Review of Files and Exceptions

Write programs that accomplish each task. Use proper conventions for variable names, input prompts, output statements, and program structure. Do *not* assume that the user will enter the correct data type or value for any input, or that files contain the correct data types or values. You may wish to use the files in `review_exceptions_files.zip` for testing.

1. A text file contains a list of Cartesian coordinates, (x, y). Each line of the file consists of two integers, separated by a comma, such as `3,15` or `-2,7`. Write a program that reads the file, then determines the domain (set of x-values) and range (set of y-values) for the coordinates.

2. **Steganography** (from the Greek, translated as "concealed writing") is the process of embedding a message, file, image or video inside of another message, file, image or video. One simple method of hiding a message in a text file is by appending whitespace to the end of each line. Since most text editors do not display whitespace, this information is essentially invisible. Write a program that reads a text file, and *decodes* the whitespace at the end of each line to display a secret message. For our purposes, let a space have a value of 1 and a tab (`\t` in Python) a value of 5. The sum of the whitespace values corresponds to a letter's position in the alphabet, where A=1, B=2, etc. Note that it is possible to encode a letter in multiple ways: the letter 'F' could be space/tab, tab/space, or six spaces. A line with no tabs or spaces after it represents a space in the secret message. Whitespace at the start of a line, or within a line, is not part of the message.

3. Write a program that will *encode* a hidden message in a text file using the whitespace scheme above. Your program should use a *random* encoding for each letter, so that it is more difficult to discern a pattern (e.g. the letter 'K', which is 11th in the alphabet, could be encoded as tab/tab/space, tab/space/tab, space/tab/tab, space/tab/5 spaces, etc.). Ensure that there are enough lines of text to encode the full message.

4. An **IPv4 address** (Internet Protocol, Version 4) consists of four integers in the format $N_0.N_1.N_2.N_3$, where N_i is a value between 0 and 255. To save space, a list of IPv4 addresses is stored in a binary file, where every four bytes represent the values of N_i. Write a program that reads a binary file containing an arbitrary number of IPv4 addresses, then outputs the list of addresses to a new binary file in ascending order. For example, the address 192.128.1.0 would come after the address 192.67.12.1, but before the address 255.32.0.3. Hint: rather than using a complex sorting mechanism to parse all four values of N_i, you may wish to revisit the `join()` and `format()` string methods instead.