1. **Title of the module**

COMP6630 (CO663) – Programming Languages: Applications and Design

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 6

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 or Spring

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

Pre-requisites: COMP5200 Further Object-Oriented Programming; COMP5450 Functional and Concurrent Programming

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

Computer Science and related programmes, including year in industry versions.

1. **The intended subject specific learning outcomes.
On successfully completing the module students will be able to:**
	1. Describe and compare programming paradigms: declarative, functional, imperative, object-oriented.
	2. Solve small-scale programming problems in a range of programming languages.
	3. Compare programming languages objectively, in order to choose the right tool for a given task.
	4. Make effective use of a range of tools, such as editors, compilers and interpreters.
2. **The intended generic learning outcomes.
On successfully completing the module students will be able to:**
	1. Demonstrate comprehension of the trade-offs involved in design-choices.
	2. Make effective use of IT facilities for solving problems.
	3. Manage their own learning and development, through self-directed study and working on continuous assessment.
3. **A synopsis of the curriculum**

This module shows students what trade-offs are involved in designing a programming language, and how those trade-offs ultimately influence programmer productivity. The module starts with a quick, example-based introduction to the basics of programming languages. It then continues with a series of problems that are each solved in several programming languages. After each problem, we stop and reflect on which language features help and which hinder. Finally, towards the end of the module, several of the language features previously identified are discussed in a more general setting. Indicative examples are:

* Basics of programming languages, such as: C++, C#, Dart, Go, Haskell, Java, Javascript, MATLAB, OCaml, Pyret, Python, Scala, Swift, R, Racket, Rust.
* Problem solving, in multiple languages. The problems will involve concepts such as parsing, evaluation, trees, graphs, memoization, randomization, big data algorithms, reactive user interfaces.
* Language features: pattern matching, first order functions, polymorphism, effects, exceptions, types, algebraic data types, modules, objects, classes.
1. **Reading list (Indicative list, current at time of publication. Reading lists will be published annually)**

Bird, R. (2014). *Thinking Functionally with Haskell*, Cambridge University Press.
Hutton, G. (2016) *Programming in Haskell*, 2nd edition. Cambridge University Press
Krishnamurthi, S. (2015) [available for free online]. *Programming Languages: Application and Interpretation*.
Lopes, C.V. (2014). *Exercises in Programming Style*. Chapman and Hall/CRC.
Minsky, Y., et al. (2013). *Real World OCaml*, O'Reilly Media.

1. **Learning and teaching methods**

Total contact hours: 32 hours

Private study hours: 118 hours

Total study hours: 150 hours

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

Programming Assignment (30%)

Participatory Exercises (15%)

Paper Presentation (25%)

Examination 2 hours (30%)

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)***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Module learning outcome** | 8.1 | 8.2 | 8.3 | 8.4 | 9.1 | 9.2 | 9.3 |
| **Learning/ teaching method** |  |  |  |  |  |  |  |
| **Private Study** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| **Lectures** | **X** | **X** | **X** |  |  |  |  |
| **Practical Sessions** | **X** | **X** | **X** | **X** | **X** | **X** |  |
| **Assessment method** |  |  |  |  |  |  |  |
| In class assessment |  | **X** | **X** | **X** |  | **X** |  |
| Programming assessment |  | **X** | **X** | **X** |  | **X** | **X** |
| Group project | **X** |  |  | **X** | **X** | **X** | **X** |
| Examination | **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 computer programming in today's technological innovation. The programming languages covered by this module are international, 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 the delivery of revised version | Section revised | Impacts PLOs (Q6&7 cover sheet) |
|  |  |  |  |  |
|  |  |  |  |  |

Revised FSO Jan 2018