Experiment No. 6
Serialization in Python using Pickle
Date of Performance: 21/02/2024
Date of Submission: 28/02/2024

Experiment No. 6

Title: Serialization in Python using Pickle

Aim: To study and implement serialization using Pickle in Python

Objective: To introduce serialization and deserialization using Pickle module in Python

Theory:

Serialization and deserialization play crucial roles in data handling, especially in scenarios where data needs to be stored or transmitted efficiently. Pickle, being a built-in module in Python, simplifies this process by offering a convenient way to serialize and deserialize Python objects.

One important aspect to note about Pickle is its ability to handle complex data structures seamlessly. It can serialize and deserialize not only basic data types like strings and integers but also more complex objects like lists, dictionaries, and even user-defined classes.

Additionally, Pickle provides support for protocol versions, allowing developers to choose the appropriate protocol based on factors such as compatibility and efficiency. The protocol version determines the format of the serialized data and can impact factors like file size and serialization/deserialization speed.

It's worth mentioning that while Pickle is powerful and convenient, it's not without limitations. One notable limitation is that the serialized data is not human-readable, making it unsuitable for scenarios where human-readable data is required. Also, Pickle may not be the most efficient solution for large datasets or scenarios where interoperability with non-Python systems is a requirement.

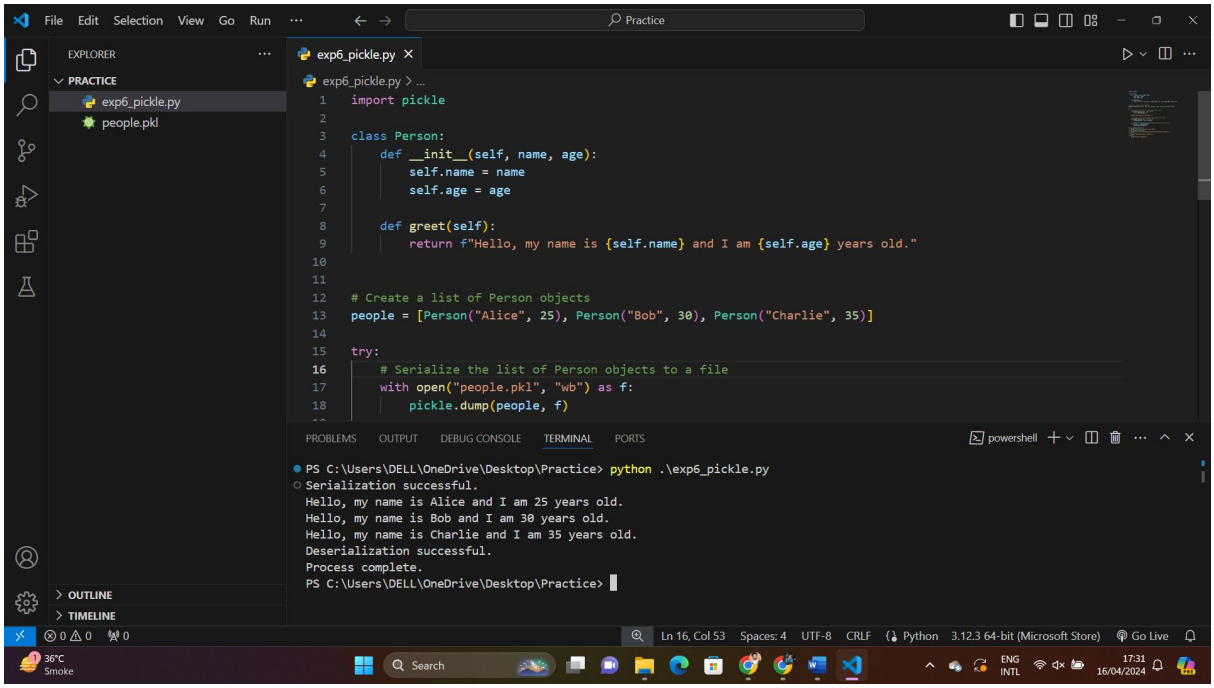
Despite these limitations, Pickle remains a valuable tool in the Python ecosystem for many use cases, offering a quick and straightforward solution for serialization and deserialization tasks. By understanding its capabilities and limitations, developers can leverage Pickle effectively to manage data in their Python applications.

Code:.

```
import pickle


class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def greet(self):
        return f"Hello, my name is {self.name} and I am {self.age} years old."


# Create a list of Person objects
people = [Person("Alice", 25), Person("Bob", 30), Person("Charlie", 35)]


try:
    # Serialize the list of Person objects to a file
    with open("people.pkl", "wb") as f:
        pickle.dump(people, f)


    print("Serialization successful.")


    # Deserialize the list of Person objects from the file
    with open("people.pkl", "rb") as f:
        loaded_people = pickle.load(f)
    # Iterate over the deserialized objects and greet each person
```

```
    for person in loaded_people:
        print(person.greet())


except FileNotFoundError:
    print("File not found error occurred.")
except pickle.PickleError:
    print("Error occurred during serialization/deserialization.")
else:
    print("Deserialization successful.")
finally:
    print("Process complete.")
```

Output:



Conclusion:

The experiment successfully demonstrates the serialization and deserialization of Python objects using the Pickle module. By serializing instances of the Student class into a file and deserializing them back, we have effectively stored and retrieved object data. This process showcases the practical utility of serialization for data persistence and transfer in Python programming. Through Pickle, we can easily maintain the state of objects across sessions, enhancing the versatility and efficiency of our code.