Mid-Term Test

Introduction

The concept of software testing during interviews is not a new one and is something that you are likely to experiences throughout your career. The concept this type of test is to give you a set of tasks to complete within a specified time frame. These tasks often range from developing small programs to writing or discussing pseudo code. This type of testing is not only designed to test your technical skills but also other factors such as creativeness and ability to come up with innovative solutions under pressure. In the midterm assessment, you are to complete three tasks of varying complexity within the time frame of one week. These tasks unlike ones you might find in an industry setting are designed purely for you to exhibit the skills you have learned within this unit and to challenge yourself.

Overview

In this assessment you will:

1. Create a program that reads in a set of rugby scores and outputs the winning team.
2. Create a program that reads in an encrypted string, determines the encryption technique and outputs the decrypted string.
3. Create a program that reads in a string, formats that string based on a set of rules and outputs some information related to the grammar and spelling.

Important Notes:

1. This assessment has a 15% weighting for the course unit.
2. You can complete the tasks in any order that you like e.g if you complete tasks 1 and 3 you will be awarded marks for the criteria met in those tasks.
3. All programs must run using Python 3.8, so if you create it on your personal computer you must ensure it will work when being marked - you are advised to test your solutions using the Computer Science VM image or on a laboratory machine.
4. You are permitted to use the "os", "argparse" and "re" libraries. **All other non-standard libraries are not permitted**. For a full list of standard libraries which are available on a fresh python 3.8 installation please refer to the the documentation at the following link: https://docs.python.org/3.8/library/
5. Submission of only one ".py" file is permitted for each task. The program must be named exactly as specified in each of the program specifications (shown later in this document) otherwise the auto-markers will not pick up your program and may result in a mark of zero being awarded for that task.

6. It is **critical** that you follow the instructions carefully when handling input/output. Marks are only awarded for functionality and any deviation from the specification can significantly impact the time it takes to mark your work and may affect the mark awarded.

7. The command line arguments for each program are as follows and must be in the order shown (your username is in the format of a12345bc):

   - python3 rugby_[yourUsername].py ./[input_folder] ./[output_folder]
   - python3 decrypt_[yourUsername].py ./[input_folder] ./[output_folder]
   - python3 spellcheck_[yourUsername].py ./EnglishWords.txt ./[input_folder] ./[output_folder]
     - spellcheck is all lower case

8. You do not need to create any folders only read from them using arguments. For example, your program will read in all of the .txt files in the ./[input_folder] and create the output .txt files in the ./[output_folder]. The names of these folders are examples, the names of these may differ during the marking process. You do not need to know what we will name these folders during marking.

9. Your program must iterate through all possible input files in the input folder. So if there are 5 input files there should be 5 output files and the names should match up with the input file (e.g if a input file is called **test_file1.txt** then the output file will be **test_file1_[yourUsername].txt**).

10. Your programs should account for all punctuation (spaces & $\backslash n$ are not punctuation).
    The full list of punctuation in the English Language is: *the period, question mark, exclamation point, comma, colon, semicolon, dash, hyphen, brackets, braces, parentheses, apostrophe, quotation mark, and ellipsis.*

    - # or @ are not counted as punctuation in any program.
    - The following programs do not consider punctuation at all (i.e. None of the inputs for the following will contain punctuation):
      - Morse Code
      - Caesar +3
      - Hexidecimal
    - The ellipsis (. . .) is considered to be 1 punctuation

11. You can write your own test cases to check your work. The ones provided are only samples and different test cases will be used during marking.

12. Use the **updated** Output Example files which have been provided for you in midterm_files.zip as your guide for formatting your outputs. These have been updated.

Program 1 - Rugby

Problem

Rugby is a complicated sport with many different rules and scoring opportunities, which makes it is a complicated process to calculate scores for both teams and as such which team is the winner. Here we present the task of creating a program that reads in a list of scores from two teams (T1 & T2) and outputs the score into a .txt file.

Objectives

Your program must...

1. Be able to run using command line arguments in the form of:
    - python3 rugby_[yourUsername].py ./[input_folder] ./[output_folder]
2. Read in a .txt file specified by [input file path] containing a single series of team scoring between team 1(T1) and team 2(T2) in the form of:
    - T1tT2pT2pT1pT1d
    - where T1 or T2 specifies the team and t, c, p or d specifies the score type
3. Determine the score of each team based on the scoring types and quantity:
    - t = 5 points (Try)
    - c = 2 points (Goal kick)
    - p = 3 points (Penalty)
    - d = 3 points (Drop goal)
4. Compare the scores to determine a winner:
    - The winner is the team with the most points after all scores have been calculated
    - Teams can draw
5. Output the results to a .txt file at the output path containing **only** The final score in the form of "T1:T2":
    - Example: 11:6

Program 2 - Encryption

Problem

Encryption is something you likely encounter everyday. This could be communication encryption used by many popular communication apps or when downloading media on the internet. Originally encryption was used to disguise messages between individuals in order to keep secrets. Here we present the task of creating a program that can identify three ciphers and decrypt a string encoded with one of them.

