MODULE DESCRIPTION FORM

نموذج وصف المادة الدراسية

Module Information معلومات المادة الدراسية					
Module Title	**Programming Fundamentals I**			**Module Delivery**	
Module Type	**Core**			☒ **Theory** ☒ **Lecture** ☒ **Lab** ☐ **Tutorial** ☒ **Practical** ☐ **Seminar**	
Module Code	**CYS1103**				
ECTS Credits	**7**				
SWL (hr/sem)	**175**				
Module Level	UG1		**Semester of Delivery**	1	
Administering Department	Cybersecurity		**College**	College of Computer Science & Information Technology	
Module Leader	Ali Kareem Abdul Raheem		**e-mail**	alialmujab@uowa.edu.iq	
Module Leader's Acad. Title	Lecturer		**Module Leader's Qualification**	Ph.D.	
Module Tutor	Ali Kareem Abdul Raheem		**e-mail**	alialmujab@uowa.edu.iq	
Peer Reviewer Name	Nabil Sadiq Abdul abbas		**e-mail**	nabeel@uowa.edu.iq	
Scientific Committee Approval Date	24/12/2025		**Version Number**	1.0	

Relation with other Modules العلاقة مع المواد الدراسية الأخرى			
Prerequisite module	None	**Semester**	
Co-requisites module	None	**Semester**	

\

Department Head Approval **Dean of the College Approval**

Module Aims, Learning Outcomes and Indicative Contents أهداف المادة الدراسية ونتائج التعلم والمحتويات الإرشادية	
Module Aims أهداف المادة الدراسية	1. Introduction to Programming: The module aims to introduce students to the fundamental concepts and principles of programming. It provides an overview of programming languages, their purpose, and their role in software development. 2. Basic Programming Constructs: The module aims to familiarize students with the basic programming constructs such as variables, data types, operators, and expressions. It focuses on teaching them how to use these constructs to write simple programs. 3. Control Structures: The module aims to introduce students to control structures such as loops and conditionals. It teaches them how to use these structures to control the flow of program execution and make decisions based on certain conditions. 4. Functions and Procedures: The module aims to teach students about functions and procedures, their purpose, and how to define and use them in programming. It focuses on modular programming and code reusability. 5. Input/Output Operations: The module aims to familiarize students with input/output operations in programming. It covers techniques for reading input from the user, displaying output, and interacting with files. 6. Problem-Solving Skills: The module aims to develop students' problem-solving skills by presenting them with programming challenges and exercises. It emphasizes the importance of breaking down problems into smaller steps, designing algorithms, and implementing solutions using programming constructs. 7. Debugging and Troubleshooting: The module aims to equip students with skills in identifying and resolving common programming errors. It teaches them techniques for debugging code, tracing program execution, and handling errors effectively.
Module Learning Outcomes مخرجات التعلم للمادة الدراسية	1. Understand the basic concepts of programming: Students should be able to explain the fundamental concepts of programming, including variables, data types, control structures, and functions. 2. Write and run simple programs: Students should be able to write simple programs using a programming language, demonstrating an understanding of basic syntax and semantics. They should be able to compile or interpret their programs and execute them successfully. 3. Apply problem-solving techniques: Students should be able to analyze and break down simple problems into smaller, manageable tasks. They should demonstrate the ability to design algorithms and implement solutions using appropriate programming constructs. 4. Use programming constructs effectively: Students should be able to utilize programming constructs such as loops, conditionals, and functions to control program flow, make decisions, and perform repetitive tasks. 5. Debug and troubleshoot programs: Students should be able to identify and correct common errors in their programs. They should be able to use debugging techniques and employ strategies to troubleshoot their code

