

CS-205: Welcome (to functional programming)

Cécilia PRADIC & Monika SEISENBERGER

Swansea University

02/10/23



Cécilia Pradic

- ▶ Lecturer
- ▶ Theory group
- ▶ Research: proof/automata theory, ...
- ▶ `c.pradic@swansea.ac.uk`
- ▶ Office: CoFo 410

Monika Seisenberger

- ▶ Associate Professor, deputy HoD
- ▶ Theory group (head of the)
- ▶ Research: formal methods, XAI,...
- ▶ `m.seisenberger@swansea.ac.uk`
- ▶ Office: CoFo 405

What?

Functional programming



- ▶ Haskell
- ▶ Lectures by Cécilia
- ▶ Weeks 1-6

Logic programming



- ▶ Prolog
- ▶ Lectures by Monika
- ▶ Weeks 7-10

Why?

- ▶ Get some more programming experience
- ▶ Explore new programming **features** and **styles**
 - ▶ higher-order functions, algebraic datatypes, recursion, ...
- ▶ Get experience learning new languages

Why?

- ▶ Get some more programming experience
- ▶ Explore new programming **features** and **styles**
 - ▶ higher-order functions, algebraic datatypes, recursion, ...
- ▶ Get experience learning new languages

This is a programming module

Practice!

- ▶ A bit of live-coding during the lectures

Why?

- ▶ Get some more programming experience
- ▶ Explore new programming **features** and **styles**
 - ▶ higher-order functions, algebraic datatypes, recursion, ...
- ▶ Get experience learning new languages

This is a programming module

Practice!

- ▶ A bit of live-coding during the lectures
(light on theory, although you *can* get me to babble)
- ▶ Having completed the lab: baseline

Why?

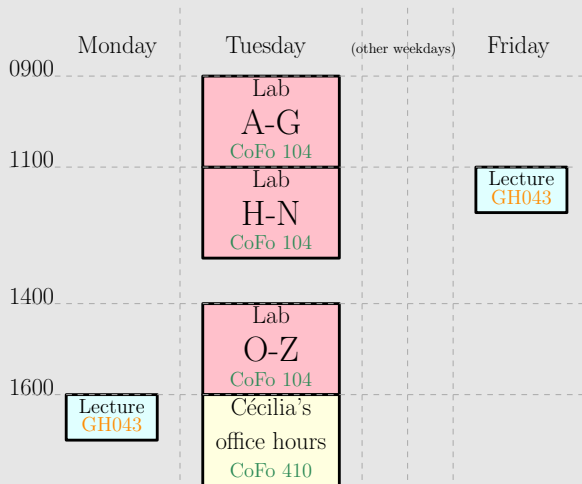
- ▶ Get some more programming experience
- ▶ Explore new programming **features** and **styles**
 - ▶ higher-order functions, algebraic datatypes, recursion, ...
- ▶ Get experience learning new languages

This is a programming module

Practice!

- ▶ A bit of live-coding during the lectures
(light on theory, although you *can* get me to babble)
- ▶ Having completed the lab: baseline
(interesting exercises: write algorithms seen in CS-270?)

How?



- ▶ $2 \times 1\text{h}$ lecture weekly
- ▶ 2h of labs
- ▶ which lab session do I go to?
→ first letter of your surname
- ▶ \exists computers but **do feel free to bring your own**
- ▶ up-to-date info on canvas

Lab rules

- ▶ Don't cheat/share solutions
- ▶ Not homework, doable within two hours (except maybe challenge tasks)
- ▶ Sign-off at the end of session/beginning of the next
- ▶ Half-marks afterwards unless an EC is submitted
(slack to deal with illness/unavailability of TAs)
- ▶ Drop me an email if you **need** to change session
(and preferably arrange to swap with someone)

<https://discord.gg/K8DH23YHWu>

- ▶ Will try to interact through that on my teaching hours (test?)
- ▶ Hopefully it will be nice (don't ruin my hopes and dreams)
- ▶ **Not the** authoritative source of materials/announcements
- ▶ Do read the readme channel and be excellent
- ▶ Thank you to (future) mod volunteer(s)

