A Programming Language

- Two variables
  - x, y
- Three operations
  - x++
  - x--
  - (x=0)? L1:L2;

L1: x++; y--; (y=0)?L2:L1
L2: ...

Fact: This language is “equivalent” to every PL!
Try writing quicksort ... or Windows, Google, Spotify,... in this language.

So why study PL?

“A different language is a different vision of life”
- Federico Fellini
So why study PL?

Programming language shapes Programming thought

PL characteristics affects how:
• Ideas are expressed
• Computation is expressed

Course Goals: Learn Languages/Constructs
New ways to:
- describe
- organize
- think about computation

“Free your mind”
-Morpheus
Enables you to

Write software that’s
• Readable
• Correct
• Extendable
• Modifiable
• Reusable

On the internet, nobody knows you’re a dog…
… a Ruby Program
… a Scala Program
… an Erlang Program

Enables you to learn new PLs

No Java (C#) 15 (10) years ago

Learn the anatomy of a PL
• Fundamental building blocks
• Different guises in different PLs

Understand PLs you already know

Enables you to design new PLs

...“who, me?”

Buried in every extensible system is a PL
• Emacs: Lisp
• Word, Powerpoint: Macros, VBScript
• Unreal: UnrealScript (Game Scripting)
• Facebook: FBML, FBJS
• SQL, Renderman, LaTeX, XML ...
Enables you to choose right PL

“...but isn’t that decided by
• libraries,
• standards,
• and my boss?”
Yes.

My goal: educate tomorrow’s tech leaders & bosses, so you’ll make informed choices

Mechanics

cseweb.ucsd.edu/classes/fa11/cse130-a/

Nothing printed, everything on Webpage!

Meetings:
• Lectures: Ranjit Jhala, Tu-Th 6:30-7:50pm @ CENTER 212
• TAs: Ming Kawaguchi, Ross Tate
• Tutor: Ayelet Bitton
• Office Hours: RJ (Th 2-4p)
  - Hours posted on web page (may change a bit)

Material

Outline:
1. Functional, OCaml, 4 weeks
2. OO, Python, 4 weeks
3. Logic, Prolog, 1 week

No recommended Text:
• Online lecture notes
• Resources posted on webpage
• Pay attention to lecture and section!

Requirements and Grading

• The good news: No Homework
• In-Class Exercises: 5%
• Midterm: 30%
• Programming Assignments (7): 30%
• Final: 35%

Grading on a curve. Two hints/rumors:
1. Lot of work
2. Don’t worry (too much) about grade
Suggested Homeworks

- On webpage after Thursday lecture
- Based on lectures, section of previous Tue, Thu
- Recommended, ungraded, HW problems are sample exam questions
- Webpage has first samples already

Weekly Programming Assignments

Schedule up on webpage

Due on Friday 5 PM

Deadline Extension:
- Four “late days”, used as “whole unit”
- 5 mins late = 1 late day
- Plan ahead, no other extensions

PA #1 online, due 4/8, 5:00 PM

Weekly Programming Assignments

Unfamiliar languages
+ Unfamiliar environments

Start Early!

Weekly Programming Assignments

Scoring = Style + Test suite

No Compile, No Score
Forget Java, C, C++ ... other 20th century PLs

Don’t complain ... that Ocaml is hard ... that Ocaml is *#@##!#!

Immerse yourself in new language

Word from our sponsor ...

- Programming Assignments done ALONE
- We use plagiarism detection software
  - I am an expert
  - Have code from all previous classes
  - MOSS is fantastic, plagiarize at your own risk
- Zero Tolerance
  - Offenders punished ruthlessly
Enough with the small talk

Say hello to OCaml

Why readability matters...

