Name: Amruth D

Date: 02ndJune2023

Course: IT FDN 110 A

Assignment 07

Python Script to Demonstrate Pickle Module and Structured Error Handling

Introduction

In this week, we will understand exception handling and pickling in python. For the assignment this week we had to write a script to demonstrate the Pickle module and Structured Error Handling.

Exception Handling in Python

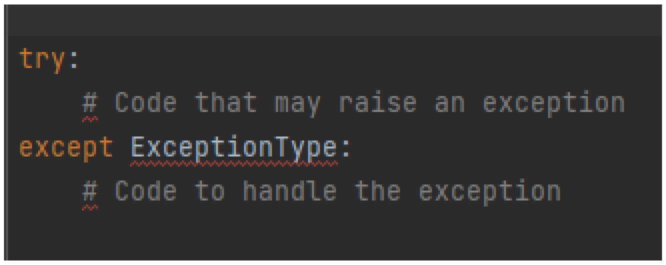
Exception handling in Python is a mechanism that allows we to handle and manage errors that occur during the execution of a program. It helps in controlling the flow of the program and gracefully handling errors without causing the program to terminate abruptly. Python provides a robust and flexible exception handling framework to catch and handle different types of exceptions

Types of Exceptions

Exceptions in Python can be of different types, such as SyntaxError, TypeError, ValueError, FileNotFoundError, KeyError, etc. Each type of exception corresponds to a specific error condition that can occur during program execution

Try-Except Block

The try-except block is used to catch and handle exceptions. The code that may potentially raise an exception is placed inside the try block. If an exception occurs, the code inside the corresponding except block is executed.

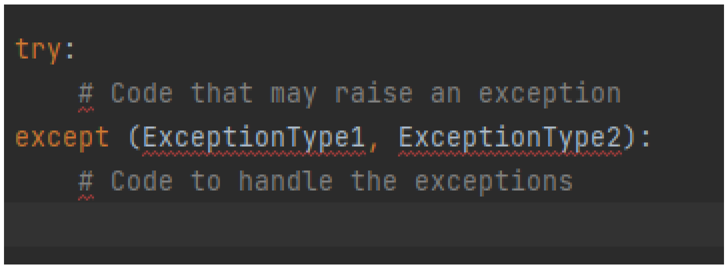


```
try:
    # Code that may raise an exception
except ExceptionType:
    # Code to handle the exception
```

We can have multiple except blocks to handle different types of exceptions.

Catching Multiple Exceptions

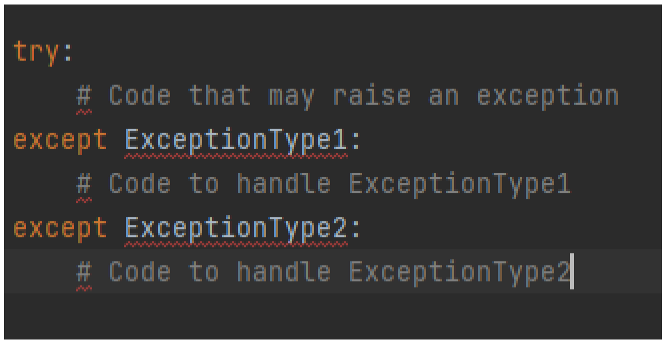
We can catch multiple exceptions using a single except block by specifying multiple exception types inside parentheses or by using multiple except blocks.



```
try:
    # Code that may raise an exception
except (ExceptionType1, ExceptionType2):
    # Code to handle the exceptions
```

Handling Specific Exceptions

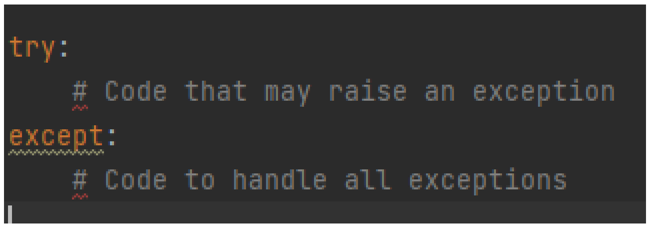
To handle specific exceptions individually, we can use multiple except blocks, each handling a specific exception type.



```
try:
    # Code that may raise an exception
except ExceptionType1:
    # Code to handle ExceptionType1
except ExceptionType2:
    # Code to handle ExceptionType2
```

