SEPTEMBER 2025

PROGRAMMING PARADIGMS

CS-117

Introduction to the course

Hans Hüttel, Department of Computer Science, Aalborg University

TODAY

- About the content of the course.
- About the way this course is organized.
- About what we should expect of each other.

The classes that follow (the “real” classes”) are very different from this one.

SHORT VERSION

- 12 sessions on functional programming in Haskell.
- The course is **mandatory**. It is challenging and important.
- DAT7 and SW7 are one class; the teaching there is in Danish.
- CS-IT7 is a different class. They are taught in English.
- The exam is common to everyone with the same learning goals and the same requirements.
- **You must be able to write, test and reason about the code you create in order to pass.**
- **It is very important to be an active participant in the teaching activities in class and to prepare for each session.**

A FEW WORDS ABOUT US

THE TEACHING ASSISTANT IS...

- MA Salah (CS-IT9)

WHO AM I?

- I am an associate professor. I have been employed at AAU since 1991.
- I am from Denmark originally.
- I have taught *Programming Paradigms* since 2010.
- This is a subject that I care deeply about. It is important, fascinating and highly useful.

WHAT ARE PROGRAMMING PARADIGMS?

A NOTION DUE TO ROBERT FLOYD

- **Programming paradigms are different ways or styles in which a program can be organized.**
- **Robert Floyd, professor of computer science at Stanford University, received the 1978 ACM Turing Award.**
- **In his paper *Paradigms of Programming* from that year, he wrote:**

  If the advancement of the general art of programming requires the continuing invention and elaboration of paradigms, advancement of the art of the individual programmer requires that he expand his repertory of paradigms.

SOME IMPORTANT PARADIGMS

Paradigm	Central notions
Imperative	**Statements**: ***Assignments***, loops, procedures, references/pointers
Object-oriented	**Objects** (and often also classes): Object definitions, modifying state variables, method calls – and imperative notions, too)
Functional	**Functions and expressions:** Function definitions, function calls, recursion
Logic	**Logical predicates:** Predicates, Horn clauses, proof search, resolution

PROGRAMMING AND PROBLEM SOLVING

- If you have a masters degree in computer science, you will get a job that requires strong analytic skills. Your employer will assume that you are able to solve problems by programming.
- **This is a course about programming and problem solving. Our emphasis is on functional programming.**

LANGUAGES AND PARADIGMS

- Different problems can be solved in a simple and more convincing way within different paradigms.
- Writing parsers and interpreters in imperative/object-oriented languages is a royal mess. Functional languages are better for this.
- Some paradigms make it easier to reason about program correctness than others! Correctness for imperative/object-oriented programs is a challenge!

LANGUAGES AND PARADIGMS

- One could write programs in the functional programming paradigm in C or C++, **but that is poorly supported by such languages**.
- Instead, one can write functional programs in Haskell, OCaml or F# – and it is well-supported by these languages!
- **It is important to know how to choose and use the right paradigm for the problem that we want to solve and a language that supports the paradigm well.**
- **This is a competence that is very important for software developers and in computing in general.**
- If you study for a masters degree you must become able to learn new competencies in the rest of your career. In computing one must be able to learn new and very different programming languages. The landscape keeps evolving!

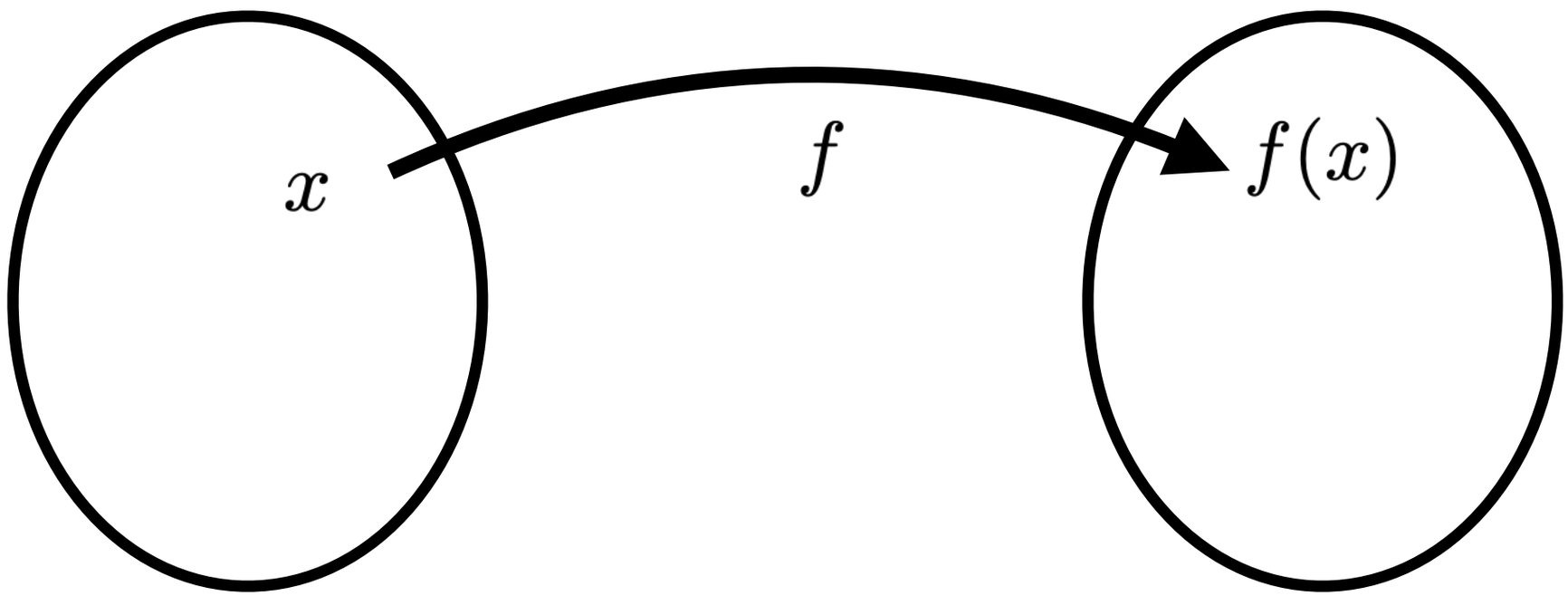
FUNCTIONAL PROGRAMMING IN HASKELL

SHORT VERSION

- The functional programming paradigm is based on the λ-calculus, a model of computability. It emerged at the same time as Turing's work.
- In a purely functional programming language (Haskell is one such language), side effects are not possible.
- The most important concepts are
  - Functions
  - Recursion
  - Lazy evaluation (especially for Haskell)
  - Algebraic data types
  - Polymorphism
  - Monads

FUNCTIONAL PROGRAMMING

- In the functional programming paradigm, we think of an algorithm as a function in the usual sense, the one we know from mathematics.



