Susan Paulen

December 1, 2020

IT FDN 110 A

Assignment 07

GitHub:

# Python Pickling Module & Error Handling

# Introduction

The two features introduced this week provide ways to make our Python code more robust and allow us to save/send serialized and deserialized data. The Pickle module can be imported to Python code in order to save to binary data files or to unpack binary data files. Error Handling can be done in a few different ways to capture errors and make a program more user-friendly.

# Pickle Module

Using the Pickle Module in Python requires importing Pickle via the command: import Pickle. This module contains functions to change an object to binary format as well as unpack a binary object to text.

Some good websites to read about Pickling and Serialization are:

* [Don’t Fear the Pickle: Using pickle.dump and pickle.load | by Manu Kalia | Better Programming | Medium](https://medium.com/better-programming/dont-fear-the-pickle-using-pickle-dump-and-pickle-load-5212f23dbbce)
* [The Python pickle Module: How to Persist Objects in Python – Real Python](https://realpython.com/python-pickle-module/#:~:text=%20The%20Python%20pickle%20module%20basically%20consists%20of,%28bytes_object%2C%20%208%20%2C%20fix_imports%3DTrue%2C%20encoding%3D%22ASCII%22%2C...%20More%20)
* [Understanding Python Pickling with example - GeeksforGeeks](https://www.geeksforgeeks.org/understanding-python-pickling-example/)
* [Serialization - Wikipedia](https://en.wikipedia.org/wiki/Serialization)

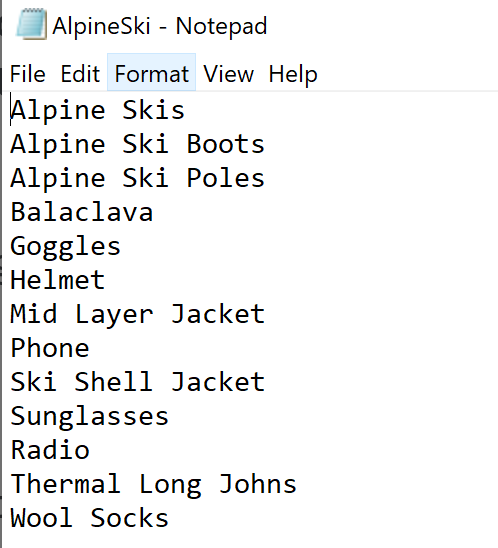
Serialization is most often used to:

1. Detect changes in data over time
2. Transfer data
3. Store data
4. Distribute objects

# Part 1: Pickle Dump

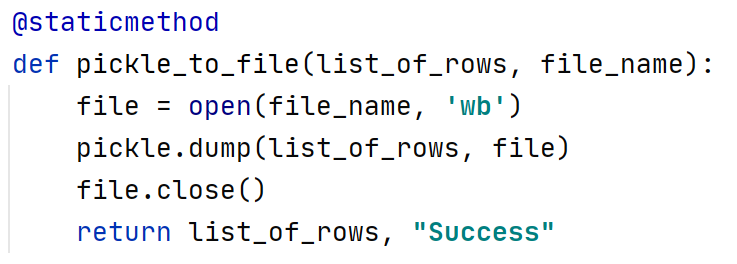
For this assignment, I created a text file called AlpineSki.txt. The text file appears like this:

Figure 1 AlpineSki.txt File Contents



This is an alpine ski trip packing list. This file will be “pickled” (serialized) into a binary file called: AlpineSki\_Pickled.txt when the following method is called from class Processor in the file AlpineSkiing.py:

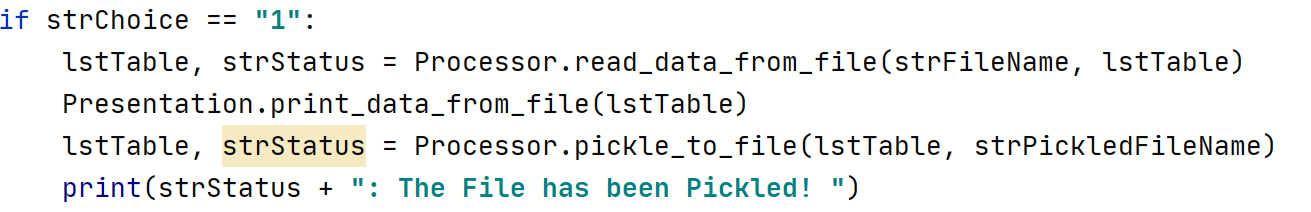
Figure 2 pickle\_to\_file method



Note that the code uses the pickle function dump to save the list (previously read from AlpineSki.txt) to a file.