Handling All Exceptions

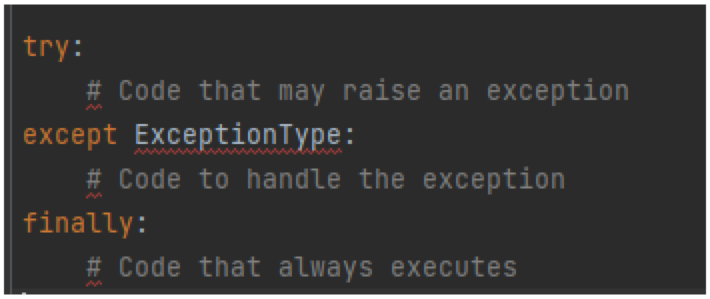
We can use a generic except block without specifying any exception type to catch all exceptions. However, it is recommended to handle specific exceptions whenever possible.



```
try:
    # Code that may raise an exception
except:
    # Code to handle all exceptions
```

Finally Block

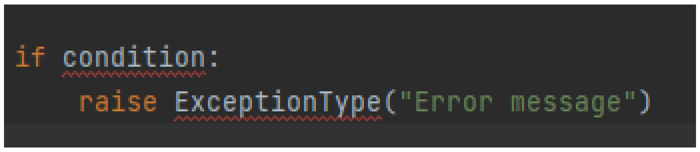
The finally block is used to specify code that should be executed regardless of whether an exception occurred or not. It is commonly used for cleanup operations like closing files or releasing resources.



```python
try:
    # Code that may raise an exception
except ExceptionType:
    # Code to handle the exception
finally:
    # Code that always executes
```

Raising Exceptions

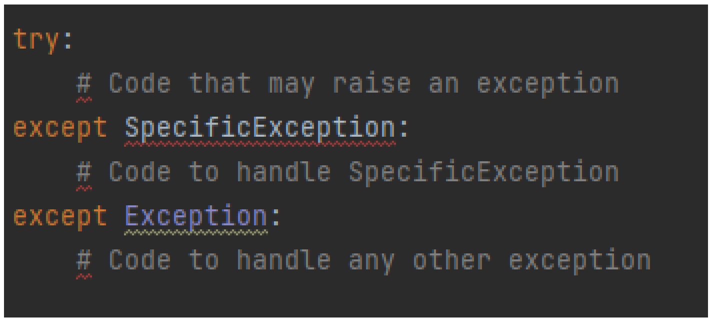
We can manually raise exceptions using the raise statement. This allows we to generate and throw exceptions when certain conditions are met.



```python
if condition:
    raise ExceptionType("Error message")
```

Exception Handling Hierarchy

The Python exception classes are organized in a hierarchy. We can handle exceptions at different levels of the hierarchy, from more specific exceptions to more general ones. The general Exception class should be handled at the end, as it can catch any exception that is not explicitly handled by more specific except blocks.



```python
try:
    # Code that may raise an exception
except SpecificException:
    # Code to handle SpecificException
except Exception:
    # Code to handle any other exception
```

Exception handling in Python provides a way to gracefully handle errors and failures, enabling we to write robust and reliable programs. It is essential for writing defensive code, handling unexpected scenarios, and ensuring the stability of your application.

Pickling in Python

Pickling in Python refers to the process of serializing Python objects into a binary representation, which can be saved to a file, transmitted over a network, or stored in a database. The pickle module in Python provides a convenient way to serialize and deserialize objects, allowing them to be easily stored and retrieved.

Serializing Objects

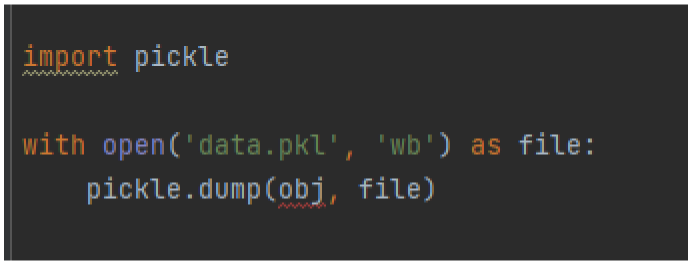
Pickling allows you to convert Python objects into a byte stream. The objects can be of any built-in data type (e.g., integers, floats, strings, lists, dictionaries) or custom-defined classes.

