Introduction to Functional Programming

Course introduction and first steps in programming

Some slides are based on Graham Hutton's public slides

Today

- Introduction to the course
- What is a computer?
- Programming languages
- Tools
- Writing code

General

- Functions
- Testing

Specific

Goal of the course

GÖTEBORGS UNIVERSITET
CHALMERS

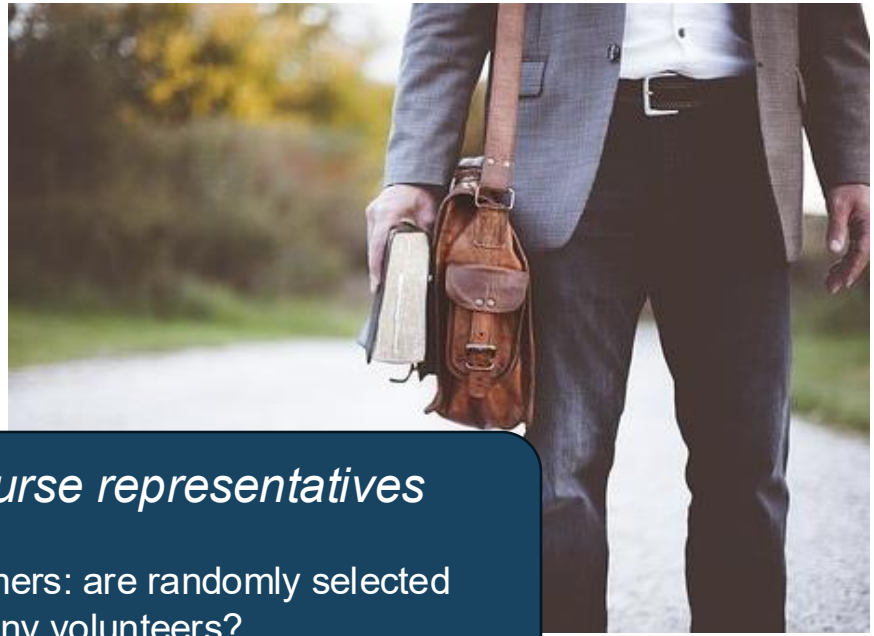
- Start from the basics
- Introduce basic concepts of computer science
- Learn to write small-to-medium sized programs in Haskell

GÖTEBORGS UNIVERSITET

CHALMERS

Teaching team

- Daniël Apol,
- Naïm Favier,
- Albin Jonzon,
- Nour Aldeen Dabora,
- Eli Uhlin,
- Alexander Appelin,
- Daniel Cole,
- Sebastian Sandstig,
- Linda Jonsson,
- David Bohman,
- Jonathan Otten,
- Nithit Sathornkit,
- Erik Dahlkvist.



Course representatives

- Chalmers: are randomly selected
- GU: any volunteers?

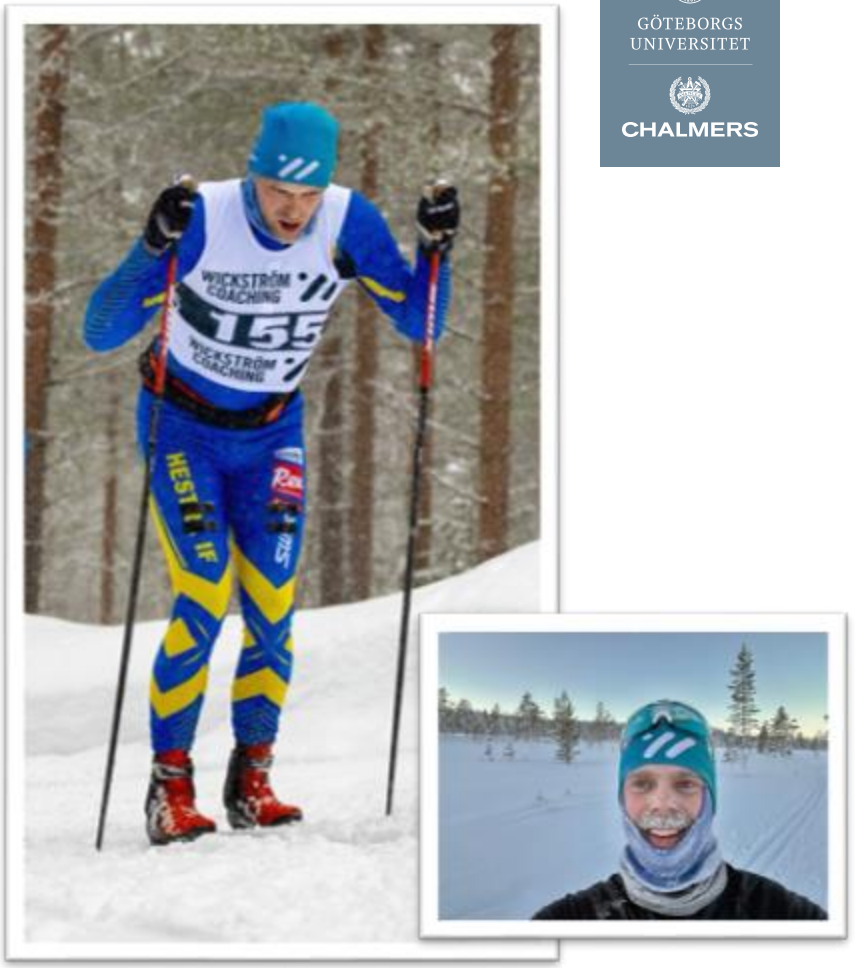
GÖTEBORGS UNIVERSITET

CHALMERS

Alex Gerdes

- Dutchman living in Sweden
- MSc (civilingenjör) OU/UU
- PhD in Computer Science
- Senior Lecturer (universitetslektor)
- Programansvarig DV
- Enhetschef FP

- Teaching: FP, data structures
- Research:
  - FP in education,
  - Property-based testing



COURSE STRUCTURE

GÖTEBORGS UNIVERSITET

CHALMERS

Canvas

- The Canvas course room will have all up-to-date information relevant for the course:
  - Schedule (links to TimeEdit)
  - Lectures: slides, live code, videos, reading suggestions
  - Lab assignments
  - Exercises
  - Last-minute changes (also via Discord)
  - And more...

