Big Ideas for CS 251
~~Theory of Programming Languages~~
Principles of Programming Languages

CS251 Programming Languages
Spring 2016, Lyn Turbak

Department of Computer Science
Wellesley College

Discussion: Programming Languages

Your experience:

- What PLs have you used?
- Which PLs/PL features do you like/dislike. Why?

More generally:

- What is a PL?
- Why are new PLs created?
  - What are they used for?
  - Why are there so many?
- Why are certain PLs popular?
- What goes into the design of a PL?

General Purpose PLs

Java Python Perl
Fortran
ML JavaScript
Racket
Haskell
C/C++ Ruby
CommonLisp

Domain Specific PLs

Excel HTML CSS
OpenGL R
Matlab
LaTeX
IDL
Swift PostScript

Programming Languages: Mechanical View

A computer is a machine. Our aim is to make the machine perform some specified actions. With some machines we might express our intentions by depressing keys, pushing buttons, rotating knobs, etc. For a computer, we construct a sequence of instructions (this is a ``program'') and present this sequence to the machine.

– *Laurence Atkinson, Pascal Programming*

Programming Languages: Linguistic View

A computer language … is a novel formal medium for expressing ideas about methodology, not just a way to get a computer to perform operations. Programs are written for people to read, and only incidentally for machines to execute.

– *Harold Abelson and Gerald J. Sussman*

“Religious” Views

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense. – *Edsger Dijkstra*

It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration. – *Edsger Dijstra*

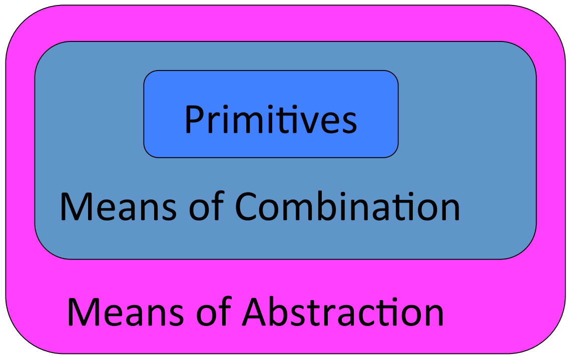
You're introducing your students to programming in C? You might as well give them a frontal lobotomy! – *A colleague of mine*

A LISP programmer knows the value of everything, but the cost of nothing. - *Alan Perlis*

I have never met a student who cut their teeth in any of these languages and did not come away profoundly damaged and unable to cope. I mean this reads to me very similarly to teaching someone to be a carpenter by starting them off with plastic toy tools and telling them to go sculpt sand on the beach. - *Alfred Thompson, on blocks languages*

A language that doesn't affect the way you think about programming, is not worth knowing. - *Alan Perlis*

Programming Language Essentials



Think of the languages you know. What means of abstraction do they have?

PL Parts

Syntax: *form* of a PL

- What a P in a given L look like as symbols?
- Concrete syntax vs abstract syntax trees (ASTs)

Semantics: *meaning* of a PL

- *Static Semantics:* What can we tell about P before running it?
  - Scope rules: to which declaration does a variable reference refer?
  - Type rules: which programs are well-typed (and therefore legal)?
- *Dynamic Semantics*: What is the behavior of P? What actions does it perform? What values does it produce?
  - Evaluation rules: what is the result or effect of evaluating each language fragment and how are these composed?

Pragmatics: *implementation* of a PL (and PL environment)

- How can we evaluate programs in the language on a computer?
- How can we optimize the performance of program execution?

Syntax (Form) vs. Semantics (Meaning) in Natural Language

Furiously sleep ideas green colorless.

Colorless green ideas sleep furiously.

Little white rabbits sleep soundly.

