Programming Principles (CSP1150)

Assignment: Individual programming assignment ("Typing Test" program)
Assignment Marks: Marked out of 20, worth 20% of unit
Due Date: 7 April 2025, 9:00AM

A Message Regarding Academic Integrity

In recent semesters this unit has experienced academic misconduct, including seeking or paying for solutions online, using the work of other CSP1150 students, generating code using AI, and working on individual assessments with other students. These incidents have resulted in students receiving zero marks for the assessment or entire unit, suspensions, and even expulsion from the university.

If you start your assignments in a timely fashion, engage with the unit content, and regularly seek help and feedback from unit staff and ECU learning support staff, these issues will not apply to you.

Background Information

This assignment tests your understanding of and ability to apply the programming concepts we have covered in the unit so far, including the usage of variables, input and output, data types and structures, selection, iteration and functions. Above all else, it tests *your ability to design and then implement a solution to a problem* using these concepts.

Contact your tutor if you have questions or do not understand any part of the assignment, or if you would like help or feedback on your work. You should do this *at least once* prior to submission.

Assignment Overview

You are required to design and implement a program that allows the user to complete a typing test, assessing them on their speed and accuracy when typing sentences. The user can the number of rounds and whether the test should be case-sensitive at the start of the program, and after the final round the program displays a summary and breakdown of their results.

Details of the program requirements can be found in the "Program Requirements" section.

The entirety of this program can be implemented in under 125 lines of code (including the list of sentences, but not including any optional additions). *This number is not a limit or a goal – it is based upon an average quality implementation that includes comments and blank lines. It is simply provided to prompt you to ask your tutor for advice if your program significantly exceeds it.*

Tip: You can (and should) make a start on this assignment as early as the end of **Week 2**.

When starting out with programming, it is common to encounter difficulties that temporarily slow or stop your progress. The sooner you start, the more time you have to work through difficulties.

Program Output Example

To help you visualise the program, here is an example screenshot of the program being run:

```
Welcome to Greg's Typing Test!
Choose number of rounds (3-20): 3

In strict mode, your input is case-sensitive.
Would you like to play in strict mode (y/n)?: y

Okay, this test will last 3 rounds, and your input is case-sensitive.
Press Enter to begin!

Round 1 of 3.  Your sentence is:
  The C programming language was developed by Dennis Ritchie in 1972.
> The C programming language was developed by Dennis Ritchie in 1972.
Accuracy: 100%, WPM: 48.
Press Enter to continue.

Round 2 of 3.  Your sentence is:
  SQL, or Structured Query Language, is used in relational databases.
> SQL, or Structured Query Language, is used in telational databases.
Accuracy: 99%, WPM: 51.
Press Enter to continue.

Round 3 of 3.  Your sentence is:
  The first iPhone, running iOS, was introduced by Apple in 2007.
> The first iphone, running ios, was intrroduced by Aple in 2007
Accuracy: 93%, WPM: 41.
Press Enter to continue.


Test Complete!
Results:
  Highest Accuracy: 100%
  Average Accuracy: 97%
  Highest WPM: 51
  Average WPM: 47

Breakdown:
  Round  Accuracy  WPM
  -----  --------  ---
  1      100%      48
  2      99%       51
  3      93%       41
```

In this example, the user chooses three rounds and to play in strict mode. The program confirms the details of the test and waits for the user to press Enter to begin.

Once the user presses enter, the program goes through three rounds, displaying a randomly selected sentence and waiting for the user to type it and press Enter each time. They are told their accuracy and words per minute each round. After the final round, their results are shown, including a round-by-round breakdown.

Text you type in the IDLE environment will be coloured as if it was code, which is why words like "in" are appearing in orange in the screenshot. This does not cause any issues and should simply be ignored.

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* as well as the code for the 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 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 or a single section of the program, and then revise and expand 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 and the Rubric and Marking Criteria document for guidance regarding 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 any parameters that the function receives and what the function returns back to the program. Pseudocode for functions should be presented *after* the pseudocode for the main part of your 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.

Two functions are required in this assignment (detailed later in the assignment brief).

