| **Assignment** | **Assigned** | **Deadline** |
| --- | --- | --- |
| **Coin Count**  **T-Level Software Development SD31** | 3 Oct 2023 | 20 Oct 2023 |

| **Learning Objectives** |
| --- |
| * Analyse a problem * Design a solution * Build and test a working solution * Evaluate the solution against the scenario |

| **Support** |
| --- |
| **These resources may help you:**  [Algorithms - GCSE Computer Science Revision - BBC Bitesize](https://www.bbc.co.uk/bitesize/guides/z7kkw6f/revision/1)  [Python Programming Examples - GeeksforGeeks](https://www.geeksforgeeks.org/python-programming-examples/)  [How to Create Flowcharts in Draw.io](https://drawio-app.com/blog/flowcharts/)  [Test Plans - Isaac Computer Science](https://isaaccomputerscience.org/concepts/prog_softeng_test_plans) |

| **Instructions** |
| --- |
| * Use Python * Internet access is allowed for research purposes * Work submitted must be your own |

| **Scenario** |
| --- |
| A local youth club has organised several coin collections for a charity during the year.  People have been very generous, and a lot of money has been collected. The coins now need to be counted and paid into the charity’s bank account.  Six club members have volunteered to count the coins. The bank has supplied plastic bags for the coins.  **The coins must be bagged in set values as shown in the table below.**  Each bag must hold exactly the value of coins shown in the table and contain only one type of coin. The volunteers sort the coins by type before counting and bagging them.  The club leader wants to check that the coins have been counted correctly by weighing each bag. At this stage, any bag containing the wrong number of coins is corrected. This checking process will take place over several sessions.  **The weight of each coin, in grams (g), is also shown in the table below.**   | **Coin** | **Bag Value** | **Weight (g)** | | --- | --- | --- | | £2 | £20 | 12.00 | | £1 | £20 | 8.75 | | 50p | £10 | 8.00 | | 20p | £10 | 5.00 | | 10p | £5 | 6.50 | | 5p | £5 | 2.35 | | 2p | £1 | 7.12 | | 1p | £1 | 3.56 | |

| **Scenario (continued…)** |
| --- |
| **The youth club leader wants a computer program to check the coin count.**  **The program must:**   * ~~allow the user to input the volunteer’s name, type of coin and weight of bag~~ * ~~validate the coin type~~ * ~~indicate the number of coins to be added or removed to correct an inaccurate bag weight~~ * ~~maintain running totals of the number of bags checked and total value~~ * ~~provide an option to display the total number of bags checked and total value~~ * ~~monitor the accuracy of the volunteers counting the coins~~ * ~~provide an option to display a list of the volunteers, abena, showing:~~   + ~~the total number of bags they have counted~~   + ~~the number of bags they counted correctly, as a percentage of their total.~~ * ~~Save the data in a text file called CoinCount.txt~~ * ~~Load CoinCount.txt at the beginning of each session.~~ * ~~Update CoinCount.txt at the end of each session.~~   Your task is to analyse this problem and to design, implement, test and evaluate a programmed solution.  You must create test data to check that your program works as intended.  **Some sample test data is shown in the table:**   | **Volunteer** | **Coin** | **Bag Weight (g)** | **Correct** | | --- | --- | --- | --- | | Abena | 5p | 235.00 | Y | | Malcolm | 1p | 356.00 | Y | | Jane | £2 | 120.00 | Y | | Andy | £1 | 166.25 | N | | Sandlip | 50p | 160.00 | Y | | Liz | 20p | 250.00 | Y |   **Create a folder called Report and share it with your tutor.**  **Save all your evidence for assessment in this folder.**  **Save your evidence as instructed in each task.** |

| **Task 1: Analysis** |
| --- |
| **1hr recommended**  You should include an introduction summarising the overall problem.  The problem should be broken down into sub-problems. You should write a description of each sub-problem you identify and explain your selection of sub-problems.£  State any assumptions you have made.  **Save your work in the Report folder as a document called Analysis.** |