Concrete Syntax: Absolute Value Function

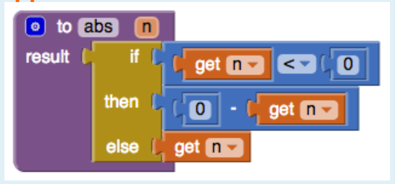
Logo: to abs :n ifelse :n < 0 [output (0 - :n)] [output :n] end

Javascript: function abs (n) {if (n < 0) return -n; else return n;}

Java: public static int abs (int n) {if (n < 0) return -n; else return n;}

Python:
```
def abs(n):
  if n < 0:
    return -n
  else:
    return n
```

App Inventor:

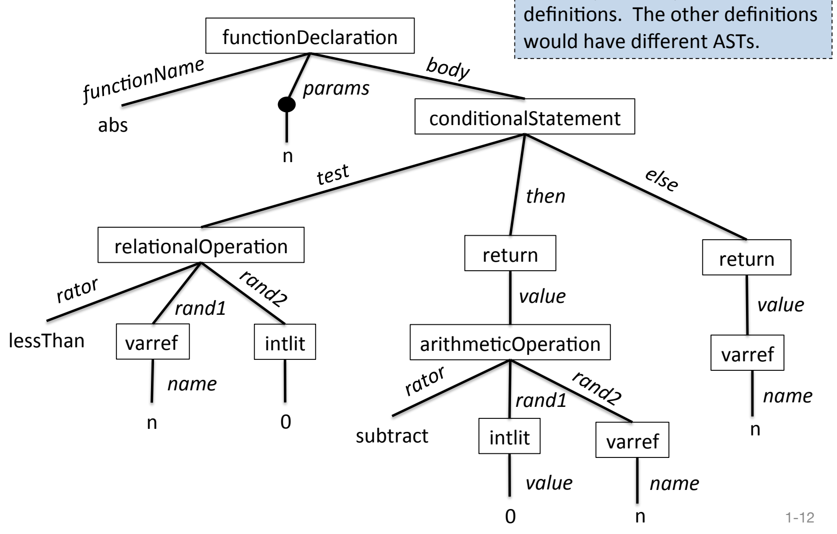


Scheme: (define abs (lambda (n) (if (< n 0) (- n) n)))

PostScript: /abs {dup 0 lt {0 swap sub} if} def

Abstract Syntax Tree (AST): Absolute Value Function

This AST abstracts over the concrete syntax for the Logo, JavaScript, and Python definitions. The other definitions would have different ASTs.



Semantics Example 1

What is the meaning of the following expression?

```
(1 + 11) * 10
```

Semantics Example 2

Suppose `a` is an array (or list) containing the three integer values 10, 20, and 30 in the following languages. What is the meaning of the following expressions/statements in various languages (the syntax might differ from what's shown).

	`a[1]`	`a[3]`	`a[2] = "foo"`	`a[3] = 17`
Java	20	dynamic index out of bounds error	static type error	dynamic index out of bounds error
C	20	returns value in memory slot after a[2]	static type error	Stores 17 in memory slot after a[2]
Python	20	dynamic list index out of range error	stores "foo" in third slot of a	dynamic list index out of range error
JavaScript	20	"undefined" value	stores "foo" in third slot of a	Stores 17 in a[3]
Pascal	20	static index out of bounds error	static type error	static index out of bounds error
App Inventor	10	30	stores "foo" in second slot of a	Stores 17 in third slot of a

PL Dimensions

PLs differ based on decisions language designers make in many dimensions. E.g.:

- *First-class values:* what values can be named, passed as arguments to functions, returned as values from functions, stored in data structures. Which of these are first-class in your favorite PL: arrays, functions, variables?
- *Naming*: Do variables/parameters name expressions, the values resulting from evaluating expressions, or mutable slots holding the values from evaluating expressions? How are names declared and referenced? What determines their scope?
- *State*: What is mutable and immutable; i.e., what entities in the language (variables, data structures, objects) can change over time.
- *Control*: What constructs are there for control flow in the language, e.g. conditionals, loops, non-local exits, exception handling, continuations?
- *Data*: What kinds of data structures are supported in the language, including products (arrays, tuples, records, dictionaries), sums (options, oneofs, variants), sum-of-products, and objects.
- *Types*: Are programs statically or dynamically typed? What types are expressible?

