Midterm

Friday was . . . interesting?

Spent the weekend reflecting.

Learning experience for all of us.

Debrief:
  - what we learned
  - what we’re going to do moving forward

Midterm: What’s the point?

Preparation, study, review.

Feedback for me and feedback for you.

Just one part of demonstrating your ability.

Practice performing under pressure.
Midterm: How did we do?

Preparation, study, review. ✓

Feedback for me and feedback for you.

Just one part of demonstrating your ability.

Practice performing under pressure. ✓

Midterm: Feedback / Analyzing Results

Feedback for me:
- despite best efforts, test a bit long / difficult
- in the end, distribution very informative
- y’all did *extremely* well considering challenge

Feedback for you:
- great job preparing, keep it up!
- pretty good job being strategic, nice work 😊
- opportunity to consider motivations / priorities

Midterm: Just One Part of Grade
Midterm: Just One Part of Grade

That said, I was once an undergrad myself.

Despair = worst possible midterm outcome.

So:

$$\text{midterm} = \max (\text{midterm}_0, \text{final})$$
or (curried):

$$\text{midterm} = \max \text{midterm}_0 \text{ final}$$

Midterm: How did we do?

Preparation, study, review.

Feedback for me and feedback for you.

Just one part of demonstrating your ability.

Practice performing under pressure.

We’re actually in pretty good shape.

I really like studying programming languages.

Super stoked to explore PL with all of you.

Why?
If you are in a shipwreck and all the boats are gone, a piano top buoyant enough to keep you afloat may come along and make a fortuitous life preserver.

This is not to say, though, that the best way to design a life preserver is in the form of a piano top.

I think we are clinging to a great many piano tops in accepting yesterday’s fortuitous contrivings as constituting the only means for solving a given problem.

I really like studying programming languages.

Super stoked to explore PL with all of you.

Why?

PL helps us break free to think thoughts, ask questions, and solve problems that would otherwise be inaccessible.

More Detailed Course Motivation

• Why learn fundamental concepts that appear in all languages?
• Why use languages quite different from C, C++, Java, Python?
• Why focus on functional programming?
• Why use ML, Racket, and Ruby in particular?
• Not: Language X is better than Language Y

[You won’t be tested on this stuff]
Summary

• No such thing as a “best” PL
• Fundamental concepts easier to teach in some (multiple) PLs
• A good PL is a relevant, elegant interface for writing software
  - There is no substitute for precise understanding of PL semantics
• Functional languages have been on the leading edge for decades
  - Ideas have been absorbed by the mainstream, but very slowly
  - First-class functions and avoiding mutation increasingly essential
  - Meanwhile, use the ideas to be a better C/Java/PHP hacker
• Many great alternatives to ML, Racket, and Ruby, but each was
  chosen for a reason and for how they complement each other

What is the best kind of car?
What is the best kind of shoe?

Cars / Shoes

Cars are used for rather different things:
  - Winning a Formula 1 race
  - Taking kids to soccer practice
  - Off-roading
  - Hauling a mattress
  - Getting the wind in your hair (some of us…)
  - Staying dry in the rain

Shoes:
  - Playing frisbee
  - Going to a formal
  - Going to the beach

More on cars

• A good mechanic might have a specialty, but also understands
  how “cars” (not a particular make/model) work
  - The upholstery color isn’t essential (syntax)
• A good mechanical engineer really knows how cars work, how
  to get the most out of them, and how to design better ones
  - I don’t have a favorite kind of car or a favorite PL
• To learn how car pieces interact, it may make sense to start with
  a classic design rather than the latest model
  - A popular car may not be best
  - May especially not be best for learning how cars work
**Why semantics and idioms**

This course focuses as much as it can on semantics and idioms

• Correct reasoning about programs, interfaces, and compilers requires a precise knowledge of semantics
  - Not “I feel that conditional expressions might work like this”
  - Not “I like curly braces more than parentheses”
  - Much of software development is designing precise interfaces; what a PL means is a really good example

• Idioms make you a better programmer
  - Best to see in multiple settings, including where they shine
  - See Java in a clearer light even if I never show you Java

**Hamlet**

The play *Hamlet*:
  - Is a beautiful work of art
  - Teaches deep, eternal truths
  - Is the source of some well-known sayings
  - Makes you a better person

Continues to be studied centuries later even though:
  - The syntax is really annoying to many
  - There are more popular movies with some of the same lessons
  - Reading *Hamlet* will not get you a summer internship

**All cars are the same**

• To make it easier to rent cars, it is great that they all have steering wheels, brakes, windows, headlights, etc.
  - Yet it is still uncomfortable to learn a new one
  - Can you be a great driver if you only ever drive one car?

