**THE UNIVERSITY OF HUDDERSFIELD**

# School of Computing and Engineering

# **ASSIGNMENT SPECIFICATION – Portfolio**

|  |  |
| --- | --- |
| Module details |  |
| Module Code | CFS2103 |
| Module Title | Introduction to Procedural Programming |
| Course Title/s | MEng / BSc (Hons) Software Engineering  MComp / BSc (Hons) Computing  MSci / BSc (Hons) Computer Science  BSc (Hons) Computer Science with Games Programming  BSc (Hons) Computer Science with Cyber Security  BSc (Hons) Computer Science with the Internet of Things  BSc (Hons) Computer Science with Artificial Intelligence  BEng (Hons) Computer Systems Engineering |

|  |  |  |
| --- | --- | --- |
| Assessment weighting, type and contact details | | |
| Title | Portfolio | |
| Weighting | 60% | |
| Mode of working for assessment task | Individual  Note: if the assessment task is to be completed on an individual basis there should be no collusion or collaboration whilst working on and subsequently submitting this assignment. | |
| Module Leader | Soran Parsa [s.parsa@hud.ac.uk](mailto:s.parsa@hud.ac.uk) | Contact details:  [s.parsa@hud.ac.uk](mailto:s.parsa@hud.ac.uk) |
| Module Tutor/s | Mohamed Eban ([M.EK.Eban@hud.ac.uk](mailto:M.EK.Eban@hud.ac.uk)) | |

|  |  |
| --- | --- |
| Submission and feedback details | |
| Hand-out date | 30/09/2024 |
| How to submit your work. | Via BrightSpace (to be automatically checked by TurnItIn). |
| Submission date/s and times | **10/01/2025** by 12:00 noon – if you have any technical issues submitting your work, please contact the Module Leader as soon as possible. |
| The expected amount of independent time you should allocate to complete this assessment | 24 Hours (Typical Effort) |
| Submission type and format | * A single .zip file containing the python source code * A Word or PDF document for the report |
| Date by which your grade and feedback will be returned | **31/01/2025** |

| Additional Guidance Information | Details |
| --- | --- |
| Your responsibility | It is your responsibility to read and understand the [University regulations regarding conduct in assessment](https://www.hud.ac.uk/registry/current-students/taughtstudents/conductinassessment/).  Please pay special attention to the assessment regulations on [Academic Misconduct.](https://www.hud.ac.uk/registry/current-students/taughtstudents/academicmisconduct/)  In brief: ensure that you;   1. DO NOT use the work of another student - this includes students from previous years and other institutions, as well as current students on the module. 2. DO NOT make your work available or leave insecure, for other students to view or use. 3. Any examples provided by the module tutor should be appropriately referenced, as should examples from external sources.   Further guidance can be found in the SCEN Academic Skills Resource and UoH Academic Integrity Resource modules in Brightspace. |
| Guidance on using AI:  *\*\*Please delete the two levels not appropriate for this assessment.*  *Guidance -* [*AI- Use Level Descriptors for Assignments – iPark (hud.ac.uk)*](https://ipark.hud.ac.uk/index.php/support-guidance/assessment-design-in-the-ai-age/introduction-to-assessment-design-in-the-ai-age/ai-use-level-descriptors-for-assignments/)*\*\** | Level 1 – Not Permitted  The use of AI tools is not permitted in any part of this assessment. All work will be checked individually to ensure that AI tools are not used. In addition, the report and in-code explanations will be used to ensure originality of the work. |
| School Guidance and Support | If you experience difficulties with this assessment or with time  management, please speak to the module tutor/s, your Personal Academic Tutor, or the Student Progress Mentors. Student Progress Mentor – useful links.   * Brightspace Module - [SCE Student Progress Mentors (hud.ac.uk)](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fbrightspace.hud.ac.uk%2Fd2l%2Fhome%2F363005&data=05%7C02%7Cp.j.mather%40hud.ac.uk%7Cf77cd0f954194c6371a608dcc749a9e7%7Cb52e9fda06914585bdfc5ccae1ce1890%7C0%7C0%7C638604366398878210%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C0%7C%7C%7C&sdata=MxOgFvEXnLApRAElIJZlrungTXsxsssneTkQk7HE1tk%3D&reserved=0). * Email - [sce.progress.mentors@hud.ac.uk](mailto:sce.progress.mentors@hud.ac.uk) * Booking an appointment - [http://hud.ac/rgl](https://eur02.safelinks.protection.outlook.com/?url=http%3A%2F%2Fhud.ac%2Frgl&data=05%7C02%7Cp.j.mather%40hud.ac.uk%7Cf77cd0f954194c6371a608dcc749a9e7%7Cb52e9fda06914585bdfc5ccae1ce1890%7C0%7C0%7C638604366398896497%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C0%7C%7C%7C&sdata=gjkWxj14%2BcAXeSy90Cm2QDcspw7mn7Cqors2K%2Bgnkis%3D&reserved=0) |
| Requesting a Late Submission | It is expected that you complete your assessments by the published deadlines. However, it is recognised that there can be unexpected circumstances which may affect you being able to do so. In such circumstances, you may submit a request for an extension.  Extension applications must be submitted before the published assessment deadline has passed.  To apply for an extension, you should access the Extension System on MyHud. |
| Extenuating Circumstances (ECs) | An EC claim is appropriate in exceptional circumstances, when an extension is not sufficient due to the nature of the request.  You can access details on the procedure for claiming ECs, on the Registry website; [Consideration of Personal Circumstances - University of Huddersfield](https://www.hud.ac.uk/registry/current-students/taughtstudents/considerationofpersonalcircumstances/), where you can also access the [EC Claim Form](https://www.hud.ac.uk/media/assets/document/registry/forms/ECClaimForm23-24.docx).  You will need to submit independent, verifiable evidence for your claim to be considered.  Once your EC claim has been reviewed you will get an EC outcome email from Registry.  An approved EC will extend the submission date to the next assessment period (e.g July resit period). |
| Late Submission  (No ECs approved) | Late submission, up to 5 working days, of the assessment submission deadline, without an approved extension will result in your grade being capped to a maximum of a pass mark.  Submission after this period, will result in a 0% grade for this assessment component. |
| Tutor Referral available | YES |
| Resources | * Please note: you can access free Office365 software and you have 100 Gb of free storage space available on Microsoft’s OneDrive – [Guidance on downloading Office 365](https://students.hud.ac.uk/media/universityofhuddersfield/content/documents/computingservices/office365/Office365-AppsDownloadGuide.pdf). |