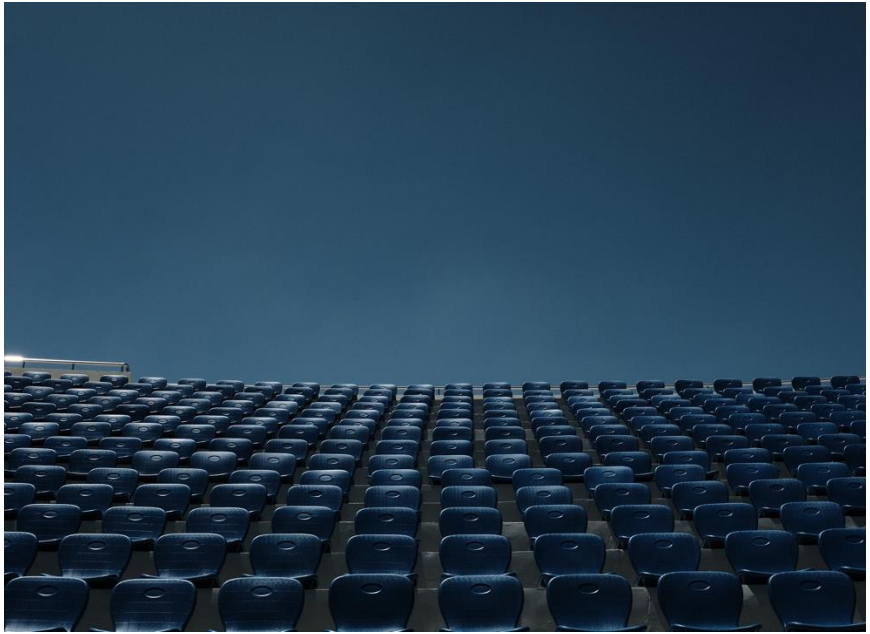
GÖTEBORGS UNIVERSITET

CHALMERS

Lectures

Why come to the lectures?

- Will introduce some of the theory
- Lots of live coding
- Will spill over in each other
- You are welcome to:
  - follow live coding with your own computer
  - use smartphone or computer to take part in quizzes
  - ... but this is completely optional!
- Will be recorded… if you continue to turn up
- Come prepared!
  - preparation instructions on Canvas



GÖTEBORGS UNIVERSITET
CHALMERS

Exercise sessions

- Mondays 10:00 – 11:45 (usually)
- Again: come prepared!
- Work on exercises together or individually
- Discuss and get some help from us
  - I will select and introduce some exercises to work on
  - Will give hints half ways during the session
  - You mostly work on your own, not a laboration
- They are split in a self-check and extra exercises
- Make sure you understand the previous week's things before you leave

GÖTEBORGS UNIVERSITET
CHALMERS

Don’t stress about the deadlines!
Focus on other parts as well!!!

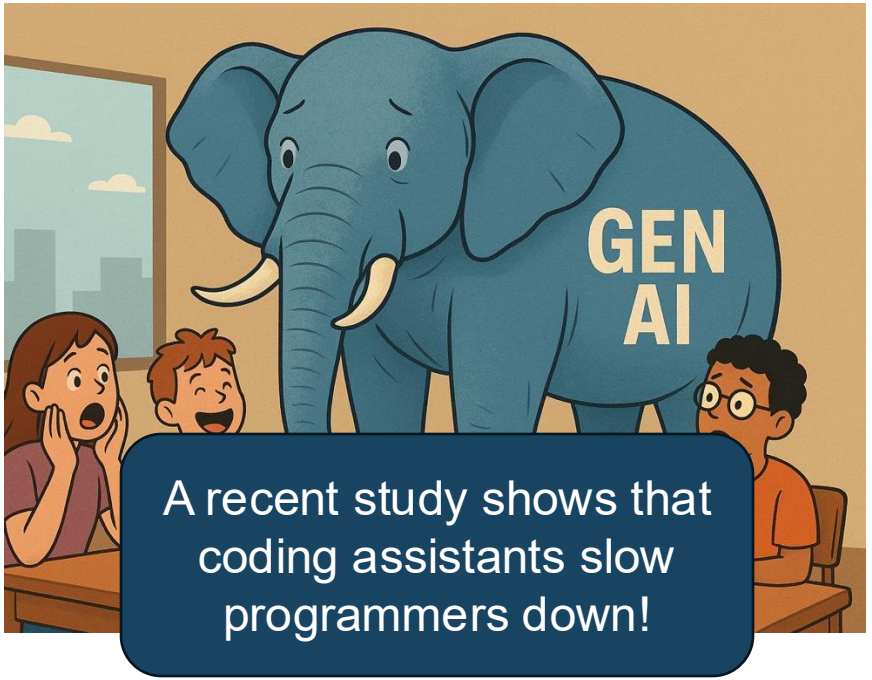
Lab assignments

- Four lab assignments
  - Some labs are divided into multiple parts
- Deadlines always on Fridays 18:00 (and are strict!)
- Submit via Canvas, read the instructions!
- Lab 2 and 3: also *present* your solution to a supervisor
- Create own groups of three
  - You could reuse the group you may have formed during the introduction weeks
  - Will freeze the groups after a while

GÖTEBORGS UNIVERSITET
CHALMERS

The elephant in the room!