- ▶ your lab slot

Support

- ▶ your lab slot
- ▶ office hours: after the lab for me

Support

- ▶ your lab slot
- ▶ office hours: after the lab for me (if I am not in, probably I am still in the lab)

Support

- ▶ your lab slot
- ▶ office hours: after the lab for me (if I am not in, probably I am still in the lab)
(caveat: I will be on leave from the end of November)

- ▶ your lab slot
- ▶ office hours: after the lab for me (if I am not in, probably I am still in the lab)
(caveat: I will be on leave from the end of November)
- ▶ **Innovation:** computer science advisory sessions
 - ▶ Tutors and PhD students to assist you with **anything**
(or so we have been told; do respect their time and effort)
(anything \subseteq CS-205 I suppose)

- ▶ your lab slot
- ▶ office hours: after the lab for me (if I am not in, probably I am still in the lab)
(caveat: I will be on leave from the end of November)
- ▶ **Innovation:** computer science advisory sessions
 - ▶ Tutors and PhD students to assist you with **anything**
(or so we have been told; do respect their time and effort)
(anything \subseteq CS-205 I suppose)
 - ▶ Every weekday
 - ▶ 9AM-noon in CoFo 101 for general support
 - ▶ 1PM-3PM in CoFo 104 for programming support

15% Labs

- ▶ 85%: sign-off of non-challenge lab tasks
- ▶ 15%: further engagement with the module
(e.g. signing-off the challenge lab tasks, excellent coursework, ...)

15% Labs

- ▶ 85%: sign-off of non-challenge lab tasks
- ▶ 15%: further engagement with the module
(e.g. signing-off the challenge lab tasks, excellent coursework, ...)

15% Haskell coursework

- ▶ Provisional deadline/release dates: October 30th/November 13th
- ▶ Probably individual, on autograder

15% Labs

- ▶ 85%: sign-off of non-challenge lab tasks
- ▶ 15%: further engagement with the module
(e.g. signing-off the challenge lab tasks, excellent coursework, ...)

15% Haskell coursework

- ▶ Provisional deadline/release dates: October 30th/November 13th
- ▶ Probably individual, on autograder

70% Pen-and-paper exam in January

- ▶ **Significant coding portion**
- ▶ Some questions about the implementation of Haskell/Prolog

`https://wiki.haskell.org/Learning_Haskell`

- ▶ Textbook for the module: Programming in Haskell by Graham Hutton
- ▶ Thanks to him for letting us use and modify his slides!
- ▶ `https://www.cs.nott.ac.uk/~pszgmh`

That's all for the admin!

Pressing questions of general interest?

Next: finally some Haskell/functional programming

Let us get to know Haskell

Strong recommendation

Install Haskell on your own machine!

- ▶ Instruction on canvas
- ▶ You will have IDE support! (and all other obvious benefits!)
- ▶ Happy to try to lend a hand with that in labs

Let us get to know Haskell

Strong recommendation

Install Haskell on your own machine!

- ▶ Instruction on canvas
 - ▶ You will have IDE support! (and all other obvious benefits!)
 - ▶ Happy to try to lend a hand with that in labs
-
- ▶ **Currently the main haskell package on windows labs does not work Sorry!!!**
 - ▶ We will use Hugs on the lab machines in the meantime, few differences

Let us get to know Haskell

Strong recommendation

Install Haskell on your own machine!

- ▶ Instruction on canvas
 - ▶ You will have IDE support! (and all other obvious benefits!)
 - ▶ Happy to try to lend a hand with that in labs
-
- ▶ **Currently the main haskell package on windows labs does not work** Sorry!!!
 - ▶ We will use Hugs on the lab machines in the meantime, few differences

To follow along right now (if you have it already installed)

Launch ghci

- ▶ computing numerical values $2/3$, `div 3 2`

Demo time!

- ▶ computing numerical values $2/3$, `div 3 2`
- ▶ computing some lists of numbers `[1,5,7] ++ [1..4]`

Demo time!

