SCHOOL OF
COMPUTER AND SECURITY SCIENCE

AUSTRALIA ECU EDITH COWAN UNIVERSITY

CSP1150/CSP5110 Programming Principles

Assignment 2: Individual programming project (WordChain)
Assignment Marks: Marked out of 30, (30% of unit)
Due Date: 28 May 2018, 9:00AM

Background Information

This assignment tests your understanding of and ability to apply the programming concepts we have covered throughout the unit. The concepts covered in the second half of the unit build upon the fundamentals covered in the first half of the unit.

Assignment Requirements

You are required to design and implement two related programs:

- "**wordchain.py**", a CLI program that allows the user(s) to play a word game. Every time the game is played, some data about it is stored in a text file. Develop this program before "logviewer.py".
- "**logviewer.py**", a GUI program that lets the user view the information in the text file and uses it to determine some gameplay statistics. Develop this program after "wordchain.py".

The following pages describe the requirements of both programs in detail.

Starter files for both of these programs are provided along with this assignment brief, to help you get started and to facilitate an appropriate program structure. *Please use the starter files.*

Pseudocode

As emphasised by the case study of Module 5, it is important to take the time to properly *design* a solution before starting to write code. Hence, this assignment requires you to *write and submit pseudocode of your program design for "wordchain.py"*, but not "logviewer.py" (pseudocode is not very well suited to illustrating the design of an event-driven GUI program). Furthermore, while your tutors are happy to provide help and feedback on your assignment work throughout the semester, they will expect you to be able to show your pseudocode and/or explain the design of your code.

You will gain a lot more benefit from pseudocode if you actually attempt it *before* trying to code your program – even if you just start with a rough draft to establish the overall program structure, and then revise and refine it as you work on the code. This back and forth cycle of designing and coding is completely normal and expected, particularly when you are new to programming. The requirements detailed on the following pages should give you a good idea of the structure of the program, allowing you to make a start on designing your solution in pseudocode.

See Reading 3.3 for further information and tips regarding writing good pseudocode.

Write a *separate section of pseudocode for each function* you define in your program so that the pseudocode for the main part of your program is not cluttered with function definitions. Ensure that the pseudocode for each of your functions clearly describes the parameters that the function receives and what the function returns back to the program.

It may help to think of the pseudocode of your program as the content of a book, and the pseudocode of functions as its appendices: It should be possible to read and understand a book without necessarily reading the appendices, however they are there for further reference if needed.

The only function required in the "wordchain.py" program is detailed later in the assignment brief.

The following pages describe the requirements of both programs in detail.

Details of "wordchain.py"

"wordchain.py" is a program with a Command-Line Interface (CLI) like that of the programs we have created throughout most of the unit. The program (without optional additions and enhancements) can be implemented in under 150 lines of code – If your program significantly exceeds this, ask your tutor for advice. Everything you need to know in order to develop this program is covered in the first seven modules of the unit. This program should be developed before "logviewer.py".

This program will implement a simple word game which can be played by multiple users who take turns on the same computer, although you will be playing it alone as you write and test your code.

The game begins by asking how many people wish to play, and then prompting you to enter a name for each of the players. The game will then continually cycle through each player and ask them to enter a word that matches the following criteria:

- The word must start with the letter that the previous word ended in.
  - e.g. If the previous word was "duck", the next word must start with "k".
  - For the first word of the game, a random letter of the alphabet is chosen.
- The game will randomly select whether the word must be a noun, a verb or an adjective.
- The word must not have been previously used in this game.

If the player enters a word that matches all of the criteria, the length of the "word chain" increases by one and the game moves on to the next player. For example, if the game asked you to enter a verb starting with "t", then "try" would be valid (as long as it had not been used earlier in the game).

The goal of the game is to obtain the longest word chain by continually entering valid words. It is not intended to be competitive or require quick thinking – there are no winners or losers, and no time limit for entering a word. You are likely to get bored of playing before you break the chain.

