THE INTERNATIONAL QUALIFICATIONS AND ASSESSMENT CENTRE (IQAC)

«Approved»

[illegible]

2025 y. «18» iyul

INTERNATIONAL QUALIFICATIONS AND ASSESSMENT CENTRE XALQARO MALAKALAR VA BAHOLASH MARKAZI IQAC

Programme	**International Foundation Year Diploma in Information Technology (RQF)**		
Unit Number/ Unit Title	**Unit 4 Programming concepts and application development**		
Cohort Code:	L03PAD-U4		
Unit Level	Level 3		
Total GLH	Total qualification time 200/ Total Guided learning hours 90/ Self-guided learning hours 110		
Credits	20 CATS/ 10 ECTS		
Lecturer			
Start Date		**End Date**	

Unit Aims	The aim of this unit is to introduce learners to the fundamental concepts and the development of applications. This unit covers the basic syntax and structure of programming languages, the distinction between high-level and low-level languages, and the different programming paradigms such as procedural, object-oriented, and functional programming. Learners will develop practical skills in writing and debugging simple programmes, using comments and proper code formatting, and understanding algorithms and data structures. By the end of this unit, learners will be equipped with the foundational programming knowledge and skills necessary to create, analyse, and troubleshoot basic programmes, forming a crucial stepping stone for advanced study and careers in software development and related fields.
Differentiation Strategies *(e.g. planned activities or support for individual learners according to their needs)*	The total number of students to be in the lesson is approximately 20. This is a multicultural group of students predominantly between the ages of 24 – 45, with numerous ethnic, gender, and creed background. These are UK academic level 5 students; hence it is assumed that they have practical, theoretical, or technological knowledge and understanding of a subject or field of work to find ways forward in broadly defined, complex contexts. These students must be able to generate information, evaluate, synthesise the use information from a variety of sources. Various

	approaches to addressing the various identified students needs will be adopted throughout the lesson. Such will include:- 1. Progressive tasks 2. Digital resources 3. Verbal support 4. Variable outcomes 5. Collaborative learning 6. Ongoing assessment 7. Flexible-pace learning
Equality & Diversity	Variety of teaching techniques will be employed to ensure that the needs of each individual learner are met.
Safeguarding & Prevent	Safeguarding policies and the Prevent duty are strictly observed to ensure the safety, well-being, and inclusivity of all students and staff.
Health & Safety	SIRM H&S policies will be maintained.
Learning Resources	**Teaching and Learning Materials**
	• "Programming: Principles and Practice Using C++" by Bjarne Stroustrup • "Python Crash Course" by Eric Matthes • "Java: A Beginner's Guide" by Herbert Schildt • "Introduction to the Theory of Computation" by Michael Sipser • "Programming Logic and Design" by Joyce Farrell • "Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin

Learning Outcome	Assessment Criteria
LO1: Understand basic programming concepts and paradigms.	1.1 Define key programming concepts (variables, data types, control structures). 1.2 Explain different programming paradigms (procedural, object-oriented, functional). Demonstrate understanding of algorithms and their role in programming.
LO2: 2. Develop and implement simple applications.	2.1 Write code using a high-level programming language (e.g., Python, Java). 2.2 Design and implement basic algorithms to solve problems. Debug and test programs to ensure functionality and reliability.
LO3. Apply software development principles and practices.	3.1 Describe the software development life cycle (SDLC) phases. 3.2 Apply version control and documentation practices in programming projects. Discuss ethical considerations in software development.

No	Learning Outcome / Topic	Learning and Teaching Activities	Which assessment criteria does the session relate to?	Day/month/ year/ signature
1.	**Introduction to Programming**	Lecture: What is programming? Demo: "Hello World" in Python. Discussion: How software impacts daily life.	**LO1.1**	
2.	**Variables & Data Types**	Practical Lab: Declaring variables. Hands-on: Using integers, floats, strings, and booleans in simple programs.	**LO1.1**	

