CIS 1904: Haskell

Course Staff

Instructor: Me (Cassia Torczon)

- PhD student in programming languages
- Office Hours: 3:30 - 4:30 p.m. on Thursdays, or by appointment
  - Location: GRW 571

TA: Claire Wang

- PhD student in machine learning and programming languages
- Office Hours: 3:00 - 4:00pm on Wednesdays
  - Location: Levine 612

Course Policies

Homework:

- Roughly weekly, due Mondays (sorry) at 11:59pm
- Intended to take ~3h/homework; if it takes more, please come to OH
- You may use Hoogle and discuss *high-level* ideas with classmates
- **You may NOT view, edit, or copy another student's code**
  - For homework only – you may work together in class

Course Policies

LLM use is strongly discouraged but not forbidden

- If you use any resources beyond Hoogle or class materials, **you must provide a full list of materials used**, including LLM transcripts
- **If you cannot explain your answer, points may be taken off**
- LLM policy is subject to change over the semester

Course Policies

Why we discourage LLMs:

- This course teaches foundational skills
  - You need these to assess and effectively use LLM output
  - Jobs expect you to have these
- Output is often wrong (I have graded Copilot code that did not even compile)
- Your questions help us teach better!
  - I won’t know which parts of the material are confusing otherwise

Course Policies

Class:

- Attendance is required
  - **Please do not come if you're sick!** Just email me, ideally before class.
  - Grades will be roughly 10% participation, 90% homework
- Mix of lecture and in-class exercises
- Most classes will have associated readings

We will use:

- Course website: these policies, general course information
- GitHub: for distributing homework
- Gradescope: for turning in homework
- Ed: for asking questions
- Canvas: for final grades
- Poll Everywhere: for some in-class practice problems
- Email, if you need to contact me directly

What is this class about?

What is this class about?

Haskell

What is this class about?

Haskell as a key example of a programming language that is:

- functional
- pure
- lazy
- statically-typed

What is a functional language?

- Functional programming: a *paradigm* for program design based on **applying** and **composing functions**
  - Designed to “look like math”
  - Evolved out of logic
- A few alternatives to functional programming:
  - Imperative programming (common in C)
  - Object-oriented programming (common in Java)
- Many languages support multiple paradigms but may be designed more with some in mind than others

What is a functional language?

- Functional programming: a *paradigm* for program design based on **applying** and **composing functions**
  - Designed to “look like math”
  - Evolved out of logic
- A few alternatives to functional programming:
  - Imperative programming (common in C)
  - Object-oriented programming (common in Java)
- Many languages support multiple paradigms but may be designed more with some in mind than others

OCaml is designed to mix all of these!

What is a pure language?

A pure language has no “side effects” like:

- Printing
- Reading from memory
- Anything except evaluating a term down to a simpler term
  - e.g. $2 + 2 + 2 \rightarrow 4 + 2 \rightarrow 6$

- Easy to reason about
- Helps avoid classes of bugs (especially concurrency bugs)
- We will see later how to write real-world code with this restriction
  - (It’s monads)

What is a lazy language?

In a lazy language, programs do not get evaluated until their results are needed!

```
foo :: Int → Bool
foo x = True
```

If multiplying any two numbers takes one second, and function overhead is negligible, how long does it take to run `foo (217^81)`?

What is a lazy language?

In a lazy language, programs do not get evaluated until their results are needed!

```
foo :: Int → Bool
foo x = True
```

If multiplying any two numbers takes one second, and function overhead is negligible, how long does it take to run `foo (217^81)`?

- In most languages, 80s (we have to calculate 217^{81} before returning `True`)
- In Haskell, 0s — we notice that we never need to know the value of `x` to evaluate the function, so we just return `True` right away!

What is a statically-typed language?

- Static typing: types are known *at compile time*

What is a statically-typed language?

- Static typing: types are known *at compile time*
  - e.g., the programmer annotates terms with their types (as in OCaml)
    ```
    let x : int = 1904
    ```
  - e.g., the **compiler** is able to infer types for many terms (also in OCaml)
    ```
    let s = “Haskell”
    ```
- facilitates analysis and optimization during compilation
- immediate feedback and refactoring support from IDE
  ```
  let y = s + 5
  Error: expression has type string, expected type int
  ```
- provides documentation and design support
  ```
  let x = fun1 (fun2 arg1 arg2) arg3
  ```

What is a statically-typed language?

- Dynamic typing: information we get from types is determined at *runtime*
  - e.g., as in Python
  - often quicker to write without annotations
  - sometimes gives the programmer more freedom

```
x = 1904    # type(x) returns <class 'int'>
x = "Haskell"   # type(x) returns <class 'str'>
```

What is a statically-typed language?

- Dynamic typing: information we get from types is determined at *runtime*
  - e.g., as in Python
  - often quicker to write without annotations
  - sometimes gives the programmer more freedom

```
x = 1904    # type(x) returns <class 'int'>
x = "Haskell"   # type(x) returns <class 'str'>


y = x + 5       # crashes
```

Why Haskell?

- It’s an excellent example of a functional, pure, lazy, statically-typed language
- Useful tool for learning about abstractions used in such languages
- It’s modular and concise, with strong correctness guarantees
- It’s fun :)

Installation!

https://github.com/cassiatorczon/cis1904-spring26

> week00 > homework > instructions.md