Python Advanced Assignment-18

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

Ans.

Text files store data in a human-readable format using standard character encoding like ASCII or UTF-8, which means each byte represents a character. Binary files, on the other hand, store data in a machine-readable format with no encoding applied. This means the data is stored as raw binary data, representing different types like images, audio, or compiled code. Text files use newline characters to indicate line breaks, while binary files do not.

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 better suited for storing human-readable information, such as logs, configuration files, and documents that need to be manually edited or read. Binary files are preferred when storing complex data structures or when you require efficiency in both space and speed, such as storing images, audio, video files, or serialized data.

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

Ans.

One issue is that Python integers can be of arbitrary size, so writing them directly to binary form might result in platform-specific formats or data corruption. Additionally, the endianness (byte order) and integer size must be considered. Without proper serialization, it’s hard to maintain compatibility across systems.

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

Ans.

The `with` keyword automatically handles the opening and closing of a file, ensuring that resources are properly released after file operations. This prevents issues like leaving files open (which can lead to memory leaks or locked files) and makes code cleaner and more concise.

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.

- Reading: Yes, when reading a line of text using methods like `readline()`, Python includes the trailing newline (`\n`) character.

- Writing: Python does not automatically append a newline when writing a line of text using `write()`. You need to explicitly add `\n` if required.

Q6: What file operations enable random-access operation?

Ans.

Random-access file operations are enabled by the `seek()` and `tell()` methods:

- `seek(offset, whence)`: Moves the file pointer to a specific position.

- `tell()`: Returns the current position of the file pointer.

These operations allow you to read from or write to arbitrary positions in the file without reading it sequentially.

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

Ans.

The `struct` package is most useful when you need to work with binary data that requires a specific format, such as reading from or writing to binary files, communicating over a network with binary protocols, or working with C-style structs in Python. It allows you to pack and unpack data into a specific byte format.

Q8: When is pickling the best option?

Ans.

Pickling is best when you need to serialize and deserialize complex Python objects (such as dictionaries, lists, or class instances) to and from a file or across a network. It is useful when you want to save Python data in a format that preserves object structure and relationships for later use.

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

Ans.

The `shelve` package is ideal when you need a persistent, dictionary-like object storage where you can store Python objects (including complex data structures) on disk with dictionary semantics. It is useful for simple databases where you need to store and retrieve Python objects by key without needing a full-fledged database system.

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

Ans.

A key restriction with `shelve` is that keys must be strings. Additionally, objects stored in a shelve must be serializable via the pickle module, meaning you can't store objects that aren't pickleable (e.g., open file handles or certain types of connections).