**Saint Augustine’s College, Sydney**

**Software Engineering Year 11: Programming Fundamentals**

Assessment Task Notification and Mark Criteria

|  |  |  |
| --- | --- | --- |
| Assessment Task Notification | | A green shield with a white text  Description automatically generated |
| **Year:** | 11 |
| **Course:** | Software Engineering |
| **Task Topic:** | Programming Fundamentals |
| **Task Type:** | Python Programming Project & Documentation |

|  |
| --- |
| Assessment Task No: 1 |
| Due Date: Friday 4th April 2025 |
| Weighting: 25% |

|  |
| --- |
| Outcomes Assessed |
| SE-11-01 describes methods used to plan, develop and engineer software solutions  SE-11-06 applies tools and resources to design, develop, manage and evaluate software  SE-11-07 implements safe and secure programming solutions  SE-11-09 manages and documents the development of a software project |

|  |
| --- |
| Task Overview |
| |  | | --- | | Your assignment is to create a Python solution for an educational quiz game. The educational piece of software is to be an interactive solution, designed to engage students while reinforcing their knowledge in various subject areas. The software could be designed for either primary or secondary students, depending on the categories and question bank you create.  The program will include an **i**ntuitive command-line interface that displays clear directions and responses to the user’s requests. It will present a series of randomised multiple-choice questions, allow the user to select from a range of categories, track their score, and provide feedback on incorrect answers.  The project requires following the Software Development Lifecycle (SDLC) in a structured approach. You are required to fully document your planning, development, and testing stages in a prescribed format. Additionally, you must use version control to regularly push changes to an online GitHub repository. | |