- Generative AI can certainly be helpful when coding
- Tempting to use a coding assistant during the labs, but don't!!!
- You need to know how to program yourselves, before you can take full advantage
- GenAI is well trained on basic stuff, but not as much on challenging tasks
- We need to have a focus on *specification* and *validation*, nothing new on the horizon!
- We are going to have a lecture on this, and a part of a lab

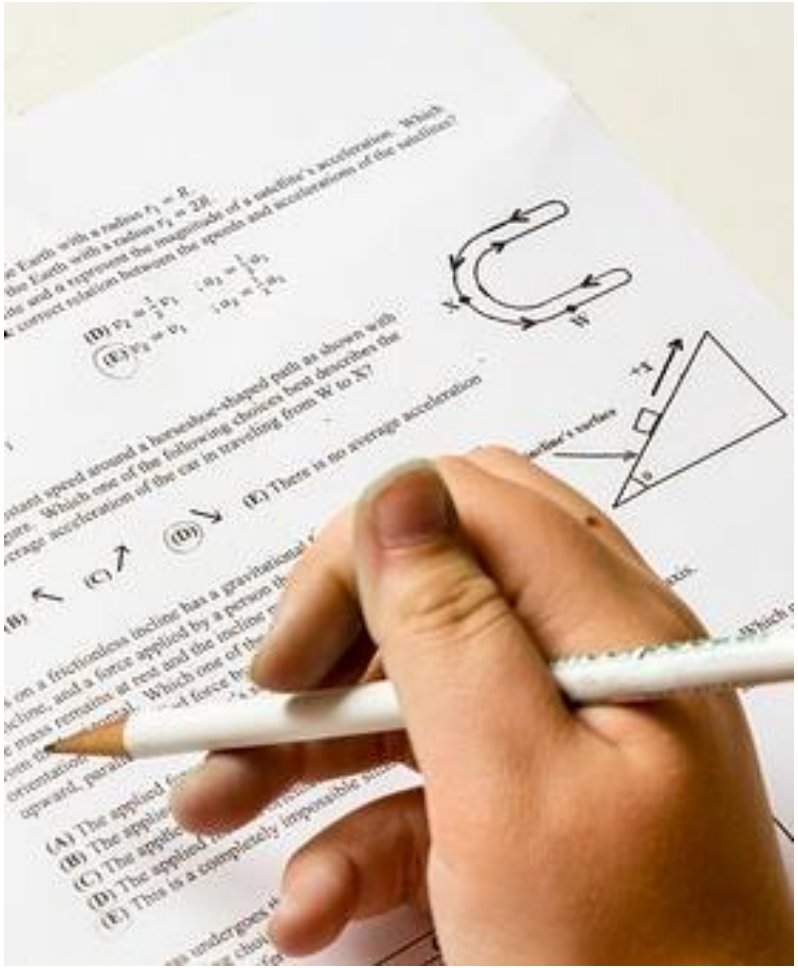


GÖTEBORGS UNIVERSITET
CHALMERS

Getting help

- Weekly exercise sessions
  - Personal help to understand material
- Lab supervision
  - Specific questions about lab assignment at hand
  - Done on campus, come prepared
- Discord
  - General questions, worries, discussions
  - Finding lab partners

First exercise session on Tuesday will be dedicated to installing the development environment! (and getting started with the labs/exercises)