The pickle Module

The pickle module is a standard library module in Python that provides functions for pickling and unpickling objects. It uses a binary format to serialize objects.

Pickle Dumping

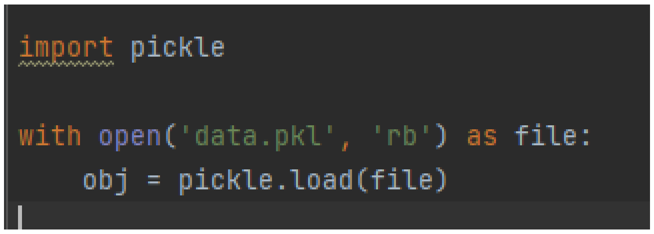
The pickle.dump(obj, file) function is used to pickle an object and save it to a file. The obj parameter represents the object to be pickled, and the file parameter represents the file object where the pickled data will be stored.



```
import pickle

with open('data.pkl', 'wb') as file:
    pickle.dump(obj, file)
```

Pickle Loading

The pickle.load(file) function is used to unpickle an object from a file. The file parameter represents the file object from which the pickled data will be read.



```
import pickle

with open('data.pkl', 'rb') as file:
    obj = pickle.load(file)
```

Pickling and Unpickling Multiple Objects

It is possible to pickle multiple objects successively and then unpickle them in the same order using the pickle.dump() and pickle.load() functions. The order of pickling and unpickling should match.

Limitations

While pickling is a powerful feature, there are some limitations to keep in mind. Not all objects can be pickled, including objects that have file handles open, network connections, or certain types of Python functions.

Security Considerations

Pickle files should be treated with caution, as unpickling data from an untrusted source can lead to security vulnerabilities. Only unpickle data from trusted sources to avoid potential risks.

Alternatives to Pickling

If you need to serialize objects in a more interoperable format or share data with non-Python applications, you can consider using alternative serialization formats like JSON, YAML, or Protocol Buffers.

Pickling provides a convenient way to serialize and store Python objects. It is commonly used for caching, storing application state, inter-process communication, and data persistence. However, it is important to be mindful of the security implications and limitations of pickling when working with sensitive or untrusted data.

GitHub Webpage

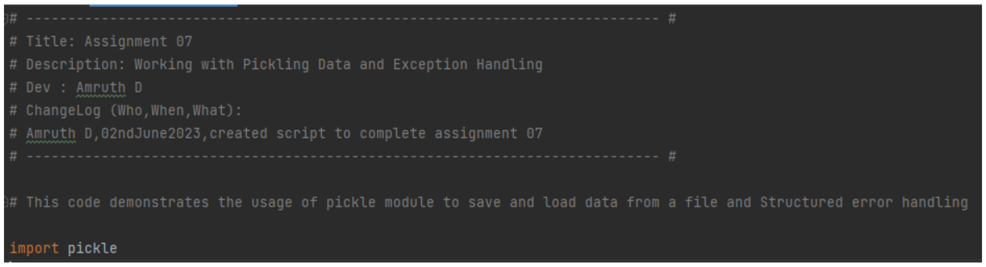
GitHub not only serves as a code repository but also provides the capability to host simple webpages. These webpages have the extension ".md" and follow a specific style guide to achieve a desired appearance. They can serve as front pages for repositories hosted on GitHub.

To access the GitHub webpage created for the assignment, please visit the following link: **https://amruthd-92.github.io/IntroToProg-Python/**

Writing The Assignment Script

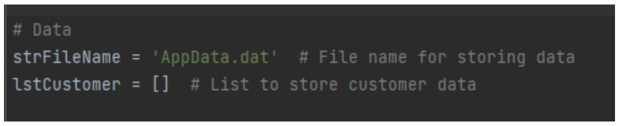
For the assignment 07 we had to write a script to demonstrates the usage of the pickle module to save and load data from a file, and structured error handling to handle invalid input values for customer ID and name. The script was completed by going through the below steps.