Tip: Do not attempt to implement the entire program at once. Work on one small part (a single requirement or even just *part* of a requirement) and only continue to another part once you have made sure that the previous part is working as intended *and that you understand it*.

It can also be useful to *create separate little programs* to work on or test small sections of code. This allows you to focus on just that part without the rest of the program getting in the way, and allows you to work on a section without necessarily having completed prior/other sections.

Program Requirements

In the following information, numbered points describe a *core 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.

1. Define a list of about 20 strings containing the sentences that the program can select for tests. The sentences should contain about 10 words each and include some proper nouns and acronyms.

2. Print a message welcoming the user to your typing test (include your name in the message) and, then prompt them to choose a number of rounds between 3 and the number of items in the sentence list. Then, ask the user whether they would like to play in "strict mode" (in which the test is case-sensitive). The program should re-prompt the user until they enter valid responses, and display error messages if they enter something invalid.

- *Store the number of rounds as an integer, and whether they are playing in strict mode as a boolean.*
- *The program should not crash if a non-integer value is entered when prompted for number of rounds.*

3. Print a message confirming the settings of the test (rounds and strict mode), and create a list of sentences to use for the test (one for each round), selected randomly from the list of sentences. Then, use the "`input()`" function to display a "Press Enter to begin" message.

- *Use "`random.sample()`" to ensure that the sentences selected for the test are all different.*
- *The `input()` function is used here (and in Requirement 4.2) simply to make the program wait until the user presses Enter. There is no need to assign the input to a variable in these cases.*

4. Enter a loop that goes through each of the sentences selected for the test. In that loop…

4.1. Print which round the user is up to out of the total number of rounds, e.g. "Round 1 of 5", and then print the current sentence. Prompt for the user's input, recording the current timestamp immediately before and after the `input()` function to calculate the time taken.

- *The "`time.time()`" function returns the current timestamp as a float (in seconds, with decimal places). Subtract the earlier timestamp from the later one to get the number of seconds taken.*

4.2. Calculate the accuracy and words per minute (WPM) of the user's input by calling the "`calculate_accuracy()`" and "`calculate_wpm()`" functions (detailed on Page 5) and display them, followed by using "`input()`" to display "Press Enter to continue".

- *Round accuracy and WPM values to the nearest integer whenever you display them, and display accuracy as a percentage (e.g. "78%").*

5. After completing all rounds/sentences, the loop from Requirement 4 ends. Display the user's results, consisting of their highest and average accuracy and their highest and average WPM.

- *Implementing this (and Requirement 6) involves adding code to earlier parts of the program to record information during the test, e.g. Using lists to store the accuracy and WPM of each round's input.*

6. Finally, display a round-by-round breakdown of the test (as pictured on Page 2) consisting of the round number, accuracy and WPM of each round's input.

The "calculate_accuracy" Function

Create a function named "`calculate_accuracy()`" and use it when implementing Requirement 4.2. The function requires three parameters:

- "`target_sentence`", a string of the sentence the user was asked to type.
- "`user_input`", a string of what the user typed.
- "`strict`", a boolean of whether the use chose strict mode for the test.

The function should compare the `target_sentence` and `user_input` strings and return an integer between 0 and 100 based upon their similarity, i.e. the user's accuracy. If `strict` is False, make sure that the comparison is *not* case-sensitive, e.g. by converting both strings to lowercase before comparing them.

> **Important Note:** Guidance regarding *how to compare the strings* will be provided via a Discussion on the unit site. We will take an approach that strikes a balance between robustness and simplicity.

The "calculate_wpm" Function

Create a function named "`calculate_wpm()`" and use it when implementing Requirement 4.2. The function requires two parameters:

- "`user_input`", a string of what the user typed.
- "`time_taken`", a float of the number of seconds taken to enter the input.

The function should split `user_input` into words using the "`.split()`" string method and use the number of words and the `time_taken` to calculate words per minute. Return the WPM rounded to the nearest integer.

Ensure that the functions do exactly what is specified above and nothing more – it is important to adhere to the stated specifications of a function, and deviating from them will impact your marks. If you are thinking of creating *additional functions* in your program, please consult with your tutor. Creating functions that add to the complexity of a program without benefiting it is a common issue.