Programming Paradigms

- *Imperative (e.g. C, Python)*: Computation is step-by-step execution on a stateful abstract machine involving memory slots and mutable data structures.
- *Functional, function-oriented) (e.g Racket, ML, Haskell)*: Computation is expressed by composing functions that manipulate immutable data.
- *Object-oriented (e.g. Simula, Smalltalk, Java)*: Computation is expressed in terms of stateful objects that communicate by passing messages to one another.
- *Logic-oriented (e.g. Prolog)*: Computation is expressed in terms of declarative relationships.

Note: In practice, most PLs involve multiple paradigms. E.g.

- Python supports functional features (map, filter, list comprehensions) and objects
- Racket and ML have imperative features.

Paradigm Example: Quicksort

```
void qsort(int a[], int lo, int hi) {
  int h, l, p, t;

  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
          l = l+1;
      while ((h > l) && (a[h] >= p))
          h = h-1;
      if (l < h) {
          t = a[l];
          a[l] = a[h];
          a[h] = t;
      }
    } while (l < h);

    a[hi] = a[l];
    a[l] = p;

    qsort( a, lo, l-1 );
    qsort( a, l+1, hi );
  }
}
```

```
quicksort :: Ord a => [a] -> [a]
quicksort []     = []
quicksort (p:xs) =
        (quicksort lesser)
        ++ [p]
        ++ (quicksort greater)
    where
        lesser  = filter (< p) xs
        greater = filter (>= p) xs
```

Functional Style (in Haskell)

Imperative Style
(in C; Java would be similar)

Pragmatics: Metaprogramming

PLs are implemented in terms of **metaprogams** = programs that manipulate other programs.

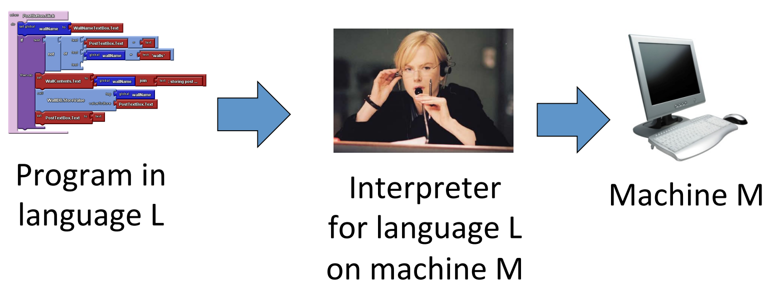
This may sound weird, but programs are just trees (ASTs), so a metaprogram is just a program that manipulates trees (think a more complex version of CS230 binary tree programs).

Implementation strategies:

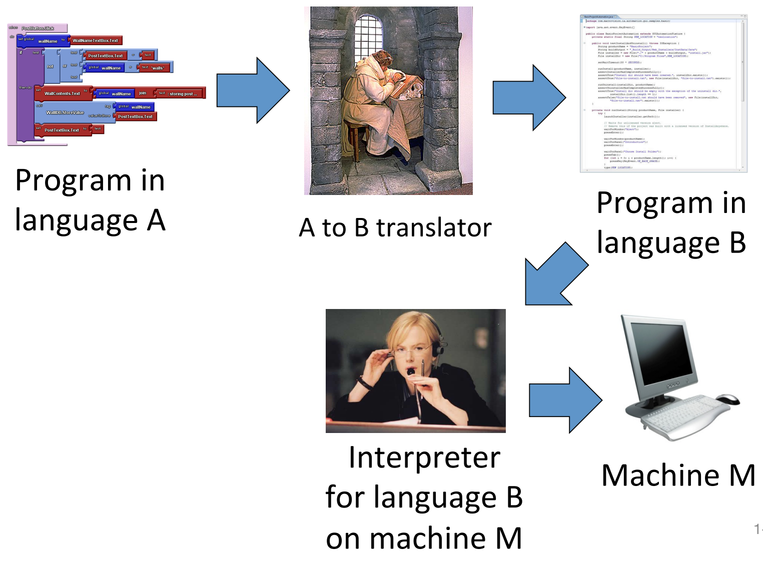
- ***Interpretation***: interpret a program P in a source language S in terms of an implementation language I.
- ***Translation (compilation)***: translate a program P in a source language S to a program P' in a target language T using a translator written in implementation language I.
- **Embedding**: express program P in source language S in terms of data structures and functions in implementation language I.