Importing the necessary module and providing initial information about the script:



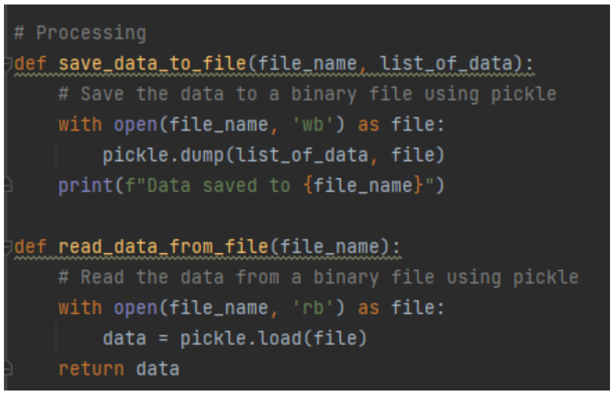
```
# ---------------------------------------------------------------------------- #
# Title: Assignment 07
# Description: Working with Pickling Data and Exception Handling
# Dev : Amruth D
# ChangeLog (Who,When,What):
# Amruth D,02ndJune2023,created script to complete assignment 07
# ---------------------------------------------------------------------------- #

# This code demonstrates the usage of pickle module to save and load data from a file and Structured error handling

import pickle
```

Initializing variables for data storage:



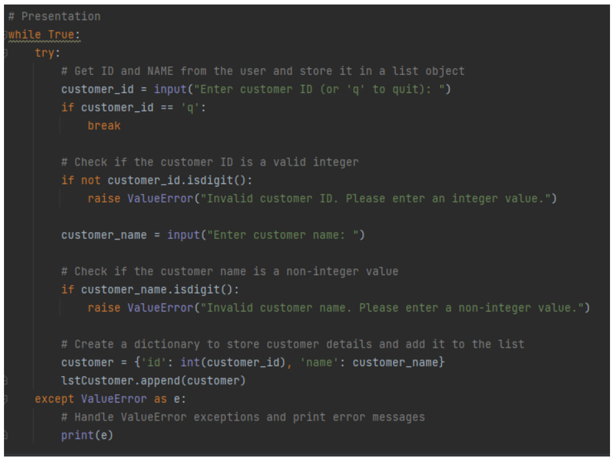
```
# Data
strFileName = 'AppData.dat'  # File name for storing data
lstCustomer = []  # List to store customer data
```

Defining functions for data processing:



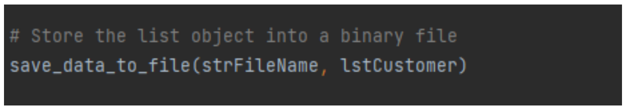
```python
# Processing
def save_data_to_file(file_name, list_of_data):
    # Save the data to a binary file using pickle
    with open(file_name, 'wb') as file:
        pickle.dump(list_of_data, file)
    print(f"Data saved to {file_name}")

def read_data_from_file(file_name):
    # Read the data from a binary file using pickle
    with open(file_name, 'rb') as file:
        data = pickle.load(file)
    return data
```

Presentation block (user interaction and data input):



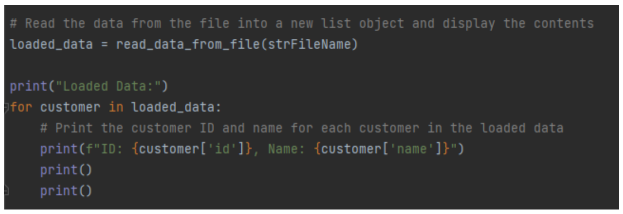
```python
# Presentation
while True:
    try:
        # Get ID and NAME from the user and store it in a list object
        customer_id = input("Enter customer ID (or 'q' to quit): ")
        if customer_id == 'q':
            break

        # Check if the customer ID is a valid integer
        if not customer_id.isdigit():
            raise ValueError("Invalid customer ID. Please enter an integer value.")

        customer_name = input("Enter customer name: ")

        # Check if the customer name is a non-integer value
        if customer_name.isdigit():
            raise ValueError("Invalid customer name. Please enter a non-integer value.")

        # Create a dictionary to store customer details and add it to the list
        customer = {'id': int(customer_id), 'name': customer_name}
        lstCustomer.append(customer)
    except ValueError as e:
        # Handle ValueError exceptions and print error messages
        print(e)
```

Storing the list object into a binary file:



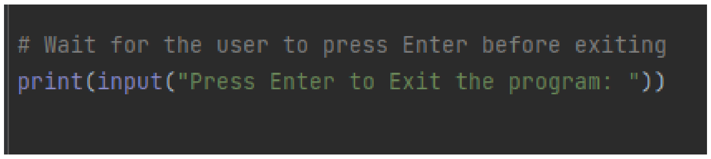
```python
# Store the list object into a binary file
save_data_to_file(strFileName, lstCustomer)
```

