Software Workshop I

Assignment 1

Marks available: 100

Date issued: 06/11/2023

Deadline: 09/11/2023

Set by: Jacqui Chetty

Introduction

This assignment consists of several questions.

Submission instructions

Submit your work by the deadline above as a zip archive.

In your assignment, do not output anything to the console. Do not put any code that gets user input.

You must not use regular expressions in your solution. ***Use of regular expressions (RegEx) in your solution will result in a mark of zero.*** A RegEx is a sequence of characters that forma a search pattern. For example, searching a string to see if ti starts with "The" and ends with "Spain", would look as follows:

```
import re

text = "The rain in Spain"

Find = re.search("^The.*Spain$", txt)
```

However, you may use the following methods:

⇒ split()
⇒ append()
⇒ lower()
⇒ len()
⇒ ord()

⇒ isdigit()

No other in-built methods may be used, and libraries are not allowed to be imported and made use of.

Before you start:

- You are provided with a video that contains the following:
  - running the completed solution – what each output for each menu option does / looks like
  - how to execute pytests
- You are provided with skeleton code to assist you with the build-up of your project. Unzip the project and open it in PyCharm.
- Click on assignment1.py to find the skeleton code. You will build your assignment up here, working on each function. The main() function has been provided to you, make sure that you understand the calls from main().
- Any existing function names, variable / lists that form part of the skeleton code MUST NOT be changed. This can result in zero marks being awarded, so be careful about this.
- The following modules / files form part of the project:
  - Ensure that it ssignment1.py – this is where you build up your code, main() is complete and no alterations are required in this function
  - text_to_clean.txt – this is the initial file with the corrupted text (used in question 1)
  - student_names.txt – this file is empty and will be filled once you have completed question 1
  - password.txt – this file contains several invalid passwords (used in question 3)

- create_emails.txt – this file is empty and will be filled once you have completed question 4
- unique_ids.txt – this file is empty and will be filled once you have completed question 4
- addresses.txt – this file contains addresses and will be used in question 5

- You will notice that there is another .py module called test_assignment1.py. This module consists of the tests that you can run for each function, to verify if your code works logically. I suggest that you complete a function, test it, and then move on to the next function.
- When you run the tests for the first-time you may get a message "Invalid Python Interpreter".
  - Click Configure Python Interpreter
  - Choose add new interpreter
  - Choose add local
  - A window prompts you about the interpreter, click on the button **use existing** and press OK.

Good luck!!

Let's begin -> go to assignment1.py and you can crack on with building your solution.

Question 1: clean_up() **[12 marks]**

Write the function for clean_up(), where the function must perform the following tasks:

- open the text_to_clean.txt file
- read the contents into the function
- write the code to inspect the contents from the file and ensure that only the following characters form part of the text:

⇒ Lower case characters
⇒ Upper case characters
⇒ Blank spaces
⇒ Full stops

Once the text is cleaned of any unwanted characters, assign the cleaned characters to a string called cleaned (see given code) and write the contents of cleaned to the student_names.txt file (this file should be there as part of the project files). Close all files and return the cleaned string, called cleaned(), to main().

Note for the student_names.txt file: Each full name should appear one below the other (i.e. one line per full name), with the carriage return being the last character on each line

Question 2: build_id() **[13 marks]**

Write the function for build_id(), where you need to create an id from the names created in question 1 in the student_names.txt file. Open the student_names.txt file for this function.

The id is created as follows:

If the name consists of 3 parts the id is created by taking their initials, i.e. the first character of each part of the name, converted to lower case characters where necessary. For

example, if the name is "David Robert Smith" this would lead to an id of drs.

If the id consists of 2 parts then the missing middle initial must be substituted with an 'x'. For example, Ann Jones would be axj. All uppercase letters must be converted to lower case in the id.

You may make use of an in-built function for this. Make use of the list called id_list – id_list exists in main() - to build each id and pass the list back to main(). Remember to close the file.