- ▶ computing numerical values $2/3$, `div 3 2`
- ▶ computing some lists of numbers `[1,5,7] ++ [1..4]`
- ▶ more complex function composition

Demo time!

- ▶ computing numerical values `2/3`, `div 3 2`
- ▶ computing some lists of numbers `[1,5,7] ++ [1..4]`
- ▶ more complex function composition
 - ▶ `drop 2 [1..5] ++ reverse [4,13]`

Demo time!

- ▶ computing numerical values `2/3`, `div 3 2`
- ▶ computing some lists of numbers `[1,5,7] ++ [1..4]`
- ▶ more complex function composition
 - ▶ `drop 2 [1..5] ++ reverse [4,13]`
 - ▶ `fromIntegral (10 / 3)`

Demo time!

- ▶ computing numerical values `2/3`, `div 3 2`
- ▶ computing some lists of numbers `[1,5,7] ++ [1..4]`
- ▶ more complex function composition
 - ▶ `drop 2 [1..5] ++ reverse [4,13]`
 - ▶ `fromIntegral (10 / 3)`
- ▶ checking and inferring types `(77 :: Int)^77`

(Spooky. Try not to be distressed)

(But worth saying, everything is secretly statically typed)

Still demo time!

Our first function definition!

```
sumOfInverses :: Float -> Float -> Float -- type declaration
sumOfInverses x y = 1 / x + 1 / y      -- function declaration
```

Let me explain, show how to load it and play with it

The same function in Java

```
public static float sumOfInverses(float x, float y)
{
    return 1/x + 1/y;
}
```


Now, for an advanced example

```
aFunction [] = []  
aFunction (x : xs) = aFunction pre ++ x : aFunction post  
    where pre = filter (> x) xs  
          post = filter (<= x) xs
```

Can you guess what is the value of `aFunction [2,5,4,3,7]`?

First lab: to get started!

- ▶ You will familiarize yourselves with the interpreter
- ▶ You will write a few functions yourselves
- ▶ You will familiarize yourselves with basic list and arithmetic functions

First lab: to get started!

- ▶ You will familiarize yourselves with the interpreter
- ▶ You will write a few functions yourselves
- ▶ You will familiarize yourselves with basic list and arithmetic functions

To look up documentation

<https://hoogle.haskell.org>

- ▶ not *required* for the first lab
- ▶ I will walk you through this on Friday

First lab: to get started!

- ▶ You will familiarize yourselves with the interpreter
- ▶ You will write a few functions yourselves
- ▶ You will familiarize yourselves with basic list and arithmetic functions

To look up documentation

<https://hoogle.haskell.org>

- ▶ not *required* for the first lab
- ▶ I will walk you through this on Friday

End of demo! Questions?

So what's a functional programming language (PL)?

Vibes¹ surrounding certain PL features such as

- ▶ algebraic datatypes
- ▶ parametric polymorphism
- ▶ anonymous **higher-order functions**
- ▶ implicit memory management
- ▶ static type inference
- ▶ type classes

¹see e.g. <https://cs.brown.edu/people/sk/Publications/Papers/Published/sk-teach-pl-post-linnaean/paper.pdf>

So what's a functional programming language (PL)?

Vibes¹ surrounding certain PL features such as

- ▶ algebraic datatypes
- ▶ parametric polymorphism
- ▶ anonymous **higher-order functions**
- ▶ implicit memory management
- ▶ static type inference
- ▶ type classes

Vibes!!!

- ▶ you can find examples/counter-examples among functional PLs

¹see e.g. <https://cs.brown.edu/people/sk/Publications/Papers/Published/sk-teach-pl-post-linnaean/paper.pdf>

So what's a functional programming language (PL)?

Vibes¹ surrounding certain PL features such as

- ▶ algebraic datatypes
- ▶ parametric polymorphism
- ▶ anonymous **higher-order functions**
- ▶ implicit memory management
- ▶ static type inference
- ▶ type classes

Vibes!!!