Every argument x in the domain maps to exactly one value $f(x)$ in the range

FUNCTIONAL PROGRAMMING

- **Functions are first-class citizens:** A function can take a function as its argument and can return a function as its result
- Functions are defined by **expressions**
- A program is a collection of function definitions
- **Function composition** and **recursion** are the main control structures


WELCOME TO
HASKELL
COUNTY

WHAT IS THIS, AND WHAT IS ITS TYPE?

```
plip [] = 0
plip (x:l) = 1 + (plip l)
```

TYPED FUNCTIONAL PROGRAMMING

Num is a **type class**; p can be any type found in it

```
plip :: (Num p) => [a] -> p
plip [] = 0
plip (x:l) = 1 + (plip l)
```

a can be *any* type whatsoever

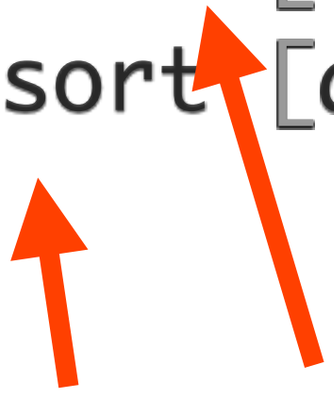
A recursive call of `plip`

Haskell has **parametric** as well as **ad hoc polymorphism** and allows us to mix them; we can easily describe functions that work on a wide variety of data in a type-safe way.

WHAT IS GOING ON HERE?

```
qsort :: (Ord a) => [a] -> [a]

qsort [] = []
qsort (x:xs) = small ++ [x] ++ big
  where small = qsort [a | a <- xs, a <= x]
        big   = qsort [a | a <- xs, a > x]
```



Recursive calls of `qsort`

TYPED FUNCTIONAL PROGRAMMING

```haskell
qsort :: (Ord a) => [a] -> [a]

qsort [] = []
qsort (x:xs) = small ++ [x] ++ big
    where small = qsort [a | a <- xs, a <= x]
          big   = qsort [a | a <- xs, a > x]
```

We declare the type of the `qsort` function (but we do not have to)

This is `quicksort` in Haskell.

Haskell has **list comprehension**. We can often describe computations in a *declarative* fashion so that we can focus more on *what* we want to compute and not so much on *how* we want to compute it.

WHAT IS GOING ON HERE?

```
term = do
                b <- base
                symbol "*"
                return (Star b)
                <|>
    do b <- base
       return b
```

ADVANCED CONTROL STRUCTURES

```
term = do
          b <- base
          symbol "*"
          return (Star b)
          <|>
          do b <- base
             return b
```

Haskell has **monads**. They allow us to structure stateful computations in a nice way. A good example of that is **parsing**.

TYPES ARE OUR FRIENDS

- Many functional programming languages (Haskell, SML, OCaml, F#, Clean…) have strong static type systems that provide strong guarantees.
- The type systems are much more expressive than the ones found in C, C++ etc. and come with strong guarantees.
- A type system is called **sound** if it guarantees that ***Well-typed programs cannot go wrong:*** If a program is well-typed, it will not exhibit certain run-time errors when executed.
- Types in Haskell can be used for specifying program properties.
- Some languages such as Coq/Gallina, Agda and Idris have even stronger type systems that are strong enough to express mathematical theories and are used for proof checking and proof development.

WHAT IS THIS?

```
data BTree = BLeaf Int | BBranch Int BTree BTree

-- sumtree :: BTree -> Int

sumtree (BLeaf x) = x
sumtree (BBranch x t1 t2) = let v1 = sumtree t1
                                v2 = sumtree t2
                            in x + v1 + v2
```

Recursive calls of `sumtree`

TYPED FUNCTIONAL PROGRAMMING

```
data BTree = BLeaf Int | BBranch Int BTree BTree

-- sumtree :: BTree -> Int

sumtree (BLeaf x) = x
sumtree (BBranch x t1 t2) = let v1 = sumtree t1
                                v2 = sumtree t2
                            in x + v1 + v2
```

The type of the `sumtree` function appears in a comment only

Haskell has **algebraic datatypes and pattern matching**. This makes it easy to describe data structures and abstract syntax trees.

Haskell has **full type inference**; we do not have to specify the types of entities unless we want to.

LAZY EVALUATION

A recursive call of `from`

```
from k = k : (from (k+1))

naturals = from 1
```

```
two = let x : y : rest = naturals
      in y
```

LAZY EVALUATION

```
from k = k : (from (k+1))

naturals = from 1

two = let x : y : rest = naturals
      in y
```

This is an infinite list!

We use pattern matching to extract the second element

Haskell has **lazy evaluation**. This is a notion of parameter passing that only evaluates the occurrences of an actual parameter that we actually need. When we use it together with recursion we can express and perform computations on infinite data structures.


HERE'S A HANDY HINT FROM MISTER NATCH:
AT HOME OR AT WORK....
GET THE RIGHT TOOL FOR THE JOB!
©R. Crumb

WHO USES FUNCTIONAL PROGRAMMING?

- Haskell is widely used within the infrastructure at many projects across **Facebook**
- **Chordify** is implemented in Haskell
- In-house tools for automotive development in **Tesla** are written in Haskell
- **Epic Games** (Fortnite etc.) are now introducing a new functional language called Verse for game scripting
- **Jane Street**, a major fintech company, uses OCaml for all its software
- **Amazon** uses Erlang to implement SimpleDB for database services as a part of the Elastic Compute Cloud (EC2).
- **Humio** (in Aarhus) uses Elm and Scala for developing web applications
- The **Grammarly** tool uses CommonLisp
- **NASA** uses functional programming (see https://uniavisen.dk/en/making-software-that-was-tested-on-nasa-planes/ where a Danish student talks about his NASA experience with this)

(etc. etc.)

WHY DID I CHOOSE HASKELL FOR THIS COURSE?

(It is not the only possible choice but we have to make one.)

- Haskell is a **well-designed** programming language whose features can be explained in a succinct manner.
- Haskell shows how important problems that are a pain to solve using imperative programming can be quite easily handled **without lots of boilerplate code**.
- Haskell is **pure** (no imperative features, unlike OCaml, F# and other languages of the ML family) so it puts full emphasis on the functional programming paradigm.
- Haskell shows how far you can get with a programming language that has a **powerful type system**.

THE LEARNING GOALS OF THE COURSE

PROGRAMMING AND PROBLEM SOLVING

- Programming =
  understanding a problem +
  devising a solution +
  writing code to solve the problem +
  testing code +
  debugging and optimizing code
- **Programming involves all of these activities.**

The goal is that you become fluent in functional programming and its underlying principles and can use them well.

- To be able to understand and apply the concepts of functional programming to solve programming problems
- **To be able to write, test and debug non-trivial programs in Haskell yourself.**
- To be able to "speak Haskell" with the correct terminology.
- These are **skills** you must master **yourself**.