Reading the data from the file, displaying the contents, and printing customer details:



```python
# Read the data from the file into a new list object and display the contents
loaded_data = read_data_from_file(strFileName)

print("Loaded Data:")
for customer in loaded_data:
    # Print the customer ID and name for each customer in the loaded data
    print(f"ID: {customer['id']}, Name: {customer['name']}")
    print()
    print()
```

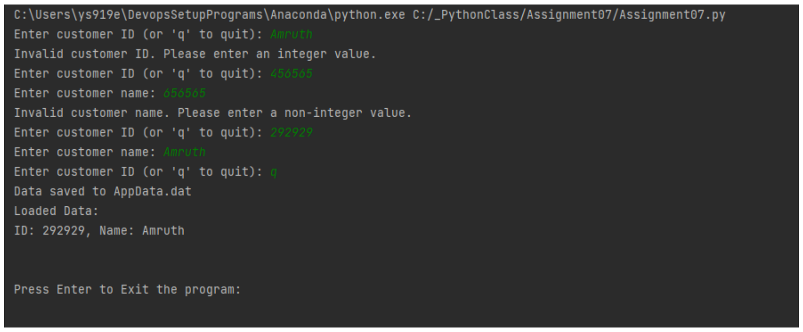
Waiting for the user to press Enter before exiting the program:



```
# Wait for the user to press Enter before exiting
print(input("Press Enter to Exit the program: "))
```

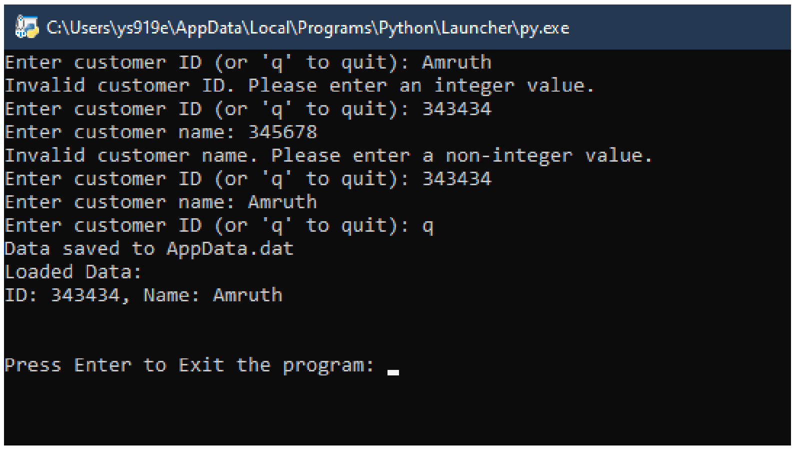
Executing the Python Script

Now that the script is written, the next step is to execute the script to check the correctness.



```
C:\Users\ys919e\DevopsSetupPrograms\Anaconda\python.exe C:/_PythonClass/Assignment07/Assignment07.py
Enter customer ID (or 'q' to quit): Amruth
Invalid customer ID. Please enter an integer value.
Enter customer ID (or 'q' to quit): 456565
Enter customer name: 656565
Invalid customer name. Please enter a non-integer value.
Enter customer ID (or 'q' to quit): 292929
Enter customer name: Amruth
Enter customer ID (or 'q' to quit): q
Data saved to AppData.dat
Loaded Data:
ID: 292929, Name: Amruth


Press Enter to Exit the program:
```

Output of script in PyCharm



```
C:\Users\ys919e\AppData\Local\Programs\Python\Launcher\py.exe
Enter customer ID (or 'q' to quit): Amruth
Invalid customer ID. Please enter an integer value.
Enter customer ID (or 'q' to quit): 343434
Enter customer name: 345678
Invalid customer name. Please enter a non-integer value.
Enter customer ID (or 'q' to quit): 343434
Enter customer name: Amruth
Enter customer ID (or 'q' to quit): q
Data saved to AppData.dat
Loaded Data:
ID: 343434, Name: Amruth


Press Enter to Exit the program:
```

Output of script in Command Prompt

Summary

In the Seventh week, we learnt about exception handling and pickling.

Writing the code for this week's assignment was little more exiting as we got together all the knowledge we acquired over the last 7 weeks. Writing python scripts now is getting more habitual and comfortable.