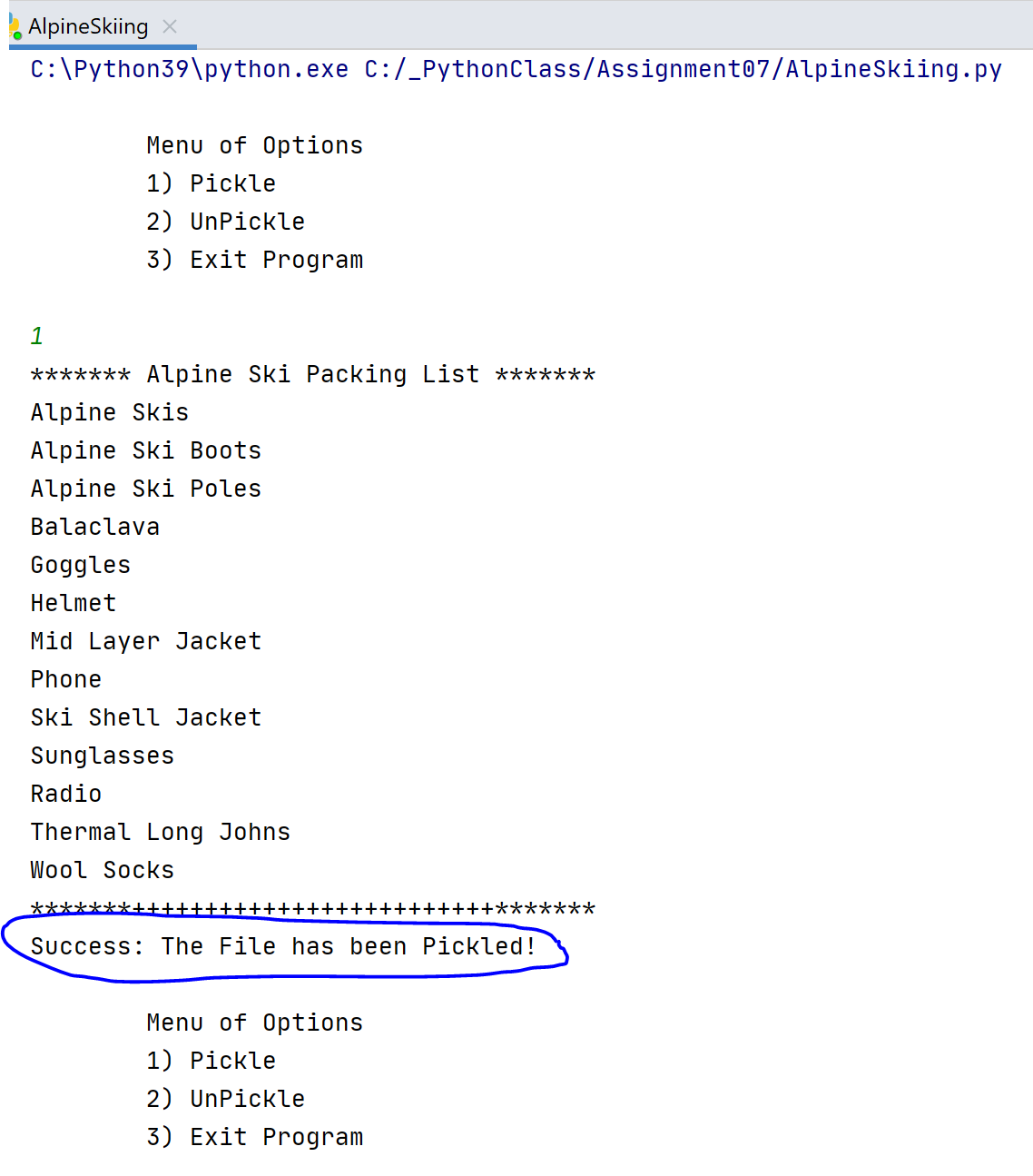
This code is chosen by the user by selecting 1 from the menu of options per the code below. Note that the arguments lstTable and strPickledFileName (hardcoded to “AlpineSki\_Pickled.txt”) are passed to pickle\_to\_file as the parameters list\_of\_rows and file\_name.

