Module Code	CSU34016
Module Name	Introduction to Functional Programming
ECTS Weighting[1]	Choose an item.
Semester taught	Choose an item.
Module Coordinator/s	Dr Andrew Butterfield
Module Learning Outcomes	On successful completion of this module, students will be able to: LO1. Develop programs in a high level functional language LO2. Analyse and structure program designs in terms of functional concepts LO3. Understand the concept of higher-order programming inherent in functional languages LO4. Apply a number of functional programming techniques and tools to develop effective functional systems LO5. Use logical proof techniques to reason about functional programs. LO6. Understand and use appropriate software development tools
Module Content	Functional programming languages present a powerful, abstract, and important direction in programming languages. The high level of abstraction and the expressive syntax makes program decomposition and composition unusually easy, while the close connections to the underlying semantics make formal reasoning tractable. Systems such as Google's "Map/Reduce" framework demonstrate the influence of this approach, and the importance to a computer scientist of understanding it. This modules uses the functional programming language called "Haskell" to introduce key concepts such as how to compute using functions, and how these languages can easily support powerful features such as pattern-matching, recursion, strong typing, type polymorphism, higher-order functions, laziness, and type classes. Practical exercises are based around the use of software development tools (Haskell "stack") that support test automation. We also study the theoretical and formal underpinnings of such languages (lambda calculus), and how much easier it is to do formal reasoning regarding program proprties using logical proofs.
Teaching and Learning Methods	The course is delivered mainly through a mix of lectures and tutorials, with regular short graded exercises being given to assess learning outcomes. There are a few laboratory slots early in the course, whose sole purpose is to ensure that all students can work effectively with the software development tools.

[1] TEP Glossary

Assessment Details[2]

Assessment Component	Brief Description	Learning Outcomes Addressed	% of total	Week set	Week due
Examination	2 hour written examination	LO1, LO2, LO3, LO4, LO5	80%	n/a	n/a
Exercise0	Tool Usage	LO6, LO1	2%	1	2
Exercise1	Basic Programming	LO1, LO2, LO6	4%	2	4
Exercise2	Higher Order Programming	LO3, LO2, LO1, LO6	4%	4	6
Exercise3	Application Program and Proof (part 1)	LO4, LO5, LO3, LO2, LO6	5%	6	8
Exercise4	Application program and Proof (part 2)	LO4, LO5, LO3, LO2, LO6	5%	9	11

Reassessment Details

Examination (2 hours, 100%)

Contact Hours and Indicative Student Workload

Contact Hours (scheduled hours per student over full module), broken down by:	**34 hours**
lecture	22 hours
laboratory	2 hours
tutorial or seminar	11 hours
other	0 hours
Independent study (outside scheduled contact hours), broken down by:	**72 hours**
preparation for classes and review of material (including preparation for examination, if applicable)	36 hours
completion of assessments (including examination, if applicable)	36 hours
Total Hours	**106 hours**

Recommended Reading List

Module Pre-requisites

Prerequisite modules: none

Other/alternative non-module prerequisites: familiarity with a main-stream (imperative) programming language, such as Java, C, python, or similar, and some experience in programming with that language. **No** prior experience with functional programming languages such as Haskell, ML, OCaml, Scheme, LISP, etc., is required.

Module Co-requisites

Module Website

Last Update

11/07/2019 by Your Name

[2] TEP Guidelines on Workload and Assessment