THE VOCABULARY OF THE COURSE

- Static vs. dynamic types.
- **Types and type classes**. Structured datatypes: Lists, pairs, user-defined datatypes
- Parametric and ad hoc-**polymorphism**
- **Functions**: The concepts of functions and closures, including lambda expressions.
- **Pattern matching**
- The relation to the lambda calculus and to type systems for the lambda calculus
- **Lazy evaluation** and infinite data structures
- Representing stateful computations in a pure functional language
- **Monads** and functors
- **Continuations**

MOODLE

MOODLE

- Moodle is the learning management system we use at AAU.
- All information about courses and projects and your education is made available on Moodle.
- You sign in to Moodle using your AAU user name (email address) and two-factor authentication.
- You **must** check the Moodle page of CS-IT7 regularly for information about the semester.
- You **must** check the Moodle page of each course that you follow for information.
- Visit http://moodle.aau.dk now. Find the course page. Find the entry for next week's session. Is there information about the exam?

INSTALLING AND RUNNING THE HASKELL SYSTEM

WHAT YOU WILL NEED

- You must have a laptop computer for class and for your exams at AAU. **A smartphone or a stationary computer will not be enough.**
- You must have your own installation of the Haskell interpreter `ghci` and become able to use it.
- You must use `ghci` in class and at the exam in this course.

HOW TO GET HOLD OF HASKELL

- Haskell is available for Windows, for macOS and for widely used Linux and FreeBSD, Open BSD and NetBSD distros. **You will also need a laptop computer for the exam.**
- Haskell works with VisualStudio and the Stack platform supports the development of larger pieces of software in Haskell. While you are making your first steps in Haskell, **please do yourself a favour and use a simple setup**.
- We will use the Haskell interpreter `ghci`.
- I use `ghci` together with the text editor Emacs.
- First install the Haskell system. Next install Emacs. Finally install the Haskell mode for Emacs. See Moodle for more details.

go back

WHERE DO WE FIND HASKELL?

- https://www.haskell.org/downloads/ has everything you need
- For Linux og BSD you can use package managers such as `apt-get` og `nix`
- For macOS you can use `homebrew`
- Windows has a version of `ghcup`.

No matter which platform you use:

1. First install `ghcup`
2. Then use `ghcup` to install `ghc` and `cabal`
3. You *may* also want to install `stack`. `stack` is used for managing software projects in Haskell. But we will not need it in this course.

10-MINUTE BREAK FOR INSTALLING EVERYTHING

HTTP://WWW.HASKELL.ORG

THE EXAM

WHAT IS THE EXAM SUPPOSED TO CHECK?

- The goal of every exam is to check that the student has met the learning goals.
- At the exam of *Programming Paradigms*, you must be able to demonstrate **that you can solve programming problems on your own using functional programming and using the programming environment.** The goal is that you become able to solve the problems in the problem set on your own and become able to **test** that your solution makes sense and is correct by using GHCi.

WHAT IS THE EXAM SUPPOSED TO CHECK?

- You must be able to **run and test and debug** your code during the exam using your own installation of the Haskell interpreter.
- **Any piece of code that contains syntax errors will receive 0 points.**
- Solving a problem means that you produce a piece of Haskell code that can actually be executed using GHCi (you must check this using GHCi during the exam) and works as intended.

WHAT HAPPENS AT THE EXAM?

- A 3-hour written exam. The exam takes place in a large hall.
- The exam consists of a collection of programming problems that you are asked to solve using your computer and the Haskell interpreter GHCi.
- The problem set contains programming problems **that you have never seen before**. The problems are of the same nature as the ones you will meet in class.
- Your answers must be in the form of a simple text file with .hs suffix **that must be able to compile**.

WHAT HAPPENS AT THE EXAM?

- At the exam each one of you must have access to
  - A laptop computer with a working installation of GHCi. You are supposed to use throughout the exam
  - The course textbook (other relevant written course material from Moodle will be provided in hardcopy)
  - An official compendium
  - These are the only aids allowed at the exam; Google, ChatGPT, “own notes” etc. are not allowed.
- Your answers must be in the form of a simple text file with .hs suffix **that must be able to compile**.

WHAT HAPPENS AT THE EXAM?

- The exam submissions are assessed by
  - the teacher of the course (me)
  - two external examiners with substantial expertise in functional programming:
    - Torben Ægidius Mogensen (associate professor, University of Copenhagen)
    - Jesper Bengtson (associate professor, IT University of Copenhagen)
- The education system in Denmark makes use of external examiners in order to ensure that assessment is fair.
  - The result of the exam is a grade on the Danish grading scale.

THE DATE OF THE EXAM

- The date of the exam is not known yet.
- But it will probably be the *first* exam in January 2026.
- Written exams for very large classes are always scheduled as the first ones. This is because
  - this is an exam that is common to many degree programmes and involves a large group of students
  - assessment requires more time when external examiners are involved, since not everyone may be able to mark exams papers on the same days.

YOU HAVE THREE ATTEMPTS

- If you should fail this exam, you have only two more attempts left. The next chance comes in February 2026.
- The third and last attempt will be in January 2027.
- After that you will have to apply to the board of studies for an exemption to get a 4th (final) attempt. In extraordinary cases, a 5th attempt can be granted.

TEACHING AND ASSESSMENT MUST BE PROPERLY ALIGNED

- The problems in the problem set of the exam will be reminiscent of programming problems that we meet in class.
- Therefore: The only sensible way of preparing for the exam is to take part in the activities of the course.
  - **"Taking part" does NOT mean "watching others solve programming problems".**
  - **"Taking part" means solving programming problems yourself.**

HOW ARE THE TEACHING ACTIVITIES STRUCTURED?

ACTIVE LEARNING!

- In this course the activities are there to help you meet the learning goals. I have designed these activities based on my experience with the subject and with teaching in general.
- Before we meet in class you are supposed to prepare for class.
- The presentation of new material is in the form of YouTube videos by the author of the textbook.
- In class, the main activities are
  - solving and sharing solutions and thoughts about programming problems
  - discussions.

THE FLIPPED CYCLE OF THE COURSE

Read description of session and watch video

Read text

Activities in class

A SESSION IN PROGRAMMING PARADIGMS

- **Any time before we meet:**
  1. Read the introduction to the session and try to understand the learning goals. Always do this first.
  2. Watch the video content and read the text.
  3. Try to solve the two small preparation problems.
  4. Write down the most important question that you have about the topic of this session. **(do this in groups)**
- **You can watch the video and read the text in any order that you want**
- It is a good idea to set aside a certain day (or more than one day) and a certain time of day for preparing for a session. You may want to prepare together with fellow students.

WHAT HAPPENS IN CLASS?

1. Presentation of learning goals (by me)
2. Discussion of preparation problems (students)
3. List of questions from you (students)
4. A question for discussion (everyone)
5. A programming problem to be solved (everyone)
6. Discussion of a typical stumbling block (by me)
7. More programming problems to be solved (everyone)
8. Discussion: What was easy? What was challenging? (everyone)

WHAT ELSE HAPPENS IN CLASS?

