# BIT246[:](https://webems.rmit.edu.vn/exchweb/bin/redir.asp?URL=http://blackboard.rmit.edu.vn/webapps/portal/tab/_1_1/%2520/webapps/portal/frameset.jsp?tab_id=_2_1%26url=%252fwebapps%252fblackboard%252fexecute%252flauncher%253ftype%253dCourse%2526id%253d_922_1%2526url%253d) Object Oriented RAD

# Assignment 2 – Software Application – Ruby Project

**Due Date: Week 6, Week 8 and Week 12**

**Date of Demonstration: Notify on class**

**Assignment Weightage: 35%**

**Total Possible Marks:**

**Submission1 (5 marks) + Submission2 (15marks) + Submission3 (15marks) = 35 marks**

A company often needs to retrieve documents on its hard drive based on those documents containing words or phrases. It thus needs a system that allows the user to enter the required phrase or word and to then find the document/s containing that phrase and to print them to screen.

**This Assignment contains HURDLE TASK – Question Answer session for all Submissions – the task that must be passed in order to pass the whole Assignment or Submissions of Assignment.**

The project is to be delivered in several Submissions:

**Submission 1 (10 marks – due week 6)**

Write Ruby code and answer following questions. Upload rb files (with comments) on Moodle:

1. Create a text file named Story1234.txt (1234 should replace by your student ID), type the following statements into the file and save it:

Ruby is an interpreted, high-level, general-purpose programming language. It was designed and developed in the mid-1990s by Yukihiro "Matz" Matsumoto in Japan. Ruby is dynamically typed and uses garbage collection and just-in-time compilation.

Next, create a Ruby program named Submission1Q11234.rb (1234 should replace by your student ID) and add the program logic needed to perform the following tasks:

* Verify that the file exists.
* If the file does not exist, display an error message.
* If the file exists, read it and test whether it contains the phrase “a little boy”
* If it does display the contents of the text file otherwise display phrase not found.
* Add appropriate comments e.g., Name, Student ID, functionality of a variable, brief explanation of logic and flow of your program embedded inside the program code.

1. Create a Ruby program file named Submission1Q21234.rb (1234 should replace by your student ID) and add the program logic needed to perform the following tasks:
   * Prompt the user to type the path and name of a folder whose contents will be displayed. Integrate your student id with the name of the variables to store path and name of the folder e.g. path1234 or folder1234.
   * Capture and analyze user input to determine whether the specified folder exists.
   * Tell the user whether or not the folder exists.
   * If the folder exists, prompt the user to press Enter in order to view the folder’s contents.
   * If the folder does not exist, display an error message. Integrate your student id with the error message e.g. “James - error1234 : Folder does not exists”.
   * Add appropriate comments e.g., Name, Student ID, functionality of a variable, brief explanation of logic

**Submission 2 (15 marks – due week 8)**

Create a Ruby program file named Submission2Q11234.rb (1234 should replace by your student ID) and add the program logic needed to perform the following tasks:

1. Request two user inputs - a term/word/phrase and a directory from the user through command prompt.
2. Search for user input term/word/phrase in all ***docx documents (.docx)*** in the specified directory and sub-directories (user input directory).
3. Output the list of names of the documents containing the user given term/word/phrase.

Output total number of documents containing the term/phrase.

Output total number of occurrences of the term/phrase in a specific document.

1. Create a new ***word file*** and write the given term in it if there is no match found for the given term/word/phrase in all word documents in the given directory.
2. Require no fancy user interface. User input will be provided through command prompt. It must be implemented by using Ruby.
3. Implemented by using classes, objects, methods and constructor.
4. ***Use rubyXL gem***. (https://rubygems.org/gems/rubyXL/versions/3.3.29). Install all development dependencies.
5. Add appropriate comments e.g., Name, Student ID, functionality of a variable, brief explanation of logic

**Submission 3 (15 marks – due week 12)**

This prototype will then be transported to a rails framework with a user interface similar to the example below.

1. The user will provide the input through web page in rails. Web page should contain two text fields (1. Word/phrase and 2. Root directory) and a search button in excel files.
2. Search results will be display on web page as well.
3. In this Submission for every match found, the filename including the full path will be printed to the browser and the content of the document will be printed into a text. Please note the following picture is just an example giving you hints how to achieve the interface. Your interface should show the path and content of excel files.

Graphical user interface, text, application

Description automatically generated

1. If there is no match found for the given term/word/phrase in **all excel files** in the given directory/folder, upload a file/photo (deciding by you) to your searched directory using rails.
2. Add appropriate comments e.g., Name, Student ID, functionality of a variable, brief explanation of logic

**Plagiarism**

All used sources must be properly acknowledged with references and citations, if you did not create it. Quotations and paraphrasing are allowed but the sources must be acknowledged. Failure to do so is regarded as plagiarism and the minimum penalty for plagiarism is failure for the assignment. The act of given your assignment to another student is classified as a plagiarism offence. Copying large chucks and supplying a reference will result in zero marks as you have not contributed to the report. Copying from Youtube or other videos is also plagiarism (including transcripts). Citation in a video can be included as credits at the end.

**Late Submission**

**Late submission of assignments will be penalised as follows:**

* For assignments 1 to 10 days late, a penalty of 5% (of total available marks) per day.
* For assignments more than 10 days late, a penalty of 100% will apply. Which means, the assignment will not be marked and attract a fail grade.

Your submission must be compatible with the software (PDF/Word/Video/Zip) in Melbourne Polytechnic, Computer Laboratories/Classrooms.

