College of Engineering, Construction and Living Sciences
Bachelor of Information Technology
IN628: Programming 4
Level 6, Credits 15

Assessment 02: Language Exploration

Assessment Overview

For this assessment, you will use Ruby to build an implementation of the game, Word Mastermind. We won't be covering the basic features of Ruby formally in class; you will be learning these 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, if the code is piano & the guess is night, the feedback is [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**
In Class Checkpoints	15%	1, 2, 3	CRA	Cumulative
Roguelike	45%	1, 2, 3	CRA	Cumulative
Language Exploration	25%	1, 2, 3	CRA	Cumulative
Theory Exam	15%	3, 4, 5	CRA	Cumulative

Conditions of Assessment

You will complete this Language Exploration assessment outside timetabled class time, however, there will be availability during the teaching sessions to discuss the requirements and progress of this assessment. This assessment will need to be completed by Friday, 12 June 2020 at 5pm.

Pass Criteria

This assessment is criterion-referenced with a cumulative pass mark of 50%.

Submission Details

You must submit your program files via **GitHub Classroom**. Here is the link to the repository you will be using for your submission – https://classroom.github.com/a/EW9FDA4F. For ease of marking, please submit the marking sheet with your name & student id number via **Microsoft Teams** under the **Assignments** tab.

Authenticity

All parts of your submitted assessment must be completely your work and any references must be cited appropriately.

Policy on Submissions, Extensions, Resubmissions & Resits

The school's process concerning **Submissions, Extensions, Resubmissions and Resits** complies with Otago Polytechnic policies. Students can view policies on the Otago Polytechnic website located at https://www.op.ac.nz/about-us/governance-and-management/policies.

Extensions

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

Resubmissions

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 resubmission will be C-.

Learning Outcomes

At the successful completion of this course, students 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.
4. Demonstrate sound programming and software engineering practices independent of the environment or tools used.
5. Explain the theoretical issues surrounding programming language design and development.

Instructions

Application Requirements - Learning Outcomes 1, 2, 3

The Language Exploration application must have the following functional requirements:

- Launch without modification
- Be entirely console-based. Do not submit any GUI code
- Load its list of potential words from an external text file (02-word-list.txt) provided when it is first launched. The word list may not be modified
- Randomly select a word (code) at each round. This word must not contain duplicate letters or special characters
- Allow a fixed number of guesses for each round. Each guess is a five-letter word from the keyboard by the player
- After each guess, display the number of remaining guesses in some way
- Clearly indicate a win or loss
- 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
- Not throw any exceptions or crash during runtime
- Fulfil the special commenting requirements discussed below

Code Commenting - Learning Outcomes 1, 2, 3

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.

IN628 Programming 4
Semester 1, 2020

Assessment 02: 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

OTAGO POLYTECHNIC
Te Kura Matatini ki Otago

Assessment 02: 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.			