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

The two file types may look the same on the surface, but they encode data differently. While both binary and text files contain data stored as a series of bits (binary values of 1s and 0s), the bits in text files represent characters, while the bits in binary files represent custom data.

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?

In text file data is stored in Unicode or ASCII that can be read by program and user both. So, if data need to be readable by user text file is more suitable to store data. A small error in a textual file can be recognized and eliminated when seen. Whereas, a small error in a binary file corrupts the file and is not easy to detect. Text files are human readable and it's easy to diff/grep/otherwise work with them.

A binary file is usually very much smaller than a text file that contains an equivalent amount of data. For image, video, and audio data this is important. Small files save storage space, can be transmitted faster, and are processed faster. I/O with smaller files is faster, too, since there are fewer bytes to move. In addition, binary formats also offer advantages in terms of speed of access.

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

As we know that Python's built-in input() function always returns a str(string) class object. So for taking integer input we have to type cast those inputs into integers by using Python built-in int() function.

Reading integers from binary file in Python:

The read method returns a sequence of bytes as a string. To convert from a string byte-sequence to binary data, use the built-in struct module:

import struct

print(struct.unpack('i', fin.read(4)))

Note that unpack always returns a tuple, so struct.unpack('i', fin.read(4))[0] gives the integer value that you are after.

You should probably use the format string '<i' (< is a modifier that indicates little-endian byte-order and standard size and alignment - the default is to use the platform's byte ordering, size and alignment). According to the BMP format spec, the bytes should be written in Intel/little-endian byte order.

An alternative method which does not make use of 'struct.unpack()' would be to use NumPy:

import numpy as np

f = open("file.bin", "r")

a = np.fromfile(f, dtype=np.uint32)

'dtype' represents the datatype and can be int#, uint#, float#, complex# or a user defined type. See numpy.fromfile.

Personally prefer using NumPy to work with array/matrix data as it is a lot faster than using Python lists.

As of Python 3.2+, you can also accomplish this using the from\_bytes native int method:

file\_size = int.from\_bytes(fin.read(2), byteorder='big')

Note that this function requires you to specify whether the number is encoded in big- or little-endian format, so you will have to determine the endian-ness to make sure it works correctly.

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

The with keyword in Python is used as a context manager. As in any programming language, the usage of resources like file operations or database connections is very common. But these resources are limited in supply. Therefore, the main problem lies in making sure to release these resources after usage. If they are not released, then it will lead to resource leakage and may cause the system to either slow down or crash.

As we know, the open() function is generally used for file handling in Python. But it is a standard practice to use context managers like with keywords to handle files as it will automatically release files once its usage is complete.

So by using with keyword we don’t need to open and close the file every time. It will automatically close the file once the work with the file is done.

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?

Python readline() is a file method that helps to read one complete line from the given file. It has a trailing newline (“\n”) at the end of the string returned. You can also make use of the size parameter to get a specific length of the line.

In order to print without newline in Python, you need to add an extra argument to your print function that will tell the program that you don't want your next string to be on a new line. Here's an example: print("Hello there!", end = '') print("It is a great day.")

We can open the file in append access mode i.e. ‘a’, using ‘with open’ statement too, and then we can append the text at the end of the file.

# Open a file with access mode 'a'

with open("sample.txt", "a") as file\_object:

# Append 'hello' at the end of file

file\_object.write("hello")

If we open the file in write ‘w’ mode. It will overwrite the previous line that was present in the file.

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

Purpose of linecache module in Python’s standard library is to facilitate random access to any text file, although this module is extensively used by Python’s traceback module to generate error trace stack. Further prettyprints of reading are held in a cache so that it saves time while reading lines repeatedly.

The most important function in this module is getline() which reads a specified line number from given file. Following is the list of functions −

getline(file, x)

This function returns xth line from file. If not present it will return empty string. If the file is not present in current path, function ties to locate it in directories in sys.path – the module search path.

clearcache()

If prettyprint of previous getline() function is no longer needed, you can clear the cache by this function.

checkcache()

This function checks if the cache is valid. This is useful if files in cache may have been changed on disk.

lazycache()

Seed the cache for filename with module\_globals. The module loader will be asked for the source only when getlines is called, not immediately.

getlines()

This function returns lines from file in the form of list object.

updatecache()

This function updates cache entries and returns a list of lines.

To demonstrate use of linecache functionality, first we build a text file to store the famous Zen of Python (list of software principles that influence design philosophy of Python). Output of ‘import this’ is redirected to zen.txt by following code −

import sys, io

zen = io.StringIO()

old\_stdout = sys.stdout

sys.stdout = zen

import this

sys.stdout = old\_stdout

f=open('zen.txt','w')

f.write(zen.getvalue())

f.close()

When above code is executed, zen.txt will be created in current directory. We shall use this text file to read lines from it with getline() function.

To read 4th line from file

>>> linecache.getline('zen.txt',4)

'Explicit is better than implicit.\n'

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

This module converts between Python values and C structs represented as Python bytes objects. Compact format strings describe the intended conversions to/from Python values. The module’s functions and objects can be used for two largely distinct applications, data exchange with external sources (files or network connections), or data transfer between the Python application and the C layer.

Q8. When is pickling the best option?

“Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. The advantage of using pickle is that it can serialize pretty much any Python object, without having to add any extra code. Its also smart in that in will only write out any single object once, making it effective to store recursive structures like graphs.

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

The shelve module implements persistent storage for arbitrary Python objects which can be pickled, using a dictionary-like API. The shelve module can be used as a simple persistent storage option for Python objects when a relational database is overkill. The shelf is accessed by keys, just as with a dictionary.

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

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.