GÖTEBORGS UNIVERSITET

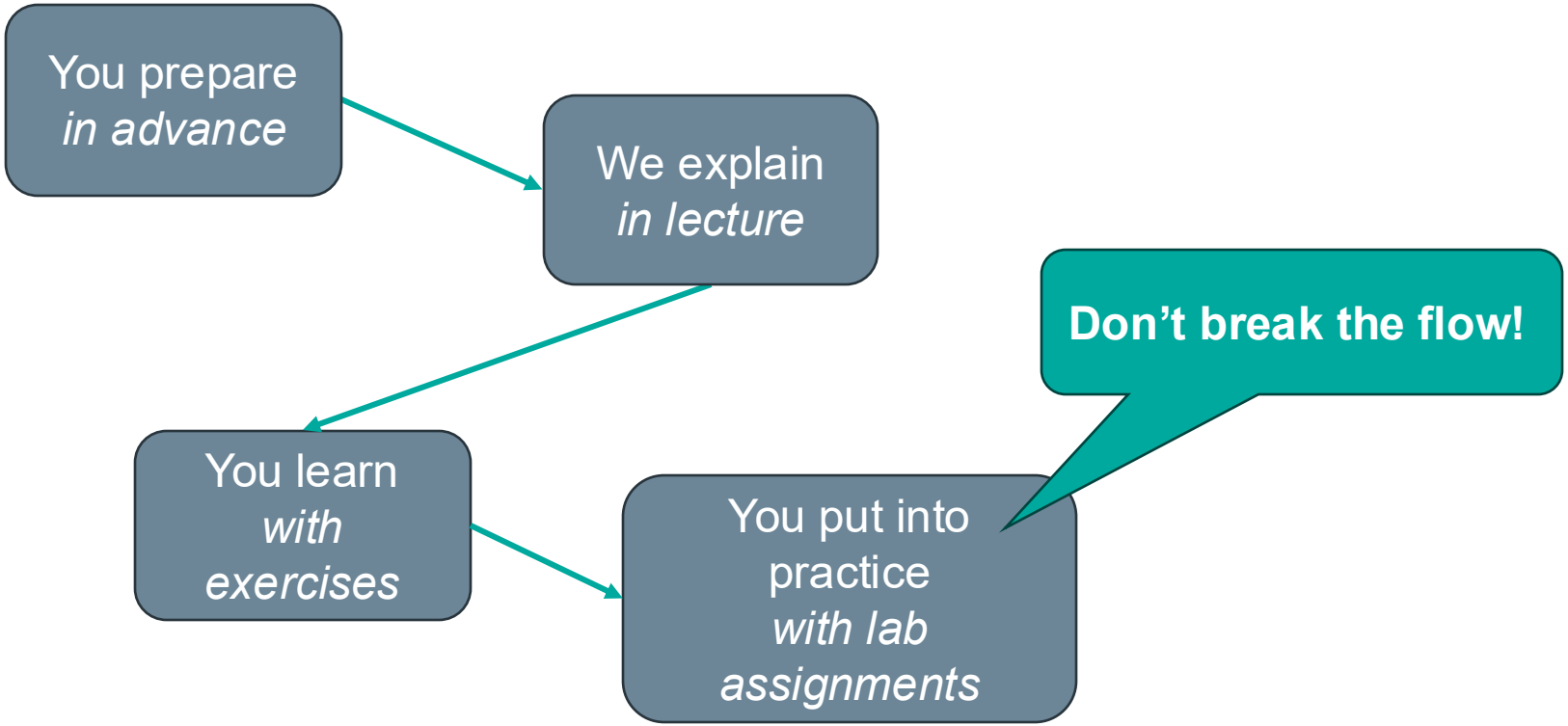
CHALMERS

Assessment

New for this year!

- Written (digital) exam (4.5 hp)
  - Consists mostly of small programming problems to solve
  - You need Haskell “in your fingers”
  - More details in Canvas
  - U-3-4-5 grading scale
- Course work (3 hp)
  - Complete all labs successfully
  - Pass/Fail grading scale

The Flow


You prepare
in advance
We explain
in lecture
You learn
with exercises
You put into practice
with lab assignments
Don't break the flow!

GÖTEBORGS UNIVERSITET

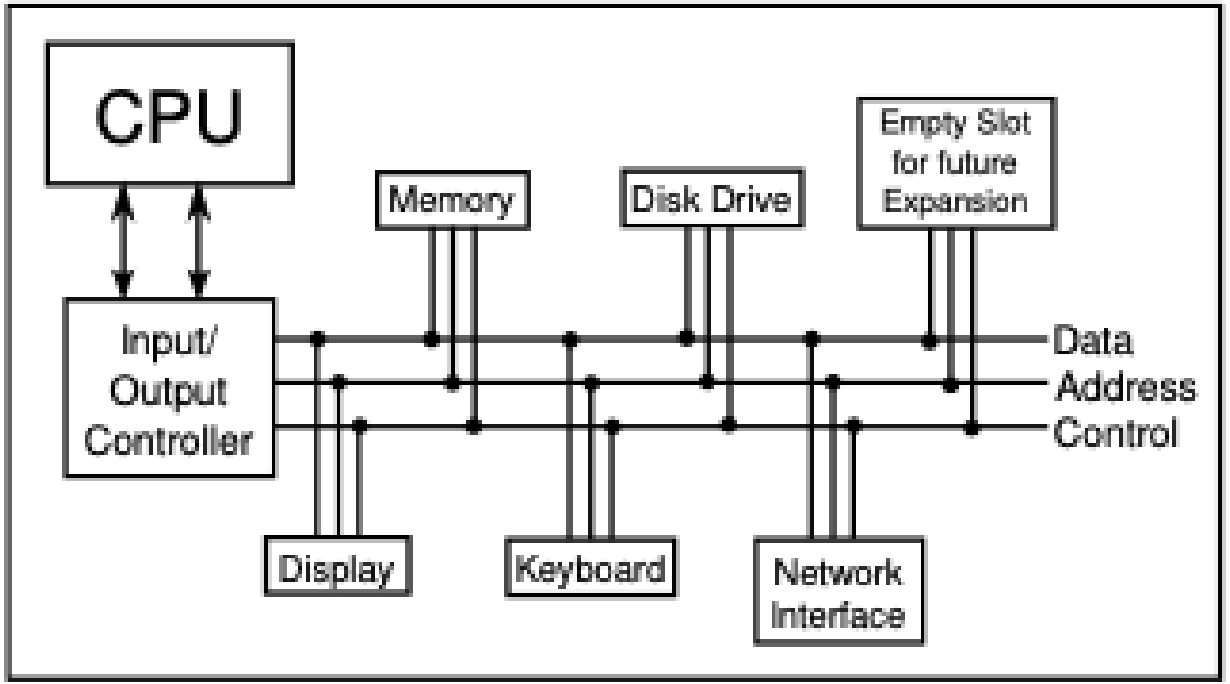
CHALMERS

A risk

- 8 weeks is a short time to learn programming
- The course is fast paced
  - Each week we learn a lot
  - Catching up again is hard
- So do keep up!
  - Read the material for each week
  - Make sure you can solve the problems
  - Reflect and plan! (remember ‘framtidskoden’ course)
  - Go to the weekly exercise sessions
  - From the start!