- There are also problems that you can solve on your own if you have more time
- Along the way there may be *"microlectures"* about problems that we discover that we should talk about
- I will set up a folder for each session that you can use for uploading content
- The problem set will be available at the beginning of each session

IN OTHER WORDS...

- **You must prepare before class.** We use the class activities as a way to finalise our understanding.
- Classes are **not** lectures but active learning sessions.
- My goal is to help you learn this material. This is meant to be a transparent structure.
- It is important that you talk to each other and talk to me and the teaching assistant. We are (also) here to answer all your questions.

STARTING OVER

The general strategy when programming is always the same: Get an idea, make a specification, code, fix bugs, test, revise code.

In many other ways, you have to start from scratch to learn functional programming. **It is important to never underestimate the fact that you have to start over.**

Haskell is not “a weird version of Java” or “a weird version of Python”. Haskell is based on a completely different programming paradigm that you must learn to master.

The programming environment is an important friend. **If you become familiar with the type system and know how to test your code you can check that your solution to a problem is correct. You must learn to do this.**

STARTING OVER

Try comparing it to the differences between

- learning to drive a car and
- learning to drive a motorcycle

A motorcycle is **not** a car with two wheels missing, a strange steering wheel and no reverse gear.

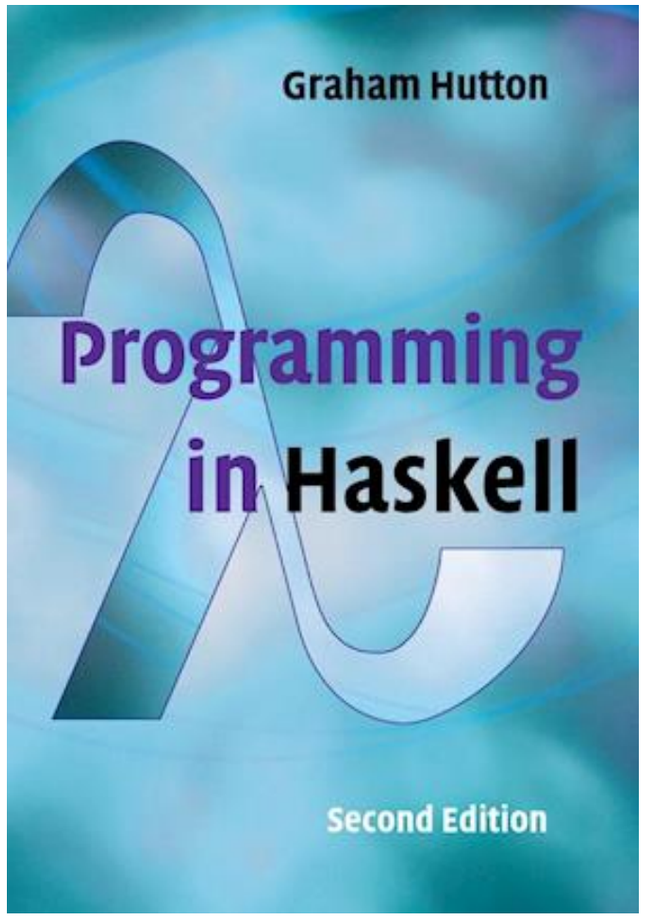
A car is **not** a motorcycle with no saddle, strange handlebars and two wheels too many.

A driving licence for one vehicle is **not** automatically a driving licence for the other one!

THE COURSE MATERIAL

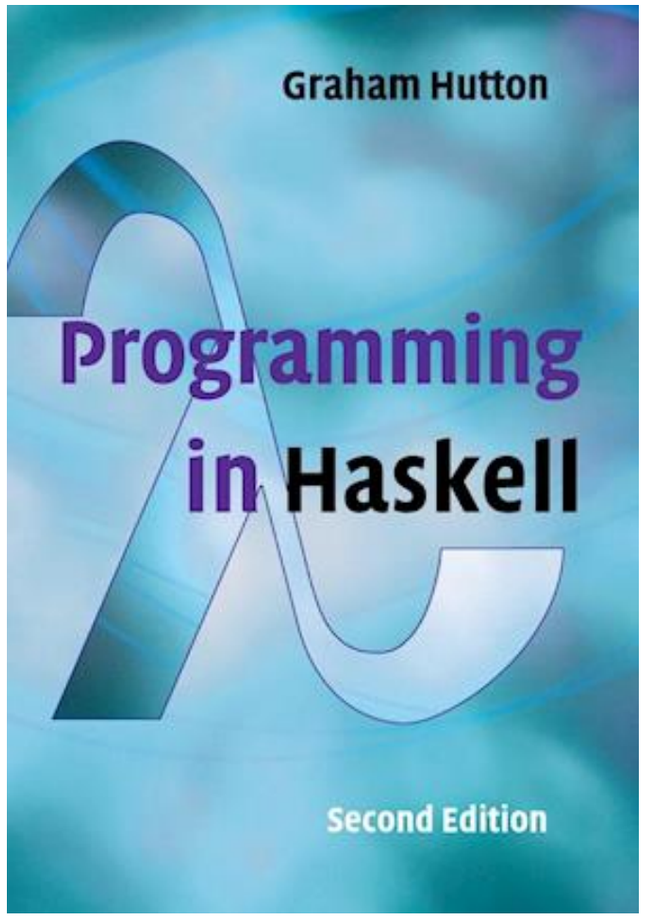
THE MAIN TEXT OF THE COURSE

- We are going to use a textbook for the entire course. We will cover almost all of the book.
- The book is well-written and concise.
- **The course videos we will use were also made by Graham Hutton!**
- I will often refer to the book so you need to become familiar with it.
- You can buy the book at FACTUM (and elsewhere).

