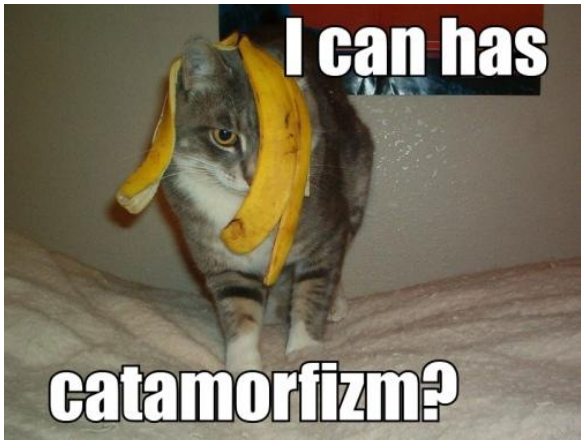
CSCI 490
Functional Programming and the Art of Recursion

Spring 2016
Lecture: TR 1:15-2:30, MC Reynolds 317
Website: `http://ozark.hendrix.edu/~yorgey/490/`

Instructor: Brent Yorgey, MC Reynolds 310
Office hours: any time my door is open, or make an appointment at
`http://byorgey.youcanbook.me`
Email: `yorgey@hendrix.edu`

Course Description

Functional programming takes as its central organizing principle the idea of evaluating functions rather than executing instructions. It supports programming at a high level of abstraction and careful control of side effects. This course explores functional programming in depth, combining practical programming experience in Haskell and Idris with exploration of the underlying theory. Planned topics include algebraic data types and pattern matching, recursion and induction, folds, lambda calculus (typed and untyped), laziness, functors, and monads. We will also spend some time doing dependently typed functional programming and computer-checked formal proofs in Idris, exploring the deep connection between functional programs and formal logic.



By the end of the semester you will get this joke

Calendar

Week	Month	Days	Topics
1	Jan	19,21	Introduction to Haskell
2		26, 28	Algebraic data types, pattern matching, logic and proof
3	Feb	2, 4	Laziness, higher-order programming, recursion patterns
4		9, 11	Polymorphism, type inference, folds
5		16, 18	Folds, structural induction
6		23, 25	Type classes, kinds, Functors
7	Mar	1, 3	Applicative functors
8		8, 10	Monads
9		15, 17	Untyped λ-calculus
			Spring break
10		29, 31	Simply typed λ-calculus
11	Apr	5, 7	Curry-Howard isomorphism, introduction to Idris
12		12, 14	Idris
13		19, 21	Open recursion
14		26, 28	Cata- and anamorphisms, free monads

Evaluation

Evaluation will be based on

- problem sets (40%),
- presentations and participation (10%),
- quizzes (25%), and
- a final project (25%).

Problem sets

Problem sets will be assigned weekly. They will typically have both a written component and a programming component. Let's not beat around the bush: they are going to be very challenging! This is where most of your learning will take place in the course.

Problem sets will be completed in groups of 2 or 3. You may pick your own groups (though I am happy to serve the role of matchmaker if you need help finding a group).

Assignments must be turned in electronically via Moodle. Problem sets must be written up as `.lhs` files, which are simultaneously valid Haskell source files and also can be typeset via LaTeX using the `lhs2tex` tool. You will turn in your problem set both in `.lhs` and typeset PDF format.

Occasionally there may be some additional written component which you are free to write up in some other format. Acceptable formats for such components include

- PDF
- OpenDocument (`.odt`)
- plain text (`.txt`)

Proprietary formats such as Microsoft Word (`.doc`, `.docx`) or Apple Pages are *not* acceptable. Documents submitted in such formats will receive a score of zero.

Each student has three late days to spend throughout the semester as they wish. Simply inform me any time *prior* to the due date for an assignment that you wish to use a late day; you may then turn in the assignment up to 24 hours late with no penalty. Multiple late days may be used on the same assignment. There are no partial late days; turning in an assignment 2 hours late or 20 hours late will both use 1 late day.

Partners working on an assignment together must each use a late day in order to turn in the assignment up to 24 hours late. Late days are non-transferrable.

Presentations

Each week, several students will be randomly selected and assigned a problem from the upcoming problem set. The selected students will be responsible to present a solution and facilitate a class discussion of their assigned problem when the problem set is due the following week.

