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

Ans- Text files store data as plain text that can be read and edited by humans using a simple text editor, while binary files store data in a format that can only be read and understood by computers. Text files use character encoding such as ASCII or Unicode, while binary files use binary code.

**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?**

Ans- Text files are often preferred when the data needs to be easily readable and editable by humans, such as for storing configuration settings, log files, or source code. They can also be useful for sharing information across different platforms, as text files can be opened and edited using a wide range of software programs.

Binary files, on the other hand are typically used for storing large amounts of data that are not intended to be directly read or edited by humans, such as images, audio, video, and executable files. They can be more efficient in terms of storage space and processing speed, as they do not require the overhead of encoding and decoding text data. Thus, binary files can also be more complex and may require specialized software to open and interpret the data correctly.

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

Ans- There are a few issues with using binary operations to read and write a Python integer directly to disk like Endianness, Portability and Maintenance. For these reasons, it is often better to use text-based formats, such as JSON or CSV, to store and exchange data in a way that is more portable, human-readable, and easier to maintain.

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

Ans- The ‘with’ statement in Python provides a way to automatically handle the opening and closing of files in a safe and efficient way. When using the ‘with’ keyword, the file is automatically closed when the block of code inside the ‘with’ statement is exited, whether by normal execution or by an exception being raised. This eliminates the need to manually close the file, reducing the risk of file leaks and making the code more concise and readable. Using the ‘with’ keyword for file I/O in Python provides a cleaner and safer way to handle file operations, making the code more readable, maintainable, and less error-prone.

**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?**

Ans- When you read a line of text using the ‘readline()’ method in Python, the trailing newline character (\n) is included in the string that is returned. This is true for both text files and binary files opened in text mode.

Similarly, when you write a line of text using the ‘write()’ method in Python, you are responsible for including the newline character if you want it to be added to the file. If you don't include the newline character, the next line of text that is written to the file will be appended immediately after the last character of the previous line.

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

Ans- In Python, you can perform random-access operations on a file using the ‘seek()’ method, which allows you to move the file pointer to a specific location within the file. However, for random-access operations to work, you must open the file in binary mode using the "rb" or "wb+" mode and with help of ‘tell()’ method returns the file's current position, as an integer number of bytes from the beginning of the file.

With these operations, you can move the file pointer to a specific position and read or write data from that position. This enables you to perform random-access operations on a file.

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

Ans- There are some insights on when the ‘struct’ package might be useful in general. The ‘struct’ module in Python provides a way to convert native Python data types such as integers and floats into a string of bytes and vice versa. This is useful when working with binary data formats such as network protocols, file formats, and data exchange formats.

The ‘struct’ module is useful when working with binary data formats, which can occur in various programming domains such as file I/O, network programming, and hardware interaction.

**Q8. When is pickling the best option?**

Ans-Pickling is a serialization process in Python that allows you to convert a Python object into a stream of bytes, which can be stored in a file or sent over a network, and later be unpickled to reconstruct the original object. The pickle module in Python provides an easy-to-use interface for this serialization process.

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

Ans- The ‘shelve’ package in Python provides a high-level interface for persistent storage of Python objects in a dictionary-like format. The ‘shelve’ module is built on top of the ‘dbm’ module, which provides a simple key-value store interface to a database. There are some situations where the shelve module might be the best option like Storing and retrieving large amounts of data, Caching, Storing user preferences and Web scraping.

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

Ans- One special restriction when using the ‘shelve’ module compared to using other data dictionaries is that the keys used in the ‘shelve’ database must be strings. This is because the ‘shelve’ module uses the keys as filenames to store the corresponding values as pickled data.