Experiment No. 6
Serialization in Python using Pickle
Name : Vaidehi D. Gadag
Branch/Div.: Comps-1 (C47)
Date of Performance: 24/02/2024
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.

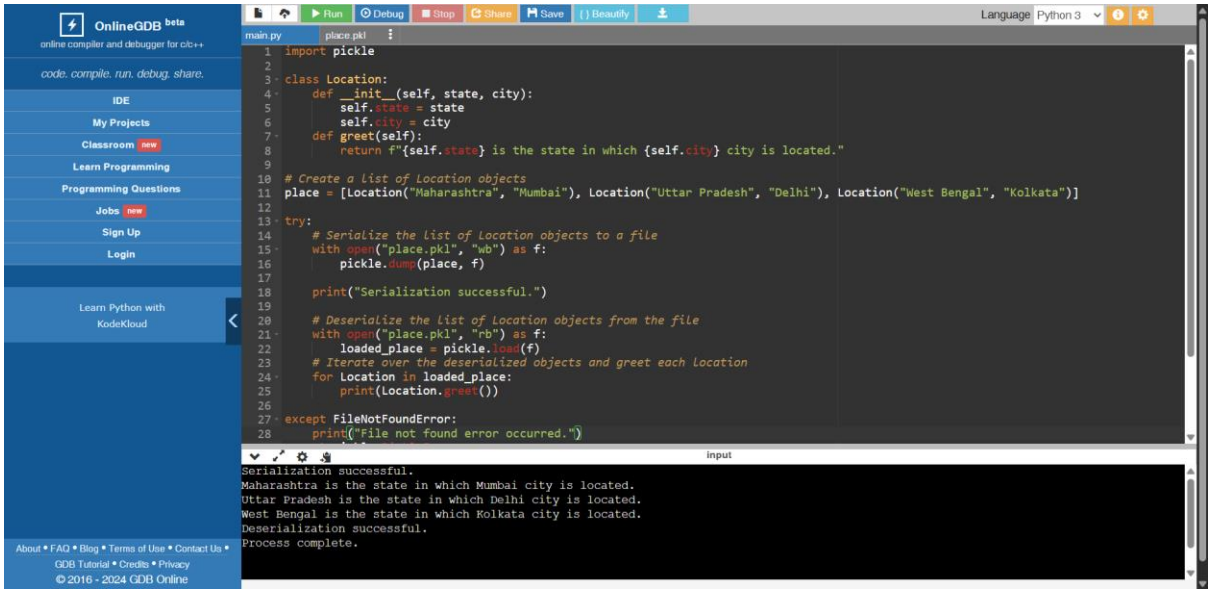
Despite these limitations, Pickle remains a valuable tool in the Python ecosystem for many use cases, offering a quick and straightforward solution for serialization and deserialization tasks. By understanding its capabilities and limitations, developers can leverage Pickle effectively to manage data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

Code:

```
import pickle

class Location:

    def __init__(self, state, city):

        self.state = state

        self.city = city

    def greet(self):

        return f"{self.state} is the state in which {self.city} city is located."


# Create a list of Location objects

place = [Location("Maharashtra", "Mumbai"), Location("Uttar Pradesh", "Delhi"), Location("West Bengal", "Kolkata")]


try:

    # Serialize the list of Location objects to a file

    with open("place.pkl", "wb") as f:

        pickle.dump(place, f)


    print("Serialization successful.")


    # Deserialize the list of Location objects from the file

    with open("place.pkl", "rb") as f:

        loaded_place = pickle.load(f)
```

```
    # Iterate over the deserialized objects and greet each location
    for Location in loaded_place:
        print(Location.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 process of serialization and deserialization has several practical applications in Python programming. For instance, serialization can be used for data persistence, allowing us to maintain the state of objects across different sessions. This can be particularly useful in applications where data needs to be saved and loaded between runs, such as in game development or scientific simulations.

Additionally, serialization can be used for data transfer, enabling us to send and receive object data between different parts of a program or even between different programs running on different machines. This can be useful in distributed systems or in networked applications where data needs to be exchanged between different nodes.

Overall, the Pickle module provides a powerful and flexible mechanism for serializing and deserializing Python objects. By mastering this technique, Python programmers can enhance the versatility and efficiency of their code, making it easier to manage complex data structures and transfer data between different parts of a system.