an error occurs in numeric calculations
AttributeError	Raised when attribute reference or assignment fails
Exception	Base class for all exceptions
FloatingPointError	Raised when a floating point calculation fails
ImportError	Raised when an imported module does not exist
IndentationError	Raised when indentation is not correct
IndexError	Raised when an index of a sequence does not exist
KeyError	Raised when a key does not exist in a dictionary
KeyboardInterrupt	Raised when the user presses Ctrl+c, Ctrl+z or Delete
LookupError	Raised when errors raised cant be found
MemoryError	Raised when a program runs out of memory
NameError	Raised when a variable does not exist
OSError	Raised when a system related operation causes an error
OverflowError	Raised when the result of a numeric calculation is too large
RuntimeError	Raised when an error occurs that do not belong to any specific exceptions
SyntaxError	Raised when a syntax error occurs
SystemError	Raised when a system error occurs
SystemExit	Raised when the sys.exit() function is called
TypeError	Raised when two different types are combined
ValueError	Raised when there is a wrong value in a specified data type
ZeroDivisionError	Raised when the second operator in a division is zero

Table 1. Some of the most common built-in exceptions in Python. Adapted from (W3Schools, 2023).

Demonstration on Pickling and Binary Files

This week's script is relatively simple, and it aims to demonstrate how to pickle and unpickle files. To do this, we will use the To Do List that we populated with the script that we created last week. The To Do List is a text file that contains tasks and priorities that are populated by the user by using a script that takes the user's input and appends it to the text file as a new line. The resulting To Do List from our most recent run of the To Do List script looks like this:

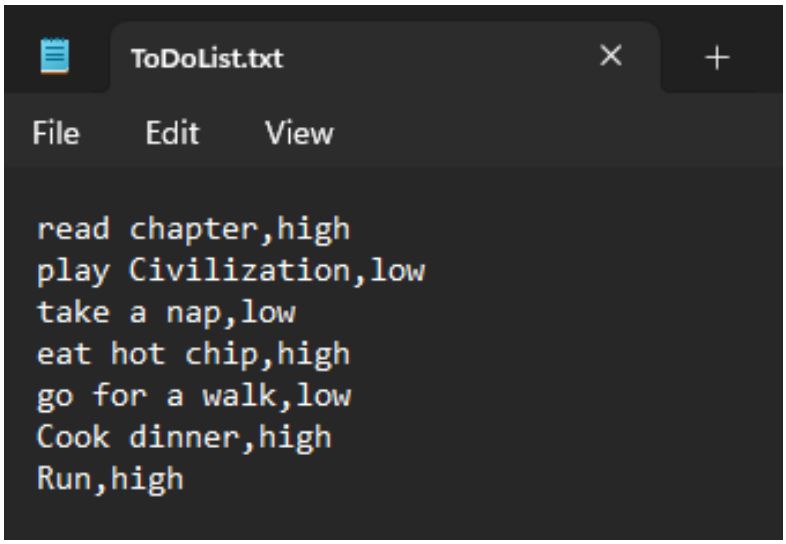

Figure 1. Simple text file showing the tasks and priorities in the to do list.

This time, we will take this information and serialize and deserialize it using the methods in the `pickle` module.

Using try-except error handling to import the module

The script begins with the implementation of a try-except error handling strategy when the `pickle` module is imported. The program encloses the import process in the `try` block of code, and prints "Imported the pickle module!" if the import is successful. In the event of an error during the import process, an `ImportError` would be raised. This is an exception that occurs when a module does not exist. For example, if the script tried to the import a module named `pikkle` instead of `pickle`, or if the module is not installed, the program would raise an exception and print: "ERROR: Could not import the module."

```
9  #Try to import the pickle module and raise an exception if pickle is not available.
10 try:
11     import pickle           #Try to import the pickle module
12     print ("Imported the pickle module!\n")
13 except ImportError:
14     print ("ERROR: Could not import the module.")  #Raise exception if import process fails
15     sys.exit()
```

Figure 2. This try-except error handling approach would display an import error if the pickle module fails to import.

Variables

After importing the `pickle` module, the script declares three variables that will be used in the processing functions. The first of these variables is the text file that will be read by the script to extract the data to be pickled, and it is named `filename`. This variable is assigned to the "ToDoList.txt" file that was generated by running the To Do List script, and that contains the values shown in figure 1. The second variable is an empty list that will store the values read by the script, this variable is named `list_from_file`. Finally, the last variable in this section is the `b_filename`, where the `b` stands for binary, as it represents the name and file extension of the binary file that will result from pickling the data.

```
18  #---------- Data ----------------------------------------------------------------
19  # Declare variables and constants
20  filename = "ToDoList.txt"       #Name of text file with original data
21  list_from_file = []             #List for lines that will be extracted from file.
22  b_filename = "AppData.dat"      #Name of binary file for pickling/unpickling
23
```

Figure 3. Declaring variables.