Figure 3 Code Executed If User Chooses Option 1 (Pickle)



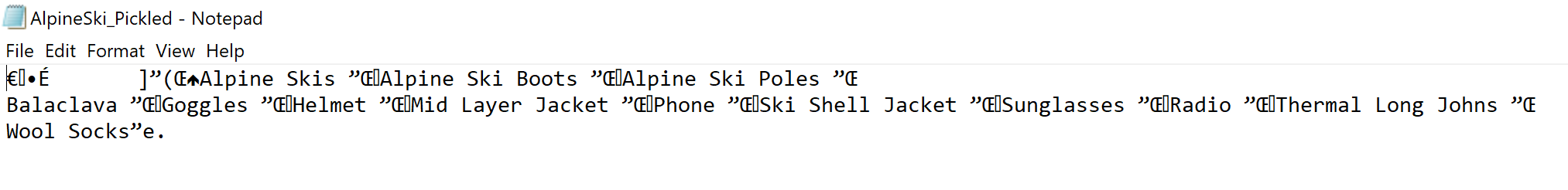
When the program runs, selecting 1 returns this result:

Figure 4 User Chooses Option 1 (Pickle)



The contents of the Pickled File “AlpineSki\_Pickled.txt” are:

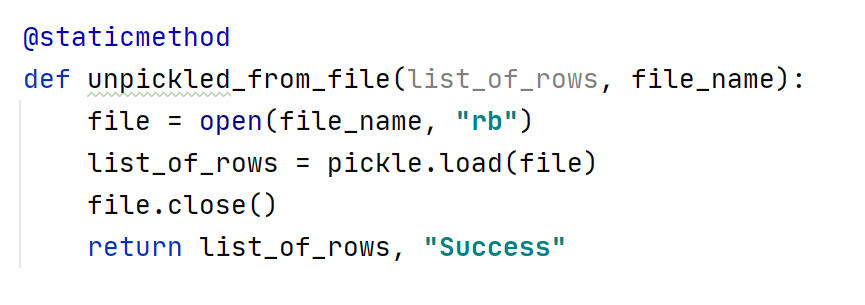
Figure 5 AlpineSki\_Pickled.txt Contents



# Part 2: Pickle Load

Any pickled (serialized) file can be unpickled (deserialized) to a list when the following method is called from the class Processor in the file AlpineSkiing.py:

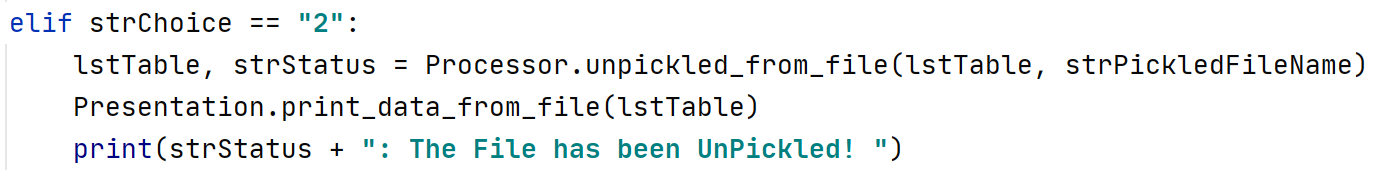
Figure 6 unpickled\_from\_file method



Note that the code uses the pickle function load to save to a list.