Once the word chain is broken (which ends the game), the program should add "log" of the game to a text file named "logs.txt". Use the "`json`" module to read and write data from/to the text file in JSON format (see Reading 7.1). Each log of a game should be a *dictionary* consisting of three keys:

- **"players"**: an integer of the number of players, e.g. 3
- **"names"**: a list of strings of the player names, e.g. ['Andrea', 'Beth', 'Carol']
- **"chain"**: an integer of the final chain length of the game, e.g. 21

The logs should be stored in a list, resulting in the file containing a *list* of *dictionaries*. The example to the right demonstrates a list of two logs.

After adding the log to the text file, the program ends.

JSON

```
[
    {
        "players": 3,
        "names": ["Andrea", "Beth", "Carol"],
        "chain": 21
    },
    {
        "players": 2,
        "names": ["Homer", "Marge"],
        "chain": 15
    }
]
```

Requirements of "wordchain.py"

In the following information, numbered points describe a *requirement* of the program, and bullet points (in italics) are *additional details, notes and hints* regarding the requirement. Ask your tutor if you do not understand the requirements or would like further information. The requirements are:

1. The first thing the program should do is create some variables that will be needed later. You do not need to use the same variable names and may find the need for additional variables. Create:

- *A `chain` variable containing 0 – this will be used to keep track of the length of the word chain.*
- *A `wordTypes` variable containing a list of 3 strings – "noun", "verb", "adjective". This will be used when randomly selecting a type of word for the user to enter.*
- *A `playerNames` variable containing an empty list – player names will be added to this list.*
- *A `usedWords` variable containing an empty list – words will be added to this list as they are used.*

2. The program should then welcome the user to the game, and prompt them to enter how many people wish to play (minimum of 2). If the user enters something invalid (not an integer, or less than 2), the program should continually re-prompt the user until they enter valid input.

- *There are some examples in Lecture 3 and tasks in Workshop 4 covering what is required.*

3. After obtaining the number of players, the program should prompt the user to enter a name for each of them. Append the names to the `playerNames` list you created earlier.

- *Use your `inputWord()` function (detailed below) to prompt the user for a player name and to ensure that they are re-prompted until they enter something valid (only letters and at least one long).*

4. Once you have a list of names, the game can begin – enter a loop that will repeat as long as the word chain remains unbroken. The body of this loop must:

4.1. Randomly select a type of word from the list of `wordTypes`.

- *The random.choice() function can be used to select a random item from a list. It can also select a random character from a string, which can be used to select a random letter of the alphabet.*

4.2. If the `chain` is 0 (i.e. if it's the first word), randomly select a letter of the alphabet to be the starting letter. Otherwise, set the starting letter to the last letter of the previous word.

4.3. Print a message specifying which player's turn it is and the criteria of the word they must enter, and then prompt the user to enter a word.

- *Use your `inputWord()` function (detailed below) to prompt the user for a word. Store the word in lowercase – this will make other parts of the program more convenient.*
- *You will need to figure out how to keep track of the current player using a variable and print their name using the `playerNames` list. The number of players variable may be useful here.*

4.4. Check if the word is valid by making sure that the first letter of the word matches the letter specified (see Requirement 4.2) and that the word does not exist in the `usedWords` list.

- *Make sure the letters/words are lowercase so you can compare them in a case-insensitive way.*
- *If the first letter is not correct or the word has been used before, print an appropriate message and the loop/game should end.*

4.5. Check if the word is recognised and of the specified word type. This will be done using a free online dictionary service by "Wordnik". Your code will send a request via the web that asks for up to 5 definitions of a specified word and word type, e.g. "potato" and "noun". Any matching definitions will be sent back as a list of dictionaries in JSON format. If the list is empty, then the word was not recognised or is not of the specified type.

