

College of Engineering, Construction and Living Sciences Bachelor of Information Technology

IN628 Programming 4

Level 6, Credits 15

## Assessment Task

Assessment 2: Language Exploration



# Assessment 2: Language Exploration

**Due Date:** Friday, 22nd November, 5.00 pm – code freeze.

**Value:** 25% of your final mark.

**Group Size:** Individual.

## Learning Outcomes Met By This Assessment:

At the successful completion of this course, learners will be able to:

1. Program effectively in an industrially relevant programming language
2. Implement a wide range of intermediate data structures and algorithms to act as modules of larger programs
3. Use an appropriate integrated development environment to create robust applications

## Assessment Overview

For this assessment, you will use **Ruby** to build an implementation of the game, **Word Mastermind**. We will be covering the basic features of **Ruby** formally in class; you will be learning the more complex features independently. The main purpose of the assessment is not just to build a simple game, rather to demonstrate your ability to effectively learn a new programming language which differs, both programmatically and syntactically, from the familiar C-family languages.

**Word Mastermind** is a variation on the classic coloured-peg puzzle game **Mastermind**, but using words and having slightly different rules. In **Word Mastermind**, the computer **(codemaker)** chooses a word **(code)** and the player **(codebreaker)** tries to figure out the word. At each turn the **codebreaker** makes a guess. The **codemaker** provides feedback about the accuracy of the guess. Specifically, for each letter in the **codebreaker’s** guess, the **codemaker** indicates one of three outcomes:

* **Exact**: The letter is an exact match to the letter in the same position in the **code**
* **Near**: The letter is contained in the **code**, but is not in the correct position
* **Miss**: The letter is not contained in the **code**

For example:

|  |  |
| --- | --- |
| **Code** | piano |
| **Guess** | night |
| **Feedback** | near exact miss miss miss |

The **codebreaker** knows that ‘i’ is the second letter in the **code,** the first letter is not ‘n’, but is somewhere in the **code** and ‘g’, ‘h’ and ‘t’ are **not** contained in the **code**. The **codebreaker** is allowed a fixed number of guesses – the fewer the guesses, the more difficult the game. If the **codebreaker** guesses the **code** within the permitted number of guess, s/he wins the round.

In your implementation of **Word Mastermind,** you will use five letter words only. A list of words is provided as a text file, which your game must load when it is launched. You must use this; it may not be modified. The codes may only be words that contain **no duplicate letters** (e.g. piano is valid but aaron is not because it contains multiple occurrences of ‘a’). You must ensure, programmatically, that only legal words are selected from the loaded word list.

Your version must implement the core game play, with the specific functional requirements shown below. The code must be elegant, technically correct, architecturally sound and written in idiomatic **Ruby**. In addition, to demonstrate your mastery of the language syntax and semantics, you will provide detailed code commenting to explain the logic of your implementation, and to describe each of the syntactic elements you used to implement that logic.

**Assessment Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Assessment Activity** | **Weighting** | **Learning Outcomes** | **Assessment Grading Scheme** | **Completion Requirements** |
| Language Exploration | 25% | 1-3 | Percentage/CRA | 50% |

## Functional requirements. The application must:

|  |  |
| --- | --- |
| **1** | Launch without modification. |
| **2** | Be entirely console-based. Do not submit any GUI code. |
| **3** | Load its list of potential words from an external text file (word-list.txt) provided when  it is first launched. The word list may not be modified. |
| **4** | Randomly select a word **(code)** at each round. This word must not contain duplicate  letters or special characters. |
| **5** | Allow a fixed number of guesses for each round. Each guess is a five-letter word from  the keyboard by the player. |
| **6** | Provide feedback about the letters in the guess as described above. You may use  whatever text-based display format you like. Make sure it is clear to the player. |
| **7** | After each guess, display the number of remaining guesses in some way. |
| **8** | Clearly indicate a win or loss. |
| **9** | Allow the user to play as many rounds of **Word Mastermind** as s/he wishes, exiting  with an appropriate keystroke. Display this clearly to the user. |
| **10** | Not throw any exceptions or crash during runtime. |
| **11** | Fulfil the special commenting requirements discussed below. |

**Commenting**

As stated above, the primary purpose of the assessment is to demonstrate your ability to learn and use a new programming language. The most direct way for you to demonstrate your mastery of **Ruby** is to explain your code thoroughly via comments. In this assessment, your code comments are not for future reference, or for the convenience of the reader, as per normal. Your code comments are where you demonstrate how well you understand the code you are submitting. To gain the full marks for commenting you must have:

* A header comment for each method, which explains in detail the input, output, effect and computational logic of that method.
* Inline commenting for every computational statement which explains in detail the syntax and logic of the construct.

Make sure your comments don’t simply translate the **Ruby** commands into English. A fully commented submission will be completely clear, at both the syntactic and semantic levels, to a reader who has never seen **Ruby** code before.

## Marking Rubric

Attached at the end of this document.