> **Tip:** Check the Discussions on the unit site on a regular basis – extra tips, examples and advice regarding the assignment will be posted there throughout the semester. You are also welcome to make your own posts if you have questions – but don't upload your work to the discussion board!
>
> Email your work to your tutor *at least once* while working on your assignment, so that they can give you advice and feedback on it. This will definitely help you to improve your work (and your mark)!

Optional Additions and Enhancements

Below are some suggestions for minor additions and enhancements that you can make to the program to further test and demonstrate your programming ability. They are *not required* and you can earn full marks in the assignment without implementing them.

- When prompting the user for the number of rounds and whether to play in strict mode at the start of the program, make it so they can press enter without typing anything to select the maximum number of rounds and to play in strict mode.
  You can make it so that the default is not to play in strict mode if you prefer.

- Allow the user to input "x" during the test to end it prematurely, skipping remaining rounds and going to the results (do not include the round where "x" was entered in the results). Be sure to mention this feature before the test starts, and account for the possibility of the user entering "x" in the first round.

- After displaying the results, if the user's average accuracy is at least 90% and average WPM is at least 40, print "Well done!" If their average accuracy is at least 95% and average WPM is at least 60, print "Excellent!"

- Adjust the "`calculate_wpm()`" function so that it considers the accuracy of the user's input when calculating WPM. Make the function receive the accuracy as a parameter, divide the accuracy by 100 (so that it is a float between 0 and 1) and multiply that by the WPM.
  This ensures that low accuracy input reduces the WPM.

- After displaying the results and breakdown, ask the user if they would like to start again. If they enter "y" or "yes", return to the start of the program. Otherwise, end the program.

Referencing Out-of-Unit Code

If your program contains any built-in functions or programming techniques *not* covered in the Core Content of a module from this unit, you must reference them within your code. This simply requires a comment above the relevant line(s) of code containing a link to the specific webpage where you learned of the function/technique. If your source of knowledge is not a website (e.g. a reading, textbook, or prior learning), reference or explain it in a comment as clearly as possible.

Failure to reference out-of-unit functions or programming techniques will result in *losing marks* in the Code Quality section of the marking key/rubric. If you are not sure whether something was covered in the Core Content of a module, it is recommended that you reference it.

Submission of Deliverables

It is strongly recommended that you send your work to your tutor for feedback at least once prior to submitting it. Once your assignment is complete, submit both your **pseudocode** (in PDF format) and **source code** (".py" file) to the appropriate location in the Assignments area of the unit site. Be sure to **include your name and student number at the top of both files** (not just in the filenames).

Academic Integrity and Misconduct

The **entirety of your assignment must be your own work** (unless otherwise referenced) and produced for the current instance of the unit. Any unreferenced content you did not create is academic misconduct (plagiarism). Assignments are submitted to plagiarism checking software which includes previous assignments, and the work submitted by all students in the unit. Submissions are also manually compared and checked. Do not make your work available online, during or after 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 – you can help someone by explaining a concept or directing them to the relevant resources, but doing any part of the assignment for them or alongside them, or showing them your work, is inappropriate.* An unacceptable level of cooperation between students on an assignment is academic misconduct (collusion). *Using AI tools (e.g., ChatGPT) is also inappropriate for assignment work. If you are having trouble with your assignment, please talk to/email your tutor.* If you are uncertain about plagiarism, collusion or referencing, simply ask unit staff.

You may be asked to **explain and demonstrate your understanding** of the work you have submitted. Your submission should accurately reflect *your* understanding and ability to apply the unit content.

Marking Key

Criteria	Marks
Pseudocode These marks are awarded for submitting pseudocode which suitably represents the design of your source code. Pseudocode will be assessed based on whether it clearly describes the steps of the program in English, and whether the program is well structured.	**6**
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/functions as specified.	**8**
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, referencing external sources, and adhering to best practise.	**6**

Total:	**20**

See the "Rubric and Marking Criteria" document provided with this brief for further details!