**Portfolio**

## 1. Assignment Aims

To develop fundamental Procedural Programming skills and create programmatic solutions to a real-world problem involving robotics and automation.

**2. Learning Outcomes**

MLO4, Apply fundamental procedural programming concepts to create programmatical solutions to problems.

MLO5, Implement, test, and debug computer programs using procedural languages.

MLO6, Write documentation using standards and tools commonly used in the industry for documenting procedural programming code.

## 3. Assessment Brief

# Introduction

You are required to submit a Python project that simulates the management of a robotic cell in an industrial setting. In this scenario, robots and human workers collaborate to assemble a product in an organized process. Your program should allow a user (factory supervisor) to manage the operations in the robotic cell, ensuring the robots and workers coordinate efficiently to assemble products step by step.

Your submission should include all files and classes that make up your program in a single ZIP file (including any required libraries). You will also provide a document (maximum 1500 words) containing a description, explanations, and screenshots that demonstrate how your program operates under different scenarios and test cases. The document should focus on the solution’s behaviour, error handling, and justification of design choices.

The assessment includes the standard of the source code (layout, indentation, identifiers, comments, etc.) as well as functionality. Code should follow the usual python conventions, Procedural Programming principles and good practice; the project will also be marked on how well the solution has been designed.

**Project Brief:**

Design and implement a Python program that simulates the coordination between n robots and m workers in a robotic cell to assemble a product. The system should allow the supervisor to manage operations, including assigning tasks, monitoring the status of robots and workers, and tracking the progress of assembly. Your system should support the following functionalities:

**Add and remove robots**: Allow the supervisor to add new robots to the robotic cell or remove existing ones. Each robot should have a unique ID and status (idle, working, or finished task).

**Add and remove workers**: Similar to robots, workers can be added or removed from the system. Each worker should have a unique ID and status (idle, working, or finished task).

**Assign tasks** to robots and workers: Tasks such as welding, assembling, or inspecting need to be assigned to robots and workers. Each task requires specific resources (e.g., 1 robot and 1 worker) and has a time duration before it can be completed. The system should dynamically assign tasks to idle robots and workers.

**Monitor task progress**: Track the progress of each task and update the status of robots and workers in real-time. Tasks should transition from not started to in progress, and finally completed. Once a task is completed, the system should mark the involved robots and workers as idle again, ready for the next task.

**Product management**: The supervisor can assign a product to the robotic cell for assembly. The product should have multiple assembly steps (e.g., welding, assembling, testing). Each step should define the type and number of resources (robots/workers) required and the order in which the steps must be completed.

**Error handling**: Handle situations such as:

Attempting to assign tasks when no robots or workers are available.

Trying to remove robots or workers that are currently engaged in a task.

Overloading the system with more tasks than can be handled by available robots and workers.

**Display status of robots, workers, and tasks**: The system should provide real-time updates on the status of each robot, worker, and task. This includes which tasks are in progress, which are completed, and the status of the robots and workers (idle or busy).

**Report completion of product assembly**: When all tasks required for the product are completed, the system should notify the supervisor that the product has been fully assembled.

**Requirements:**

**Variables, Data Types, and Control Flow (MLO4)**

**Robot and worker data**: Use dictionaries or classes to represent robots and workers. Store their unique IDs and statuses.

**Task management**: Use lists, tuples, or dictionaries to store and track tasks. Each task should store information such as the required number of robots and workers, the task type, and its current status (not started, in progress, completed).

