Overview

This is a course on functional and modular programming.

As a course on programming, the course emphasizes the analysis and solution of problems, the development of correct and efficient algorithms and data structures that embody the solutions, and the expression of the algorithms and data structures in a form suitable for processing by a computer. The focus is more on the human thought processes than on the computer execution processes.

As a course on functional programming, the course approaches programming as the construction of definitions for (mathematical) functions and (immutable) data structures. Functional programs consist of expressions that use these definitions. The execution of a functional program entails the evaluation of the expressions making up the program. Thus this course’s focus is on problem solving techniques, algorithms, data structures, and programming notations appropriate for the functional approach.

As a course on modular programming, the course approaches the construction of large programs as sets of modules that collaborate to solve a problem. Each module is a coherent collection of function and data type definitions. A module hides its private features, allowing their use only within the module, and exposes its public features, enabling their use by the other modules in the program.
This is not a course on functional or modular programming *languages*. In particular, it does not undertake an in-depth study of the techniques for implementing such languages on computers. The focus is on the concepts for programming, not on the internal details of the technological artifact that executes the programs.

Of course, we want to be able to execute our programs on a computer and, moreover, to execute them efficiently. Thus we must become familiar with some concrete programming language and use an implementation of that language to execute our programs. To be able to analyze program efficiency, we must also become familiar with the basic techniques that are used to evaluate expressions.

The academic community has long been interested in functional programming. In recent years, the practitioner community has also become interested in functional programming techniques and language features. There is growing use of languages that are either primarily functional or have significant functional subsets—such as Haskell, OCaml, Scala, Clojure, F#, Erlang, and Elixir. Most mainstream languages have been extended with new functional programming features and libraries—for example, Java, C#, Python, JavaScript, and Swift. Other interesting research languages such as Elm and Idris are also generating considerable interest.

Some versions of this course have used Haskell. Haskell is a “lazy” functional language whose development began in the late 1980’s.

In the Spring 2016 and Spring 2019 versions, we are using the hybrid language Scala, which has a powerful functional subset.

Most of the concepts, techniques, and skills learned in this Haskell-based or Scala-based course can be applied in other functional and multiparadigm languages and libraries.

More importantly, any time we learn new approaches to problem solving and programming, we become better programmers in whatever language we are working. A course on functional programming provides a novel, interesting, and, probably at times, frustrating opportunity to learn more about the nature of the programming task.

Enjoy this “functional programming” course.