Question 3: validate_ password() [15 marks]

Write the function for validate_password() that checks to see if a password is valid or not. The function should accept a password variable as a parameter, this is given to you in main(). A valid password has the following characteristics:

- It contains at least 8 characters
- It contains at most 12 characters
- It contains only alphabetic characters, digits and the underscore '_' character
- It has a mix of upper-case and lower-case characters
- It does not start with a digit
- It does not contain any of the commonly used passwords found in the password.txt file given to you

If it fails to adhere to these characteristics, then a message must be written to the list called illegal_password. For example,

- if the password is too short, return "TOO SHORT"
- if the password is too long, return "TOO LONG"
- if the password contains illegal characters, return "WRONG CHARACTERS"

- if the password does not have a mix of upper and lowercase characters, return "NOT MIXED CASE"
- if the password starts with a digit, return "LEADING DIGIT"
- if the password is one of the commonly used passwords, return "CANNOT MAKE USE OF THIS PASSWORD"

The function should accommodate multiple fails. For example, a password may be "TOO SHORT" and it may contain "NOT MIXED CASE". These are stored in illegal_password list.

Question 4: create_unique() [20 marks]

A full id includes the short id created in question 2, however now you need to also append the id with a 4-digit number. The function should accept the id_list (created in question 2). Full student id's are unique. The rules for the creation of the full id is that the id created must be appended with 0000. Consider the following examples:

If the id is drs then 4 zeros – 0000 – need to be appended to make the unique id as drs0000. Similarly, axs would be axj0000. Now imagine there is another student whose initials are also axs. As axj0000 has already been assigned, the sensible approach would be to simply append 0001 and so on. So, the 2nd axs would be axs0001. If another axs comes along then this would be axs0002.

Pass the id_list created in question 3 to this function to build each unique id. Make use of the file named unique_ids.txt and ensure that the unique id list is written to this file. Additionally, write each unique id to the file create_emails.txt, ensuring that an email address is created from the unique id. For example axs0000 would have the email address of axs0000@student.bham.ac.uk, similarly axs0001 would have an email address of axs0001@student.bham.ac.uk .

Note for the unique_ids.txt / create_emails.txt file: Each unique id / email should appear one below the other (i.e. one line per id / email), with the carriage return being the last character on each line

Question 5: create_short_address() [10 marks]

Write a function that creates short addresses from a longer one. The file addresses.txt is given to you and this file contains all the longer addresses for each student. Each address should be converted to the specified shorter format. Assume that the longer address format is as follows:

- address1, address2, city, postcode

Your function must create a short address that only contains the following:

- address1, postcode

For example, if the long address is as follows:

- 25 Smith Street, Safehaven, Birmingham, B112XX

The short address is as follows:

- 25 Smith Street, B112XX

Make use of a list called split_addrs to create the short addresses, returning the short address list (split_addrs) to main().

Question 6: validate_pcode() [20 marks]

Write a function that validates the postcode from the split_addr list created in Question 5. Create a validate_pcode list so that the result of each of the postcode characteristics, see below, forms part of the validate_pcode list. The validate_code should result in a series of 'True' or 'False' depending on the outcome of the postcode validation. Each time a code is valid, you can assign 'True' otherwise assign 'False'.

The postcode must adhere to the following:

- Verify that the length of the postcode is 6 characters. If the postal code is not 6 characters you can assign "$$$$$$", i.e. 6 zeros to the postcode.
- The first character must be an uppercase valid character from A to Z
- The second, third and fourth characters must be valid numerics
- The last 2 characters must be uppercase valid characters from A to Z

This means that the validate_pcode list will look something like this:

True True True False

In this case the length was correct, the first character was a valid uppercase character, the second, third and fourth characters were valid numerics and the False means that one or both last characters were not valid uppercase characters.

Now, as there will be many of these postcodes to validate, you need to assign a unique number to each so the final list will look as follows:

0 True True True False

1 True False False True

So, the final validate_pcode should look like this. Many rows starting from 0 each row holds 4 True / False combinations. The next row is 1 with another combination of True / False, and so on.

Question 7: ids_addr() [10 marks]

Accept the short address list from question 5 (short_addr) as a parameter to this function. Make use of the unique_ids.txt

file created in question 4. This function creates a dictionary that consists of the unique id found in unique_ids.txt and the short address, passed in as a parameter to this function. Return the dictionary named combo to main().