**HOGESCHOOL ROTTERDAM / CMI**

# Functional programming fundamentals

**INFSEN02-8/INFSEN23-8**
2018-2019

| | |
|---|---|
| Number of study points: | 4 ects |
| Course owners: | Francesco Di Giacomo, Giulia Costantini, Mohamed Abbadi |

# Module description

| | |
|---|---|
| **Module name:** | Functional programming fundamentals |
| **Module code:** | INFSEN02-8/INFSEN23-8 |
| **Study points and hours of effort:** | This module gives 4 ects, in correspondence with 112 hours:<br>• 2 X 8 hours of combined lecture and practical<br>• the rest is self-study |
| **Examination:** | Written Exam and Practical assessment |
| **Course structure:** | Lectures, self-study, and practical exercises |
| **Prerequisite knowledge:** | all INFDEV courses. |
| **Learning materials:** | • Book: Remo H. Jansen - Learning TypeScript 2.x - Second Edition<br>• Book: Benjamin Pierce - Types and Programming Languages<br>• exercises and assignments, to be done at home and during the practical part of the lectures (pdf): found on N@tschool |
| **Connected to competences:** | realiseren en ontwerpen |
| **Learning objectives:** | At the end of the course, the student:<br>• **understands** the fundamental semantic difference between functional and imperative programming. `(FP VS IMP)`<br>• **understands** reduction strategies such as $\rightarrow_\beta$. `(RED)`<br>• **understands** the basics of a functional type system. `(TYP)`<br>• **can program** with the typical constructs of a modern functional language. The language of focus is Typescript. `(FP EXT)` |
| **Course owners:** | Francesco Di Giacomo, Giulia Costantini, Mohamed Abbadi |
| **Date:** | September 17, 2018 |

# 1 General description

Functional programming and functional programming languages are increasing in popularity for multiple reasons and in multiple ways, to the point that even mainstream languages such as Python, C++, C#, and Java are being extended with more and more functional programming features such as tuples, lambda's, higher order functions, and even monads such as LINQ and async/await. Whole architectures such as the popular map/reduce are strongly inspired by functional programming.
"Java™ developers should learn functional paradigms now, even if they have no immediate plans to move to a functional language such as Scala or Clojure. Over time, all mainstream languages will become more functional" [IBM].
"LISP is worth learning for a different reason — the profound enlightenment experience you will have when you finally get it. That experience will make you a better programmer for the rest of your days, even if you never actually use LISP itself a lot." – Eric S. Raymond
"SQL, Lisp, and Haskell are the only programming languages that I've seen where one spends more time thinking than typing." – Philip Greenspun
"I do not know if learning Haskell will get you a job. I know it will make you a better software developer." – Larry O' Brien
The reason for this growth is to be found in the safe and deep expressive power of functional languages, which are capable of recombining simpler elements into powerful, complex other elements with less space for mistakes and more control in the hands of the programmer. This comes at a fundamental cost: functional languages are structurally different from imperative and object oriented languages, and thus a new mindset is required of the programmer that wishes to enter this new world. Moreover, functional languages often require more thought and planning, and are thus experienced, especially by beginners, as somewhat less flexible and supporting of experimentation.

## 1.1 Relationship with other didactic units and required knowledge

This module completes and perfects the understanding and knowledge of programming that was set up in the Development courses of the first year. The preliminary knowledge necessary to fully understand this course covers the following topics:

- Semantics of programming languages and its evaluation.
- The memory model of *Stack* and *Heap*.
- *Type systems* and type checking.
- Dynamic vs Static typing.

# 2 Course program

The course is structured into eight lectures. The eight lectures take place during the eight weeks of the course, but are not necessarily in a one-to-one correspondance with the course weeks.

## Unit 1

### Topics

- Stateful vs stateless computation
- Lambda calculus semantics.
  - Variables
  - Lambda-abstractions/functions
  - Function application
- Introduction to Typescript
- example of lambda calculus in Typescript
- Bindings and their semantics in lambda-calculus.
- Functional `if-then-else` and differences with its imperative counterpart.

## Unit 2

### Topics

- Typed lambda calculus. Typing variables, lambda abstractions, function applications.
- Recursive functions.
- Discriminate unions and records in Typescript.
- Pattern matching
- Lists with `Cons` and `Empty`
- Recursive functions on lists in Typescript.

## Unit 3

### Topics

- Higher-order functions.
- Pipe operator, function composition, map, fold, map2, fold2.
- Curry and Uncurry
- Case Study: SQL

## Unit 4

### Topics

- Immutable trees.
- Immutable Binary Search Tree. Find, Add, Remove.
- Case Study: Expression evaluation

## Unit 5

### Topics

- `map` and `fold` on Binary Search Trees.
- `map` and `fold` on `Option`.
- Functors with records of functions.

# 3 Assessment

The course is tested with two exams: a written exam and a practical exam. The final grade is determined as follows:

```
0.3 * writtenGrade + 0.7 * practicalGrade
```

**Note:** both parts must be sufficient (i.e. `>= 5.5`) to pass the exam.

**Motivation for grade** A professional software developer is required to be able to program code which is, at the very least, *correct*.
In order to produce correct code, we expect students to show: *i*) a foundation of knowledge about how the semantics of the programming language actually work; *ii*) fluency when actually writing the code.
The quality of the programmer is ultimately determined by his actual code-writing skills, therefore the written exam will contain require you to write code. This ensures that each student is able to show that his work is his own and that he has adequate understanding of its mechanisms.

## 3.1 Theoretical examination INFSEN02-8/INFSEN23-8

The general shape of an exam for `INFSEN02-8/INFSEN23-8` is made up of a short series of highly structured open questions. In each exam the content of the questions will change, but the structure of the questions will remain the same. Questions might include (but not limited to): apply the semantics of lambda calculus on a small function, determine the type of a functional program, determine the result of the execution of a functional program. A sample exam will be provided during the course.

## 3.2 Practical examination INFSEN02-8/INFSEN23-8

The practical exam requires to complete code snippets provided in the exam text. The code snippets contain simple functions covering the topics seen in class. A sample exam will be provided during the course.

# Appendix 1: Assessment matrix

| Learning objective | Dublin descriptors |
|---|---|
| `FP VS IMP` | 1, 4, 5 |
| `RED` | 1, 2, 4, 5 |
| `TYP` | 1, 4, 5 |
| **FP EXT** | 1, 2 |

Dublin-descriptors:

1. Knowledge and understanding
2. Applying knowledge and understanding
3. Making judgments
4. Communication
5. Learning skills