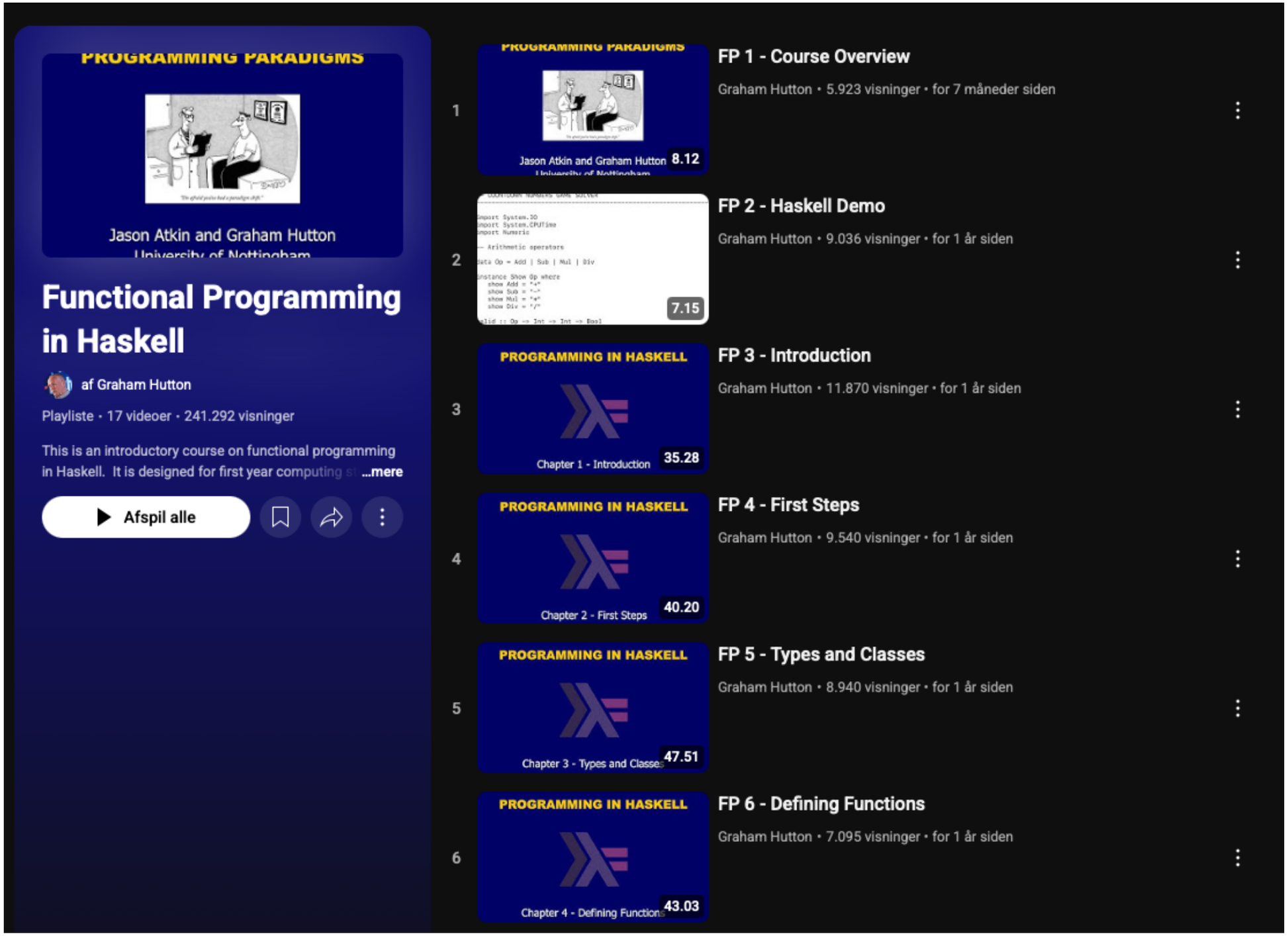


THE MAIN TEXT OF THE COURSE

- The first two chapters are available online if you order the book over the WWW and you experience problems with delivery.
- You will need the book for the exam.
- We will cover about 15 pages (one chapter) per week.
- University graduates are supposed to read and know academic books!

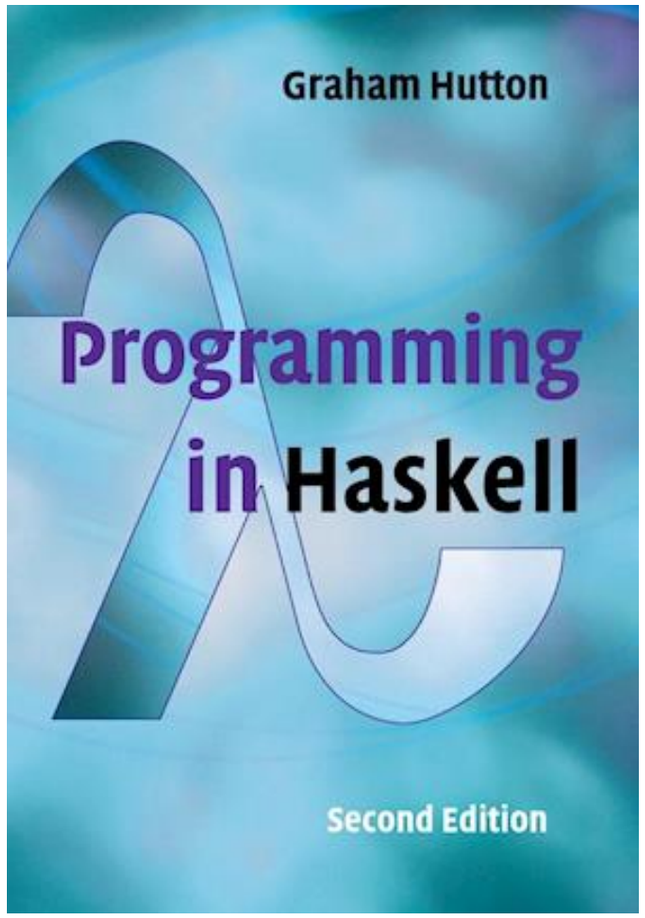


THE VIDEO CONTENT (AND BEYOND)



THE VIDEO CONTENT (AND BEYOND)

- The parts of the book that we will cover in the course are covered in videos made by Hutton himself.
- The videos are freely available on YouTube *(though not in February)*. So everything is there already for you to watch.
- There is other content available from the course webpage.