- *Since interacting with APIs over the web is not covered in the unit, **there will be a Blackboard discussion board post going over the process in detail**. It only takes a few simple lines of code!*
- *If the word is not recognised, print an appropriate message and the loop/game should end.*

4.6. If the word is valid, add 1 to `chain`, congratulate the player and print the word's definitions. Remember to also append the word to the `usedWords` list.

- *Printing the definitions simply involves looping through the list of dictionaries received from Wordnik and printing the "text" key of each dictionary (which contains the word's definition).*

5. After the chain is broken and the loop/game ends, the program should print the final chain length and record a log of the game as detailed in the previous section of the assignment brief.

- *First create a dictionary with keys of "players", "names" and "chain" and values of the number of players, `playerNames` list and `chain`.*
- *Then, try to open a file named "logs.txt" in read mode and use json.load() to load the JSON data from the file into a variable named `logs` and then close the file. If any exceptions occur, set the `logs` variable to an empty list. This will occur the first time the program is run, since the file will not exist.*
- *Finally, append the log dictionary to the `logs` list, and then open "logs.txt" in write mode and use json.dump() to write the `logs` list to the file in JSON format, then close the file.*
- *We are reading the data from the file into a variable, appending the log to the variable, then writing the variable to the file. This will work better than trying to append the log directly to the file.*

This concludes the core requirements of "wordchain.py". The following pages detail the function mentioned above, the additional requirements for CSP5110 students, optional additions and enhancements, and an annotated screenshot of the program.

Remember that you are required to submit pseudocode for your design of "wordchain.py".

The "inputWord" Function

You must define and use a function named "inputWord" as part of "wordchain.py". The function should accept one parameter named `prompt` – a string containing the message to display before waiting for input. The function should repeatedly prompt the user for input until they enter a value which consists entirely of letters and is at least one character long (i.e. the input cannot be blank or contain numbers, spaces, symbols, etc.). The function should then return the value.

- *Lecture 7 covered a string method that can be used to check if a string is entirely alphabetic.*

You are welcome to define and use additional functions if you feel they improve your program.

Additional "wordchain.py" Requirements for CSP5110 Students

If you are in CSP5110, the following additional requirements apply. If you are in CSP1150, you do not need to do implement these requirements (but you are encouraged to do so if you want). Ask your tutor if you do not understand any of the requirements or would like further information.

1. When entering player names, make sure that they are all different. If a name that has already been added to the `playerNames` list is entered, prompt the user to enter a different name.

2. At the very end of the program, obtain a random word from Wordnik and display it, along with the word's definition. This will require you to send at least one request to the Wordnik API.

Optional Additions and Enhancements for "wordchain.py"

The following are suggestions for optional additions and enhancements that you can implement to demonstrate deeper understanding of programming and deliver a product which is more "polished". They are *not required*, but you are encouraged to implement them (and others) if you can.

- Ensure correct pluralisation and grammar in everything that the program prints, e.g. ensure that you print "an adjective" rather than "a adjective" and "the chain is 1 link" long rather than "the chain is 1 links long".

- Include the date and time that the game was played as an additional key in the log details that you store into the text file at the end of the game.

- During the game, keep track of how many valid nouns, verbs and adjectives have been entered and include these details as new keys in the log of the game.

- Gradually increase the difficulty of the game by starting out with a minimum word length of 3, and increasing it by 1 every 2 "rounds" (i.e. after each player has entered 2 words). Be sure to show the minimum word length when prompting the user for a word, and check the word against the minimum length after they enter it.