INTRODUCTION TO COMPUTING

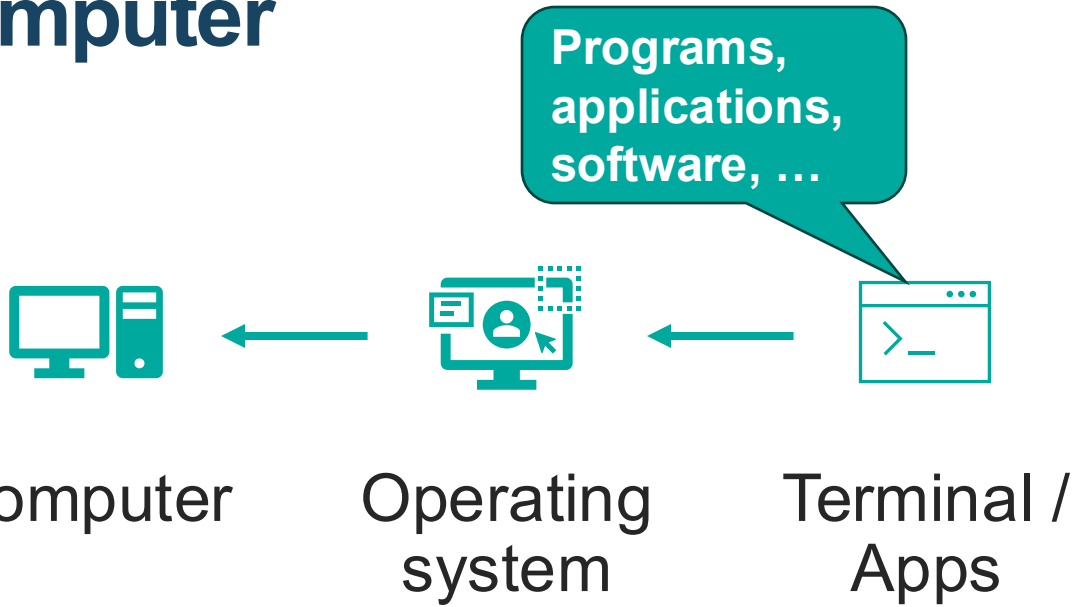
What is a computer?



Source: Introduction to Programming Using Java, D. Eck

Controlling a computer

- Computer consists of hardware resources:
  - Processor
  - Memory
  - Many other parts
- The Operating System controls the hardware
  - Allows applications to run
  - Shares the resources between apps
- Apps/terminal instructs computer what to do
  - With help from the OS



GÖTEBORGS UNIVERSITET
CHALMERS

Software = Programs + Data

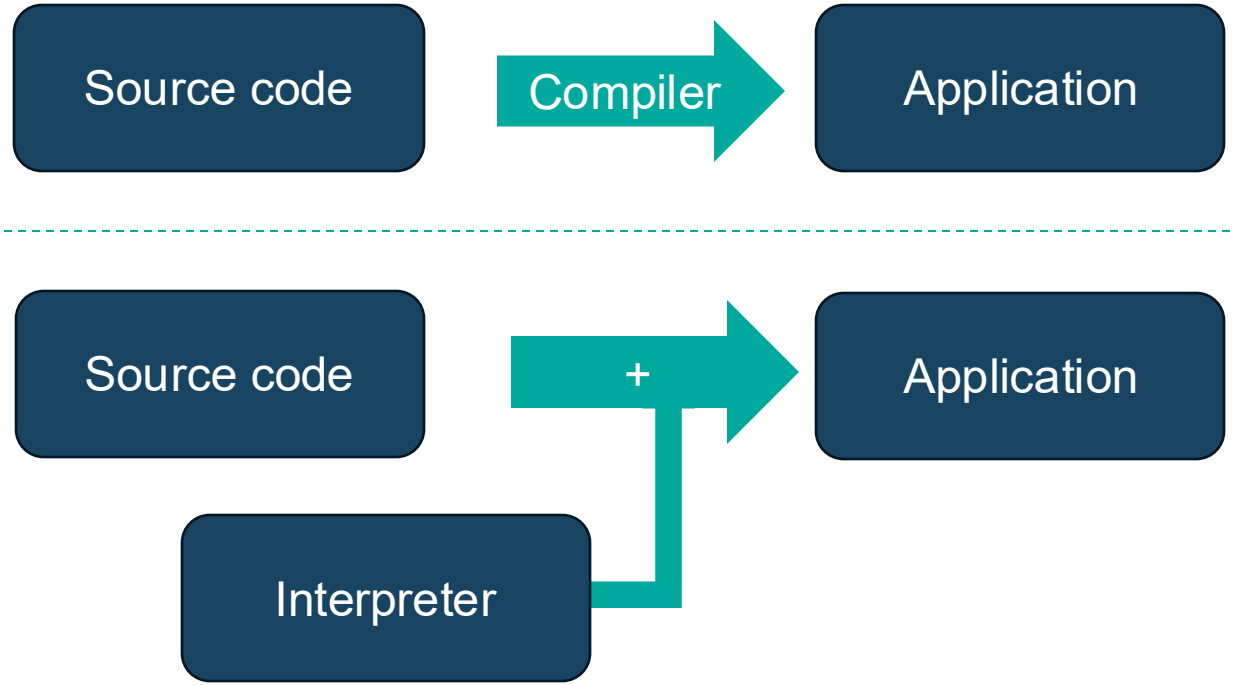
- Data is any kind of storable information, e.g.:
  - numbers, letters, email messages
  - maps, video clips
  - mouse clicks, *programs*
- Programs compute new data from old data:
  - A computer game computes a sequence of screen images from a sequence of mouse clicks
  - vasttrafik.se computes an optimal route given a source and destination bus stop

