ITI0212 Functional Programming
Spring 2026

Callum Reader Eigil Rischel

Course blurb

This course introduces *pure functional programming with dependent types* in the Lean 4 language.

Pure functional programming is the construction of programs by composition and evaluation of pure functions. Pure functions transform inputs of a specified type (A) into outputs of a specified type (B) in a deterministic fashion, without producing side-effects (such as accessing a database or writing to the terminal).

In functional programming languages, functions have a type $A \to B$ like any other, and functions themselves are ordinary *terms*. This means that we can pass functions to other functions, and write functions that manipulate functions, returning new functions. Functional programming can be contrasted with imperative programming, in which computation is represented by series of *statements* which alter the program's state.

Dependent types or *type families* allow types to depend on terms of other types. This abstract sounding principle reflects a ubiquitous pattern of thought (in mathematics and in everyday life), as when for example, we consider "number of days in a month" to depend on the month (and indeed year).

Dependent types form a bridge between computation and logic. Programming with dependent types allows two complementary methods for establishing the correctness of programs: by using types to express strong contracts on a program's behaviour, or by expressing and proving logical statements about our programs by writing programs that amount to proofs of these statements. Both methods can provide much stronger guarantees of program correctness than is possible with testing.

If you have already completed *ITT8060 Advanced Programming*, some concepts from the start of the course may be familiar, but we will quickly encounter new concepts: inductive types in general, typeclasses, monads, dependent types, propositions-as-types, theorem proving, and more. However, *ITT8060 Advanced Programming* is not a prerequisite for this class.

Lean 4

Lean 4 is a new language, with various cutting edge features, although it is not unique in this regard: languages such as `Agda` and `Idris` are broadly comparable, and all of the basic principles pure of functional programming transfer to other pure functional languages such as `Haskell`. As a young language, the ecosystem (e.g. libraries) is relatively sparse, so Lean might not be your first choice for "batteries included" programming, but this is changing by the day. In the past, `ITI0212` has been taught with `Idris`. We have switched to `Lean` mostly because it is easier to set up, and has a more mature interface for *theorem proving*, which we shall encounter later in the course.

Course structure

The main components of the course are as follows.

lecture: Each week there is a lecture, which introduces new theoretical concepts together with examples of their application.

lab: Each week there is also a pratical session, in which students complete tasks designed to build experience and competence with the concepts presented in lecture. Labs do not count towards the course grade, however. Lab sessions are the best opportunity to test your understanding and ask for help or clarification if necessary.

homework: Approximately every three weeks there is a homework assignment consisting of a number of short exercises intended to build further competence and provide a basis for assessment of student progress. You will have two weeks to complete assignments.

project: There is a course project, consisting of a larger and less precisely specified programming task, allowing students to demonstrate creative problem-solving in addition to programming skills. The project will be announced around the start of April, and you will have about a month to complete it.

exam: At the end of the course there is an individual interactive final examination, in which instructors ask students to solve short problems and explain concepts based on what they have learned in the course.

Assignment submission

Submission of homework assignments will be conducted through the TalTech GitLab server (`https://gitlab.cs.ttu.ee`).

Shortly after an assignment deadline, student submissions will be pulled for marking and feedback will be communicated to students through the GitLab issue tracker.

Student evaluation

Course grades are determined from several components, weighted as follows:

Homework assignments	1/3
Course project	1/3
Final exam	1/3

When assessing each assignment, marks will be weighted as follows:

Correctness	2/3
Style (clarity, concision, efficiency)	1/3

Academic Integrity

Collaboration and learning from one another are strongly encouraged, while copying answers and cheating are strictly forbidden. You are expected to be able to distinguish the two. If you are contemplating an action, and you're not sure into which category it falls, you should consider whether what you intend to submit for evaluation is the product of your own efforts and represents your own understanding of the concepts involved. If it is/does not, then you should not submit the work as your own.

Academic Accommodations

The course staff are committed to fostering an accessible and inclusive learning environment where all students feel welcome, comfortable, and treated fairly. If you have any concerns or suggestions for improvement, or would like to request an individual accommodation please let us know.

Syllabus

(Subject to change.)

1. Introduction
2. Inductive types and recursive functions
3. Parameterized types and generic functions
4. Function literals and higher-order functions
5. Type Classes I
6. Type Classes II: Monoids, functors, monads
7. Monads I
8. Monads II (IO, transformers)
9. Propositions as Types I
10. Correctness by construction I
11. Propositions as types II: First-order logic
12. Inductive equality
13. Correctness by construction II
14. Correctness by theorem proving I
15. Correctness by theorem proving II
16. Review