1. **Title of the module**

COMP5450 (CO545) Functional Programming

1. **School or partner institution which will be responsible for management of the module**

School of Computing

1. **The level of the module (Level 4, Level 5, Level 6 or Level 7)**

Level 5

1. **The number of credits and the ECTS value which the module represents**

15 credits (7.5 ECTS)

1. **Which term(s) the module is to be taught in (or other teaching pattern)**

Autumn

1. **Prerequisite and co-requisite modules**

CO3BB Problem solving with algorithms or equivalent experience.

1. **The programmes of study to which the module contributes**

Computer Science, CS(AI), CS(Networks), plus Year in Industry variants of these programmes.

Also available as part of MSc Computer Science or MSc Advanced Computer Science.

1. **The intended subject specific learning outcomes.  
   On successfully completing the module students will:**
   1. Have an understanding of the core concepts of functional programming: functions as the central organising principle, functions as values, side-effect free programming, evaluation via rewriting.
   2. Be able to understand and use recursion, recursive data structures (e.g., lists, trees, mutually recursive data types), and higher-order functions to solve problems.
   3. Be able to implement functional programs to solve problems using appropriate idioms in a modern functional programming language, including understanding the languages main idioms and basic libraries.
   4. Understand the structure of a functional program from the perspective of its types, including creating their own data structures to solve problems.
   5. Understand the basic theory of the lambda-calculus and its relation to modern mainstream programming languages. Understand how to read context-free descriptions of language syntax e.g. for the lambda calculus.
   6. Have an understanding of the concepts of concurrent programming from a functional perspective: processes, independence, communication, and synchronisation.
   7. Understand program properties exposed by function and concurrent programming: partiality, totality, side-effect freedom, non-termination, determinism, deadlock, and starvation.
2. **The intended generic learning outcomes.  
   On successfully completing the module students will be able to:**
   1. Understand trade-offs in different approaches and designs and make appropriate choices.
   2. Be able to make effective use of IT facilities for scholarship and research.
   3. Be able to manage their time, learning, and development effectively.
3. **A synopsis of the curriculum**

This module introduces students to the functional programming paradigm, using at least one modern functional programming language to put the core concepts into practice. The module will develop both the foundation and theory of this paradigm, as well as the practice and application of the paradigm to solve problems and build systems. The module will core topics, including:

* Functions as first-class language constructs and as a central organising principle;
* Higher-order functions and compositional programming;
* Basic semantics of functional languages;
* The role of types in programming;
* Algebraic data types and pattern matching;
* Recursion and recursive data types;
* Differences with imperative and object-oriented programming paradigms;
* Properties of programs, (e.g., purity, side-effect freedom, totality, and partiality).
* The lambda-calculus as a programming model and foundation.
* BNF grammars for representing context-free syntax, and its relation to ADTs and language manipulation.
* Testing and issues of building correct software.

The module will develop practical skills in programming and problem solving using functional programming. There will also be a chance to apply functional programming to help understand better concepts in logic and mathematics.

Later parts of the module will then consider concurrent programming in the context of functional programming, including concurrent programming models and primitives (e.g., message-passing concurrency), parallelism, synchronisation and communication, and properties of deadlock, communication-safety, and starvation.

1. **Reading list (Indicative list, current at time of publication. Reading lists will be published annually)**

“Programming in Haskell” (2nd edition), Graham Hutton, 2016

“The Haskell Road to Logic, Maths, and Programming” Kees Doets, Jan van Eijck, 2004.

“Real World Haskell: Code You Can Believe In” Bryan O’Sullivan et al, O’Reilly Media, 2008.

“Erlang Programming”, Francesco Cesarini and Simon Thompson, O'Reilly Media, 2009.

“Programming Erlang: Software for a Concurrent World”, Joe Armstrong, Pragmatic Bookshelf, 2007.

1. **Learning and teaching methods**

Total contact hours 33

Total private study hours: 117

Total module study hours: 150

1. **Assessment methods**
   1. Main assessment methods

2 In-class assessments (10% each)

2 Courseworks (approximately 15 hours of effort each, 15% each)

Examination (2hrs) (50%)

13.2 Reassessment methods

Like for like

1. **Map of module learning outcomes (sections 8 & 9) to learning and teaching methods (section12) and methods of assessment (section 13)**

Add/delete lines and columns as appropriate:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Module learning outcome** | 8.1 | 8.2 | 8.3 | 8.4 | 8.5 | 8.6 | 8.7 | 9.1 | 9.2 | 9.3 |
| **Learning/ teaching method** |  |  |  |  |  |  |  |  |  |  |
| **Private Study** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |  | **X** |
| classes | **X** | **X** | **X** | **X** | **X** | **X** |  |  | **X** | **X** |
| lectures | **X** | **X** | **X** | **X** | **X** | **X** | **X** |  | **X** |  |
| **Assessment method** |  |  |  |  |  |  |  |  |  |  |
| In-class assessments | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| Coursework | **X** | **X** | **X** | **X** |  | **X** | **X** | **X** | **X** | **X** |
| Examination | **X** | **X** |  | **X** | **X** | **X** | **X** | **X** |  | **X** |

1. **Inclusive module design**

The School recognises and has embedded the expectations of current equality legislation, by ensuring that the module is as accessible as possible by design. Additional alternative arrangements for students with Inclusive Learning Plans (ILPs)/declared disabilities will be made on an individual basis, in consultation with the relevant policies and support services.

The inclusive practices in the guidance (see Annex B Appendix A) have been considered in order to support all students in the following areas:

a) Accessible resources and curriculum

b) Learning, teaching and assessment methods

1. **Campus(es) or centre(s) where module will be delivered**

Canterbury

1. **Internationalisation**

The topics addressed by this module relate to a field which is of international importance, given the global role of computers in today's technological innovation and the use of functional programming worldwide. The topics covered by this module are international in nature, being identical worldwide and independent of traditional spoken language.

**FACULTIES SUPPORT OFFICE USE ONLY**

**Revision record – all revisions must be recorded in the grid and full details of the change retained in the appropriate committee records.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date approved | Major/minor revision | Start date of delivery of revised version | Section revised | Impacts PLOs (Q6&7 cover sheet) |
| 26/02/19 | Major | September 2020 | 6,11,13 | no |
|  |  |  |  |  |