1.	**Module Code**	**CFM2103**	
2.	**Module Title**	**Mathematical Programming**	
3.	**Schools involved in delivery**	Computing and Engineering	
4.	**Name of Course(s)**	BSc(Hons) Mathematics MMath Mathematics	
5.	**Module Leader**	Ciprian D. Coman	
6.	**Location**	Queensgate	
7.	**Module Type**	Core	
		Choose an item.	
8.	**Credit Rating**	20	
9.	**Level**	F (FHEQ 4)	
10.	**Learning Methods**	Lectures	**44** hr
		Practical Classes and Demonstrations	**22** hr
		Choose an item.	hr
		Guided Independent Study	**134** hr
		Total hours	200 hr
11.	**Pre-requisites**	None	
12.	**Recommended Prior Study**	None	
13.	**Co-requisites**	None	
14.	**Shared Teaching**	None	
15.	**Professional Body Requirements**	None	
16.	**Graded or Non Graded**	Graded	
17.	**Barred Combinations**	None	

18. Synopsis

The aim of this module is to introduce you to computer programming in general, and to Python programming specifically. You will develop an understanding of basic programming concepts and structures such as loops, conditional statements, iterations, functions, variables, scope etc. You will then go on to develop an understanding of object-oriented concepts such as data encapsulation, inheritance and polymorphism. By using Python as the programming language of choice, you will investigate mathematical problems taken from the material taught in lectures and will develop skills for the practical implementation of various types of algorithms discussed in class.

19. Learning Strategy

Contact time typically consists of lectures and tutorial sessions. Lectures will introduce the theoretical foundations of programming and related mathematical concepts, and will include example applications drawn from topics associated with various branches of mathematics (e.g., calculus, geometry, discrete mathematics, etc).

Students will be taught how to apply their skills in programming to translate various mathematical solution strategies into appropriate software solutions which they will develop in the tutorials to explore and support application of the theoretical concepts.

20. Outline Syllabus

Programming topics:

Introduction to computers and programming; algorithms; flowcharts; pseudocode;

Overview of Python (e.g., how it works, architecture, modules/libraries, etc).

Data types; variables; named constants.

Data structures (lists, tuples, arrays, dictionaries, sets); introduction to Numpy.

Decision structures and Boolean logic (e.g., the conditional ‘if-else’ clause).

Repetition structures (e.g., ‘for’ and ‘while’ loops).

Functions; parameter passing; scope (local vs. global).

Working with files; writing and reading data.

Guidelines for computer program design; code refactoring.

Introduction to user-defined data types (classes).

Aspects of inheritance and polymorphism.

Recursion.

Exception handling in Python.

Plotting mathematical functions in 2D and 3D (with Matplotlib).

Symbolic Toolbox.

	Algorithms/Mathematical content: Floating-point numbers and their computer representation; review of binary and decimal number systems; relative and absolute errors in scientific computing. Introduction to algorithm complexity; big O and little o notations; related concepts. Searching algorithms (e.g., binary search and/or linear search). Sorting algorithms (e.g., one or more of the following: bubble/insertion/merge/quick sort). Recurrence relations & mathematical induction (e.g., Fibonacci numbers, general solution of a recurrence relation, etc). Iterative methods for the solutions of algebraic equations in one unknown and/or for differential equations (e.g., Newton's method and/or finite-difference approximations for first-order ODEs).
21.	**Learning Outcomes** On successful completion of this module, students will be able to:
	Knowledge and Understanding Outcomes **1.** Demonstrate an understanding of procedural programming using Python. **2.** Demonstrate understanding of fundamental concepts in object-oriented programming using Python.
	Ability Outcomes **3.** Analyze mathematical problems and design and develop algorithms using mathematical programming tools. **4.** Implement, test, and debug computer programs in Python using a mixture of built-in and user-defined functions (or classes).
22.	**Assessment Strategy**
	22.1 ***Formative assessment*** Opportunity for formative assessment will be provided through the practical sessions. Work will be set and feedback will be provided on the students' efforts. Model answers will be provided to certain exercises for comparison and evaluation. Students will be able to gain informal feedback on work in progress by taking it to the practical for their tutor to review. Students will also have the opportunity to self-asses their work to gain a greater understanding of the assessment process.

22.2 *Summative Assessment*

Assessment tasks (including assessment weightings)

Assessment Task		Assessment Weighting (%)	Learning outcomes
1. Portfolio	2,000 words (20 notional assessment hours)	50%	1, 3
The portfolio will consist of several programming exercises designed to test the learner's understanding of procedural programming. This portfolio will be developed as work carried out in class and through directed independent study. Some of the exercises will simply check your knowledge of the Python syntax, while others will test your ability to translate basic mathematical algorithms into computer code.			
Tutor assessed			
Available for tutor re-assessment			
Not anonymously marked			

2. Portfolio	2,000 words (20 notional assessment hours)	50%	2, 4
The second piece of coursework will be mostly related to object-oriented programming, and it serves a twofold purpose. First, this coursework is meant to check the learner's understanding of the syntax for creating classes (identifying suitable instance attributes and defining the appropriate methods for handling them), implementing inheritance and operator overloading, etc. Second, the coursework is also used for evaluating the learner's ability to design useful OOP code in Python, which can be applied to solving simple mathematical problems.			
Tutor assessed			
Available for tutor re-assessment			
Not anonymously marked			
This is the final element of assessment			

Assessment Criteria

Coursework 100%: divided into two submission points weighted 50% / 50% as indicated above. The total effort of the combined components should be equivalent to 4,000 words (approximately 40 hours).

The first coursework will cover material based on procedural programming, while the second piece of coursework will be on object-oriented concepts and their application to a range of mathematical problems.

The coursework will involve developing a portfolio of programming exercises designed to solve a range of problems drawn from various areas of mathematics (e.g., calculus,

	combinatorics, number theory, geometry). The portfolio will be developed as work carried out in class and through directed independent study. This portfolio demonstrates the learner's understanding of and ability to apply programming concepts and techniques in problem solving, as well as their ability to use various mathematical libraries in Python (e.g., Matplotlib and Numpy). Tutor re-assessment is available for both Coursework components. The assessment will not be marked anonymously.
	My Reading http://library3.hud.ac.uk/myreading/lists/CFM2103