	effectively. 6. Demonstrate basic data manipulation skills: Students should be able to work with basic data structures such as arrays, lists, or strings. They should demonstrate proficiency in manipulating and accessing data stored in these structures. 7. Apply input/output operations: Students should be able to incorporate input/output operations into their programs. They should demonstrate the ability to read input from users, display output, and interact with files as needed. 8. Understand basic software development principles: Students should have an awareness of software development principles such as code organization, code reusability, and modularity. They should be able to write clear, readable, and maintainable code following coding conventions and best practices. 9. Collaborate effectively in programming projects: Students should demonstrate the ability to work collaboratively in a programming project, effectively communicating with team members, sharing code, and using version control systems.
Indicative Contents **المحتويات الإرشادية**	The indicative contents of a programming fundamentals include the following topics: 1. Introduction to Programming: o Definition and importance of programming o Overview of programming languages and their uses o Introduction to a specific programming language (e.g., Python) and its features 2. Variables and Data Types: o Introduction to variables and their purpose o Basic data types (e.g., integers, floating-point numbers, strings, booleans) o Variable declaration and assignment 3. Control Structures: o Introduction to control structures (e.g., if statements, loops) o Conditional statements (e.g., if-else, nested if statements) o Looping structures (e.g., while loop, for loop) 4. Functions and Procedures: o Definition and purpose of functions o Function declaration and invocation o Passing arguments to functions and returning values o Introduction to predefined functions and libraries 5. Problem Solving and Algorithmic Thinking: o Understanding and defining problems o Breaking down problems into smaller tasks o Developing algorithms and step-by-step solutions o Translating algorithms into code

Learning and Teaching Strategies استراتيجيات التعلم والتعليم	
Strategies	When teaching programming fundamentals to first-grade students in an Information Technology department, it is important to employ strategies that are suitable for their age and learning level. Here are some effective strategies: 1. Hands-on Activities: Use interactive and hands-on activities to engage students actively in the learning process. For example, provide puzzles, games, or physical objects that represent programming concepts like variables or loops. This approach helps make abstract concepts more tangible and enjoyable. 2. Visual Representations: Utilize visual aids such as diagrams, flowcharts, or illustrations to help students visualize programming concepts. Visual representations can assist in understanding the flow of program execution, the relationship between different programming constructs, and the logic behind algorithms. 3. Gamification: Integrate gamification elements into programming exercises and assignments. Create coding challenges, competitions, or educational games that motivate students to apply programming concepts creatively. This approach promotes active learning, problem-solving, and healthy competition among students. 4. Collaborative Learning: Encourage collaborative learning by facilitating group projects or pair programming activities. Collaborative learning fosters communication, teamwork, and the exchange of ideas among students. It also allows students to learn from each other and collectively solve programming problems. 5. Step-by-Step Approach: Break down programming concepts into small, manageable steps. Start with simple and concrete examples before moving on to more complex topics. Provide clear instructions and explanations, demonstrating each step in the process. This incremental approach helps students grasp concepts gradually and build their programming skills effectively. 6. Real-Life Examples: Connect programming concepts to real-life scenarios that students can relate to. Use examples from everyday situations, such as creating a program to calculate the total cost of items in a shopping cart or simulating a traffic light system. Relating programming to real-world applications makes it more relevant and engaging for students. 7. Interactive Online Resources: Utilize interactive online resources, educational programming games, or kid-friendly coding platforms specifically designed for young learners. These resources often provide interactive tutorials, visual programming environments, and immediate feedback, making the learning experience more interactive and enjoyable. 8. Individualized Support: Provide individualized support and feedback to students. Offer assistance to those who are struggling and provide additional challenges to those who grasp concepts quickly. Regularly

	assess students' progress and address their specific learning needs to ensure that they are making steady progress. 9. Encourage Creativity: Foster creativity by encouraging students to think creatively and find innovative solutions to programming problems. Provide opportunities for them to apply programming concepts in creative projects, such as designing simple games or creating animations. This approach encourages critical thinking, problem-solving, and the exploration of their own ideas. 10. Reflective Practice: Incorporate reflection and self-assessment activities into the learning process. Encourage students to review their own code, identify areas for improvement, and reflect on their problem-solving approaches. This reflective practice helps students develop a deeper understanding of programming concepts and improves their ability to analyze and debug their code.