|  |
| --- |
| Task Instructions |
| The project consists of two parts: the source code submitted through a GitHub repository, and the accompanying documentation.  Part A: Source Code (50%)  When developing your software, you must adhere to the following technical specifications:   * The program must be developed using Python 3.x * The program must be executed and operated entirely in the command line. * You must use Git/GitHub for version control throughout development. * You may use external Python libraries to improve functionality. Please list these in a requirements.txt file with explanations of their use and links to their official documentation.  |  | | --- | | The functional requirements for your solution include:   * **Category Selection** - Allow users to choose from a range of categories at the start of the game. * **Randomized Questions** - Randomly select a series of questions from the chosen category at the outset of the game. * **Score Tracking** - Keep a running score of correct answers during the game. * **Feedback System** - Track incorrect answers and display them at the end of the game. * **Game Loop** - Allow users to play multiple rounds of the quiz without restarting the program. | | Some marks will be awarded for creativity if students implement additional features such as:   * **Question Timer**: Set a time limit for answering questions. * **Hint System**: Allow users to request a hint for difficult questions. * **High Scores**: Save and display the highest scores from previous sessions.   **Part B: Documentation (50%)**  Documentation supporting your software development journey must be created using Microsoft Word and include the following sections:   |  |  | | --- | --- | | Section | Description | | Planning & Designing | Requirements Definition | | Structure chart | | Flow chart | | Data Dictionary | | Implementation | Link to your GitHub repository containing source code | | Testing | Test table | | Evaluation | Project reflection |   For detailed guidance on what to include in each section, refer to the marking criteria. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Marking Criteria: Part A – Source Code | | | | | |
| Criteria | **Incomplete/Non-Attempt** | **Below Expectations** | **Basic** | **Proficient** | **Advanced** |
| Algorithms and Game Logic | Identifies some relevant game logic, but it is not applicable to the scenario or does not function as intended. | Describes basic algorithms and game logic, but they contain severe flaws, leading to largely non-functional gameplay. | Constructs functional algorithms, though logic errors cause inconsistent gameplay and unexpected outcomes. | Constructs algorithms which support most of the game's logic and feature requirements. There may be minor deviations in logic that do not significantly affect the integrity of gameplay. | Constructs algorithms and game logic is optimally implemented, with a precision that ensures gameplay is fun, informative and educational. |
| 0-4 | 5-8 | 9-12 | 13-16 | 17-20 |
| Data Structures | Identifies incorrect or ineffective data structures that do not meet the task requirements. | Identifies data structures that are implemented but poorly suited for storing and managing quiz data. | Demonstrates the use of data structures that can store questions and topic areas, but they may not allow for easy maintenance. | Demonstrates the use of appropriate data structures for managing questions and topics. Additional questions and topics can be added easily. | Demonstrates the use of optimal data structures for managing questions and topics, separating this data in external file(s) for cleanliness and maintenance. |
| 0-2 | 3-4 | 5-6 | 7-8 | 9-10 |
| Efficiency | Code demonstrates a significant lack of understanding of efficiency, with critical errors or complete absence of functional logic. | Code demonstrates inefficiencies, numerous errors and poor organisation, making it difficult to comprehend or maintain. | Code demonstrates some efficiency but could be improved; there might be some overly complex functions or those that perform more than one task. | Code demonstrates efficiency; functions are generally well-organised and adhere to the single responsibility principle, with occasional non-critical inefficiencies. The mainline structure is logical, but there may be minor improvements that could enhance readability and performance. | Code demonstrates high efficiency with functions that are concise, clearly purposed, and devoid of redundancy. The mainline structure is logical and functions adhere to the single responsibility principle. |
| 0-2 | 3-4 | 5-6 | 7-8 | 9-10 |
| Syntax | No identification or lack of understanding of Python syntax conventions. | Syntax demonstrates some errors and inconsistencies, severely hindering readability and maintainability. | Demonstrates basic syntax conventions but with some inconsistencies that may make understanding and maintaining the code more challenging. | The code largely demonstrates best practice to naming and style conventions, with few deviations that do not impede understandability. | The code demonstrates PEP 8 best practices for naming, style, and convention, enhancing readability and maintainability. |
| 0-1 | 2 | 3 | 4 | 5 |
| Input Validation | No input validation has been identified and implemented. | Some input validation is demonstrated, but is poorly executed, leading to many errors and issues in user interaction. | Basic input validation is demonstrated, but not all on all relevant inputs. | Reliable input validation and error handling are demonstrated, though there might be rare edge cases not accounted for. | Code demonstrates comprehensive input validation and error handling ensure a robust and user-friendly interface. |
| 0-1 | 2 | 3 | 4 | 5 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Marking Criteria: Part B – Documentation | | | | | |
| Criteria | **Incomplete/Non-Attempt** | **Below Expectations** | **Basic** | **Proficient** | **Advanced** |
| Task Definition | Identifies minimal or no task requirements; lacks understanding. | Describes the task with limited clarity; misinterpretations are evident. | Explains the task with a reasonable interpretation of requirements. | Analyses the task, demonstrating logical alignment with game mechanics. | Critically analyses the task, integrating a strategic approach with Python and technical specifications. |
| 0-1 | 2 | 3 | 4 | 5 |
| Structure Chart | Identifies unrelated structure chart elements, showing minimal understanding of its purpose. | Identifies some structure chart elements, unrelated to the scenario. Minimal understanding of its purpose is demonstrated. | Constructs a structure chart which follows basic guidelines but may include errors or incomplete scenario coverage. | Constructs a well-conceived structure chart that breaks down the problem, applying a suitable sequence of execution with clarity and detail. | Constructs a precise and detailed structure chart, elegantly presented, demonstrating flawless logic and a clear sequence of execution. Excellently presented using draw.io. |
| 0-1 | 2 | 3 | 4 | 5 |
| Data Dictionary | Identifies few or no data dictionary elements, showing little understanding of its purpose. Missing key variables and data types. | Identifies some relevant data dictionary elements, but the dictionary is incomplete and lacks comprehensive detail. | Constructs a data dictionary that outlines key variables and data types, but may include errors or inconsistencies. | Constructs a complete data dictionary with all necessary variables, types, and meaningful examples, tailored to the software requirements. | Constructs an exhaustive and meticulous data dictionary, demonstrating careful attention to variable naming conventions and data structure integrity. |
| 0-1 | 2 | 3 | 4 | 5 |
| Flowchart | Identifies few or no flowchart elements, or the provided chart shows no resemblance to a flow chart used by developers. | Identifies some relevant flowchart symbols; may inaccurately represent the logical sequence of the quiz’s functionality. | Constructs a flowchart using correct symbols and demonstrates basic understanding of the game's functions. | Constructs a well-structured flowchart that accurately models the software's mainline and related data structures with adherence to established formatting guidelines. | Constructs an accurate representation of game logic within a flowchart, expertly crafted, demonstrating meticulous attention to detail and logical precision. Presented using draw.io. |
| 0-1 | 2 | 3 | 4 | 5 |
| Test Table | Identifies some potential tests that could be performed on software. | Identifies relevant tests and includes a basic test table, which may lack essential elements like expected outcomes or may contain inaccuracies. | Constructs a test table with a minimum of five meaningful tests, covering key aspects of the software. | Constructs a comprehensive test table with at least ten detailed tests, demonstrating a range of tests including boundary values, path coverage, and exception handling. | Constructs an exceptional test table with a robust suite of tests that address a range of test types, demonstrating understanding of boundary values, path coverage, and exception handling. Recommends bug fixes based on the tests, which have been briefly documented below the test table. |
| 0-1 | 2 | 3 | 4 | 5 |
| GitHub Repository and Version Control | Identifies the need to use version control in a software development project. | Applies limited version control methods with sparse or non-informative commit messages. | Applies basic version control using Git/GitHub demonstrated with a repository containing a README.so file. There are gaps in detail and sporadic commits. | Applies version control methodologies to the project. Repository contains detailed README and regular commit messages that, while generally informative, may occasionally lack detail. | Applies version control methodologies to the project. Demonstrates a well-organized GitHub repository with a comprehensive README, clear setup instructions, game rules, and detailed commit history reflecting progress. Care and pride has been taken to showcase the software well. |
| 0-4 | 5-8 | 9-12 | 13-16 | 17-20 |
| Project Reflection | Identifies limited reflection upon the success of the project, or provides only a brief, superficial comment on the project's perceived success. | Discusses some of the project's success and challenges encountered, though discussion may be general and not fully substantiated. | Examines the development process, though some reflections may be general and not fully substantiated. | In-depth reflection on the development process, analysing successes, challenges, and areas for future improvement. | Thorough and structured critical analysis, free from grammatical errors, offering a evaluation of the development journey and resultant learning. |