Objectives

Your program must...

1. Be able to run using command line arguments in the form of:
    - python3 decrypt_[yourUsername].py ./[input_folder] ./[output_folder]
    - where [username] is your university 8 digit identifier that normally consists of an alpha character, followed by five digits and ends with two alpha characters.
2. Read in a .txt file containing a single cipher-text (the encrypted string) in the form of:
    - binary:01010101 (your program does not need to decode binary, this is just an example)
    - (algorithm):(ciphertext)
3. Determine which encryption method is used to encrypt the string (the cipher-text):
    - Morse Code (we are using International Morse Code)
    - Caesar +3 (it is always +3)
    - Hexadecimal
4. Decrypt the cipher-text to produce the original decrypted plaintext (a sentence written in English):
    - Including spaces and numbers where appropriate
    - With **no** upper case letters
5. Output to a .txt file to the ./output_folder containing **only** the decrypted sentence:
    - "solving hex is very easy in python"
6. The input files for encryption will start with what type of decryption is to be performed. For example:
    - Hex:68 72 43
    - Caesar Cipher(+3):exw
    - Morse Code:— ....

Program 3 - Spellchecker

Problem

Spellcheckers have become common place in many word processing applications. They often identify incorrectly spelt words and suggest changes to the user to correct the error. This is achieved by searching through a common dictionary and finding words that are a close match to the original word written by the user. In this task you are to create a program that will read in a .txt file, formatting it to remove symbols/numbers etc and identify spelling mistakes within the file using a predefined dictionary of English words.

Objectives

Your program must...

1. Be able to run using command line arguments in the form of:
    - python3 spellcheck_[yourUsername].py ./EnglishWords.txt ./[input_folder] ./[output_folder]
    - ./EnglishWords.txt will be a file path to the location of the EnglishWords.txt file
2. Read in a .txt file containing a string of text:
    - Of any length
    - Including spaces, symbols, new lines and numbers
3. Format the text to:
    - Remove any numbers from the text
    - Remove any punctuation from the text
    - Transform any upper case letters into lower case variants
4. Split the file into a list of words and determine if they are correct:
    - By determining if they are contained within the "EnglishWords.txt" file which has been provided for you on Blackboard
    - We will use the EnglishWords.txt file in the state it is in when downloaded from Blackboard, any alterations you make to this file will not be considered during marking.
    - If the word appears in the EnglishWords.txt file then you can assume that the word is spelt correctly.
5. The program should count the total number of capital letters in the file, for example:
    - 'Who' would +1 to the count
    - 'THE' would +3 to the count

6. Output a .txt file of results related to formatting and spellchecking the file in the form of:

   [user_name]
   Formatting ####################
   Number of upper case letters changed: 4
   Number of punctuations removed: 5
   Number of numbers removed: 0
   Spellchecking ####################
   Number of words: 54
   Number of correct words: 54
   Number of incorrect words: 0

Output Examples

For an example layout of the output files please see the **updated** midterm_files.zip folder on Blackboard.

Preparation and Submission

All work and resources needed for your game must be pushed to your coursework git (*16321_python_coursework_[username]*). Each task .py file should have the following branch names and tags.

	Branch Name	**Tag**
Program 1	rugby	prog1_rugby
Program 2	encryption	prog2_encryption
Program 3	spellchecker	prog3_spellchecker

Additionally: Your final commit (regardless of which program is it) must also have a second tag of **Python-Midterm**. If you fail to do this then your submission will not show in SPOT. You are advised to check SPOT once you have submitted to ensure the submission was successfully received, but keep in mind SPOT only refreshes a few times a day so it may not appear instantly.

Deadline:
18:00 on Sunday the 14th November.
Automatic DASS extensions **DO NOT** apply to this test.

Assessment Type:
This activity is subject to summative assessments regulations, therefore your submission will be marked and you will receive the associated feedback. The marks you obtain (see below) count for up to 15% of your overall mark for this course unit.

Marks:
Marks are awarded on three main categories:

1. The running & testing of each program
   (a) Program 1 - Rugby
   (b) Program 2 - Encryption
   (c) Program 3 - Spell Checker

If the marker finds that the file submitted is unreadable and/or does not compile then you will not receive any marks for that program. Please be aware that we will not install external packages to make your code compile. We should be able to run your code with a basic python setup and the packages outlined at the start of this document.

Please note: Your work will not be re-marked because you disagree with the mark you were awarded. The markers all follow the same rubric which has been designed to be as fair and comprehensive as possible. Any queries with your marking can be raised with either Gareth or Stewart within a week of the marks being released, any queries made after this time will not be considered.

End of Assessment