Quicksort in C

Quicksort in Ocaml

Quicksort in J

Quicksort in OCaml
Plan (next 4 weeks)

1. Fast forward
   - Rapid introduction to what's in ML

2. Rewind

3. Slow motion
   - Go over the pieces individually

ML: History, Variants

“Meta Language”
Designed by Robin Milner
To manipulate theorems & proofs

Several dialects:
- Standard ML (SML)
  - Original syntax
- Objective Caml: (Ocaml)
  - “The PL for the discerning hacker”
  - State-of-the-art, extensive library, tool, user support
- F# (Ocaml.NET) released in Visual Studio

ML’s holy trinity

Expression ➔ Value

Type

- Everything is an expression
- Everything has a value
- Everything has a type

Interacting with ML

“Read-Eval-Print” Loop

Repeat:
1. System reads expression \( e \)
2. System evaluates \( e \) to get value \( v \)
3. System prints value \( v \) and type \( t \)

What are these expressions, values and types?
**Base type: Integers**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2+2</td>
<td>4</td>
</tr>
<tr>
<td>2 * (9+10)</td>
<td>38</td>
</tr>
<tr>
<td>2 * (9+10) -12</td>
<td>26</td>
</tr>
</tbody>
</table>

**Complex expressions using “operators”**: (why the quotes?)
- +, -, *
- div, mod

**Base type: Strings**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ab”</td>
<td>“ab”</td>
</tr>
<tr>
<td>“ab” ^ “xy”</td>
<td>“abxy”</td>
</tr>
</tbody>
</table>

**Complex expressions using “operators”**: (why the quotes?)
- Concatenation ^

**Base type: Booleans**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>1 &lt; 2</td>
<td>true</td>
</tr>
<tr>
<td>“aa” = “pq”</td>
<td>false</td>
</tr>
<tr>
<td>(“aa” = “pq”) &amp;&amp; (1&lt;2)</td>
<td>false</td>
</tr>
<tr>
<td>(“aa” = “aa”) &amp;&amp; (1&lt;2)</td>
<td>true</td>
</tr>
</tbody>
</table>

**Complex expressions using “operators”**: “Relations”:
- =, <, <=, >=
- &&, ||, not

**Type Errors**

- (2+3) || (“a” = “b”) is rejected
- “pq” ^ 9
- (2 + “a”) is rejected

**Untypable expression is rejected**
- No casting, No coercing
- Fancy algorithm to catch errors
- ML’s single most powerful feature (why?)
Complex types: Product (tuples)

- (2+2, 7>8); (4, false)
- int * bool

Complex types: Lists

- Unbounded size
- Can have lists of anything (e.g. lists of lists)
- but ...

- Triples,…
- Nesting:
  - Everything is an expression
  - Nest tuples in tuples

- All elements must have same type

- [1; “pq”];

- 'a list
- int list
- string list
- (int*string) list
- (int list) list

- [];
- [ ];
- [1;2;3];
- [1;2;3]
- [1+1;2+2;3+3;4+4];
- [2;4;6;8]
- [“a”; “b”; “c”“d”];
- [“a”; “b”; “cd”]
- [(1,”ab”);(3+4,”c”)];
- [(1,”ab”);(7,”c”)]
- [[1];[2;3];[4;5;6]];
Complex types: Lists

List operator “Cons” ::

1::[];  
1::[2];  
“a”::[“b”;“c”];

List operator “Append” @

[1;2]@[3;4;5];  
[“a”]@[“b”];

Can only “cons” element to a list of same type

Can only append two lists

… of the same type

Complex types: Lists

List operator “head” hd

hd [1;2];  
hd ([“a”]@[“b”]);

List operator “tail” tl

tl [1;2;3];  
tl ([“a”]@[“b”]);

Only take the head a nonempty list

Only take the tail of nonempty list
Recap: Tuples vs. Lists?

What’s the difference?

- **Tuples:**
  - **Different types, but fixed number:**
    - pair = 2 elts
    - triple = 3 elts
    - (3, “abcd”) (int * string)
    - (3, “abcd”, (3.5, 4.2)) (int * string * (float * float))
  
- **Lists:**
  - **Same type, unbounded number:**
    - [3;4;5;6;7] int list

So far, a fancy calculator...

... what do we need next?