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

ANS:

There are mainly two types of data files — text file and binary file. A text file consists of human readable characters, which can be opened by any text editor. On the other hand, binary files are made up of non-human readable characters and symbols, which require specific programs to access its contents.

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 used to store data more user friendly. Binary files are used to store data more compactly. In the text file, a special character whose ASCII value is 26 inserted after the last character to mark the end of file. In the binary file no such character is present.

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

ANS:

The open() function opens a file in text format by default. To open a file in binary format, add 'b' to the mode parameter. Hence the "rb" mode opens the file in binary format for reading, while the "wb" mode opens the file in binary format for writing. Unlike text files, binary files are not human-readable.

When you read from a binary file, a data type called bytes is used. This is a bit like list or tuple, except it can only store integers from 0 to 255.

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

ANS:

Using with means that the file will be closed as soon as you leave the block. This is beneficial because closing a file is something that can easily be forgotten and ties up resources that you no longer need.

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:

Python readline() method reads only one complete line from the file given. It appends a newline (“\n”) at the end of the line.

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

ANS:] Random Access Files

1. position – Returns the channel's current position.
2. position(long) – Sets the channel's position.
3. read(ByteBuffer) – Reads bytes into the buffer from the channel.
4. write(ByteBuffer) – Writes bytes from the buffer to the channel.
5. truncate(long) – Truncates the file (or other entity) connected to the channel.

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

ANS:

It is used mostly for handling binary data stored in files or from network connections, among other sources. This process needs to be done at the start of the program.

Q8. When is pickling the best option?

ANS:

Pickle in Python is primarily used in [serializing and deserializing a Python object structure](https://wiki.python.org/moin/UsingPickle). In other words, it’s the process of converting a Python object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network. The pickled byte stream can be used to re-create the original object hierarchy by unpickling the stream. This whole process is similar to object serialization in Java or .Net.

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

ANS:

The shelve module in Python's standard library is a simple yet effective tool for persistent data storage when using a relational database solution is not required. The shelf object defined in this module is dictionary-like object which is persistently stored in a disk file.

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

ANS:

The shelf dictionary has certain restrictions. Only string data type can be used as key in this special dictionary object, whereas any picklable Python object can be used as value. This is the base class for shelf implementations.