This class

• CS 476/MCS 415, Programming Language Design
• MWF 2:00-2:50 PM, Burnham Hall 209

• Course website: https://www2.cs.uic.edu/~mansky/teaching/cs476/fa19/index.html

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

• Professor: William Mansky (mansky1@uic.edu)
  • Office hours Monday 10:00-11:00 AM, Wednesday 3:00-4:00 PM in SEO 1331, and by appointment

• TA: Krishna Garg (kgarg8@uic.edu)
  • Office hours Tuesday, Thursday 11:00 AM-12:00 PM in SEO 1325

• Office hours are great for homework help!
Textbook

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

• Assignments: 30%
• Midterms (2): 40%
• Final: 30%
• 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
• Can submit up to 2 days late at a 20% penalty
Ask questions!

• In class, verbally or with Poll Everywhere

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

• If you have a question, someone else probably has the same question!
Questions

Top
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
  — Best practices, tradeoffs

• 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
What is a programming language?
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.
Three Ways of Describing a Language

• Text
  — Arithmetic operators include +, -, and *.
  — The value of “a + b” is equal to the sum of the value of a and the value of b.

• Math
  — \( O ::= + | - | * \)
    \[
    \begin{array}{c}
    a \Downarrow v_a \\
    b \Downarrow v_b
    \end{array} \quad \frac{a + b \Downarrow v_a + v_b}{a+b \Downarrow v_a + v_b}
    \]

• Code
  — type op := Plus | Minus | Times
  — eval (Op Plus a b) = eval a + eval b
Course outline

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

• OCaml: a functional language in the ML family
  — Do almost all control flow with functions
  — 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

• Math + OCaml dust = program! (type checker, interpreter, etc.)
OCaml: The Read-Eval-Print Loop

(demo)

• Can also be compiled
OCaml: Recursion, not Loops

• “Variables” in a functional language can’t be changed:
  let x = 3;;
  let add_x y = x + y;;
  add_x 2;; (* returns 5 *)
  let x = 4;;
  add_x 2;; (* still returns 5 *)
**OCaml: Recursion, not Loops**

- “Variables” in a functional language can’t be changed

```ocaml
let total = 0;
for (i = 4; i >= 0; i--){
    total = total + i;
}
```

won’t work!
OCaml: Recursion, not Loops

• “Variables” in a functional language can’t be changed
• We can use recursive functions instead:

```ocaml
let rec sum i = if i > 0 then i + sum (i - 1) else 0;;
sum 4;; (* would be sum(4) in C *)
```
Data Structures as Functions: Sets

• Contain at most one of any item

\{"a", "b"\}  

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>yes</td>
</tr>
<tr>
<td>b</td>
<td>yes</td>
</tr>
<tr>
<td>Everything else</td>
<td>no</td>
</tr>
</tbody>
</table>
Data Structures as Functions: Bags

• Also called multisets

\{“a”, “a”, “b”, “b”, “b”\}

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>Everything else</td>
<td>0</td>
</tr>
</tbody>
</table>

type bag = string -> int
Data Structures as Functions: Maps

• Map keys to values

```json
{“a”: “val_a”, “b”: “val_b”}
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>val_a</td>
</tr>
<tr>
<td>b</td>
<td>val_b</td>
</tr>
<tr>
<td>Everything else</td>
<td>“”</td>
</tr>
</tbody>
</table>

type map = string -> string
Questions
HW1 – Getting Started with OCaml

• Set up your OCaml programming environment
• Write some recursive functions on inductive data types
• Due Tuesday 9/3 at 2 PM

• Next time: syntax