

Introduction to 02157 Functional programming

The foundation of functional programming languages originates from Lambda Calculus, that was introduced by Alonso Church in the 1930ties in mathematical and logical studies of functions and function compositions. Even though Lambda Calculus was invented before computers were, Lambda Calculus captures the notion of computability and the first uncomputability results were actually expressed in Lambda Calculus terms before the appearance of concrete computers.

In the simplest untyped form, Lambda Calculus has just three kinds of expressions e :

- *variables* x
- *abstractions* $\lambda x.e$
- *applications* $e_1 e_2$

where an abstraction represents an anonymous function (the function of x given by e) and application represents function application. If the application has the form $(\lambda x.e) e_2$ then it is informally evaluated by substituting every free occurrence of x in e by e_2 (leaving out some details).

The first functional programming language LISP was published by John McCarthy in 1960. The notation of LISP is kept at a minimum directly reflected by the above described constructs of Lambda Calculus. While LISP was a dynamically scoped, untyped (dynamically typed) language, some prominent successors of LISP, that is Common LISP and Scheme, are statically scoped. LISP was originally used in the area of Artificial Intelligence.

Later the development of statically typed functional languages began. One branch is the ML family of languages (including ML, Standard ML, OCAML and F#). These languages are based on an *eager evaluation strategy*, where the argument e_2 in an application $(\lambda x.e) e_2$ is evaluated to a value, which then is substituted in for x in e . Another branch is based on a *lazy evaluation strategy*. Prominent representatives of this branch are Miranda and Haskell.

Although ML (one of the very early typed functional language) was developed to support theorem proving, it soon turned out that the typed functional programming languages actually constitute convenient general-purpose programming languages. You may get some impression on the use of functional programming languages from the following web-pages:

- <http://homepages.inf.ed.ac.uk/wadler/realworld/>
- <https://fsharp.org/testimonials/>

According to the *Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, by ACM and IEEE, December, 2013, topics covered by this course should be a mandatory part of any bachelor-level computer science education.

1 Teaching rooms, time slots and exam form

Practical information can be found here:

<http://courses.compute.dtu.dk/02157/PracticalInfo.html>.

Be aware that the exam form is: Written exam - 4 hours, **no aid**.

2 Textbook and syllabus

We will use the textbook:

- *Functional Programming using F#*, Michael R. Hansen and Hans Rischel
Cambridge University Press, 2013:

The book is available electronically in DTU's library and you can use the following link to get the book (with study discount) from Polyteknisk Boghandel:

https://polyteknisk.dk/home/dtu/soege_resultat?utf8=%E2%9C%93&q=02157&b=20&st=c&code=&exact_match=false

The textbook has a supporting webpage: <http://compute.dtu.dk/~mire/FSharpBook/>

The following parts of the book (approx. 200 pages) will be covered in the course:

- Chapter 1: Getting started
- Chapter 2: Values, operators, expressions and functions
- Chapter 3: Tuples, records and tagged values
- Chapter 4: Lists
- Chapter 5: Collections: Lists, maps and sets
- Chapter 6: Finite trees
- Chapter 7: Modules (Sections 7.1-7.4)
- Chapter 9: Efficiency
- Chapter 11: Sequences (excluding 11.8 Type providers and databases)

This part basically constitutes the syllabus — only minor changes may occur.

It is strongly recommended that you carefully study this material. You are expected to master the course topics at a technical depth that is similar to what you see in the reading material.¹

¹Experience from previous years shows that too many students underestimate the effort that is necessary.

3 Programming language and installation

We will use F# as programming language: <http://fsharp.org>. F# is an open-source, cross-platform, functional-first programming language.

You are required to install an F# system on your own laptop to be used for the exercises.

Please consult <http://fsharp.org> concerning installation of F#.

There are two popular installations:

- One that works for Mac, Linux and Windows systems is:
 1. Install first the .NET Core SDK, then
 2. install Visual Studio Code and select the Extensions icon and search for (and install) "Ionide".

This gives you the opportunity to use Visual Studio Code as a program editor; while supporting execution of programs using F# Interactive in a separate window. You can also use the .NET command-line interface (CLI) for developing, building, running, and publishing .NET applications.

- One that works for Windows and Mac systems is:
 - Install Visual Studio.

This also gives you features similar to those mentioned above.

You may have a look at [1], that gives rationale and design choices behind the development of F# as an open-source, cross-platform, functional-first programming language.

4 The course form

The course consists of lectures, exercise classes and homework. You may find a tentative lecture plan here: <http://courses.compute.dtu.dk/02157/plan.html>.

While the textbook material for a lecture is on the plan well before the lecture starts, other kind of information and material may be ready just in time.

There will be no mandatory assignments this year. This will give you more freedom; but also more responsibility concerning how to conduct your study of this course.

Please notice that the learning goals and requirements to pass the exam remain the same as in previous years despite the fact that there are no mandatory assignments. It is, therefore, strongly recommended that you organize your workload evenly over the semester according to DTU's expectations and recommendations.

5 Organization of lectures and exercises

The lectures and exercises are organized to give you an even workload and a steady progression of the concepts throughout the semester *starting from the first day*.

The lectures are intended to support your own reading. But be aware: The textbook contains much more information than covered in the lectures and the slides. That is, to get an in-depth knowledge of the concepts it is strongly recommended to carefully study the textbook material.

We use a so-called flipped classroom model where the teaching slot is used as follows:

- Exercise class from 8:00 - 10:00. Teaching assistants will be available from 8:15.
- Lecture from 10:15 to 12:00.

The exercises for a given week relate to the topics of previous week's lecture. This gives you the opportunity to study and digest the topics before you solve problems.

The exercises for a given class typically starts with simple ones checking basic knowledge of the topics, followed by exercises of increasing difficulty. It is strongly encouraged that you solve problems during the week, so that you can exploit the time slot with the TAs discussing problems you find difficult.

Notice that there will be no on-line support and no hot-line support.

You are always welcome to come to my office: Room 112 in Building 322.

6 Mini projects

In this year's version of the course, some exercises will be continued over a few weeks in order to cover different *themes*. By bundling in this way it is possible, within a single small project, to introduce different computer science topics and at the same time develop your functional programming competences. The aims are:

- To exercise functional programming concepts.
- To tell coherent stories where several topics work together.
- To link the story to topics from “neighbouring courses”.

Furthermore, these small projects are chosen so that they relate to topics from prerequisite courses and give appetizers to later courses.

The small projects will also reveal and exploit the strong mathematical basis of functional programming languages.

References

- [1] Don Syme, *The Early History of F#*, Proc. ACM Program. Lang. 4, HOPL, Article 75 (June 2020), 58 pages. <https://doi.org/10.1145/3386325>