Q1. Describe the differences between text and binary files in a single paragraph.

ANSWER.

Text files store data as plain text, usually in a human-readable format, where each character corresponds to a specific character encoding scheme like ASCII or Unicode. They contain readable characters, such as letters, numbers, punctuation, and whitespace. On the other hand, binary files store data in a format that's not intended for human consumption and is encoded in binary, comprising sequences of 0s and 1s. Binary files can store any type of data, including images, audio, video, and executable code, and are typically processed by specific programs designed to interpret their contents. Unlike text files, binary files may not be human-readable without specialized software, and their structure is often optimized for efficiency in storage and processing, utilizing complex data representations tailored to the specific data being stored.

Q2. What are some scenarios where using text files will be the better option? When would you like to use binary files instead of text files?

ANSWER.

Using text files is advantageous in scenarios where human readability and easy editing are priorities. For example, configuration files for software applications are often stored as text files because they can be easily modified with a simple text editor, allowing users to customize settings without needing specialized software. Additionally, text files are suitable for storing data that needs to be shared and understood by humans, such as documentation, plain text reports, or scripts.

On the other hand, binary files are preferred when efficiency in terms of storage space and processing speed is crucial, or when the data being stored is not intended for direct human consumption. Binary files are commonly used for storing multimedia data like images, audio, and video files, as well as compiled code and proprietary data formats. They offer faster read and write operations compared to text files and can handle complex data structures more efficiently. Additionally, binary files provide greater flexibility in representing data, allowing for optimized storage and processing tailored to specific application requirements.

Q3. What are some of the issues with using binary operations to read and write a Python integer directly to disc?

ANSWER.

Overall, while using binary operations for reading and writing data can offer benefits such as improved efficiency and reduced storage space, it also introduces complexities and potential issues that need to be carefully managed.

Q4. Describe a benefit of using the with keyword instead of explicitly opening a file.

ANSWER.

Using the `with` keyword in Python for file operations offers the benefit of automatic resource management, specifically for file handling. When you use `with`, you create a context manager that automatically handles the opening and closing of the file. This ensures that the file is properly closed when you're done with it, even if an exception occurs during the execution of your code.

Q5. Does Python have the trailing newline while reading a line of text? Does Python append a newline when you write a line of text?

ANSWER.

When you read a line of text using Python's file reading methods like `readline()` or `readlines()`, Python includes the trailing newline character ('\n') in the returned string, if present in the file. However, when you use `read()` to read the entire file into a string, the trailing newline character of the last line, if it exists, is not included in the returned string.

When you write a line of text using Python's file writing methods like `write()`, Python does not automatically append a newline character ('\n') to the end of the string you write. It only writes the exact characters you specify. If you want to include a newline at the end of each line when writing to a file, you need to explicitly include it in the string you write.

Q6. What file operations enable for random-access operation?

ANSWER.

Random-access operation refers to the ability to read or write data at any position within a file, rather than sequentially from the beginning to the end. In Python, the following file operations enable random-access operation:

1. seek(offset, whence): The `seek()` method of a file object sets the file's current position. It takes two arguments: `offset`, which indicates the number of bytes to move the position relative to a reference point determined by the `whence` argument, which defaults to 0 (the beginning of the file). `whence` can take the following values:

- 0: Beginning of the file (default)

- 1: Current position

- 2: End of the file

2. tell(): The `tell()` method returns the current position within the file as an integer number of bytes from the beginning.

Q7. When do you think you'll use the struct package the most?

ANSWER.

You'll likely use the `struct` package most frequently when dealing with binary data in various contexts, including file I/O, network programming, interfacing with C libraries, low-level data manipulation, and handling binary protocols.

Q8. When is pickling the best option?

ANSWER.

Pickling is a versatile tool in Python for serializing and deserializing Python objects, making it suitable for a wide range of applications where preserving the state and structure of Python objects is important. However, it's worth noting that pickling may not be suitable for all scenarios, such as when interoperability with non-Python systems or human-readable data formats is required. In such cases, alternative serialization formats like JSON or XML may be more appropriate.

Q9. When will it be best to use the shelve package?

ANSWER.

The `shelve` module is best suited for scenarios where you need a simple and efficient way to persistently store and retrieve Python objects in a structured manner, without the complexity of dealing with low-level file operations or external databases. However, it's important to note that `shelve` is not suitable for concurrent access from multiple processes or threads due to its file-based nature. In such cases, you may need to consider using a more robust data storage solution such as a database.

Q10. What is a special restriction when using the shelve package, as opposed to using other data dictionaries?

ANSWER.

While the `shelve` package provides a convenient way to persistently store and retrieve Python objects in a dictionary-like manner, it's important to be mindful of the picklability restriction and ensure that all objects stored in a shelf are serializable using `pickle`.