Presentations will constitute 10% of the final grade, and will be graded on preparedness, clarity, appropriate use of time, appropriate use of aids such as chalkboard, projector, or papier-mâché models, and facilitation of discussion.

Group members of the presenter will also receive the same grade as the presenter. The point of this is that it is the responsibility of the group to help prepare a presentation, even though only one person will give it.

Quizzes

There will be weekly ten-minute quizzes testing you on key aspects of the previous week's problem set.

Grading of the quizzes will be mastery-based: you can re-take a quiz as many times as you like (though you will get a different version of the quiz each time) until you master the material. Your final grade for the quiz will be the maximum of the grades achieved. The first iteration of a given quiz will be in class, in written form; subsequent iterations will use an oral format at a mutually agreeable time.

Final project

At the end of the semester, you will undertake a final project, worth 25% of your final grade, which explores a topic from the course in greater depth, or explores a functional programming topic not covered during the course. Projects may be completed either individually or in groups of two or three. The project topic is open-ended. Examples of project topics include:

- Implementing some sort of substantial system in Haskell or Idris.
- Learning about, implementing, and explaining (*e.g.* via an expository paper) some interesting functional algorithms or data structures.
- Using Idris (or some other formal theorem-proving system) to carry out some nontrivial proof(s), such as proving the correctness of a particular algorithm or data structure implementation, or proving some mathematical theorem.
- Contributing to an open-source project written in a functional language.

More specific details about the final project will be discussed as it approaches. The scheduled final exam slot will be used for final project presentations.

Attendance and submission policies

Lecture attendance is expected, but is not a formal part of your grade. I do appreciate you letting me know in advance when you must be absent for some reason.

If you are absent from lecture it is **your responsibility** to obtain notes from other student(s). Do not come to me and ask "what did I miss?".[1] On the other hand, if after obtaining notes you have specific questions or confusions regarding the topics covered, I would be happy to talk with you.

Disabilities

It is the policy of Hendrix College to accommodate students with disabilities, pursuant to federal and state law. Students should contact Julie Brown in the Office of Academic Success (505.2954; `brownj@hendrix.edu`) to begin the accommodation process. Any student seeking accommodation in relation to a recognized disability should inform the instructor at the beginning of the course.

[1] I am likely to answer that you missed the part where that is your responsibility.

Academic Integrity

All Hendrix students must abide by the College's Academic Integrity Policy as well as the College's Computer Policy, both of which are outlined in the Student Handbook.

For specific ways the Academic Integrity policy applies in this course, please refer to the Computer Science Academic Integrity Policy.

The short version is that academic integrity violations such as copying code from another student or the Internet are **easy to detect**, will be **taken very seriously**, and carry a default recommended sanction of **failure in the course**.

If you have any questions about how the Academic Integrity policy applies in a particular situation, please contact me.

Learning objectives

By the end of the course you will:

- **Induction and recursion**
  - Become comfortable with induction as a foundational tool for mathematical reasoning
  - Design appropriate recursive data types to model data
  - Write recursive programs over recursive data types in Haskell and Agda
  - Carry out inductive proofs about recursive data types, in English and in Agda
  - Understand the relationship between folds and induction
- **Type systems**
  - Understand the type systems of the simply typed λ-calculus, Haskell, and Agda
  - Understand and produce typing derivations for terms of the λ-calculus
  - Understand the role of type systems in constraining programs
  - Understand the logical interpretation of types, via the Curry-Howard isomorphism
- **λ-calculus**
  - Understand the historical and continuing role of the λ-calculus as a foundational model of logic and computation
  - Understand the definition, mechanics, and significance of the Y combinator

  - Become comfortable with the mechanics of the λ-calculus as a model of computation, including its reduction behavior and Church encoding

- **Haskell**
  - Write moderately sophisticated Haskell programs to solve specific problems
  - Understand algebraic data types and pattern-matching
  - Design, use, and prove properties of recursion patterns such as maps, filters, and folds

- **Idris**
  - Write basic data type and function definitions in Idris
  - Translate back and forth between logical statements expressed in English and as Idris types
  - Carry out inductive proofs in Idris