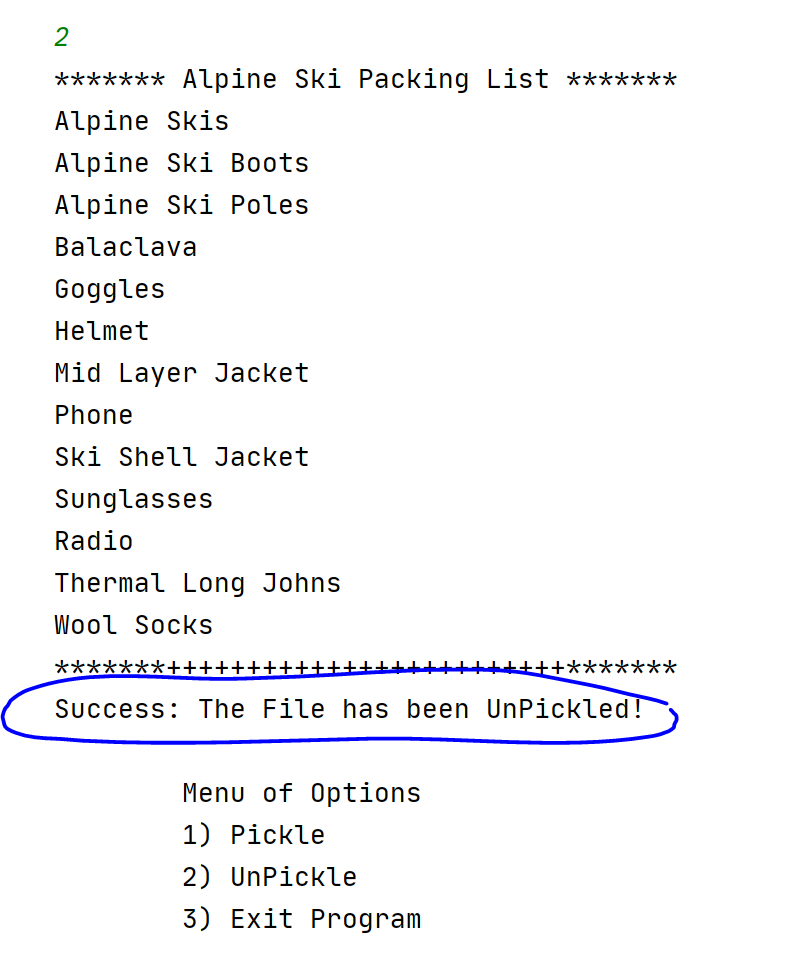
This code is chosen by the user by selecting 2 from the menu of options per the code below. Note that the arguments lstTable and strPickledFileName (hardcoded to “AlpineSki\_Pickled.txt”) are passed to unpickled\_from\_file as the parameters list\_of\_rows and file\_name.

Figure 7 Code Executed If User Chooses Option 2 (Unpickle)



When the program runs, selecting 2 returns this result:

Figure 8 User Chooses Option 2 (UnPickle)



# Error Handling

Error Handling in Python can be grouped into the following 4 categories:

1. Try-Except/Try-Except-else Clause

This clause is useful for capturing unknown code problems or issues where a file may not always be available.

1. Raise Exception with Arguments

Exceptions can be raised anywhere they are needed. Once an exception has been raised, execution will stop, and the exception will go up the call stack until the exception is handled.

1. Create Custom Exceptions in Python

Custom exceptions are created by the developer in the code as a user defined function or class.

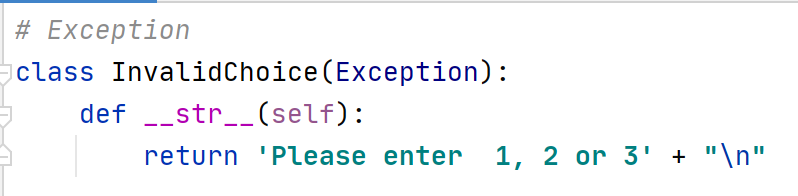
1. Python Built-in Exceptions

Built-in Exceptions are either generated by the interpreter or built-in functions. They usually have additional information that details an error’s cause such as a code or a description. Examples of built-in exceptions are: FileNotFoundError, ZeroDivisionError and ImportError.

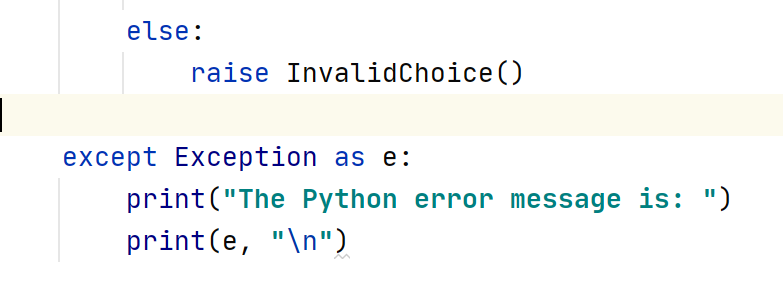
AlpineSki.py makes use of 3 of these error handling methods if anything other than 1, 2 or 3 is entered as an option.