**Product management**: Store the product assembly steps and manage the sequence of task completion. Ensure that robots and workers work on the right tasks in the correct order.

**Control flow**: Implement logic using if, else, while, and for loops to control task assignments, handle errors, and monitor the status of robots and workers. Ensure that the program checks availability of resources before assigning tasks and updates task statuses in real-time.

**Functions (MLO4)**

Break the system down into multiple functions, for example:

* add\_robot(), remove\_robot(): Manage the list of robots and their statuses.
* add\_worker(), remove\_worker(): Manage the list of workers and their statuses.
* assign\_task(): Dynamically assign tasks to available robots and workers. If none are available, the system should wait until resources are freed.
* check\_status(): Monitor the progress of tasks, robots, and workers. This function should regularly update task statuses and report any issues (e.g., idle robots or incomplete tasks).
* assign\_product(): Set up a new product to be assembled, defining the steps required and allocating resources to each step.

**Testing and Debugging (MLO5)**

**Test cases**: Include test cases that demonstrate:

* Adding and removing robots and workers.
* Assigning tasks and completing them.
* Handling errors when no robots or workers are available.
* Monitoring the progress of a multi-step product assembly.

**Error handling**: Test for common errors such as trying to remove robots or workers engaged in tasks or over-assigning tasks.

**Documentation (MLO6)**

 Provide detailed comments in the code, explaining each function, variable, and control flow logic.

 Write a 1500-word document that includes:

* A brief overview of the robotic cell management system.
* How procedural concepts (variables, loops, functions) are applied in the project.
* Testing and error-handling procedures, explaining your approach to simulating tasks and managing resources.
* Justifications for your design choices, tools, and solutions.
* Screenshots demonstrating the system in action, such as adding robots, assigning tasks,

Screenshots demonstrating the program in action (e.g., adding a book, borrowing a book, etc.).

**Submission:**

**Code**: Submit all Python files along with any necessary libraries in a single ZIP file.

**Documentation**: Submit a 1500-word document in Word or pdf format.

**Assessment Criteria:**

**Program Functionality** (25%): How well the program meets the outlined functionality.

**Code Quality** (25%): Use of appropriate data types, loops, and functions, along with efficient code structure.

**Testing and Debugging** (20%): Demonstration of thorough testing and error handling.

**Documentation** (30%): Clarity and completeness of the 1500-word report and in-code documentation.

**Rubrics**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **Excellent (70-100%)** | **Good (60-69%)** | **Satisfactory (50-59%)** | **Needs Improvement (40-49%)** | **Unsatisfactory (<40%)** |
| **1. Program Functionality (25%)**  Does the program fully meet the requirements? | - All features work as specified.  - No logical or runtime errors.  - Extra functionality (e.g., advanced search) or efficiency implemented. | - All core features work as specified.  - Few minor issues or bugs. | - Most core features implemented, though some may be incomplete or not working as intended.  - Some bugs or logical errors. | - Limited implementation of the required features.  - Significant bugs or errors.  - Some tasks incomplete. | - Does not meet core requirements.  - Little to no functionality.  - Frequent crashes or critical issues. |
| **2. Code Structure & Quality (25%)**  Are good programming practices followed (e.g., functions, loops, data types)? | - Code is well-organized, with effective use of functions and control structures.  - Efficient and concise code.  - Excellent use of data types.  - No redundant code. | - Code is well-structured, with appropriate use of functions and control structures.  - Minimal redundancy in the code.  - Good use of data types. | - Code is somewhat structured but may contain redundant sections.  - Functions used but could be more modular.  - Adequate use of control flow and data types. | - Code lacks proper structure, with significant redundancy.  - Insufficient use of functions and control flow.  - Poor choice of data types. | - Code is disorganized and difficult to follow.  - Minimal or no use of functions or control flow.  - Inappropriate data types or poorly implemented logic. |
| **3. Testing and Debugging (20%)**  Are there test cases? How well are errors handled? | -Comprehensive testing with multiple test cases demonstrating each function.  - Exceptional error handling for all possible user inputs or actions. | - Good range of test cases.  - Effective error handling, though some edge cases may be missed. | - Some test cases are provided but are not comprehensive.  - Basic error handling, but may not cover all edge cases. | - Minimal testing provided.  - Little to no error handling. | - No testing provided.  - Frequent crashes or unhandled errors during runtime. |
| **4. Documentation (30%)**  Is the code well-commented? Is the report clear and comprehensive? | - Code is thoroughly and clearly commented.  - The 1500-word document is comprehensive and well-written.  - Screenshots clearly demonstrate program functionality. | - Code is sufficiently commented, though some parts may lack detail.  - The 1500-word document is clear, with some minor details missing.  - Screenshots provided but could be more detailed. | - Code contains minimal comments.  - The 1500-word document is adequate but lacks detail.  - Screenshots provided but may not fully explain functionality. | - Code contains very few or no comments.  - The 1500-word document is vague or incomplete.  - Screenshots are unclear or missing key details. | - No comments in code.  - No 1500-word document or screenshots provided, or they do not explain the project. |