CSE341: Programming Languages

Lecture 7.5
Course Motivation

Dan Grossman

Fall 2011

Course Motivation
(Did you think I forgot? ☺)

- Why learn languages that are quite different from Java, C, C++?
- Why learn the fundamental concepts that appear in all (most?) languages?
- Why focus on functional programming?

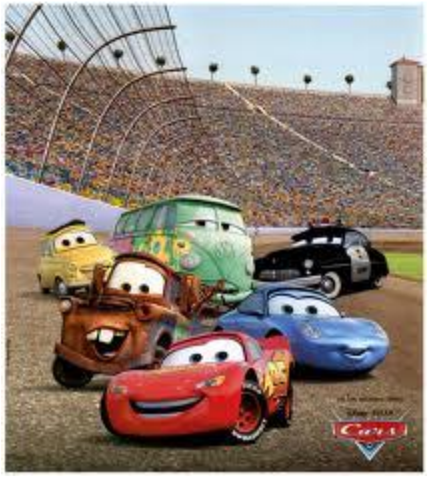
What is the best kind of car?

What is the best kind of shoes?

Cars / Shoes

Cars are used for rather different things:

- Winning the INDY 500
- 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 2004 Honda Civics) work
  - And that the syntax, I mean upholstery color, isn't essential

- A good mechanical engineer really knows how cars work, how to get the most out of them, and how to design better ones

- To learn how cars work, it may make sense to start with a classic design rather than the latest model
  - A popular car may not be a good car for learning how cars work

All cars are the same

- To make it easier to rent cars, it’s great that they all have steering wheels, brakes, windows, headlights, etc.
  - Yet it’s still uncomfortable to learn a new one

- 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, CSE431]
- Java, ML, and a language with one loop and three infinitely-large integers are "the same"
- Beware "the Turing tarpit"

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

A note on reality

Reasonable questions when deciding to use/learn a language:

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

CSE341 by design does not deal with these questions

- You have the rest of your life for that
- And technology *leaders* affect the answers

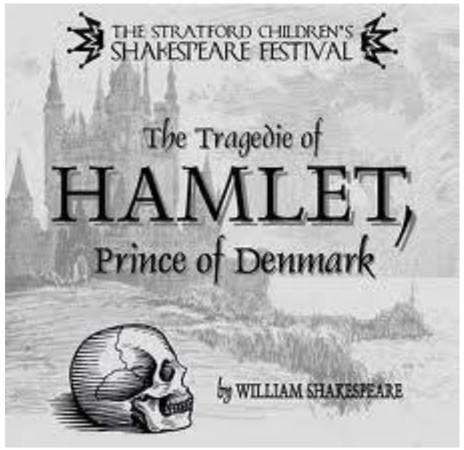
Why semantics and idioms

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 (even in college) centuries later even though:

- The syntax is really annoying to many (yet rhythmic)
- There are more popular movies with some of the same lessons (just not done as well)
- Reading Hamlet will not get you a summer internship

Functional Programming

Okay, so why 70-80% of course with *functional languages*, i.e., languages where:

- Mutation is discouraged
- One-of types via constructs like datatypes
- Higher-order functions are very convenient

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

- Garbage collection (Java didn’t exist in 1995, CSE341 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#)
- Recursion (a big fight in 1960 about this – I’m told ☺)
- …

Somehow nobody notices we are right (20 years later)

- Maybe pattern-matching, currying, hygienic macros, etc. will be next
- “To conquer” vs. “to assimilate”

Recent Surge

- Microsoft: F#, C# 3.0, LINQ
- Scala (Twitter, LinkedIn, FourSquare)
- Java 8 (?), C++ (?)
- MapReduce / Hadoop (everybody)
  - Avoiding side-effects essential for fault-tolerance here
- Haskell (dozens of small companies/teams)
- Erlang (distributed systems, Facebook chat)
- Ocaml (JaneStreet)
- …

Honestly, SML and its implementations are showing their age, but Ocaml and F# are *very* similar and other functional languages also much easier to learn

Why a surge?

My best guesses:

- Concise, elegant, productive programming
- Javascript, Python, Ruby helped break the Java/C/C++ hegemony
  - And these functional languages do some things better
- Avoiding mutation is *the* easiest way to make concurrent and parallel programming easier
- Sure, functional programming is still a small niche, but there is so much software in the world today even niches have room

See separate list of links to “where is functional used” on the course web page

Is this real programming?

- The way we use ML/Racket/Ruby in 341 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, …
  - Functional languages have all that and more
  - If we used Java the same way in CSE341, Java would seem “silly” too

The languages together

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

	dynamically typed	statically typed
functional	Racket	SML
object-oriented	Ruby	Java

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
- There are good general design principles for PLs
- A good language is a relevant, crisp interface for writing software
- Software leaders should know PL semantics and idioms
- Learning PLs is not about syntactic tricks for small programs
- Functional languages have been on the leading edge for decades
  - Ideas get absorbed by the mainstream, but very slowly
  - Meanwhile, use the ideas to be a better C/Java/PHP hacker