**Extensions:**

Under normal circumstances, extensions will not be granted. In case of extenuating circumstances—such as illness—a Special Consideration form, accompanied by supporting documentation, must be received before 3 working days from the due date. If granted, *an extension will be only granted only by the time period stated on the documentation; that is, if the illness medical certificate was for one day, an extension will be granted for one day only*. *Accordingly, the student must submit within that time limit.*

Penalties may apply for late submission without an approved extension.

**Penalties:**

Academic misconduct such as cheating and plagiarism incur penalties ranging from a zero result to program exclusion.

**ASSIGNMENT OVERVIEW AND FEEDBACK SUMMARY**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| All Assignments and feedback are provided via the Moodle site and in classes. | | | | | |
| Assignment Tasks: | **Due Date** | **SLOs** | **CLOs** | **Weight** | **Comments** |
| A1 Class Test – Two written Assignments of key concepts of object-oriented programming and design, including syntax and semantics of the programming language. | Week 4 - Test A  Week 10 - Test B | 1,2,3,4 | 1 | 20% | Individual |
| A2 Ruby Project – Suggested topic: Development and demonstration of an original advanced application system based on. Suggested topic  Submission 1 (5%) : Working with Files and Folders  Submission 2 (15%): Working with Files and folders by using Ruby gems  Submission 3 (15%): Working with Files and folders by using rails | **Week 6 - Submission 1**  **Week 8 - Submission 2**  **Week 12 - Submission 3** | **1,2,3** | **1,3** | **35%** | **Individual** |
| A3 Video Presentation – Suggested topci: A report and presenation to demonstrate an understanding of Ruby programming concept | Week 14 | 4 | 3 | 20% | Individual |
| A4 RoR Project – Suggested topic: Development and demonstration of a web-based application using MVC for multiple CRUDs for multiple models | Week 13 | 1,2,3 | 1,3 | 25% | Individual |

**Marking criteria for Submission 1**

|  |  |
| --- | --- |
| **Marking Week** | **Possible marks** |
| **Penalty - No Comments** | **-1** |
| **Penalty – Student ID is not included** | **-5** |
| **Q1** | **2.5** |
| Verify that the file exists. | 0.5 |
| If the file does not exist, display an error message. | 0.5 |
| If the file exists, read it and test whether it contains the phrase “a little boy” | 1 |
| If it does display the contents of the text file otherwise display phrase not found | 0.5 |
| **Q2** | **2.5** |
| Prompt the user to type the path and name of a folder whose contents will be displayed. | 0.5 |
| Capture and analyze user input to determine whether the specified folder exists. | 0.5 |
| Tell the user whether or not the folder exists. | 0.5 |
| If the folder exists, prompt the user to press Enter in order to view the folder’s contents. | 0.5 |
| If the folder does not exist, display an error message. | 0.5 |

**Rubric for Submission 2**

**Question Answer session must be passed (provide satisfying answer) in order to pass the whole Assignment or Submissions of Assignment.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Marking Scheme** | **Approx. % of grade** | **Excellent** | **Good** | **Reasonable** | **Poor** |
| Program specifications/  Correctness | 35% | No errors, program  always works  correctly and  meets the  specification(s). | Minor details of  the program  specification are  violated, program  functions  incorrectly for  some inputs. | Significant details  of the specification  are violated,  program often  exhibits incorrect  behaviour. | Program only  functions correctly  in very limited  cases or not at all. |
| Documentation, style | 10% | code is  well‐commented and styling and naming conventions are well implemented | comments are generally used when required and naming and styling conventions are usually followed | some comments are missing or naming and styling conventions are not always followed | Hardly any comments or adherence to naming and styling conventions |
| Robustness | 15% | The program catches all exceptions that can occur and provides all necessary information to the user when they occur | The program catches most exceptions that can occur and provides reasonable information to the user when they occur | The program catches some exceptions that can occur and provides some information to the user when they occur | The program does not catch exceptions |
| Demonstration with question answering (Hurdle task) | 40% | Demonstrates full knowledge by answering all class questions  with explanations and elaboration | Is at ease with expected answers to all questions, without elaboration | Is uncomfortable with information and is able to answer only rudimentary questions | Does not have grasp of information and cannot answer questions about subject |

**Rubric for Submission 3**

**Question Answer session must be passed (provide satisfying answer) in order to pass the whole Assignment or Submissions of Assignment.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Marking Scheme** | **Approx**  **% of grade** | **Excellent** | **Good** | **Reasonable** | **Poor** |
| Program specifications/  Correctness | 30% | No errors, program  always works  correctly and  meets the  specification(s). | Minor details of  the program  specification are  violated, program  functions  incorrectly for  some inputs. | Significant details  of the specification  are violated,  program often  exhibits incorrect  behaviour. | Program only  functions correctly  in very limited  cases or not at all. |
| Documentation, style | 10% | code is  well‐commented and styling and naming conventions are well implemented | comments are generally used when required and naming and styling conventions are usually followed | some comments are missing or naming and styling conventions are not always followed | Hardly any comments or adherence to naming and styling conventions |
| Robustness | 10% | The program catches all exceptions that can occur and provides all necessary information to the user when they occur | The program catches most exceptions that can occur and provides reasonable information to the user when they occur | The program catches some exceptions that can occur and provides some information to the user when they occur | The program does not catch exceptions |
| User Interface | 10% | Interface is neat and well organized and uses rails capabilities extensively | Interface is basically neat and well organized but does not make full use of rails capabilities | Interface is reasonable and makes some use of rails capabilities | Interface is untidy and not organized and does not use rails capabilities |
| Demonstration with question answering (Hurdle task) | 40% | Demonstrates full knowledge by answering all class questions  with explanations and elaboration | Is at ease with expected answers to all questions, without elaboration | Is uncomfortable with information and is able to answer only rudimentary questions | Does not have grasp of information and cannot answer questions about subject |