Processing

Processing in this script occurs with the execution of three custom functions, which are named:

- `save_data`
- `dump_data`
- `read_data`

These three functions are defined in the processing section of the script, as shown in figure 4.

```python
#---------- Processing  ----------------------------------------
1 usage
def save_data(filename, list_from_file):
    """ Reads data from a file into a list
    :param filename: (string) with name of file:
    :param list_from_file: (list) you want filled with file data:
    :return: (list) of rows extracted from file
    """
    with open(filename) as file:
        for row in file:
            list_from_file += [row.strip()]
    file.close()
    return list_from_file

1 usage
def dump_data(b_filename, list_from_file):
    """ Dumps data from a list into a binary file
    :param b_filename: (string) with name of the binary file:
    :param list_from_file: (list) filled with file data
    """
    b_file = open(b_filename,"ab")
    pickle.dump(list_from_file,b_file)
    b_file.close()

1 usage
def read_data(b_filename):
    """ Loads data from a binary file into a list
    :param b_filename: (string) with name of binary file:
    :return: (list) of items loaded from binary file
    """
    b_file = open(b_filename, "rb")
    data = pickle.load(b_file)
    b_file.close()
    return data
```

Figure 4. Defining functions to read, serialize and deserialize data.

The `save_data` function takes two parameters: the name of the file that contains the data to be read and pickled, and the empty list object that will be populated with the lines from the text file, which in this case is "ToDoList.txt". This function uses the `open()` method to open the file, and then implements a `for` loop to iterate over the lines and split them into separate items of a list. Then, the file is closed, and the function returns a list with all the items obtained from the text file.

The second function is `dump_data`. This function takes two parameters: the name of a binary file that will be created to pickle data, and the list of items returned by the `save_data` function. The `dump_data` function opens the binary file, which in this case is named "AppData.dat" in `"ab"` mode, which is a mode that appends data to a binary file. Then, the function uses the `pickle.dump()` method to serialize all the items in the list returned by the first function, and stores the serialized (pickled) values in the "AppData.dat" binary file.

The last function defined is `read_data`. This function unpickles all the data contained in the binary file that was created and populated by `dump_data`. To do that, the function takes one parameter, which is the name of the binary file, and uses the `pickle.load()` method to deserialize the data into a list, which is returned by the function for display in the presentation section of the script.

Presentation

The presentation section shown in figure 5 contains the execution of the three functions defined in the processing section.

```
56  #---------- Presentation  ------------------------------------------------------------ #
57  save_data(filename,list_from_file)      #Read data from ToDoList.txt
58  print('Extracted data from text file!\n')
59
60  dump_data(b_filename, list_from_file)   #Pickle data from ToDoList.txt into AppData.dat
61  print ('Pickled the data!\n')
62
63  data = read_data(b_filename)            #Unpickle data into list
64  print ('Unpickled the data!\n')
65  print("This is the data that was pickled and unpickled: " +'\n')
66  print (data,type(data))                 #Print list of unpickled items
```

Figure 5. Executing the custom functions for pickling and unpickling.

When the `save_data` function is executed, the script reads the text file, populates a list with its contents, and then displays the message: “Extracted data from text file!”. Then the `dump_data` function pickles the extracted data into the “AppData.dat” file, and displays a message that reads: “Pickled the data!”. Lastly, the data is unpickled by the `read_data` function, and a message that reads: “Unpickled the data!” is displayed, followed by the list containing the unpickled items.

Output Example

Execution of this script in PyCharm and Command Prompt is straightforward. The script runs, progress messages are printed as the functions are executed, and a final list of task and priorities is displayed back to the user after pickling and unpickling. Figure 6 shows output from running the script in PyCharm, and figure 7 shows output in Command Prompt.

