Exercises

Level 4: I/O and String Manipulation

4.1: Python Strings

1) Use the following string for each of the below exercises: **“ The Python course is the best course that I have ever taken. ”**:

- **a.** Display the length of the string.
- **b.** Find the index of the first ‘o’ in the string.
- **c.** Trim off the leading spaces only.
- **d.** Trim off the trailing spaces only.
- **e.** Trim off both the leading and trailing spaces (**use this trimmed string for all the remaining parts below**).
- **f.** Fully capitalize the string.
- **g.** Fully lowercase the string.
- **h.** Display the number of occurrence of the letter ‘d’ and of the word ‘the’.
- **i.** Display the first 15 characters of the string.
- **j.** Display the last 10 characters of the string.
- **k.** Display characters 5-23 of the string.
- **l.** Find the index of the first occurrence of the word ‘course’.
- **m.** Find the index of the second occurrence of the word ‘course’.
- **n.** Find the index of the second to last occurrence of the letter ‘t’, between the 7th and 33rd characters in the string.
- **o.** Replace the period (.) with an exclamation point (!).
- **p.** Replace all occurrences of the word ‘course’ with ‘class’.

2) Save the following Windows file-path into a string variable: **C:\Users\Me\Desktop\MyTable.csv**. Perform the following operations:

- **a.** Extract the filename with extension from the path.
- **b.** Extract the file extension only.
- **c.** Add another folder (can name it whatever you’d like) between ‘Desktop’ and the filename.

3) Create a list as follows: **[‘C:’, ‘Users’, ‘Me’, ‘Desktop’, ‘MyTable.csv’]**. Perform the following:

- **a.** Join the list together to create a valid pathname.
- **b.** Insert another folder into the list, between ‘Desktop’ and ‘MyTable.csv’ and join the resulting list to create a valid pathname.

4) Create a program that does the following:

- **a.** Prompts the user for name, age (integer), and country of residence. Display the information as follows: **<name>** is **<age>** years old and lives in **<country>**.
- **b.** Do the same as above, but using a decimal number for the age. Display the number with one decimal place.
- **c.** Do the same as above, but make the country name all caps.

Write a separate version of the above using format flags, a version using the **format** function with numeric placeholders, a version using the **format** function with keyword placeholders, and a version using f-Strings – which is cleanest?

5) Convert your **Timer** class' print statement to use f-Strings instead of concatenating strings.

Note that you are expected to use f-Strings in all your code from here on out.

4.2: Logging

For all of the below exercises, be sure to use the previously-learned string formatting to log well-formed messages. Log messages should be informative and explicit as possible. For example, if logging an error for an input value, the message should contain the actual input value, why it's bad, and what the code expects instead.

1) Modify your **Timer** class to use a logging statement (info level) instead of a print statement.

2) Modify your Timer class as follows:
- **a.** Add a class-level *warnThreshold* variable, which defaults to 1 minute.
- **b.** When printing the time taken, use a warn-level log statement instead of info-level if the time taken exceeds the warn threshold.

3) Add logging statements to your **Loan** class. This should consist of the following:
- **a.** Anytime an exception is thrown (i.e., when an incorrect Asset type is passed-into the initialization function), log an error prior to raising the exception.
- **b.** Debug-level logs which display interim steps of calculations and return values for the Loan functions.
- **c.** Info-level logs to display things like 't is greater than term' in the loan functions.
- **d.** At the point the exception is caught (in your **main** function) use **logging**.**exception** to display the exception in addition to a custom message.
- **e.** Add a warn log to the recursive versions of the waterfall functions (that they are expected to take a long time, so the explicit versions are recommended).

4) Play around with your Loan and Timer classes to trigger logging statements. Switch logging levels in your **main** code to demonstrate how to turn on/off different levels of logging.

4.3: File I/O

1) To practice file management, do the following in order (using Python code):
- **a.** Create a new directory.
- **b.** Rename the above directory.
- **c.** Delete the above directory.
- **d.** Create another directory and create two text files in this directory.
- **e.** Delete one of the text files from the above directory.
- **f.** Rename the remaining text file.
- **g.** Create a subdirectory within the above created directory.
- **h.** Move the remaining text file into the subdirectory.
- **i.** Remove the top level directory with all its contents (using a single function call). Be careful!

2) Write code that searches your entire computer for all files of extension **py**. Hint: Use **os.walk** and any necessary string manipulation functions.

3) Write code that searches your entire computer for all **pdf** files which reside in any directory (at any level) that contains the string 'gr' in its name.

4) Open a brand-new file and write to it (should write several lines).

5) Open the file from 4) and read it. Display each line.

6) Open the file from 4) and append to it.

7) Create a program which does the following:
- **a.** Gives the user a choice of two options: (1) Add Loan, (2) Write file and exit.
  - **i.** If user enters '1', prompt the user for the type of Loan, its asset name/value, its face amount, rate, and term. Each prompt should occur one after the other. After the last prompt, save the entry into a Loan object, notify the user that the loan has been recorded, and return to the main menu.
  - **ii.** If user enters '2', loop through all the entered loans and write them to a file. The file should be in extension **.csv**. To do this properly, each sub-entry (loan type, asset name, asset value, amount, rate, and term) should be separate by a comma. Each loan should be separated by a newline.
- **b.** To verify that your generated .csv is a valid .csv file, try opening it in Excel once it has been generated. You should see six columns and the number of rows should reflect the number of loans.

8) As a follow-up, create a third option: (3) Read loan .csv file. This option should:

- **a.** Ask the user to enter a file path of the loan .csv file.
- **b.** Load the .csv file into **Loan** objects.
- **c.** Add an additional (fourth) option to display the WAR and WAM of all the loans.

Note that this is going to be very useful to read in actual Loan data in our ABS model in the case study. Additionally, we will be writing our ABS model results into CSV files.