**Rangitoto College Internal Assessment Resource**

**Achievement Standard:** 91897

**Standard title:**  Use advanced processes to develop a digital technologies outcome

**Credits:** 6

**Version:** 1

**Resource title:** Typing Tester

| **Achievement** | **Achievement with Merit** | **Achievement with Excellence** |
| --- | --- | --- |
| Use advanced processes to develop a digital technologies outcome. | Use advanced processes to develop an informed digital technologies outcome. | Use advanced processes to develop a refined digital technologies outcome. |

**Student/Akonga instructions**

**Introduction/Kupu Arataki**

This assessment activity requires you to plan, trial, test and develop a computer program. You will utilise a development process to help you make informed decisions throughout the coding, testing and trialling of your program and show ongoing refinement to improve the functionality and quality of your program.

You will be assessed on how effectively you plan your development, decompose the outcome into smaller components, and test and refine your program. You will not be assessed on the specific tools and techniques used to develop the program, however, you will be assessed on how well you have applied your chosen tools and techniques (based on testing and trialling) to develop a high-quality outcome (e.g. follows PEP8, PEP257, well-structured, logical, flexible, robust and comprehensively tested).

You have approximately 7 weeks to complete this task.

There is a Term 1 Week 10 checkpoint when you are expected to submit an appropriate amount of work on your project (including all evidence to date) to your teacher.

The final completion date for this assessment is Friday 21st of May.

**Task/Hei Mahi**

Scenario

Typing is currently an invaluable skill. How else are you going to write code? Speech-to-text voice recognition has come a long way in recent years, yet it is still not widely socially accepted to speak to your ‘device’ (Google Glass didn’t exactly take off, did it?). You may also want to communicate with someone without having everyone else in the room hearing exactly what’s in your message. It is generally accepted that the only way to improve typing ability, like most things, is with practise. Training must be at a difficulty commensurate with a person’s skill. Difficulty can be based on length of words, the specific letters used in words, use of special characters, etc. Ability is then measured in words per minute, with accrruac yplaaying acrucial role.

Your task is to make a program that tests/trains typing ability. The type of program you make to do this is largely down to you.

There are many examples of different typing programs/games at <http://www.wordgames.com/>. You may investigate these at home to analyse and select possible components/gameplay functionality for your program or you can design your own from scratch. You are not marked for your design.

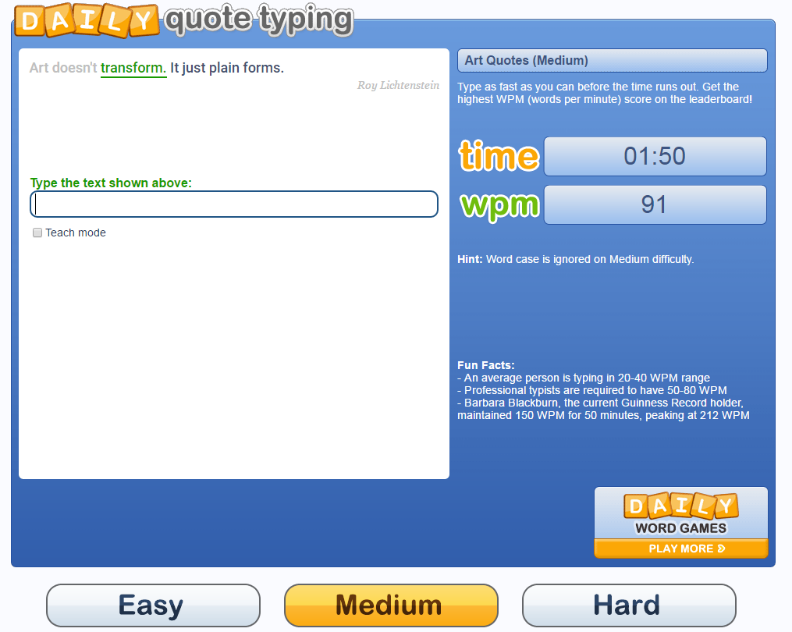
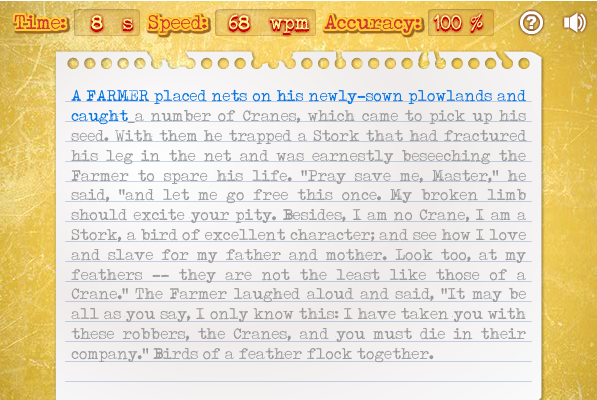
Some versions have objects travelling towards the player or down the screen and require you to enter the text before it reaches the player (this one shoots at them as you get each letter)



<http://zty.pe/>

There are also other versions where you copy out sentences and it measures your time, words per minute (wpm) and accuracy.