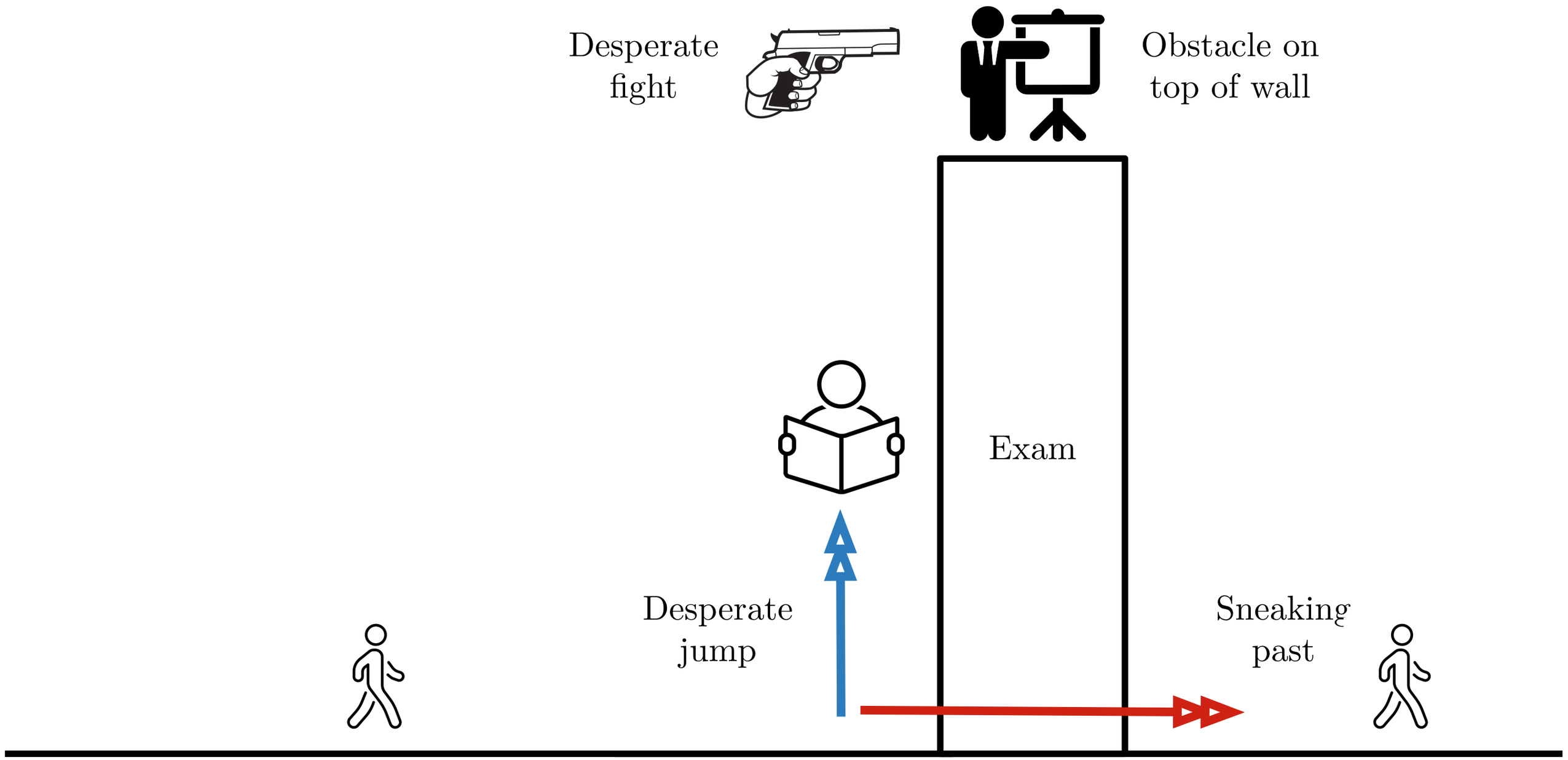


HOW TO APPROACH TEACHING AND LEARNING

A MATTER OF MUTUAL TRUST

- It is important that you can trust me as the teacher of this course.
  - Everything I say about the learning goals and what we expect from students is actually the case.
  - Everything I say about the teaching activities is based on many years of experience.
  - Everything I say about the exam is actually the case.
- It is important that I can trust you as students.

A LOW LEVEL OF TRUST



A HIGH LEVEL OF MUTUAL TRUST

New insights

Guide on
the side

Exam

Activites in
class

IS IT SAFE TO ASK QUESTIONS IN CLASS?

OF COURSE IT IS.

ASKING QUESTIONS IS WHAT STUDENTS SHOULD ALWAYS DO.

WHAT SOME STUDENTS APPEAR TO BELIEVE

- Some students appear to believe that it is wrong or even dangerous to ask questions.
- They seem to believe that the teacher will then seek revenge and punish them for having asked a question.
- This is a sign of low levels of trust.

PLEASE REMEMBER...

We are here **to help students learn**. Classes are not instruments for silencing or punishing students. The classes in *Programming Paradigms* are **active learning sessions** where it is important that we can have a dialogue based on trust.

The teaching staff at universities in Denmark is not allowed to intimidate or threaten students. If I ever did that, that would be a serious breach of the Code of Conduct at AAU. I would lose my job.

I do not necessarily know the names of most students. Even if I wanted to punish you for asking a question in class, it would be almost impossible to find out who you are.

PLEASE REMEMBER...

When a student fails, it simply means more work and more worries for the teaching staff. There is no joy in failing anyone.

When a student fails, the university loses potential income.

The assessment happens at the exam and only then. **We use two external examiners from other Danish universities.** The grade of each student must be agreed upon by the external examiners and myself and is based on the exam submission and nothing else.

If you have reason to believe that the assessment was not fair, **contact the study secretary or the semester coordinator.**

Even if I *could* get away with failing students for asking questions in class, what good would it do me?

MY THOUGHTS

- **My job is (among other things) to help students learn.**
- It is highly unfortunate when students appear to be struggling but do not ask questions.
- I cannot help students if they do not ask for help.

THE WORST AND THE BEST...

- **The worst thing** that can happen to those that ask a question is that they may not get an answer immediately.
- **The best thing** that can happen to those that ask a question is that they learn something new and that their fellow students also learn something.

ASKING QUESTIONS IS GOOD FOR EVERYONE

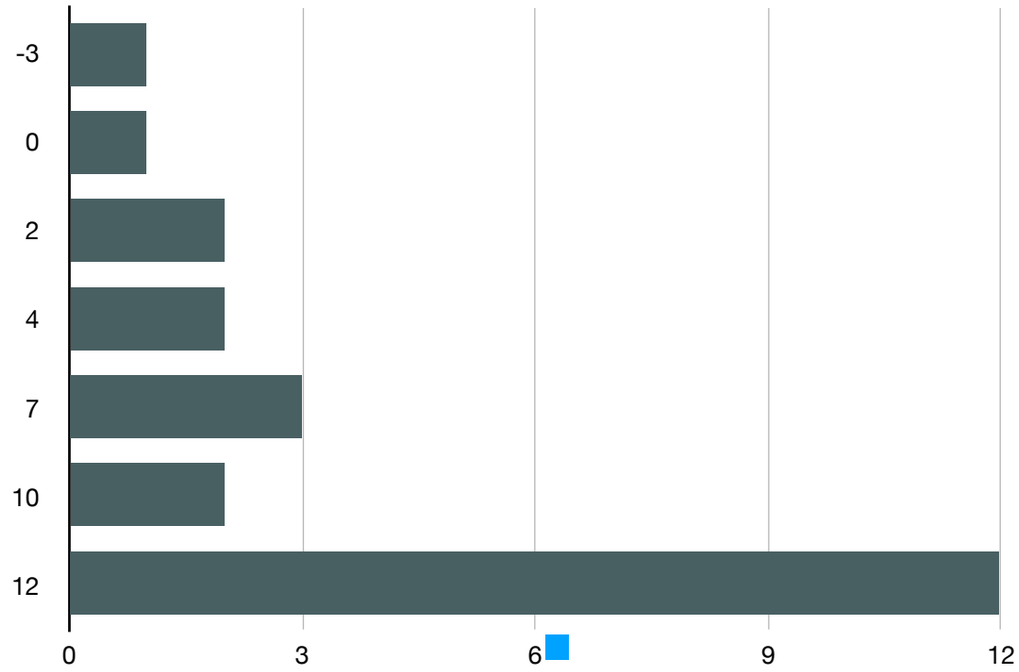
- When you ask questions in class, we know that you are interested in learning.
- Asking questions can prevent you from failing.
- Asking questions can also help other students learn!

