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

Ans:- *Binary files typically contain a sequence of bytes, or ordered groupings of eight bits.When creating a custom*

*# file format for a program, these bytes are arranged into a format that stores the necessary information for*

*# the application. Binary file formats may include multiple types of data in the same file, such as image, video,*

*# and audio data. This data can be interpreted by supporting programs, but will show up as garbled text in a*

*# text editor.*

*# Text files are more restrictive than binary files since they can only contain textual data.However, unlike*

*# binary files, they are less likely to become corrupted. While a small error in a binary file may*

*# make it unreadable, a small error in a text file may simply show up once the file has been opened. Text files*

*# may be saved in either a plain text (.TXT) format and rich text (.RTF) format. A typical plain text file*

*# contains several lines of text that are each followed by an End-of-Line (EOL) character. An End-of-File (EOF)*

*# marker is placed after the final character, which signals the end of the file. Rich text files use a similar*

*# file structure, but may also include text styles, such as bold and italics, as well as page formatting*

*# information. Both plain text and rich text files include a (character encoding) scheme that determines how the*

*# characters are interpreted and what characters can be displayed. Since text files use a simple, standard*

*# format, many programs are capable of reading and editing text files.*

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 include small size and versatility. Kilobytes or megabytes smaller than the same data stored in*

*# other formats, they can be rapidly and massively exchanged via email or disk. Most can be opened on computers*

*# running diverse operating systems, using very basic software. Binary files is that they are more efficient.*

*# In terms of memory, storing values using numeric formats, rather than as text characters, tends to use less*

*# memory. 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?

Ans:- *When we read or write a python integer using binary operations*

*# a)Binary operations deal with raw data*

*# b)one needs to identify how many bytes one would read or write.*

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

Ans:- *When a file is opened using the 'with' keyword, if some exceptions occur after opening a file, or at the end*

*# of the file it automatically does the closing of the file. There by not leaving an file in open mode and there*

*# would no need to explicitly close 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?

Ans:- *While reading a newline of text from atext file, python reads the newline also.*

*# While writing the python doesnt append a new line at end of line. It has to be handled explicitly.*

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

Ans:- *The file operations like seek(pos,orig), tell() enable random access operations.*

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

Ans:- *The struct package is mostly used while converting a common python types into 'C' language types.This is done*

*# by packing python variables into data fields of specific sizes.So, when we try read , right number of bytes are*

*# read. This useful when interacting with existing binary files.*

*# Reading and writing a single integer using struct package*

**from** struct **import** pack, unpack, calcsize

**def** write\_file(fname,int\_n):

**with** open(fname,'wb') **as** f:

bss**=**pack('h',int\_n)

f**.**write(bss)

**def** read\_file(fname):

**with** open(fname,'rb') **as** rf:

bss**=**rf**.**read(calcsize('h'))

**return** unpack('h',bss)

write\_file('struct\_file1.dat',155)

read\_file('struct\_file1.dat')

(155,)

Q8. When is pickling the best option?

Ans:- *Once a file is created it can be read by other python programs. The functions available in the pickle package*

*# take care how to represent the data is written in th file.*

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

Ans:- *Shelve package is used to pickle data but treats the treats the entire file as dictionary. The location of any*

*# object is looked up according to its key and is returned easily.*

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

Ans:- *A “shelf” is a persistent, dictionary-like object. The difference with “dbm” databases is that the values*

*# (not the keys!) in a shelf can be essentially arbitrary Python objects — anything that the pickle module can*

*# handle. This includes most class instances, recursive data types, and objects containing lots of shared*

*# sub-objects. The keys are ordinary strings.*