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&#39;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?

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

Text files store data in a human-readable format, typically encoded using character sets like ASCII or UTF-8, with content structured as lines of text. In contrast, binary files store data in a raw, byte-by-byte format, which is not meant to be human-readable. Text files interpret data according to encoding standards, automatically handling line breaks and character representations, while binary files treat data as a sequence of bytes, preserving the exact byte sequences as written, making them suitable for non-textual data like images or executables.

### **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?**

Text files are better when dealing with data that is human-readable and will be processed or edited by humans, such as configuration files, logs, or CSV files. They are also more portable across different systems due to standard encodings. Binary files are preferable when working with non-textual data like images, audio files, or when exact byte-for-byte fidelity is required, such as for serialized data, compressed files, or executable programs, where any alteration to the data could corrupt the file.

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

Using binary operations to read and write a Python integer directly to disk can lead to issues such as endianess, where the byte order of data can vary between different systems (big-endian vs. little-endian), potentially causing misinterpretation of data. Additionally, the size of the integer (e.g., 32-bit vs. 64-bit) might differ across platforms, leading to compatibility issues. Direct binary operations also lack the flexibility and abstraction that higher-level formats like pickling provide, which automatically handle these concerns.

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

Using the with keyword ensures that files are properly closed after their suite finishes, even if an exception occurs. This reduces the risk of resource leaks and avoids the need for explicitly calling close() on the file object. It also simplifies the code by eliminating the need for try/finally blocks to ensure file closure.

### **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?**

When reading a line of text using Python's readline() or iterating over a file object, the trailing newline character is included in the returned string unless it’s the last line of the file, which might not have a newline. When writing a line of text using write(), Python does not automatically append a newline; you must include \n at the end of the string if you want to ensure that each line is properly separated.

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

Random-access operations are enabled by the seek() and tell() methods in Python. The seek() method allows you to move the file pointer to a specific position in the file, enabling reading or writing from that point, while tell() returns the current position of the file pointer. These operations are particularly useful for binary files or when working with large files where reading sequentially is inefficient.

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

The struct package is most useful when you need to handle binary data that follows a specific format, such as reading or writing binary files that store structured data like numbers or packed data fields. It’s also useful for working with data received from or sent to low-level systems like network protocols or hardware devices, where precise control over the data layout and byte order is required.

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

Pickling is the best option when you need to serialize complex Python objects, including custom classes, dictionaries, lists, and more, to store them on disk or transmit them over a network. It allows you to save the state of an object and later reconstruct it with minimal effort, making it ideal for saving program states, caching data, or passing data between different parts of a program.

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

The shelve package is best used when you need a simple, persistent, dictionary-like object that stores Python objects in a database file. It’s useful when you want to store a collection of objects that can be accessed via keys, much like a Python dictionary, but with the added benefit of persistence across program runs. This makes it ideal for small to medium-sized applications that need to persist data without setting up a full-fledged database.

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

A special restriction when using the shelve package is that the keys in the shelve database must be strings. Additionally, the values must be picklable Python objects, meaning they must be able to be serialized and deserialized by the pickle module. This contrasts with standard dictionaries, which can use a broader range of immutable types as keys and do not impose serialization requirements on values.