**Introduction.**

In module seven we begin by learning to work will files using pickling and structured exception handling. We will also be learning how to create more complex GitHub web pages. This new web page will include basic commands in a language called “markdown”.

***Week 7 Labs***

**Lab 7.1**

**T**his lab was relatively easy to do. Create some data, save data to file, and read the data using pickling. First, we create a string and list for creating the data. Next save the data to the file by appending the binary (that is what “ab” stands for). pickle.dump is used to serialize(convert into a byte stream) Python objects and write them to a file or a binary stream. Serialization is the process of converting complex data structures, like dictionaries or objects, into a format that can be stored and later reconstructed. You can see each one of these steps in the example below Figure 6.1.

A screenshot of a computer program

Description automatically generated

***(Figure 6.1)***

Next we have the script read the data from the file using pickling again. We simply have it open the file and read the binary text by inputing “rb”. Shown in figure 6.2.

A screenshot of a computer program

Description automatically generated

***(Figure 6.2)***

The final step of this lab saves the list object to a binary file. Then the script loads the data by using the read\_data\_from\_file that I defined in the first block of the script and I present that data back to the user by simply making a print command for the loaded data. In figure 6.3 you can see how this was executed. This concludes lab 7.1.

A screenshot of a computer program

Description automatically generated

***(Figure 6.3)***

**Exception handling**

An exception in python is an unexpected event that occurs during program execution. Errors that occur at runtime are called ***exceptions*** or ***logical errors****.* These occur when; try to open a file that does not exist, try to didvide a number by zero, or try to import a module that does not exist.

* The try…except block is used to handle exceptions in python.

try:

# code that may cause exception

except:

# code to run when exception occurs

For each try block, there can be zero or more except blocks. Multiple except blocks allow us to handle each exception differently. The argument type of each except block indicates the type of exception that can be handled by it. For example,

try:

even\_numbers = [2,4,6,8]

print(even\_numbers[5])

except ZeroDivisionError:

print("Denominator cannot be 0.")

except IndexError:

print("Index Out of Bound.")

# Output: Index Out of Bound

Here, we are trying to access a value to the index 5. Hence, index error exception occurs



The ZeroDivisionError exception is skipped and the IndexError is executed.

* **Try with else clause.**

In some situations you might want to run a certain block of code if the code block inside try runs without any errors.

For these cases, you can use the optional else keyword with the try statement.

Here’s an example:

A screenshot of a computer

Description automatically generated

If we pass an odd number.

A black text on a white background

Description automatically generated

If we pass an even number, the reciprocal is computed and displayed.

A black text on a white background

Description automatically generated

However, if we pass a 0, we get the ZeroDivisionError as the code block inside the else is not handled by preceding.

A computer error message

Description automatically generated

* **Python try…finally**

In Python, the finally block is always executed no matter whether there is an exception or not .

The finally block is optional. For each try block, there can be only one finally block.

Lets see an example,

A screenshot of a computer code

Description automatically generated

Output

A close up of a message

Description automatically generated

In this example, we are dividing a number by 0 inside the try block. Here, this code generates an exception. The exception is caught by the except block, then the finally block is executed.

To learn more on exception handling visit: [Python Exception Handling (With Examples) (programiz.com)](https://www.programiz.com/python-programming/exception-handling)

**Pickling**

Picklingis the process whereby a python object hierarchy is converted into a byte stream. Unpickling is the process by which original Python objects are retrieved from the string representation i.e., from the pickle file. It’s a process that involves serializing (convering) a python object into a format that can be stored or transmitted, and later deseralized (reconstructed) to obtain the original object.

The term “pickling” comes from the analogy with the process of preserving vegetables by pickling them in a jar. In python, pickling allows you to preserve the state of complex data structures, including dictionaries, lists, and classes, so they can be saved to afile, database, or storage medium.

Python provides the built-in “pickle” module to facilitate pickling and unpickling.

Here I provided some script as an example of pickling:

A screenshot of a computer program

Description automatically generated

**Assignment 07**

In assignment seven the task was to demonstrate how pickling and structured error handling work. I kept my code really simple so it would be clear on how exactly each process works.

1. The first step is to define my data structure. I made some simple data consisting of name, age, and city you live in. You can see this below in figure 7.1.

A screenshot of a computer

Description automatically generated

(Figure 7.1 )

1. Next, I write the data to a file using the pickle method. When the file opens it will write binary, hence the ‘wb’. The ‘w’ opens the file in write mode and the ‘b’ indicates the file should be treated as a binary file. See figure 7.2 for an example. I begin to add in some error handling, often referred to as “try-except “ blocks.

A screen shot of a computer

Description automatically generated

(Figure 7.2)

1. The third step is to unpickle and read the binary file. Similar to when you open and write a binary file, this time it will open and read the binary final. That is what the ‘rb’ stands for in the script below in figure 7.3. Again, I use structured error handling in the form of “try-except” block.

A screen shot of a computer program

Description automatically generated

(Figure 7.3)

1. The final section of the code encapsulates the main logic of the script. It initializes the filename, saves data to a file using pickling, and then loads data from the file using unpickling. The code inside this block will only execute when the script is run directly, ensuring that these operations are performed when the script is intended to be used as a standalone program. If this script is imported into another script, this block won’t be executed, allowing the functions and data to be used in a modular way. See figure 7.4 for example.

A screenshot of a computer code

Description automatically generated

(Figure7.4)

Here is what the text file looks like when writing binary code using pickling.

A screenshot of a computer

Description automatically generated

Just to make sure the script ran as it should, I ran the script in the OS shell also. Below is my example of how it ran.

A screenshot of a computer program

Description automatically generated

**Summary**

Pickles are a very useful tool for simplifying your script and storing data. Using pickles for serialization and deserialization in python can streamline data storage and transmission. However, it’s essential to incorporate robust exception handling practices to gracefully handle errors that can occur during pickling and unpickling.