| **Task 2: Design** |
| --- |
| **3hrs recommended**  **Algorithms**  Design algorithms, using pseudo-code or flowcharts, that show a logical solution to each sub-problem.  You should include inputs, processes, outputs, validation checks and the programming constructs that you will use when you produce your program.  You should show how the algorithms will link together and lead to an overall solution.  **Save your algorithms in the Report folder as a document called Design.**  **Initial test plan**  You should complete the relevant sections of the test plan template provided to produce an initial test plan that will demonstrate your strategy for testing your solution.  **Save your initial test plan in the Report folder as a document called TestPlan.**  **Save a copy of TestPlan in the Report folder as a document called Debugging.**  **This will be used in Tasks 3 and 4.** |

| **Task 3: Implementation** |
| --- |
| **8hrs recommended**  You should translate your design into a program. Ensure that your program is clear and easy to understand.  Add the results of any tests carried out during the implementation stage to the Debugging document.  Save the updated Debugging document in the Report folder.  Create a subfolder called Implementation in the Report folder.  Save your source code and all the files required to execute the program, including your CoinCount.txt file, in the subfolder. |

| **Task 4: Testing, Refining, and Evaluation** |
| --- |
| **2hrs recommended**  You should complete the Debugging document by adding any further tests carried out at this stage, including the results of retesting following the correction of any errors.  **Save the completed Debugging document.**  Evaluate your solution by explaining how well your program meets each of the requirements that you identified in your analysis and describing any refinements that you made to your program during design and implementation.  **Save your evaluation in the Report folder as a document called Evaluation.** |

| **Mark Scheme** |
| --- |
| | **Analysis** | | | | | --- | --- | --- | --- | | **Excellent** | **Good** | **Requires Improvement** | **Unacceptable** | | The project demonstrates a deep understanding of the requirements, including feasibility analysis and consideration of user needs. | The project adequately analyses the requirements with some areas that could be further refined or clarified. | The project's analysis is limited, lacking clarity, completeness, or consideration of user needs. | The project's analysis is missing or inadequate, leading to a lack of direction. |  | **Design** | | | | | --- | --- | --- | --- | | **Excellent** | **Good** | **Requires Improvement** | **Unacceptable** | | The project includes well-documented algorithms using pseudo-code or flowcharts for each sub-problem, showcasing a logical and comprehensive solution. | The project presents effective algorithms for most sub-problems, demonstrating a logical solution with clear documentation. | The project's algorithm design is lacking in detail or coherence, with some sub-problems not adequately addressed or unclear connections between algorithms. | The project lacks clear algorithm design, making it challenging to understand the logical solution for sub-problems or how these solutions combine into an overall solution. |  | **Implementation** | | | | | --- | --- | --- | --- | | **Excellent** | **Good** | **Requires Improvement** | **Unacceptable** | | The project meets all the specified requirements of the brief and functions flawlessly, providing a comprehensive solution.  The project has a well-designed, intuitive, and visually appealing user interface.  It effectively handles all possible errors and exceptions, providing informative error messages and ensuring graceful degradation. | The project meets most of the requirements and functions effectively, with minor issues or missing features.  The project has a decent user interface that is reasonably intuitive and visually presentable, with some room for improvement.  It adequately handles most expected errors and exceptions, providing useful error messages. | The project partially meets the requirements of the brief and has some functional issues or missing features.  The project's user interface lacks design elements, usability, or aesthetics, and requires significant improvements.  It also lacks proper error handling in some areas, leading to unexpected behaviour or cryptic error messages. | The project does not meet the specified requirements of the brief and lacks basic functionality.  The project has a poorly designed and unintuitive user interface that hampers user experience and usability.  It has inadequate or no error handling, resulting in frequent crashes, undefined behaviour, or unclear error messages. |  | **Testing, Refining, and Evaluation** | | | | | --- | --- | --- | --- | | **Excellent** | **Good** | **Requires Improvement** | **Unacceptable** | | The project includes comprehensive testing plans and effectively evaluates the software against the specified requirements.  It demonstrates thorough testing, including unit, integration, and user acceptance testing, with clear documentation of test cases and results. | The project incorporates testing and evaluation processes but may lack comprehensive coverage or clear documentation of testing efforts.  It demonstrates some level of testing, but there is room for improvement. | The project lacks a structured testing and evaluation approach, resulting in limited or inconsistent testing efforts.  Test cases and results may be incomplete or unclear. | The project lacks any meaningful testing and evaluation, leaving the software's functionality unverified and unvalidated. | |

| **Overall Grade:** |  |
| --- | --- |