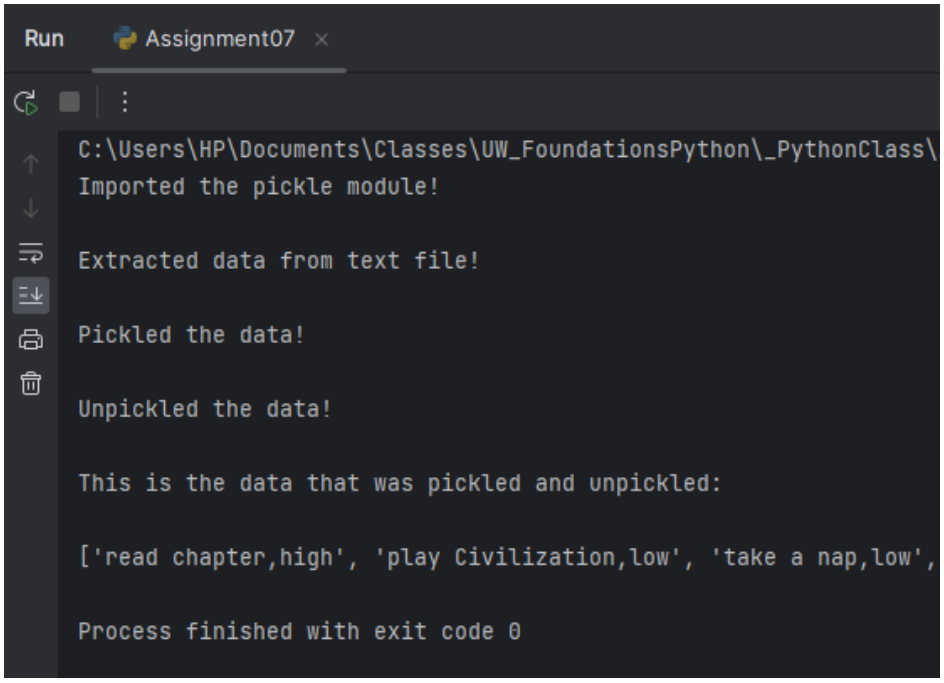

Figure 6. Output in PyCharm.

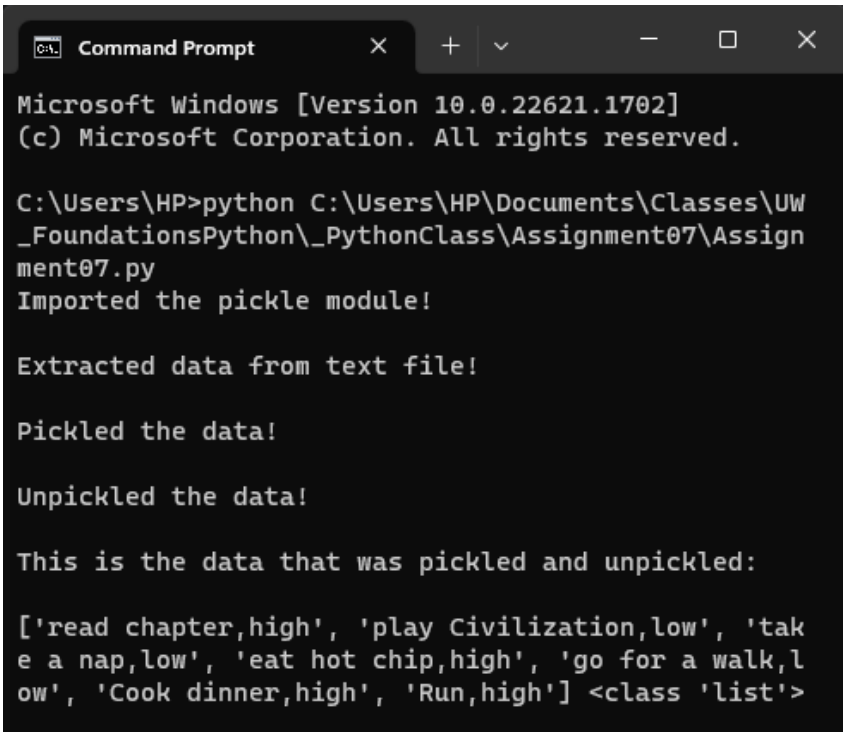

Figure 7. Output in Command Prompt.

Summary

In this paper we explored the concepts of serialization and deserialization in Python by implementing the methods in the `pickle` module. Serialization refers to the concept of

converting objects into byte streams that cannot be read by humans, but that can be understood by computers. Serialized data can be stored in binary files that are efficient and transmittable over a network, and many different kinds of complex data can be stored in a binary file. The `pickle` module uses the `dump` and `load` methods to pickle and unpickle the data. We also explored the concept of error handling by using `try` statements and `except` clauses, and the variety of built-in exceptions that exist in Python to handle different kinds of errors.

References

Arne Hjelle, G. ((n.d.)). *Python import: Advanced Techniques and Tips*. Retrieved from Real Python: https://realpython.com/python-import/#modules

Dawson, M. (2010). *Python Programming for the Absolute Beginner* (Third Edition ed.). Boston, MA, United States of America: Course Technology PTR.

GeeksForGeeks. (2018, November 13). *Understanding Python Pickling*. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/understanding-python-pickling-example/

Mastromatteo, D. (n.d.). *The Python pickle Module: How to Persist Objects in Python*. Retrieved from Real Python: https://realpython.com/python-pickle-module/

Python.org. (2023). *Built-in Exception*. Retrieved from Python.org: https://docs.python.org/3/library/exceptions.html

Python.org. (2023, May 30). *pickle — Python object serialization*. Retrieved from Python.org: https://docs.python.org/3/library/pickle.html

W3Schools. (2023). *Python Built-in Exceptions*. Retrieved from W3Schools: https://www.w3schools.com/python/python_ref_exceptions.asp