This class

• CS 476/MCS 415, Programming Language Design
• MWF 2:00-2:50 PM, Burnham Hall 317

• Course website: https://www2.cs.uic.edu/~mansky/teaching/cs476/fa18/index.html

• Lectures recorded, available on Blackboard
• Discussion board via Piazza, assignments via Gradescope
Instructors

• Professor: William Mansky ([mansky1@uic.edu](mailto:mansky1@uic.edu))
• Office hours Monday 9:30-10:30 AM, Wednesday 3:00-4:00 PM in SEO 1127, and by appointment

• TA: Jack Blandin ([blandin1@uic.edu](mailto:blandin1@uic.edu))
• Office hours Tuesday, Thursday 5:00-6:00 PM in SEO 1127
Textbook

• *Types and Programming Languages*, Pierce, 2002
• Available online through the library
Grading

• Assignments: 30%
• Midterm: 35%
• Final: 35%
• For 4-credit students, project: 25%
• Participation: up to 5% extra credit
Assignments

• Programming assignments in OCaml: write an interpreter for a language/feature, implement a type checker, etc.

• Written homework: try out logical systems, write proofs about programs

• Collaboration encouraged, but you must write up your own solution, cite sources

• Submit via Gradescope

• Two free late days; after that, 20% penalty per day late for up to two days
Ask questions!

• In class – stop me anytime!

  — Can ask/answer anonymously
  — Can post privately to instructors
  — Can answer other students’ questions
Programming Language Design

• Up till now, you’ve interacted with PLs as *users*
• We’ll look at PLs as *designers* – syntax, features, specification, intended behavior
• And also *implementers*
  — Compile into another language (see CS 473), e.g. C, Java, OCaml
  — Write an interpreter, e.g. JavaScript, Python, Ruby
  — Some of both
Structure of a language

• Syntax
  — Concrete: what do programs look like?
  — Abstract: what are the pieces of a program?

• Semantics
  — Static: which programs make sense?
  — Dynamic: what do programs do when we run them?

• Pragmatics
  — Implementation: how can we actually make the semantics happen?
  — IDE, tool support, etc.
Language Implementation

• Lexing
• Parsing
• Type checking
• (optimization)
• Execution OR translation, then execution
Course outline

• Language types: imperative, functional, OO, logic, ...
• Syntax: grammars, abstract syntax trees
• Operational semantics
• Interpreters
• Type systems: checking, inference, safety
• Program verification
The OCaml Programming Language

• OCaml: a functional language in the ML family
  — Closely related to SML, F#
  — Designed to operate on elements of programming languages

• Strongly-typed functional language with references, based on lambda calculus with pattern-matching
OCaml: The Read-Eval-Print Loop

(demo)

• Can also be compiled
Inductive Data Types

• Define a type by giving a list of cases

type season = Spring | Summer | Fall | Winter

type value = Intval of int
  | Stringval of string | Floatval of float

type intlist = Nil | Cons of int * intlist
Cons (1, Cons (2, Cons (3, Nil)))
Pattern-Matching and Recursion

type season = Spring | Summer | Fall | Winter

let get_temp s =
  match s with
  | Spring -> 70
  | Summer -> 80
  | Fall -> 70
  | Winter -> 30
Pattern-Matching and Recursion

type value = Intval of int
  | Stringval of string | Floatval of float

let print_val v =
  match v with
  | Intval i -> print_int i
  | Stringval s -> print_string s
  | Floatval f -> print_float f
Pattern-Matching and Recursion

type intlist = Nil | Cons of int * intlist

let rec length l =
  match l with
  | Nil -> 0
  | Cons (i, rest) -> length rest + 1
Common OCaml Errors
HW1 – Getting Started with OCaml

• Set up your OCaml programming environment
• Write some recursive functions on inductive data types
• Due Monday 9/3 at 2 PM
• No late days

• Next time: syntax