DBLN CPSI 4491

This course is being offered at Griffith College, CAPA's academic partner in Dublin. The Irish academic system differs from the US, particularly with grading. Griffith College professors expect students to undertake a good deal of independent study to achieve a high mark in their classes. For additional information about this class, please contact the Boston Program Advising Team at 1-800-793-0334.

Programming Paradigms

Continuous Assessment: 60%
Exam: 40%

Intended Module Learning Outcomes

On successful completion of this module learners will be able to:

1. explain and exemplify clearly the defining characteristics of the functional, imperative and object-oriented paradigms;
2. develop solutions to problems using a functional style;
3. write higher-order functions and write programs that use them;
4. use pattern matching when writing functions;
5. prove properties of functional programs using structural induction;
6. outline the main features of the imperative paradigm;
7. develop programs using stepwise refinement and use loop invariants;
8. outline the main features of the object-oriented paradigm;

9. develop programs using the object-oriented model;
10. explain the need for contracts and write classes that implement contracts;
11. write classes in a functional style and write higher-order methods for encapsulated data structures;
12. consider, discuss, and criticise questions of style and aesthetics in programming;
13. develop solutions to problems using each of the three paradigms.

Module Objectives

Programming is, to paraphrase Wittgenstein, a language game and language forms an essential part of our thought process. It is, to quote Boole, *an instrument of human reason, not merely a medium for the expression of thought.* A programming language influences the way we think about programs and the way we construct our solutions to problems. To quote Perlis: *a language that doesn't affect the way you think about programming is not worth knowing*. A programming language is not just a collection of statements it also encompasses in its design a notion of composition, a paradigm that allows *good* programs to be constructed. By *good* I mean programs that are constructed to reflect a particular style of programming. A language is said to support a particular paradigm of programming if it provides ways to write safe, efficient programs in that style. The main objective of this module is to build on the work covered in both first year and second year programming courses. It does this by focusing on the three paradigms: imperative, object-oriented and functional. Both the imperative and object-oriented paradigms are covered in previous courses. Therefore, a greater emphasis is placed on the functional paradigm. This is important in two respects. Firstly, it is a fundamental paradigm supported by many widely used programming languages (Miranda, Haskell, Ocaml, Erlang, Scala, F#). Secondly, it makes it possible to write powerful higher order methods in the object-oriented paradigm. This integration of the two programming paradigms is supported in both Scala(Java Framework) and F#(.Net Framework). A key objective in the module is to provide our learners insight into this new development in programming styles. (It also plays a role in the Concurrent Programming module in 4th year)

Module Curriculum

Introduction

- History of programming languages and their creators.
- Outline of main aspects of three paradigms: Imperative, Object-Oriented and Functional.

Functional Paradigm

- Simple functions and recursion.
- Nested functions and tail recursion.
- Higher-order functions.
- Processing list data structure with first order methods and pattern matching.
- Sorting. Higher order operations on lists: map, filter, partition, forall, exists, foldleft and foldright.
- For comprehension, case classes and text processing.
- Reasoning about functional programs using structural induction.

Imperative Paradigm

- Stepwise program construction, Hoare triplets {P} S {Q} and program construction with proof of correctness(Dijkstra).
- Use of loop invariants.
- Data Structures and analysis of performance

Object-Oriented Paradigm

- System design around data objects in system rather than functionality.
- Communication through interfaces.
- Functional objects and immutable state.
- Operator overloading and higher-order interfaces.
- Composition, inheritance, abstract class, traits (interfaces).
- Stateful objects, design by contract and state invariants (Meyer).
- Genericity, functional data structures and collections.