There are features that are included in all/most of these “games” (i.e. user instructions, time taken, accuracy, wpm, randomness, GUI, replayable, stored text). You must integrate these in your program.



**Specifications for your program**

* **Use/Gameplay** should be **effective**
  + Your program should have **existing text** to **test** the **user against**. This may be dictionary type text, or it may be quotes/sentences/paragraphs etc. This should be **stored in a text file(s)** that your program reads.
  + Your program must **‘mark’** the **text entered** **against** the **supplied text** to **check correctness**. This may be at each letter or each word.
  + You must **measure** and **display** the **time taken**, the **speed in words per minute** (wpm) and the **accuracy**.
  + The program must be **able to run the test/game again** **within the same game instance** (e.g. you shouldn’t need to close your program and run it again in order to ‘play’ again).
  + There **should be some randomness** each time your program is run – it shouldn’t always use the same texts in the same order.
  + The program should **not crash through incorrect file formats**. If the text files are incorrect, they should be rejected before program use begins.
  + Your program **must use a Graphical User Interface** (GUI)
* **Program code** should be **flexible**
  + You must **create and use functions** where appropriate
  + You must follow **good coding standards** and use **global variables appropriately**

Steps

1. Decide on an appropriate planning methodology, and what project management and version control tools you will use to manage your program development.
2. Set up any necessary planning/project management tools.
3. Research any content for your program and use your chosen methodology/tools to plan out the development and structure of your program. For example:
   * Will there be levels, lives, time limits?
   * Explain what relevant implications are important to consider in the development of your program?
   * Decompose your program into the different components you need to incorporate into the final program (e.g. start-up screens/GUI interface, different levels, sub-programs/functions/methods).
4. Throughout your development, you must trial multiple components. For example, this could include different ways to present the user interface, different ways of keeping score, etc. Select the best components to include in your final program, based on the results of your testing and trialling.
5. Use your selected version control tools/techniques to save successive versions of your code and keep evidence of how you created the program in an ongoing manner (e.g. screenshots showing your file structure with appropriately named versions/program components, including brief annotations of the changes made in each version).
6. Ensure your testing and trialling includes both expected cases and relevant boundary cases (e.g. what happens when the wpm gets to a certain number?). You may want to get other students and/or your family/whanau to test your program at each stage and provide feedback to help you improve your final program. Using others to test the program will help to ensure it is comprehensively tested for many different cases (including expected and relevant boundary cases). Note the improvements that could be made based on the testing and implement your changes.
7. Throughout the development of your program code, ensure that you document your program with appropriate variable/module names and comments that describe code function and behaviour. Follow the common conventions of your programming language (e.g. naming conventions or rules for program layout, i.e. PEP8).
8. Comprehensively test your final program to ensure that it functions correctly and is of high-quality (e.g. bug free, has a well-presented and easy-to-use interface, contains all the required information).
9. Discuss how the information from planning, testing and trialling of the components of your game program assisted you to develop a high-quality outcome. Annotated screen shots may help with this.
10. Show how your program has addressed the relevant implications.

**Appendix**

Examples of relevant implications include:

* social – *suitable for people/groups who will use it and social situations it will be used in*
* cultural – *inclusive, inoffensive*
* legal – *copyright issues*
* ethical – *appropriate use of content*
* intellectual property – *rights of owner identified, rights for use*
* privacy – *appropriate access to sensitive information*
* accessibility – *considering wide range of users*
* usability – *easy to use, understand*
* functionality – *performs expected task, works as most would expect it to*
* aesthetics – *appearance suitable for intended user, follows basic design principles*
* sustainability and future proofing – *easy to maintain, update without major rebuild*
* end-user considerations – *meets requirements of intended user, suitable for their demographic*
* health and safety – *follows all health and safety codes (during process and for end users)*