GÖTEBORGS UNIVERSITET
CHALMERS

Programming languages

- Programs are written in *programming languages*
  - Not natural language, must be unambiguous
  - Syntax and semantics
- There are hundreds of different programming languages, each with their strengths and weaknesses
- A large system will often contain components in many different languages

Compiler/interpreter



GÖTEBORGS UNIVERSITET
CHALMERS

Two major paradigms

Imperative programming:

- Instructions are used to change the computer's state:
  - `x := x+1`
  - `deleteFile(”slides.pdf”)`
- Run the program by following the *instructions* top-down
- Describing *how* to solve

Functional programming:

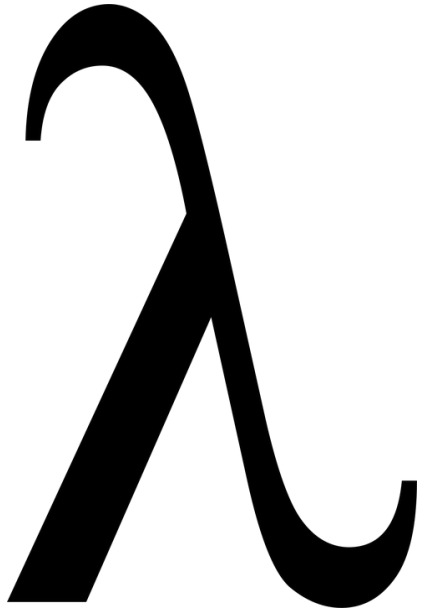
- Functions are used to *declare* dependencies between data values:
  - `y = f(x)`
  - `x = 32`
- Dependencies drive evaluation
- Describing *what* to solve

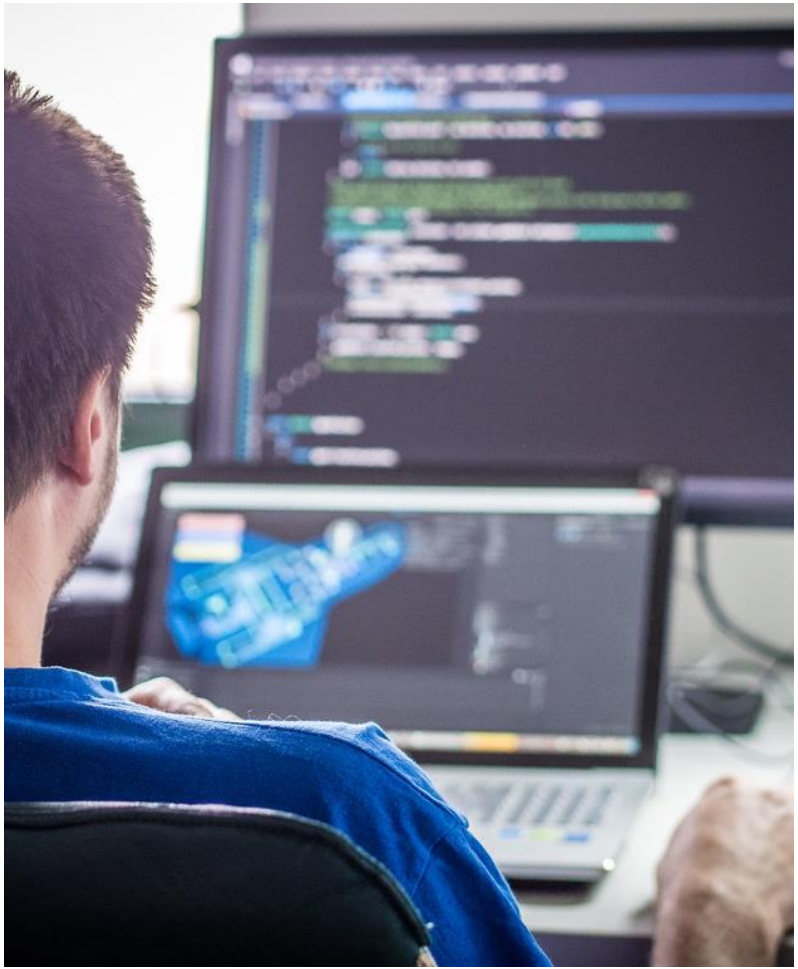
GÖTEBORGS UNIVERSITET

CHALMERS

Functional programming

- *Functions* are:
  - used to declare dependencies between data values: `y = f(x)`
  - the basic building blocks of (functional) programs
  - used to *compose* functions into other functions
  - *only* dependent on the argument (in so-called pure functions)
- *Functional programming* is a *style* of programming in which the basic method of computation is the *application of functions to arguments*
- A *functional programming language* is one that *supports* and *encourages* the functional style