1. If something other than 1, 2 or 3 is entered, the code falls through to the else statement which raises a custom class.
2. The user defined exception InvalidChoice is raised when anything other than 1, 2 or 3 is entered as an option. This is a custom exception. Notice that it uses the base Exception class as the parent. In this way, the user-defined exception class raises exceptions as any other exception class does.

Figure 9 Custom Exception

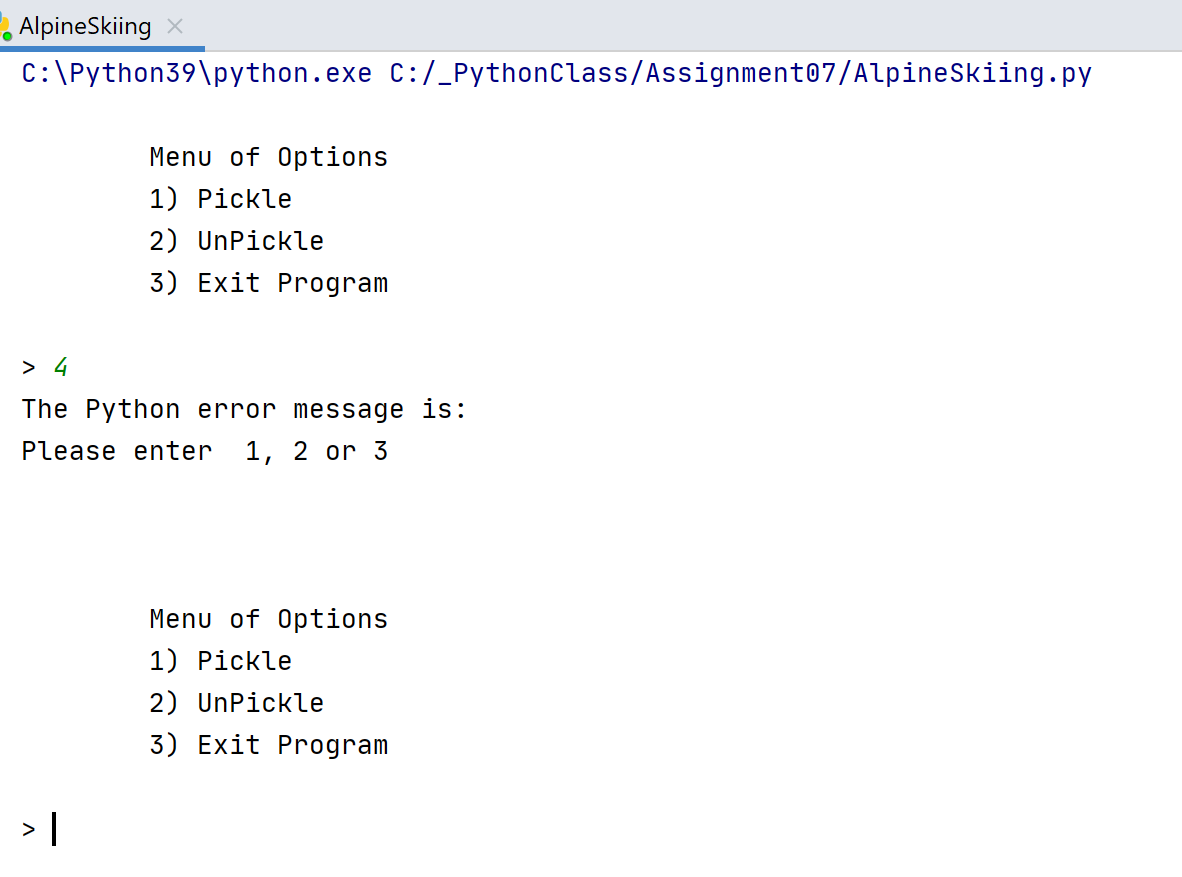


1. Note that the except clause invokes the Exception parent which incorporates the custom exception message returned from InvalidChoice as “e”.