Student Workload (SWL)

الحمل الدراسي للطالب محسوب لـ 15 اسبوعا

Structured SWL (h/sem) الحمل الدراسي المنتظم للطالب خلال الفصل	78	**Structured SWL (h/w)** الحمل الدراسي المنتظم للطالب أسبوعيا	5
Unstructured SWL (h/sem) الحمل الدراسي غير المنتظم للطالب خلال الفصل	97	**Unstructured SWL (h/w)** الحمل الدراسي غير المنتظم للطالب أسبوعيا	7
Total SWL (h/sem) الحمل الدراسي الكلي للطالب خلال الفصل	175		

Module Evaluation

التقييم التكويني للطالب

		Time/Number	Weight (Marks)	Week Due	Relevant Learning Outcome
Formative assessment	Quizzes	5	8% (8)	2, 4, 5, 7, 9	1, 2, 3, 4
	Projects	1	5% (5)	10	All
	Attendance and Lab execution	10	15% (15)	2, 3, 4, 5, 6, 7, 8, 9, 10, 11	All
	H. W.	5	7% (7)	2, 4, 5, 7, 9	2, 3, 4, 5, 7 8, 9, 10, 11
	Weekly Class Activity with Attendance	5	5% (5)	2, 3, 4, 5, 6, 7, 8, 9, 10, 11	All
	Formative assessment	40%(40)			
Summative assessment	Midterm Exam	2hr	10%(10)	7	All
	Final Exam	3hr	50% (50)	16	
Total assessment			100% (100 Marks)		

Delivery Plan (Weekly Syllabus)	
	Material Covered
Week 1	Introduction to Programing Fundamentals
Week 2	Learn how to change problem to algorithm
Week 3	Pseudo Code and Flowchart
Week 4	Variables, Assignment Statements, and Expressions
Week 5	Augmented Assignment Operators
Week 6	Encode characters using ASCII
Week 7	**MidTerm Exam**
Week 8	Case Study: Minimum Number of Coins
Week 9	Write Boolean expressions using comparison operators
Week 10	if Statements and Common Errors in Selection Statements
Week 11	Logical Operators and Generating Random Numbers
Week 12	Loops and Nested loop
Week 13	Keywords Break and Continue
Week 14	**Preparation for final exam**

Delivery Plan (Weekly Lab. Syllabus)	
	Material Covered
Week 1	Lab 1: Introduction to Programming Fundamentals
Week 2	Lab 2: Getting started with C++
Week 3	Lab 3: Reading Input from the Console
Week 4	Lab 4: learn variables
Week 5	Lab 5: learn Assignment Statements, and Expressions
Week 6	Lab 6: Read strings from the keyboard and Encode characters using ASCII
Week 7	**MidTerm Exam**
Week 8	Lab 8: Write Boolean expressions using comparison operators
Week 9	Lab 9: implement if Statements
Week 10	Lab 10:learn how to read Common Errors in Selection Statements
Week 11	Lab 11: learn Logical Operators
Week 12	Lab 12: Generating Random Numbers
Week 13	Lab 13: while loop, for loop, and Nested loop
Week 14	**Preparation for final exam**

Learning and Teaching Resources
مصادر التعلم والتدريس

	Text	Available in the Library?
Required Texts	C++:The Complete Reference, Fourth Edition Herbert Schildt McGraw-Hill/Osborne New York Chicago San Francisco Lisbon London Madrid Mexico City Milan New Delhi San Juan Seoul Singapore Sydney Toronto	No
Recommended Texts	OqeiliSalch, prof. Department of IT-AL-Balqa Applied University	No
Websites	https://www.w3schools.com/cpp/cpp_intro.asp	

Grading Scheme
مخطط الدرجات

Group	Grade	التقدير	Marks (%)	Definition
Success Group (50 - 100)	**A -** Excellent	امتياز	90 - 100	Outstanding Performance
	B - Very Good	جيد جدا	80 - 89	Above average with some errors
	C - Good	جيد	70 - 79	Sound work with notable errors
	D - Satisfactory	متوسط	60 - 69	Fair but with major shortcomings
	E - Sufficient	مقبول	50 - 59	Work meets minimum criteria
Fail Group (0 – 49)	**FX –** Fail	راسب (قيد المعالجة)	(45-49)	More work required but credit awarded
	F – Fail	راسب	(0-44)	Considerable amount of work required

Note: Marks Decimal places above or below 0.5 will be rounded to the higher or lower full mark (for example a mark of 54.5 will be rounded to 55, whereas a mark of 54.4 will be rounded to 54. The University has a policy NOT to condone "near-pass fails" so the only adjustment to marks awarded by the original marker(s) will be the automatic rounding outlined above.