Program Output Example

To help you to visualise the program, here is an example of the program being used:

Note: To save space, I have truncated long definitions and only included the first definition per word.

```
Welcome to WordChain!
How many players (minimum of 2)?: 3
Enter name of player 1: Andrea
Enter name of player 2: Beth
Enter name of player 3: Carol

Andrea is up next.
Enter a noun beginning with "H"!
> hotel
Good job, Andrea - "hotel" is a noun defined as...
  • An establishment that provides lodging and usually meals an

The word chain is now 1 link long!

Beth is up next.
Enter a noun beginning with "L"!
> lobster
Good job, Beth - "lobster" is a noun defined as...
  • Any of several edible marine crustaceans of the family Homa

The word chain is now 2 links long!

Carol is up next.
Enter an adjective beginning with "R"!
> rotten
Good job, Carol - "rotten" is an adjective defined as...
  • Being in a state of putrefaction or decay; decomposed.

The word chain is now 3 links long!

Andrea is up next.
Enter an adjective beginning with "N"!
> new
Good job, Andrea - "new" is an adjective defined as...
  • Having been made or come into being only a short time ago;

The word chain is now 4 links long!

Beth is up next.
Enter a noun beginning with "W"!
> water
Good job, Beth - "water" is a noun defined as...
  • A clear, colorless, odorless, and tasteless liquid, H2O, es

The word chain is now 5 links long!

       ... and so on until an invalid word is entered ...

Andrea is up next.
Enter an adjective beginning with "K"!
> kitch
Word chain broken - "kitch" does not appear to be an adjective!

Final chain length: 21
Game log saved.
```

The program welcomes the user, then asks the user to enter the number of players.

The program then asks the user to enter a name for each player.

Beginning with the first player, the game asks for a word of a random type – a noun. Since this is the first word of the game, the starting letter is randomly chosen.

Andrea answers correctly, the word's definition(s) are shown and the game continues…

The next player, Beth, is asked for a word beginning with the last letter of the previous word – L. The random word type is again a noun.

Beth answers correctly, the word's definition(s) are shown and the game continues…

The next player, Carol, is asked for a word beginning with the last letter of the previous word – R. The random word type is an adjective.

Carol answers correctly, the word's definition(s) are shown and the game continues…

Back to Andrea for a word beginning with the last letter of the previous word – N. The random word type is an adjective again.

Andrea answers correctly, the word's definition(s) are shown and the game continues…

Beth is up, needing a noun beginning with W.

She answers correctly, the word's definition(s) are shown and the game continues…

The game continues in this manner...

Oh no, Andrea has misspelt "kitsch" (can't blame her) and so the game ends!

The final chain length is shown and the log of the game is stored before the program ends.

Details of "logviewer.py"

"logviewer.py" is a program with a Graphical User Interface (GUI), as covered in Module 9. The entirety of this program can be implemented in under 150 lines of code – If your program exceeds this, ask your tutor for advice. Everything you need to know in order to develop this program is covered in the first nine modules of the unit. This program should be developed after "wordchain.py". To ensure compatibility and consistency, you must use the "`tkinter`" module to create the GUI. You will also need to use the "`tkinter.messagebox`", and "`json`" modules.

This program uses the data from the "logs.txt" file. The program should load all of the data from the file *once only* - when the program begins. The program simply allows the user to view the logs created by "wordchain.py" as well as some basic statistical information.

The user can press the "Next Log" button to advance through the logs until they reach the last log. Once they have reached the last log, clicking the button should display a "no more logs" messagebox.

Clicking the "Show Stats" button should show a messagebox that displays the total number of games, average number of players, and maximum chain length of all logged games.

See the content of Module 9 for examples of creating a class for your program's GUI. A starter file has been provided to assist you. The following pages detail the requirements of the program.