Run Time Error Handling in AlpineSki.py:

Figure 7 Run-Time Error Handling of an Invalid Option



Some good websites to read about Python Exception Handling are:

* [Python Exception Handling: A Quick Tutorial for Learning (techbeamers.com)](https://www.techbeamers.com/python-exception-handling/)
* [Exception Handling in Python | Pythontic.com](https://pythontic.com/language/exception%20handling/introduction#:~:text=Exception%20Handling%20in%20Python%3A%201%20Python%20has%20built-in,code%20for%20handling%20the%20exception.%20More%20items...%20)
* [Python - Exceptions Handling - Tutorialspoint](https://www.tutorialspoint.com/python/python_exceptions.htm)
* [Built-in Exceptions — Python 3.9.1rc1 documentation](https://docs.python.org/3/library/exceptions.html)

# Summary

Error Handling is essential in every program but most especially in programs where errors can be expected and even predicted; such as user entry and data formats that can be changed by other systems or developers.

The Pickle Module can be useful for serialization and deserialization within a program. However, it has security issues so should be used only for internal program purposes and not for deserialization of external binary files.

# Appendix A - Python Code

*# ---------------------------------------------------------------------------- #  
# Title: An Alpine Skiing Packing List  
# Description: Demonstration of Pickling and Exception Handling  
# ChangeLog: (Who,When,What):  
# SPaulen, 12/01/2020, Created  
# ---------------------------------------------------------------------------- #  
# Import Pickle Module*import pickle  
  
*# Declare Variables and Classes*strFileName = **"AlpineSki.txt"** *# The name of the text data file*strPickledFileName = **"AlpineSki\_Pickled.txt"** *# The name of the binary data file*lstTable = [] *# A list that acts as a 'table' of rows*intChoice = 0 *# Captures the user option selection*strStatus = **""** *# Captures the status of an processing functions  
  
  
# Exception*class InvalidChoice(Exception):  
 def \_\_str\_\_(self):  
 return **'Please enter 1, 2 or 3'** + **"**\n**"***# Processing*class Processor:  
 @staticmethod  
 def read\_data\_from\_file(file\_name, list\_of\_rows):  
 list\_of\_rows.clear() *# clear current data* try:  
 file = open(file\_name, **"r"**)  
 for row in file:  
 if **"**\n**"** in row:  
 row = row.replace(**"**\n**"**, **" "**)  
 list\_of\_rows.append(row)  
 file.close()  
 return list\_of\_rows, **"Success"** except:  
 return None, **"Success"** @staticmethod  
 def pickle\_to\_file(list\_of\_rows, file\_name):  
 file = open(file\_name, **'wb'**)  
 pickle.dump(list\_of\_rows, file)  
 file.close()  
 return list\_of\_rows, **"Success"** @staticmethod  
 def unpickled\_from\_file(list\_of\_rows, file\_name):  
 file = open(file\_name, **"rb"**)  
 list\_of\_rows = pickle.load(file)  
 file.close()  
 return list\_of\_rows, **"Success"***# Presentation*class Presentation:  
 @staticmethod  
 def print\_data\_from\_file(list\_of\_rows):  
 print(**"\*\*\*\*\*\*\* Alpine Ski Packing List \*\*\*\*\*\*\*"**)  
 for row in list\_of\_rows:  
 print(row)  
 print(**"\*\*\*\*\*\*\*+++++++++++++++++++++++++\*\*\*\*\*\*\*"**)  
  
 @staticmethod  
 def print\_menu\_tasks():  
 print(**'''  
 Menu of Options  
 1) Pickle  
 2) UnPickle  
 3) Exit Program  
 '''**)  
  
  
*# Main Body*while True:  
 try:  
 Presentation.print\_menu\_tasks()  
 intChoice = input(**"> "**)  
 if intChoice == 1:  
 lstTable, strStatus = Processor.read\_data\_from\_file(strFileName, lstTable)  
 Presentation.print\_data\_from\_file(lstTable)  
 lstTable, strStatus = Processor.pickle\_to\_file(lstTable, strPickledFileName)  
 print(strStatus + **": The File has been Pickled! "**)  
 elif intChoice == 2:  
 lstTable, strStatus = Processor.unpickled\_from\_file(lstTable, strPickledFileName)  
 Presentation.print\_data\_from\_file(lstTable)  
 print(strStatus + **": The File has been UnPickled! "**)  
 elif intChoice == 3:  
 print(**"Goodbye!"**)  
 break  
 else:  
 raise InvalidChoice()  
  
 except Exception as e:  
 print(**"The Python error message is: "**)  
 print(e, **"**\n**"**)