|  |
| --- |
| Experiment No. 6 |
| Serialization in Python using Pickle |
| Name: Krisha Chikka |
| Roll no. : 31 SE/Div1 |
| Date of Performance: 12/02/2024 |
| Date of Submission: 26/02/2024 |

**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.")

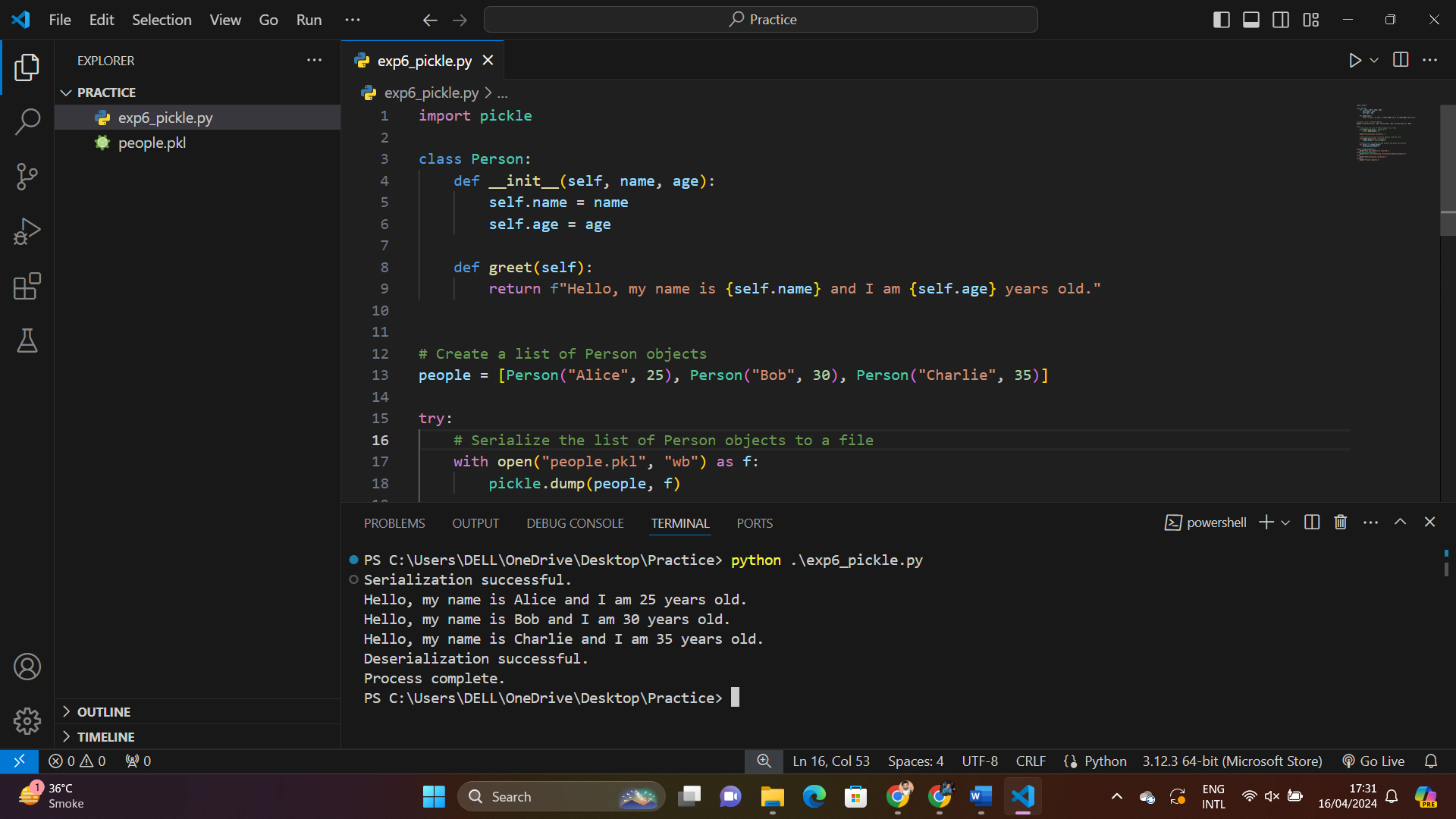
else:

    print("Deserialization successful.")

finally:

    print("Process complete.")

**Output:**



**Conclusion:**

In conclusion, the experiment on serialization in Python using Pickle provides valuable insights into the efficient handling of data within Python applications. Pickle, being a built-in module, simplifies the serialization and deserialization process, allowing developers to store and retrieve complex Python objects with ease. Through the experiment, we explored Pickle's capabilities in serializing custom objects like classes and efficiently storing them in files. Additionally, we learned about Pickle's support for protocol versions and its ability to handle various data structures seamlessly. While Pickle offers convenience and simplicity, it's important to consider its limitations, such as non-human-readable serialized data and potential inefficiencies with large datasets. Overall, this experiment enhances our understanding of serialization techniques in Python and equips us with a valuable tool for managing data in Python applications effectively.