GÖTEBORGS UNIVERSITET

CHALMERS

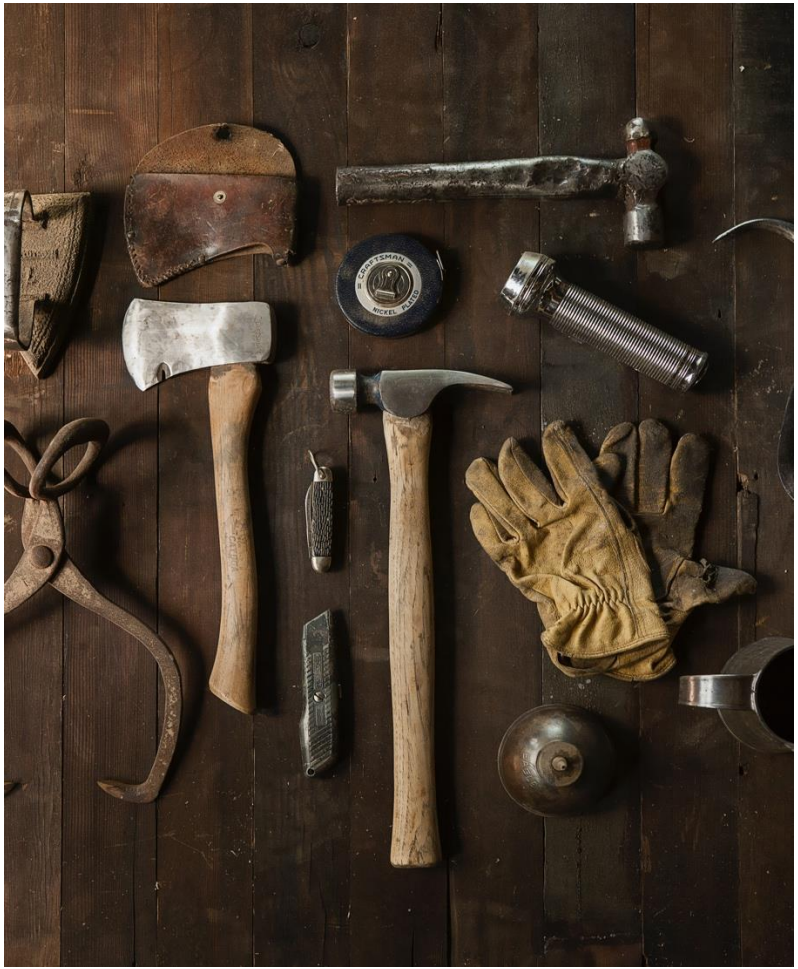
Teaching programming

- We want to give you a broad basis
  - Easy to learn more programming languages
  - Easy to adapt to new programming languages
  - Appreciate differences between languages
  - Become a better programmer!
- This course uses the functional programming language *Haskell* (http://haskell.org/)

Why Haskell?

GÖTEBORGS UNIVERSITET
CHALMERS

- Haskell is a very *high-level language*
  - Lets you focus on the important aspects of programming
- Haskell is expressive and concise
  - Can achieve a lot with a little effort
- Haskell is good at handling complex data and combining components
- Haskell is defining the state of the art in programming language development
- Haskell is *not* a particularly high-performance language
  - Prioritizes programmer-time over computer-time



GÖTEBORGS UNIVERSITET

CHALMERS

We need tools!

- Editors
  - Many out there, we recommend *Visual Studio Code*
  - Integrated Development Environment (IDE)
- Terminal
  - Giving commands to OS: `ls`, `pwd`, `cp`, `cd`, …
  - Starting the interpreter or compile
- Files, directories
  - Organize!
- Docker
  - Virtualisation technique, a kind of mini OS
  - We offer a stable and complete development environment

GÖTEBORGS UNIVERSITET

CHALMERS

MENTI!

GÖTEBORGS UNIVERSITET
CHALMERS

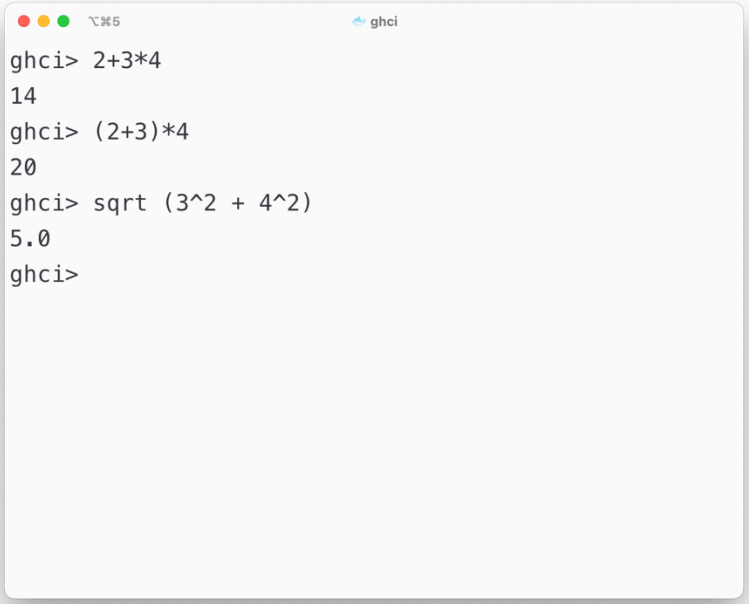
Glasgow Haskell Compiler

- GHC is the leading implementation of Haskell, and comprises a compiler and interpreter
- The interactive nature of the interpreter makes it well suited for teaching and prototyping
- GHC is freely available
- Starting GHCi:
  - The interpreter can be started from a terminal command prompt by simply typing `ghci`
  - The GHCi prompt `ghci>` means that the interpreter is ready to evaluate an expression

GÖTEBORGS UNIVERSITET
CHALMERS

GHCi example

- For example, it can be used as a desktop calculator to evaluate simple numeric expressions



GÖTEBORGS UNIVERSITET
CHALMERS

Function application

- In *mathematics*, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space

```
f(a, b) + c d
```

Apply the function `f` to `a` and `b`, and add the result to the product of `c` and `d`

