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

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?

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

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

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?

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

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

Q8. When is pickling the best option?

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

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

Answer:

Q1. Text files and binary files differ in the way they store and represent data. Text files store data in a human-readable format where each character represents a printable character, and special characters such as newlines and tabs have specific escape sequences. Binary files, on the other hand, store data in a machine-readable format, where each character represents a byte of information. Text files are typically smaller in size and can be easily edited with a text editor, while binary files are larger in size and require specific software to read and modify.

Q2. Text files are a better option when the data is primarily textual, such as documents, configuration files, and logs. Binary files are suitable when the data is non-textual, such as images, videos, audio files, and serialized Python objects. Binary files are also faster to read and write since they do not require encoding and decoding operations.

Q3. One of the issues with using binary operations to read and write a Python integer directly to disc is that the byte order of the integer may not be consistent across different platforms. This can cause compatibility issues when moving the file between platforms with different byte orders.

Q4. The with keyword provides a cleaner syntax for opening and closing files, automatically handling the closing of the file after the block is executed, even if an exception occurs. This ensures that the file is closed properly, preventing resource leaks and potential data corruption.

Q5. When reading a line of text in Python, the trailing newline character is included as part of the line. When writing a line of text, Python appends a newline character at the end of the line, unless specified otherwise.

Q6. The seek() and tell() file operations enable random-access operations on a file. The seek() operation allows moving the file pointer to a specific byte location, while the tell() operation returns the current position of the file pointer.

Q7. The struct package is most commonly used in scenarios where binary data needs to be packed and unpacked, such as parsing network packets, reading binary files, and interacting with low-level hardware.

Q8. Pickling is the best option when a Python object needs to be serialized and deserialized, preserving its state and class information. This is useful in scenarios where the object needs to be stored to disk or transmitted over a network.

Q9. The shelve package is best used when a dictionary-like interface is required to store and retrieve Python objects to and from disk. Shelve provides persistent storage of Python objects, allowing them to be retrieved across multiple sessions.

Q10. The shelve package restricts the keys in the dictionary to strings, whereas other data dictionaries do not have this restriction. This means that the keys must be serialized to strings before they can be used in the shelve package.