Writing To Text Files

In addition to reading data from text files, Python can write data as well. There are two commands for writing data: `write()`, which writes a single string, and `writelines()`, which writes a sequence of strings. Only strings can be written to text files. Data that are not strings, such as integers or floats, must be converted to strings before writing, using the `str` function. Attempting to write non-string data to a file will crash the program.

Files can be opened for **writing**, or for **appending**. Writing to a file will erase all existing content, while appending to a file adds new data to the end of the file. In either case, a new file will be created if it does not already exist. Use `open()` with either `"w"` or `"a"` as an argument.

To write data to multiple lines in a text file, it is necessary to concatenate line endings (either `\n` on *nix and Mac, or `\r\n` on Windows) and data manually. Compare the following two code snippets:

Code:
```
f = open("fname", "w")
f.write("hello")
f.write("there")
f.close()
```

Output:
```
hellothere
```

Code:
```
f = open("fname", "w")
f.write("hello"+"\n")
f.write("there")
f.close()
```

Output:
```
hello
there
```

Writing text to a file may not occur immediately (it is system dependent), but closing a file will write all data that have not been previously written. Therefore, it is very important to close a file after it has been opened for writing. Like reading data, using the `with` keyword will automatically close the file, even if your program is interrupted.

Common Commands For Writing To Text Files

Description	Command
Open a file for *writing* and assign to a file object, f.	`f = open("FILENAME", "w")`
Open a file for *appending* and assign to a file object, f.	`f = open("FILENAME", "a")`
Alternate method, using `with` (closes file automatically).	`with open("FILENAME", "w") as f:` `  # CODE GOES HERE`
Write a string, s, to a text file.	`f.write(s)`
Write a list, L, of strings to a file (can use a tuple too).	`f.writelines(L)`
Close a file when finished.	`f.close()`

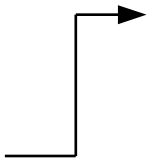
Writing To Text Files

Answer the following questions.

1. Describe a situation where you might want to preserve the content of a file to which you are writing, but appending data is not appropriate. What would you need to do instead?

2. What will be the output of the code below? Explain how to “correct” the code.

```
L = [1, 3, 5, 7, 9]
f = open("numbers.txt", "w")
for num in L:
      f.write(str(num))
f.close()
```



```
f = open("numbers.txt", "r")
print("The numbers are: ")
for line in f:
      print(line)
f.close()
```

Write programs that accomplish each task. Use proper conventions for variable names, input prompts, output statements, and program structure. You may assume that all files contain data in the formats specified. You may download the file `write_files.zip` for testing.

3. Extend the inventory program from the Reading Text Files worksheet to allow the user to add new items to the end of the file.

4. Extend the inventory program from the Reading Text Files worksheet to allow the user to modify items (description or stock level), and delete items.

5. It is bad practice to store passwords as plain text. Instead, programs typically apply a mathematical operation to the password and store the result instead. Write a program that asks the user to create a password, then generate an “encrypted” string by adding 128 to the ASCII value of each character. For example, the capital letter A would become 65 + 128 = 193, or the character ‘Á’. Write this string to a file, `password.txt`. The password ‘D5b’, for example, would appear as ‘Äµâ’. As a follow-up, write a program that will read `password.txt` then ask the user to enter the original password in three attempts. Display a message indicating success or failure.

6. A game is played where a random number between 1-20 is generated. The player attempts to guess the number. If the player guesses correctly, they receive 3 points. If they are 1 off, they receive 2 points. If they are 2 off, they receive 1 point. Otherwise, the player receives 0 points. Play 10 rounds of this game, and calculate the player’s final score. Use a file, `hiscores.txt`, to keep track of "high scores" for this game. Initially, the file contains five lines. Each line consists of a player name ("NONE" to start) and a score, separated by a comma (0 to start). Your program should read the contents of this file into a list. After a player has finished, their score should replace an entry if applicable, and the file should be updated with the new list of names/scores.