• And maybe PLs are more like cars, trucks, boats, and bikes

• So are all PLs really the same…

**Are all languages the same?**

Yes:
  - Any input-output behavior implementable in language X is implementable in language Y [Church-Turing thesis]
  - Java, ML, and a language with one loop and three infinitely-large integers are “the same”

Yes:
  - Same fundamentals reappear: variables, abstraction, one-of types, recursive definitions, …

No:
  - The human condition vs. different cultures (travel to learn more about home)
  - The primitive/default in one language is awkward in another
  - Beware “the Turing tarpit”
Functional Programming

Why spend 60-80% of course using functional languages:
– Mutation is discouraged
– Higher-order functions are very convenient
– One-of types via constructs like datatypes

Because:
1. These features are invaluable for correct, elegant, efficient software (great way to think about computation)
2. Functional languages have always been ahead of their time
3. Functional languages well-suited to where computing is going

Most of course is on (1), so a few minutes on (2) and (3) …

Ahead of their time

All these were dismissed as “beautiful, worthless, slow things PL professors make you learn”

• Garbage collection (Java didn’t exist in 1995, PL courses did)
• Generics (List<T> in Java, C#), much more like SML than C++
• XML for universal data representation (like Racket/Scheme/LISP/…)
• Higher-order functions (Ruby, Javascript, C#, …)
• Type inference (C#, Scala, …)
• Recursion (a big fight in 1960 about this – I’m told ☺)
• …

The future may resemble the past

Somehow nobody notices we are right… 20 years later

• “To conquer” versus “to assimilate”
• Societal progress takes time and muddles “taking credit”
• Maybe pattern-matching, currying, hygienic macros, etc. will be next

Recent-ish Surge, Part 1

Other popular functional PLs (alphabetized, pardon omissions)
• Clojure http://clojure.org
• Erlang http://www.erlang.org
• F# http://tryfsharp.org
• Haskell http://www.haskell.org
• OCaml http://ocaml.org
• Scala http://www.scala-lang.org

Some “industry users” lists (surely more exist):
• http://www.haskell.org/haskellwiki/Haskell_in_industry
• http://ocaml.org/companies.html
• In general, see http://cufp.org
**Recent-ish Surge, Part 2**

Popular adoption of concepts:
- C#, LINQ (closures, type inference, …)
- Java 8 (closures)
- MapReduce / Hadoop
  - Avoiding side-effects essential for fault-tolerance here
- …

**Why a surge?**

Dan’s best guesses:
- Concise, elegant, productive programming
- JavaScript, Python, Ruby helped break the Java/C/C++ hegemony
- Avoiding mutation is the easiest way to make concurrent and parallel programming easier
  - In general, to handle sharing in complex systems
- Sure, functional programming is still a small niche, but there is so much software in the world today even niches have room

**The languages together**

SML, Racket, and Ruby are a useful combination for us

<table>
<thead>
<tr>
<th></th>
<th>dynamically typed</th>
<th>statically typed</th>
</tr>
</thead>
<tbody>
<tr>
<td>functional</td>
<td>Racket</td>
<td>SML</td>
</tr>
<tr>
<td>object-oriented</td>
<td>Ruby</td>
<td>Java</td>
</tr>
</tbody>
</table>

*ML*: polymorphic types, pattern-matching, abstract types & modules  
*Racket*: dynamic typing, “good” macros, minimalist syntax, eval  
*Ruby*: classes but not types, very OOP, mixins  
  [and much more]

If we had more time: 
*Haskell*: laziness, purity, type classes, monads  
*Prolog*: unification and backtracking  
  [and much more]

**But why not…**

Instead of SML, could use similar languages easy to learn after:
- OCaml: yes indeed but would have to port all my materials 😞  
  - And a few small things (e.g., second-class constructors)
- F#: yes and very cool, but needs a .Net platform  
  - And a few more small things (e.g., second-class constructors, less elegant signature-matching)
- Haskell: more popular, cooler types, but lazy semantics and type classes from day 1

Admittedly, SML and its implementations are showing their age (e.g., *andalso* and less tool support), but it still makes for a fine foundation in statically typed, eager functional programming
But why not…

Instead of Racket, could use similar languages easy to learn after:

- Scheme, Lisp, Clojure, ...

Racket has a combination of:
- A modern feel and active evolution
- “Better” macros, modules, structs, contracts, ...
- A large user base and community (not just for education)
- An IDE tailored to education

Could easily define our own language in the Racket system
- Would rather use a good and vetted design

But why not…

Instead of Ruby, could use another language:

- Python, Perl, JavaScript are also dynamically typed, but are not as “fully” OOP, which is what I want to focus on
  - Python also does not have (full) closures
  - JavaScript also does not have classes but is OOP

- Smalltalk serves our OOP needs
  - But implementations merge language/environment
  - Less modern syntax, user base, etc.

Is this real programming?

- The way we use ML/Racket/Ruby can make them seem almost “silly” precisely because lecture and homework focus on interesting language constructs

- “Real” programming needs file I/O, string operations, floating-point, graphics, project managers, testing frameworks, threads, build systems, ...
  - Many elegant languages have all that and more
    - Including Racket and Ruby
  - If we used Java the same way, Java would seem “silly” too

A note on reality

Reasonable questions when deciding to use/learn a language:

- What libraries are available for reuse?
- What tools are available?
- What can get me a job?
- What does my boss tell me to do?
- What is the de facto industry standard?
- What do I already know?

Our course by design does not deal with these questions
- You have the rest of your life for that
- And technology leaders affect the answers