A PARTICULAR CONCERN I HAVE

THE JANUARY 2025 EXAM

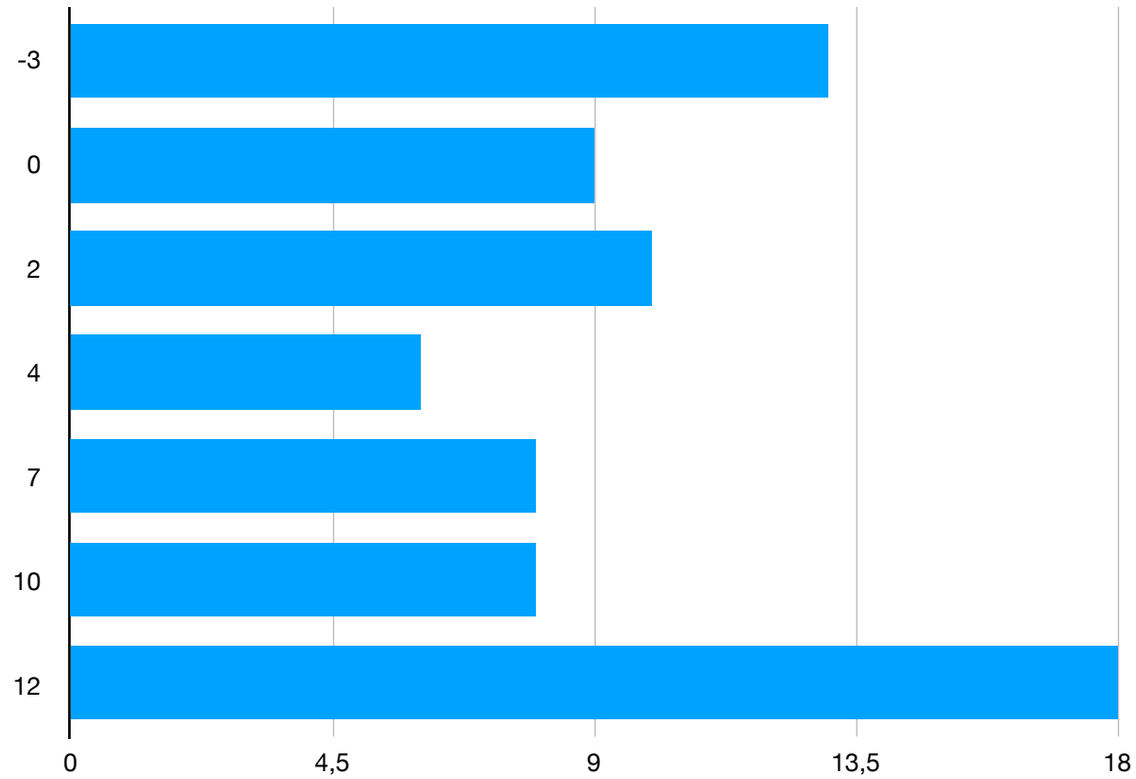
Grade distribution - DAT7

-3	1
0	1
2	2
4	2
7	3
10	2
12	12



Grade distribution - SW7

-3	13
0	9
2	10
4	6
7	8
10	8
12	18

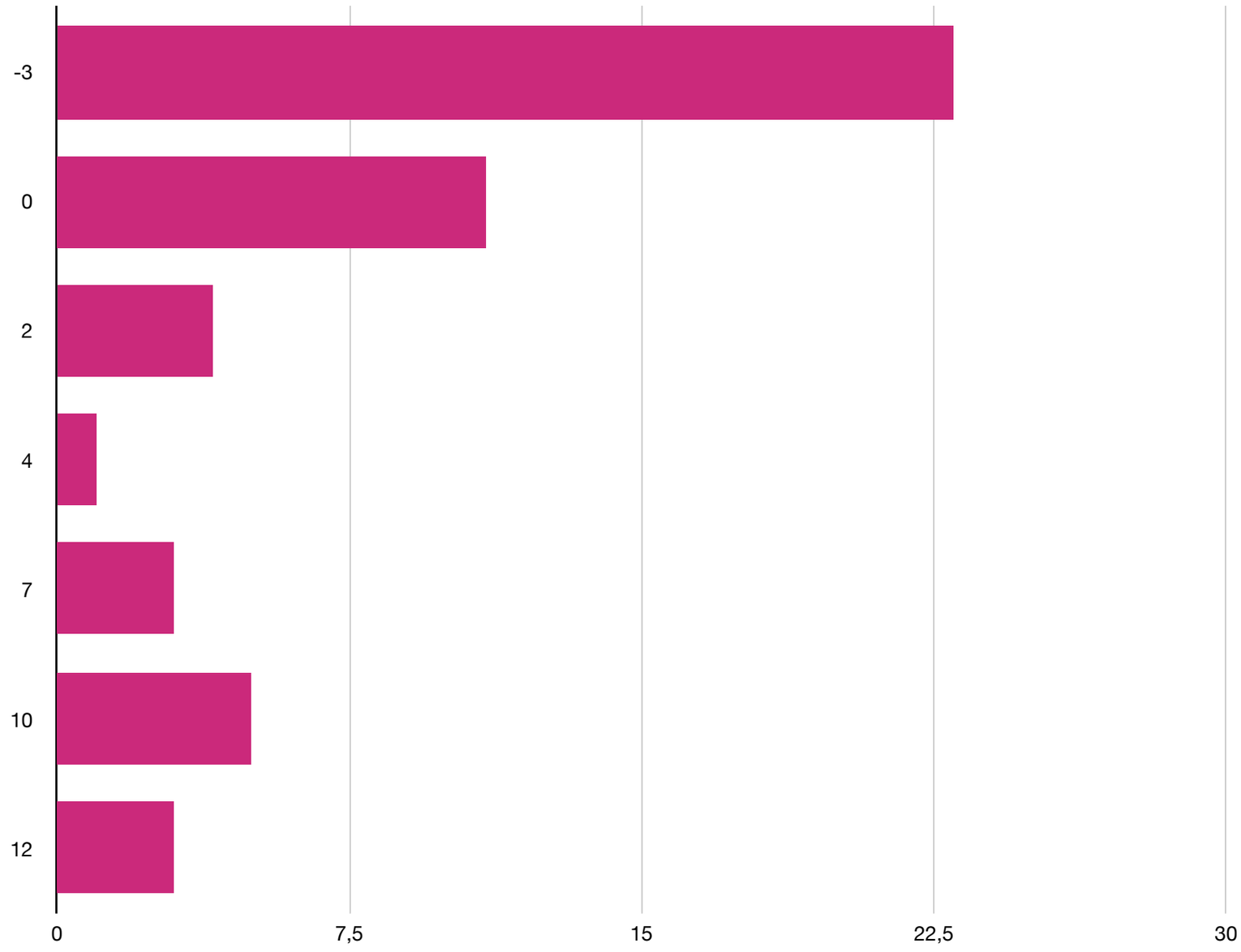


Overall pass rate: 75 %

THE JANUARY 2025 EXAM

Grade distribution - CS-IT7

-3	**23**
0	11
2	4
4	1
7	3
10	5
12	3



34 out of 50 CS-IT7 students failed. -3 is the lowest grade in Denmark. 2 is the lowest passing grade.