- ▶ you can find examples/counter-examples among functional PLs
- ▶ you can find some of these among “dysfunctional” PLs

¹see e.g. <https://cs.brown.edu/people/sk/Publications/Papers/Published/sk-teach-pl-post-linnaean/paper.pdf>

So what's a functional programming language (PL)?

Vibes¹ surrounding certain PL features such as

- ▶ algebraic datatypes
- ▶ parametric polymorphism
- ▶ anonymous **higher-order functions**
- ▶ implicit memory management
- ▶ static type inference
- ▶ type classes

Vibes!!!

- ▶ you can find examples/counter-examples among functional PLs
- ▶ you can find some of these among “dysfunctional” PLs
- ▶ complicated socio-technical history

¹see e.g. <https://cs.brown.edu/people/sk/Publications/Papers/Published/sk-teach-pl-post-linnaean/paper.pdf>

So what's a functional programming language (PL)?

Vibes¹ surrounding certain PL features such as

- ▶ algebraic datatypes
- ▶ parametric polymorphism
- ▶ anonymous **higher-order functions**
- ▶ implicit memory management
- ▶ static type inference
- ▶ type classes

Vibes!!!

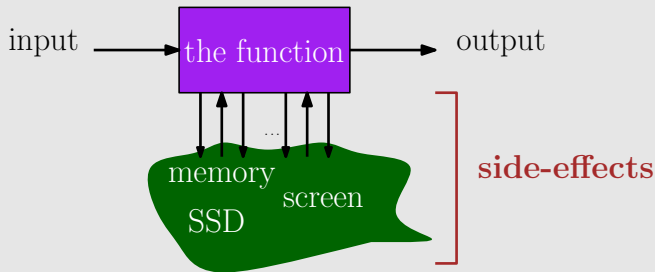
- ▶ you can find examples/counter-examples among functional PLs
- ▶ you can find some of these among “dysfunctional” PLs
- ▶ complicated socio-technical history
- ▶ but Haskell has all of these features

¹see e.g. <https://cs.brown.edu/people/sk/Publications/Papers/Published/sk-teach-pl-post-linnaean/paper.pdf>

Haskell: a different point of view on functions

Peculiarity of Haskell

No way to do **side-effects**



Examples of side-effects:

- ▶ Reading/writing a file
- ▶ Printing to the screen
- ▶ Changing a value in memory

⇒ **no mutable variables, no loops!**

Example of a side effect in Java

```
static public void sumSuffixes(int[] arr)
{
    for(int i = arr.length-2; i >= 0; --i)
        arr[i] += arr[i+1];
}
```

- ▶ A single return value
- ▶ Operation: alter the global memory (depending on arguments)
- ▶ An auxiliary mutable variable `i`



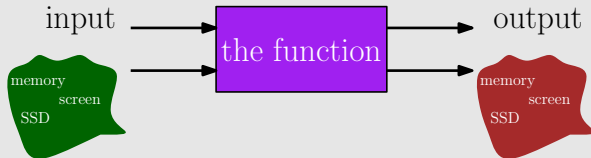
- ▶ Closer to mathematical functions
- ▶ Constrain the programming style in interesting ways
(easier to reason about code)
- ▶ No loss of expressiveness: recursion instead of loops

But using side-effects is possible!

It is even necessary sometimes

(people do want to write files)

The Haskell way: treat “real world states” as data to pass around



- ▶ \exists *very* nice abstractions to deal with that
- ▶ so all is well

Some modern functional PLs

Legacy from three big academic traditions

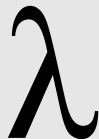
ML



Haskell



LISP dialects



- ▶ Different set of features/design philosophies
- ▶ Big inspiration for new features in more mainstream languages
- ▶ Influenced greatly the design of
 - ▶ Rust
 - ▶ dependently typed languages/proofs assistants

Some examples of big Haskell projects

Some that were somehow relevant to me

- ▶ Pandoc - “a universal document converter”
- ▶ xmonad - a tiling windows manager (similar to the one I am using right now)
- ▶ hakyll - a static blog generator

More at https://wiki.haskell.org/Applications_and_libraries