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

Text files contain human-readable text with characters encoded in formats like ASCII or UTF-8, making them suitable for documents, code, or configuration files. Binary files, on the other hand, store data in a non-human-readable format with no character encoding, making them ideal for storing complex data like images, videos, or executable programs. Binary files preserve the exact bytes, while text files may undergo encoding and decoding transformations.

Q2. In which scenarios are text files better, and when should you use binary files?

Text files are better for storing and editing human-readable data like text documents, configuration files, and source code. Binary files are preferred when you need to store or transmit non-textual data such as images, audio, video, or complex data structures like databases and executables.

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

When using binary operations, endianness (byte order) differences between systems can cause compatibility issues. Additionally, binary files are not human-readable, making debugging and manual inspection challenging.

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

Using the 'with' keyword (context manager) for file operations ensures that the file is properly closed after use, even if an exception occurs. This improves code reliability and avoids resource leaks.

Q5. Does Python include the trailing newline when reading a line of text? Does it append a newline when writing a line of text?

Python includes the trailing newline when reading a line of text using methods like `readline()`. When writing a line of text using methods like `write()`, you need to include the newline character (`\n`) explicitly if desired.

Q6. Which file operations enable random-access operation?

File operations like `seek()` and `tell()` enable random-access operations, allowing you to move to a specific position in a file and retrieve the current position, respectively.

Q7. When is the struct package most commonly used?

The struct package is commonly used for working with binary data files, especially when you need to pack and unpack binary data structures or read/write data in a specific format, such as handling network protocols or file formats.

Q8. When is pickling the best option?

Pickling is the best option when you need to serialize Python objects (e.g., custom classes, data structures) into a binary format for storage, transmission, or inter-process communication within the Python environment.

Q9. When is it best to use the shelve package?

The shelve package is best used when you need a simple persistent storage solution for Python objects that can be accessed like dictionaries. It's suitable for storing and retrieving data in a key-value format, especially in scenarios where you want to maintain state across program runs.

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

The shelve package imposes a limitation on keys; they must be strings. Unlike other data dictionaries, shelve keys cannot be of any data type; they must be text-based.