## Submission Details

* Project files must be submitted via GitHub Classroom. Here is the link to the repository you will be using for submission – [https://classroom.github.com/a/5ZRJ- y9I](https://classroom.github.com/a/5ZRJ-y9I)
* Your primary code files must be named ***<your\_op\_username>.rb***. Only files with this exact naming format will be marked.
* If you submit a multi-file solution, you are responsible for ensuring that all secondary files are correctly included in the build – you will need to explore the **Ruby** syntax for this.

## Passing Criteria

To pass this assessment, you must gain a mark of 50%.

## Authenticity

All parts of your submitted assessment must be completely your own work and any references must be cited appropriately using APA 6th edition style referencing.

## Policy on Submissions, Resubmission and Resits

The school’s process in relation to submissions, extensions, resubmissions and resits complies with Otago Polytechnic Policies. Students can view policies on the Otago Polytechnic Website located at <http://www.otagopolytechnic.ac.nz/>. Students may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are completed within a short time frame (usually no more than 5 working days) and usually must be completed within the timing of the course to which the assessment relates. Resubmissions will be available to students who have made a genuine attempt at the first assessment opportunity. The maximum grade awarded for a resubmission will be C-.

## Extensions

Please familiarise yourself with the assessment due date. If you need an extension, please contact your Course Coordinator prior to the due date. If you require more than a week's extension, a medical certificate or support letter from you manager may be needed.

# Assessment 2: Language Exploration Assessment Rubric

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **10-9** | **8-7** | **6-5** | **4-0** |
| **Code Commenting** | All header comments thoroughly explain the input, output, effect and computational logic of each method.  All inline comments thoroughly explain the Ruby syntax and logic of construct of each computational statement. | Most header comments clearly explain the input, output, effect and computational logic of each method.  Most inline comments clearly explain the Ruby syntax and logic of construct of each computational statement. | Some header comments explain the input, output, effect and computational logic of each method.  Some inline comments explain the Ruby syntax and logic of construct of each computational statement. | Header comments not implemented or do not explain the input, output and computational logic of each method.  Inline comments not implemented or do not explain the Ruby syntax and logic of construct of each computational statement. |
| **Program Structure** | Program demonstrates thorough structure on all of the following:   * General architecture e.g., classes, methods, concise naming of variables * Idiomatic use of control flow and data structures * Sufficient modularity, e.g., classes, methods have a single purpose * Efficient algorithmic logic | Program demonstrates clear structure on most of the following:   * General architecture e.g., classes, methods, concise naming of variables * Idiomatic use of control flow and data structures * Sufficient modularity, e.g., classes, methods have a single purpose * Efficient algorithmic logic | Program demonstrates structure on some of the following:   * General architecture e.g., classes, methods, concise naming of variables * Idiomatic use of control flow and data structures * Sufficient modularity, e.g., classes, methods have a single purpose * Efficient algorithmic logic | Program does not demonstrate structure on any of the following:   * General architecture e.g., classes, methods, concise naming of variables * Idiomatic use of control flow and data structures * Sufficient modularity, e.g., classes, methods have a single purpose * Efficient algorithmic logic |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Functionality & Robustness** | Program opens without errors and does not need to be modified to be run.  Code is randomly selected from an external file of five-letter words at the start of each round. Code does not contain duplicate letters and special characters.  All guess feedback, remaining guesses and win/loss correctly computed and clearly displayed.  Play another round and exit the game with an appropriate.  Thorough handling of incorrectly formatted values. No exceptions thrown and other crashes during runtime. | Program does open, though needs to be modified to be run.  Code is randomly selected from an external file of five-letter words at the start of each round. Code does not contain duplicate letters, though contains special characters.  Most guess feedback, remaining guesses and win/loss computed correctly, though not displayed clearly.  Play another round and exit the game with an appropriate keystroke, though not displayed clearly.  Handling of incorrectly formatted values mostly implemented. No exceptions thrown, though other crashes during runtime. | Program needs to be modified to be open and run.  Code is randomly selected from an external file of five-letter words at the start of each round. Code does contain duplicate letters and special characters.  Some guess feedback, remaining guesses and/or win/loss computed, though not displayed.  Play another round and/or exit the game with an appropriate keystroke, though not displayed.  Some handling of incorrectly formatted values. Some exceptions thrown and/or crashes during runtime. | Program cannot be opened or program is empty.  Code is not randomly selected from an external file of five-letter words at the start of each round. Code may be hard-coded.  Minimal or no guess feedback, remaining guesses or win/loss computed and displayed.  No keystroke to play another round or exit the game.  Minimal or no handling of incorrectly formatted values. Frequent exceptions thrown and crashes during runtime. |

**Marking Cover Sheet**



Assessment 2: Language Exploration

**IN628 Programming 4**

Level 6, Credits 15

**Bachelor of Information Technology**

Name: Date:

Learner ID:

Assessor’s Name:

Assessor’s Signature:

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Out Of** | **Weighting** | **Final Results** |
| Code Commenting | **10** | **35** |  |
| Program Structure | **10** | **40** |  |
| Functionality & Robustness | **10** | **25** |  |
| **Final Result** | | | **/100** |
| This assessment is worth 25% of the final mark for the Programming 4 course. | | | |