PASSIVE STUDENTS

In 2024 there was a large segment of CS-IT7 students who sat in the back of the room and did **nothing at all**. They showed up for class every week but that was all they did.

All of them were self-paying students.

We saw no programming activity among the passive segment and never had any questions from them.

The passive segment simply sat and watched while the other students took part in the class activities. Some had computers with them, some did not.

The teaching assistants said to me that **“It is as if these students do not care.”**

PASSIVE STUDENTS

I sent email to groups about preparation problems, but the groups with passive students never replied.

When I tried to talk to the passive students, they would say nothing.

I tried to have a meeting with everyone after class in order to reach out to them and to tell them that I was worried. The meeting was announced on Moodle and in class. Some of the passive students left the room when the meeting began.

HOW DID IT END?

It did not end well. All of the passive students received the grade -3 at the exam.

To get the grade -3 required fewer than 25 points out of 100. Many passive students got fewer than 10 points.

The passive students said that they were surprised by what was expected from them at the exam and by what the exam was like. **However, all information about the January exam was available on Moodle in early December.**

The passive students seem to have ignored all information about the course, including the cover page of the problem set for the exam.

PRETENDING TO PROGRAM

The exam submissions showed that the passive students never used or installed GHCi and were copying text that they did not understand and that was completely irrelevant. Many could not even write comments in code.

Competent Haskell programmers (such as the examiners) can spot a “pretend-programmer” immediately. The idea that one can “fool the examiner” is a worrying assumption.

A VERY LOW LEVEL OF TRUST

Why did the passive students believe that they could

- ignore the teaching activities
- ignore the information about the course
- ignore attempts by the teaching staff to communicate
- never learn to program or even use GHCi

and still be able to pass? **Why did they even attend class?**

I still have no idea.

But these kinds of behaviour are signs of not caring about the course and of a very low level of trust.

We could not make the passive students see that their attitude would prevent them from passing. It was **extremely frustrating**.

THIS MUST NOT HAPPEN AGAIN

- The passive students from CS-IT7 also failed many other exams. We have discussed this in meetings at departmental level – the initiative was mine.
- The Director of Studies and I agree that we must make sure that this will not happen again.
- This is the reason why CS-IT7 now has its own class in this course and that the department now spends time on onboarding activities. I have been given extra teaching hours for this separate class, because I am going to devote more time to CS-IT7.
- My goals are
  - to help you learn and to create **trust** among us
  - to make sure that everyone will engage with the course and use it to become **active** and **sincere** learners


After hustling his way to Carnegie Hall, the penguin suddenly felt the pressure
WUMO

PLEASE REMEMBER...

- In order to learn this, you have to devote time it throughout the semester.
- If you could learn everything in a day or two, the administration would place the exams at the start of the semester and save a lot of resources.
- And I would not put so much effort into teaching but lean back and concentrate on my research.

...AND PLEASE ALSO REMEMBER...

Later on, in your working life, your future colleagues will have to step in and compensate for all the knowledge, skills and competences that you do not have.

Your future colleagues will not appreciate someone who pretends to program and tries to rely on rote memorisation.

MY LAST WORDS

WHAT YOU SHOULD EXPECT FROM ME AS A TEACHER

- I want you to learn functional programming and learn it well, and I approach my teaching in that spirit. Functional programming is an interesting and important topic.
- I will be well-prepared for every session.
- I will provide general guidelines for installing the relevant software (here: Haskell)
- I will provide feedback to selected problems from the sessions in the form of short videos posted to Moodle
- I will always keep you informed via Moodle
- **I will answer all your questions as best as I can. If there are questions that I cannot answer here and now, I will find an answer.**

WHAT YOU CAN EXPECT FROM ME AS A TEACHER

- I will be open about what may be difficult in this course and try to refrain from saying that something is “really easy”.
- You are very welcome to talk to me in my office; if I am not there, send me a mail message such that we can meet later.
- If you find mistakes and omissions on the Moodle page, I will correct them within a reasonable amount of time when you alert me to them.
- If you are looking for additional resources about the content of the course, I will try to find suitable resources.
- **I will pay attention to your comments and take them seriously. I will evaluate the course with you as it proceeds.**

WHAT I EXPECT FROM US ALL (YOU AND I)

- When we are together, we are well-prepared and contribute to the course
- We strive to make this a good experience for everyone involved
- We are honest about it, when something seems difficult
- We listen to each other and talk to each other
- We remember that learning takes time and effort and has to do with gaining as much experience as possible, not some kind of magical talent that one was born with *(I have met seen a baby that was familiar with functional programming)*

WHAT I EXPECT FROM YOU

- **Please be here at 12:30. Class begins at 12:30, not 12:33 or 12:45.**
- **That you try to make the best out of this course and we offer.** Remember that the teaching activities are meant to help you learn. We can succeed if we all try to do our best.
- **That you are sincere, take this course seriously and use it to develop good habits that you can make use of your working life. Staying away does not help improve anything and it will not help you learn.**
- **That when you are in class, you spend your time on activities related to the course.**
- **That you remember that this is a challenging topic and requires an effort on your part throughout the semester.**

WE ARE HERE FOR YOU!

- The teaching assistants and I are here to help you meet the learning goals of this course.
- I have organised activities that are meant to ensure this.
- **We want to help you learn this.**

PLEASE REMEMBER!

- You may come from a place where students are expected to behave in a very different manner. In some places, a good student is a passive student and learning = reproducing word for word.
- **You are studying for a masters degree at a university in Europe now. In this part of the world, learning goals are real and important and students are supposed to show initiative.**

PLEASE REMEMBER!

- At this university, the teaching staff and local students expect you to
  - Be active and sincere and take part in the teaching activities.
  - Communicate with the teaching staff and your fellow students.
  - Make sure that you remain updated about information about the course. That includes reading this information.
  - **Have faith in us: What we tell you is actually true. We are completely sincere about this course and what it is. Your principle should be to act in the same way towards everyone else.**

OFFICE HOURS

- Come talk to me!
- My office is **room 2.2.57** at SLV300 (first floor).
- Or mail me at hans@cs.aau.dk .

BEFORE WE MEET AGAIN NEXT TUESDAY

- **Make sure that you have installed the Haskell platform and a suitable environment** (I strongly recommend a simple setting). You must be able to use it – you will need it throughout the course and for the exam.
- Load and experiment with a simple Haskell program (found on Moodle under the entry for session 1).
- **Read the description of session 2.**
- **Watch the video and read the text for session 2.**
- **Buy the textbook** (chapters 1 and 2 are available online).
- **Think of the question that is most important to you right now**, write it down and share it with us in class.
- **Attempt to solve the small preparation problems** for session 1.
- Some groups should prepare a short presentation of their solutions to the discussion problems. Any takers?