GÖTEBORGS UNIVERSITET
CHALMERS

Function application

- In *Haskell*, function application is denoted using a space, and multiplication is denoted using `*`

- Moreover, function application is assumed to have a *higher priority* than all other operators:

```
f a + b
```

means `(f a) + b`

rather than `f (a + b)`

```
f a b + c * d
```

As previously, but in Haskell syntax

GÖTEBORGS UNIVERSITET

CHALMERS

Function application, examples

Mathematics:

```
f(x)

f(x,y)

f(g(x))

f(x,g(y))

f(x)g(y)
```

Haskell:

```
f x

f x y

f (g x)

f x (g y)

f x * g y
```

GÖTEBORGS UNIVERSITET
CHALMERS

Variables and arguments

- Functions are *abstractions* of calculations, which we want to perform with varying values
- To capture the varying parts of a calculation we can introduce *variables*, which abstract away from particular values and thus *vary*
- When we apply a function on a value, we *substitute* the variable with the given value
- The given value in a function application is also called an *argument*
  - An argument or the given value can be regarded as the input to a function
- A different name for a variable in a function is a *parameter*
  - A function is parametrized over a variable

```
f x = x * 3 + 1

ghci> f 3
10
```

`f` is applied to an argument in this case the value 3. In the calculation (definition) of `f` we can substitute `x` with the value 3.

LIVE CODING!

I may not have time to cover everything in each lecture.

You are expected to do/learn/read the rest on your own!

GÖTEBORGS UNIVERSITET
CHALMERS

Composing functions

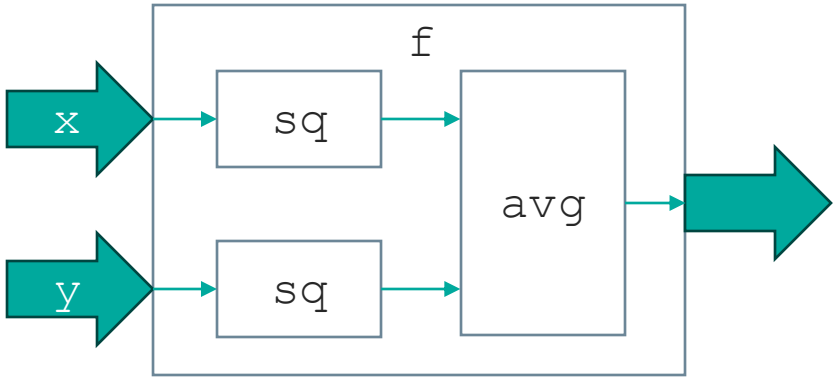
- We can create more complex functions by reusing *composing* other existing functions.
- An application written in a functional programming languages consists of many functions that work together to solve (complex) problems.

```
avg x y = (x + y) / 2

sq x = x * x

f x y = avg (sq x) (sq y)
```

- We can regard functions as boxes, which we can connect to other boxes. The output of a particular box can be the input of another box.



GÖTEBORGS UNIVERSITET
CHALMERS

Haskell scripts

- As well as the functions in the standard library (`Prelude`), you can also define your own functions
- New functions are defined within a *source code file* (also called a script), which is a text file with a sequence of definitions
- By convention, Haskell source code files usually have a `.hs` suffix (also called extension) on their filename.

GÖTEBORGS UNIVERSITET

CHALMERS

My first source code

- When writing Haskell source code, it is useful to keep two windows open,
  - one running an editor for the source code,
  - and the other running GHCi

- Start an editor, type in the two function definitions to the left, and save the script as

  `test.hs`

```
double x = x + x

quadruple x = double (double x)
```

My first source code

- Leaving the editor open, in another window start up GHCi with the new script:

```
$ ghci test.hs
```

- Now both the standard library and the file `test.hs` are loaded, and functions from both can be used:

`test.hs`

```
ghci> quadruple 10
40
ghci> even (double 3)
True
```

GÖTEBORGS UNIVERSITET
CHALMERS

My first source code

- Leaving GHCi open, return to the editor, add the following two definitions, and resave:

```
dec x = x - 1

pytha a b = sqrt (a^2 + b^2)
```

- GHCi does not automatically detect that the script has been changed, so a *reload* command must be executed before the new definitions can be used

```
ghci> :reload
Ok, one module loaded.
ghci> dec 10
9
ghci> pytha 3 4
5.0
```

GÖTEBORGS UNIVERSITET

CHALMERS

Useful GHCi commands

Command	Meaning
`:load <name>`	load source file *<name>*
`:reload`	reload current source file
`:type <expr>`	show type of *<expr>*
`:?`	show all commands
`:quit`	quit GHCi

GÖTEBORGS UNIVERSITET
CHALMERS

Naming requirements

- Function and argument names must begin with a lower-case letter. For example:

```
myFun    fun1    arg_2    x'
```

- By convention, list arguments usually have an *s* suffix on their name. For example:

```
xs    xss    ns
```

GÖTEBORGS UNIVERSITET

CHALMERS

Conditional expressions

- As in most programming languages, functions can be defined using *conditional expressions*
- Conditional expressions can be nested:
- Note: in Haskell, conditional expressions must always have an `else` branch, which avoids any possible ambiguity problems with nested conditionals

`abs` takes an integer `n` and returns `n` if it is non-negative and `-n` otherwise

```
abs n = if n >= 0 then n else -n


signum n = if n < 0 then -1 else
             if n == 0 then 0 else 1
```

GÖTEBORGS UNIVERSITET

CHALMERS

MENTI!

GÖTEBORGS
UNIVERSITET

CHALMERS