3.	**Operators & Expressions**	Workshop: Using arithmetic and comparison operators. Problem-solving: Writing expressions to calculate simple values.	**LO1.1**	
4.	**Input & Output (I/O)**	Practical Lab: Building an interactive console program that takes user input and prints a formatted response.	**LO2.1**	
5.	**Control Structures: Conditionals**	Code-along: Creating if/elif/else statements. Challenge: Building a simple grading system or login checker.	**LO1.1, LO2.1**	
6.	**Control Structures: Loops**	Practical Lab: Using for and while loops. Task: Writing a program to calculate sums or print patterns.	**LO1.1, LO2.1**	
7.	**Data Collections: Lists & Arrays**	Workshop: Storing and manipulating data in lists. Activity: Creating a simple shopping list or quiz program.	**LO1.1, LO2.1**	
8.	**Data Collections: Dictionaries**	Code-along: Using key-value pairs. Problem-solving: Building a simple employee directory or product database.	**LO1.1, LO2.1**	
9.	**Algorithms & Problem Decomposition**	Group Activity: Breaking down a real-world problem (e.g., making tea) into step-by-step instructions.	**LO1.3**	
10.	**Algorithm Design: Pseudocode**	Workshop: Writing pseudocode for a simple task (e.g., calculating average). Peer review of logic clarity.	**LO1.3, LO2.2**	
11.	**Algorithm Design: Flowcharts**	Practical Session: Using draw.io or Lucidchart to create a flowchart for a login process or vending machine.	**LO1.3, LO2.2**	
12.	**Programming Paradigms Overview**	Lecture & Discussion: Comparing Procedural, OOP, and Functional paradigms with real-world analogies.	**LO1.2**	

13.	**Procedural Programming**	Code-along: Organizing code into functions. Task: Converting a monolithic script into separate functions.	**LO1.2, LO2.1**	
14.	**Object-Oriented Programming (OOP) Basics**	Lecture & Demo: Introduction to Classes and Objects. Practical: Defining a simple Car or Student class.	**LO1.2**	
15.	**Functional Programming Basics**	Demo: Using built-in functions like map() and filter(). Concept discussion: Pure functions and immutability.	**LO1.2**	
16.	**Setting Up the Development Environment**	Guided Install: Setting up Python and VS Code. Tutorial: Running and debugging a simple script.	**LO2.1**	
17.	**Writing & Running Your First Program**	Independent Practical: Building a personal bio program that uses variables, I/O, and formatting.	**LO2.1**	
18.	**Building with Functions/Methods**	Practical Lab: Writing reusable functions. Challenge: Creating a simple calculator with functions for each operation.	**LO2.1, LO2.2**	
19.	**Introduction to Debugging**	Workshop: Using the IDE debugger. Activity: Intentionally bugging a program and having peers find the errors.	**LO2.3**	
20.	**Testing Strategies**	Practical Lab: Writing simple test cases for a function. Discussion: The importance of testing in development.	**LO2.3**	
21.	**Capstone Project 1: Problem Definition & Design**	Project Work: Choosing a final project (e.g., quiz, calculator). Creating pseudocode and a flowchart for the solution.	**LO2.2, LO3.2**	
22.	**The Software Development Life Cycle (SDLC)**	Lecture & Case Study: Walking through the SDLC phases for a well-known application like a mobile app.	**LO3.1**	

23.	**SDLC Models: Waterfall vs. Agile**	Group Role-play: Simulating a project using a Waterfall plan vs. an Agile (sprint-based) approach.	**LO3.1**	
24.	**Introduction to Version Control with Git**	Demo & Practical: Initializing a repo, making commits. Task: Tracking the versions of a simple text file.	**LO3.2**	
25.	**Using GitHub for Collaboration**	Practical Lab: Pushing a local repository to GitHub. Activity: Cloning a partner's repo and adding a feature.	**LO3.2**	
26.	**Code Documentation & Comments**	Workshop: Writing meaningful comments and a README file for the capstone project. Peer review for clarity.	**LO3.2**	
27.	**Code Readability & Style Guides**	Practical Session: Refactoring messy code for readability. Using linters (e.g., Pylint) to check for style.	**LO3.2**	
28.	**Ethics: Intellectual Property & Licensing**	Discussion: Scenarios involving copying code from Stack Overflow. Exploring open-source licenses (MIT, GPL).	**LO3.3**	
29.	**Ethics: Privacy, Security & Bias**	Case Study Analysis: How a poorly designed algorithm can lead to biased outcomes or privacy leaks.	**LO3.3**	
30.	**Capstone Project 2: Implementation**	Project Lab: Core development time for the final application, with instructor support and peer debugging.	**LO2.1, LO2.2, LO2.3**	
31.	**Capstone Project 3: Version Control & Docs**	Project Lab: Using Git to manage the project, writing final documentation, and creating a presentation.	**LO3.2**	
32.	**Debugging & Testing Clinic**	Support Session: A dedicated lab for troubleshooting capstone projects and writing final test cases.	**LO2.3, LO3.2**	

33.	**Project Presentations & Demo**	Presentation: Students present their working application, explain their code, and demonstrate its functionality.	**LO2.1, LO3.2**	
34.	**Course Review & Future Pathways**	Review Session: Key concepts recap. Discussion: Next steps in programming (web dev, data science, etc.).	**LO1, LO2, LO3**	