Constructor of the GUI Class of "logviewer.py"

The **constructor** (the "`__init__`" method) of your GUI class must implement the following:

1. Create the main window of the program and give it a title of "WordChain Log Viewer".
   - *You may also wish to set other settings such as the minimum size of the window.*

2. Try to open the "logs.txt" file in read mode and load the JSON data from the file into an attribute named "`self.logs`", and then close the file.
   - *If any exceptions occur (due to the file not existing, or it not containing valid JSON data), show an error messagebox with a "Missing/Invalid file" message and use the "`destroy()`" method on the main window to end the program. Include a "`return`" statement in the exception handler after destroying the main window to halt the constructor so that the program ends cleanly.*

3. Create a "`self.nextLog`" attribute to keep track of the next log to display and set it to 0.
   - *This attribute represents an index number in the list of logs loaded from the text file (`self.logs`).*

4. Use `Frame`, `Label` and `Button` and widgets from the "`tkinter`" module to implement the GUI depicted and described on the previous page.
   - *You will save time if you design the GUI and determine exactly which widgets you will need and how to lay them out before you start writing the code.*
   - *You are welcome change the layout of the GUI, as long as the functionality is implemented.*
   - *Do not set the text for the labels that will contain log data at this point – they will be set in the "`showLog()`" method.*
   - *See Reading 9.1 for information regarding various settings that can be applied to widgets to make them appear with the desired padding, colour, size, etc.*
   - *The "fill" and "anchor" settings may be useful – See Reading 9.1 to find out how to use them!*

5. Finally, the constructor should end by calling the "`showLog()`" method to display a the first log in the GUI, and then call "`tkinter.mainloop()`" to start the main loop.

That is all that the constructor requires. The following page detail the methods needed in the program, the additional requirements for CSP5110 students and optional additions and enhancements. You are *not* required to submit pseudocode for your design of "logviewer.py", as pseudocode is not particularly well suited to illustrating the design of an event-driven GUI program.

Methods of the GUI Class of "logviewer.py"

Your GUI class requires two methods to implement the functionality of the program - "`showLog()`" and "`showStats()`". As part of the GUI class of "logviewer.py", you must write these methods. I have described the required functionality of the methods rather than giving step-by-step instructions: It is up to you to design and implement the necessary code.

1. The "`showLog()`" method is responsible for displaying a the details of a log in the GUI.
   - *This method is called at the end of the constructor (to display the first log) and whenever the "Next Log" button is clicked.*
   - *Use the `self.nextLog` attribute to select a dictionary of log data from the `self.logs` list.*
   - *Add 1 to `self.nextLog` so that the next log is selected when the method is next called.*
   - *Show a "no more logs" messagebox if `self.nextLog` is outside the range of `self.logs`.*
   - *The "`configure()`" method (see Reading 9.1) can be used on a widget to change the text it displays. Alternatively, you can use a `StringVar` to control the text of a widget.*

2. The "`showStats()`" method is responsible for determining some simple statistics about the logs in `self.logs`, and showing a messagebox containing those statistics.
   - *This method is called whenever the "Show Stats" button is clicked.*
   - *The method must calculate the total number of games played (i.e. how many logs are in the list), the average number of players per game (rounded to the nearest integer), and the maximum chain length of all games.*
   - *How you determine this information is up to you, but is likely to involve looping through `self.logs` and using variables to count / keep track of things as needed.*

These two methods are all that are required to implement the functionality of the program, but you may write/use additional methods if you feel they improve your program.

Additional "logviewer.py" Requirements for CSP5110 Students

If you are in CSP5110, the following additional requirements apply. If you are in CSP1150, you do not need to do implement these requirements (but you are encouraged to do so if you want). Ask your tutor if you do not understand any of the requirements or would like further information.

1. The layout and presentation of your GUI will be assessed more critically. Make appropriate use of frames, alignment, fill/expand and padding settings to create a clear and consistent layout. Use font colour, size or styles to make important text stand out.

2. During the constructor, check if the `self.logs` list contains only one item. If so, disable the buttons and show a messagebox saying "Only 1 log found - Navigation and statistics disabled."

