Unit Description Form

Course Description Form

Faculty of Engineering / Department of

Unit Information
Course Information

Unit Title	**Computer Programming**	Unit delivery
Unit Type	secondary	نظريه ☒ حاضر ☒ المختبر ☒ تعليمي ☐ عملي ☐ ☐ Seminar
Unit Code	BME-12-04	
ECTS Credits	8	
SWL (ساعة / SEM)	75	

Unit level	2	Delivery Semester	2
Department of Administration	Biomedical Engineering	College	Faculty of Engineering
Unit Commander	Karrar aqeel huseein	E-mail Address	karrar.aqeel@uowa.edu.iq
Title of Unit Commander	Assistant Lecturer	Unit Commander Qualifications	Master
Unit Teacher		E-mail Address	
Peer Reviewer Name		E-mail Address	E-mail Address
Date of accreditation of the Scientific Committee	22/1/2025	Version number	1.0

Relationship with other units
Relationship with other subjects

Prerequisites Unit	No	Semester	
Common Requirements Unit	No	Semester	

Unit objectives, learning outcomes and how-to contents Course objectives, learning outcomes and instructional contents	
Objectives of the Unit Course Objectives	1. **Teaching the basics of programming**: Understand basic concepts such as variables, conditional statements, and loops. 2. **Proficiency in programming languages**: Enable students to write programs using languages such as C and C++. 3. **Algorithm Design**: Develop the ability to design effective algorithms to solve software problems. 4. **Understanding data structures**: Learn how to use different data structures such as arrays and lists. 5. **Application of object-oriented programming (OOP):** Teaching object-oriented programming principles such as objects and classes. 6. **Teaching debugging techniques**: improving debugging and code analysis skills. 7. **Apply advanced programming concepts**: Enable students to use advanced programming libraries and frameworks.
Unit Learning Outcomes Learning outcomes of the course	Understand programming principles: Gain knowledge of programming basics such as variables, conditional statements, and loops. Proficiency in programming languages: Ability to write programs using languages such as C and C++. Algorithm Design: Develop skills to design and implement effective problem-solving algorithms. Use data structures: Effectively apply data structures such as arrays, lists, and trees. Object-oriented programming (OOP): Understand and apply object-oriented programming principles such as objects and classes. Error analysis and correction: Develop debugging skills and improve code. Apply advanced concepts: the use of software libraries and frameworks, and the programming of multi-threaded applications. 1.
Indicative Contents Indicative Contents	1. Basic programming concepts: Learn the basics of programming such as variables, graphic types, and conditional structures. 2. C/C++ Programming: Learn C or C++ as an application development tool. 3. Algorithms: The study of how algorithms are designed and implemented to solve software problems. 4. Data structures: Learn how to use structures such as threaded lists, arrays, trees. 5. Object-oriented programming (OOP): Learn the principles of object-oriented programming such as objects and classes. 6. Debugging: Techniques for finding and correcting errors in code. 7. Advanced concepts: Learn programming using libraries and frameworks, and programming multi-threaded applications.

Learning and Teaching Strategies Learning and Teaching Strategies

Strategies	1. Active Learning: Encourage students to actively participate by solving exercises and problems themselves, enhancing their understanding of mathematical concepts. 2. Collaborative learning: teamwork to solve mathematical problems, helping to exchange ideas and develop analytical skills. 3. Project-based learning: Using applied mathematical projects that link mathematics to everyday life, such as studying statistics or engineering designs. 4. Ongoing Assessment: Conduct regular quizzes and exercises to track students' progress and identify points that need to be strengthened. 5. Interpretation and Discussion: Encourage students to explain their solutions and ways of thinking to stimulate deep understanding and improve communication skills.

Student Workload (SWL) The student's academic load is calculated for 15 weeks			
SWL منظم (h / sem) Regular academic load of the student during the semester	35	**SWL regulator(h/s)** Regular student load per week	5
SWL غير منظم (h / sem) Irregular academic load of the student during the semester	35	**Unregulated SWL (h/s)** Irregular student academic load per week	5
SWL إجمالي (h / sem) The student's total academic load during the semester	75		

Unit Evaluation Course Evaluation					
	As	**Time/Number**	**Weight (tags)**	**Week due**	**Related learning outcomes**
Formative Assessment	**Contests**	2	10% (10)	5, 10	LO #1 , 2, 10 and 11
	Assignments	2	10% (10)	2, 12	LO #3 , 4, 6 and 7
	Projects /Laboratory.	1	10% (10)	continuous	every
	report	1	10% (10)	13	LO #5 , 8 and 10
Final Assessment	**Midterm Exam**	2 hr	10% (10)	7	LO #1-7
	Final Exam	2 hours	50% (50)	16	every
	Overall Rating		100% (100 degree)		

Grading chart

Grading chart

group	degree	Appreciation	Tags (%)	definition
An-Najah Group (50 - 100)	**A -** Excellent	privilege	90 - 100	Outstanding Performance
	B - Very Good	Very good	80 - 89	Above average with some errors
	C - Good	Good	70 - 79	Proper work with noticeable errors
	D - Satisfactory	medium	60 - 69	Fair but with significant shortcomings
	E - sufficient	Acceptable	50 - 59	The work meets the minimum standards
Group failure (0 – 49)	**FX -** Failed	Deposit (in (processing	(45-49)	More work required but credit granted
	F - Failed	Failure	(0-44)	Large amount of work required

Note: Signs that are more than 0.5 decimal places greater than or below the full mark will be rounded higher or lower (for example, a score of 54.5 will be rounded to 55, while a mark of 54.4 will be rounded to 54. The university has a policy of not tolerating "imminent traffic failure", so the only modification to the marks granted by the original mark(s) will be the automatic rounding described above.