Final Exam

As also indicated in forthcoming email:

• This Friday, 12:00-1:00PM
• Intention is to focus primarily on material since the midterm
  − Including topics on homeworks and not on homeworks
  − May also have a little ML, just like the course has had
• You will need to write code and English
• Sample exams were written for two hours; our exam will include a subset of the material

Victory Lap

A victory lap is an extra trip around the track
  − By the exhausted victors (us)

Review course goals
  − Slides from Introduction and (skipped) Course-Motivation

Some big themes and perspectives
  − Stuff for five years from now more than for the final

Please do your course evaluations!!!

[From Lecture 1]

• Many essential concepts relevant in any programming language
  − And how these pieces fit together
• Use ML, Racket, and Ruby languages:
  − They let many of the concepts "shine"
  − Using multiple languages shows how the same concept can "look different" or actually be slightly different
  − In many ways simpler than Java
• Big focus on functional programming
  − Not using mutation (assignment statements)
  − Using first-class functions (can’t explain that yet)
  − But many other topics too

What is the best programming language?

What characteristics would you want in a best programming language?
What is the best kind of car?

What is the best kind of shoes?

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
- Staying dry in the rain

Shoes:

- Playing basketball
- 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 paint 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

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” and “Maslow’s Hammer”

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#, now Java, …)
• Type inference (C#, Scala, …)
• Recursion (a big fight in 1960 about this – I’m told ☹)
• …

Benefits of No Mutation

[An incomplete list]

1. Can freely alias or copy values/objects: Unit 1
2. More functions/modules are equivalent: Unit 4
3. No need to make local copies of data: Unit 5
4. Depth subtyping is sound: Unit 8

State updates are appropriate when you are modeling a phenomenon that is inherently state-based

– A fold over a collection (e.g., summing a list) is not!

Some other highlights

• Function closures are really powerful and convenient…
  – … and implementing them is not magic
• Datatypes and pattern-matching are really convenient…
  – … and exactly the opposite of OOP decomposition
• Sound static typing prevents certain errors…
  – … and is inherently approximate
• Subtyping and generics allow different kinds of code reuse…
  – … and combine synergistically
• Modularity is really important; languages can help

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

Our languages, together

SML, Racket, and Ruby (along with Java) are a useful combination

<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]

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

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
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

Beware Maslow's Hammer

From the syllabus

Successful course participants will:
• Internalize an accurate understanding of what functional and object-oriented programs mean
• Develop the skills necessary to learn new programming languages quickly
• Master specific language concepts such that they can recognize them in strange guises
• Learn to evaluate the power and elegance of programming languages and their constructs
• Attain reasonable proficiency in the ML, Racket, and Ruby languages and, as a by-product, become more proficient in languages they already know

What now?

• Use what you learned whenever you reason about software!
• CSE 401
• CSE 402
• CSE 505

Does PL research (cf. uwplse.org) design new general-purpose languages? Not really; it does cool stuff with same intellectual tools!

Some current UW projects
– 3D-printing tools
– Checker framework
– Rosette
– Language for microfluidics
– Verified software written in Coq (which is quite SML-like)

The End

Thank you for a great quarter!

Don’t be a stranger!

Time for ask-me-almost-anything questions?