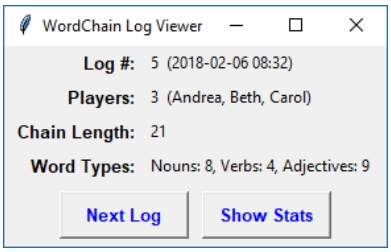
Optional Additions and Enhancements for "logviewer.py"

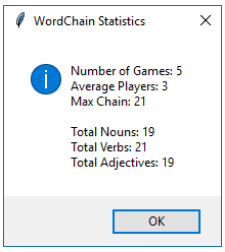
The following are suggestions for optional additions and enhancements that you can implement to demonstrate deeper understanding of programming and deliver a product which is more "polished". They are *not required*, but you are encouraged to implement them (and others) if you can.

- Enhance the program's log navigation capabilities to allow for more than simply going from start to end. You could make the program go back to the start after reaching the end, implement a "Previous Log" button, and possibly also "First Log" and "Last Log" buttons.

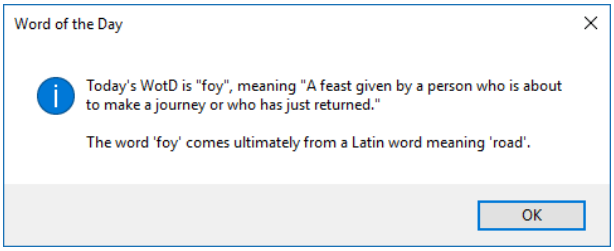
- Allow the user to enter a log number and go directly to that log. This can be achieved via an `Entry` widget, or a dialog box (research "tkinter.simpledialog" and "askinteger").
    - If you implement this and/or the previous addition, try to do so as efficiently as possible with minimal repeated code – e.g. create a method that receives a log number as a parameter and shows it, which can be used for all navigational needs.

- If you implemented the second and/or third optional addition to "wordchain.py", make sure that these additional details are included when viewing logs and statistics, e.g.





- Include a "Word of the Day" button in the GUI that, when clicked, shows a messagebox containing Wordnik's word of the day for the current date. The messagebox should also include the word's definition (the first one, if multiple are provided) and "note" text, e.g.



Submission of Deliverables

Once your assignment is complete, submit both the **pseudocode for wordchain.py** (PDF format) and the **source code for "wordchain.py" and "logviewer.py"** to the appropriate location in the Assessments area of Blackboard. An assignment cover sheet is not required, but be sure to **include your name and student number at the top of all files (not just in the filenames)**.

Referencing, Plagiarism and Collusion

The entirety of your assignment **must be your own work** (unless otherwise referenced) and produced for the current instance of the unit. Any use of unreferenced content you did not create constitutes plagiarism, and is deemed an act of academic misconduct. All assignments will be submitted to plagiarism checking software which includes previous copies of the assignment, and the work submitted by all other students in the unit.

Remember that this is an **individual** assignment. *Never give anyone any part of your assignment – even after the due date or after results have been released. Do not work together with other students on individual assignments – helping someone by explaining a concept or directing them to the relevant resources is fine, but doing the assignment for them or alongside them, or showing them your code is not appropriate.* An unacceptable level of cooperation between students on an assignment is collusion, and is deemed an act of academic misconduct. If you are uncertain about plagiarism, collusion or referencing, simply contact your tutor, lecturer or unit coordinator and ask.

You may be asked to **explain and demonstrate your understanding** of the work you have submitted.

Marking Key

Marks are allocated as follows for this assignment. The marks are divided between both programs.

Criteria	Marks
Pseudocode These marks are awarded for submitting pseudocode which suitably represents the design of your source code. As there are no fixed standards for pseudocode, it will be assessed on the basis of "does it help in understanding/describing the structure of the program?"	**5**
Functionality These marks are awarded for submitting source code that implements the requirements specified in this brief, in Python 3. Code which is not functional or contains syntax errors will lose marks, as will failing to implement requirements as specified.	**15**
Code Quality These marks are awarded for submitting well-written source code that is efficient, well-formatted and demonstrates a solid understanding of the concepts involved. This includes appropriate use of commenting and adhering to best practise.	**10**

Total:	**30**