Bootstrapping puzzles: how do we write a Java-to-x86 compiler in Java?

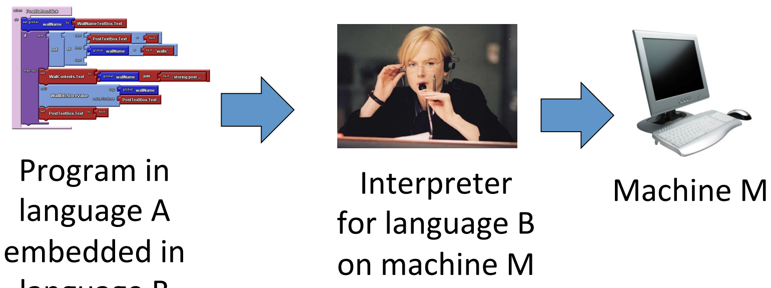
Metaprogramming: Interpretation



Metaprogramming: Translation

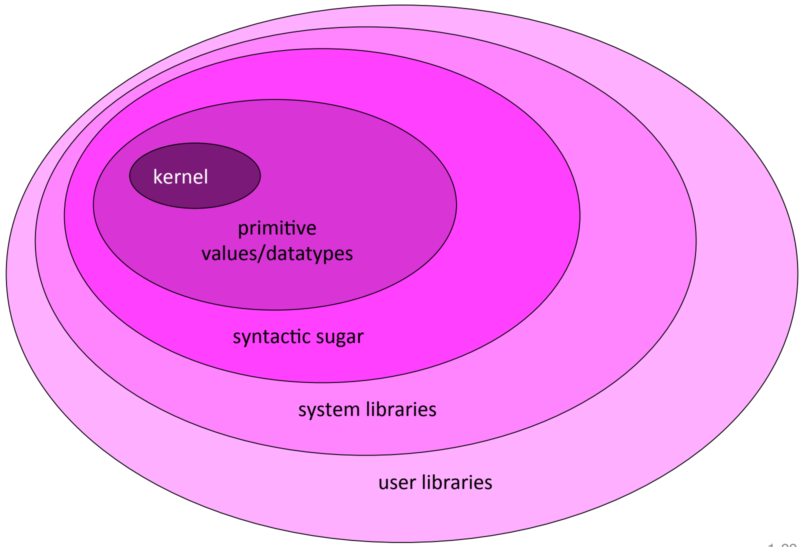


Metaprogramming: Embedding


Program in language A embedded in language B
Interpreter for language B on machine M
Machine M

1-21

Pragmatics: Programming Language Layers



1-22

Why? Who? When? Where? Design and Application

- Historical context
- Motivating applications
  - Lisp: symbolic computation, logic, AI, experimental programming
  - ML: theorem-proving, case analysis, type system
  - C: Unix operating system
  - Simula: simulation of physical phenomena, operations, objects
  - Smalltalk: communicating objects, user-programmer, pervasiveness
- Design goals, implementation constraints
  - performance, productivity, reliability, modularity, abstraction, extensibility, strong guarantees, …
- Well-suited to what sorts of problems?

1-23

Why *study* PL?

- Crossroads of CS
- Approach problems as a *language designer.*
  - *"A good programming language is a conceptual universe for thinking about programming"*
    -- Alan Perlis
  - Evaluate, compare, and choose languages
  - Become better at learning new languages
  - become a better problem-solver
  - view API design as language design
- Ask:
  - Why are PLs are the way they are?
  - How could they (or couldn't they) be better?
  - What is the cost-convenience trade-off for feature X?

1-24

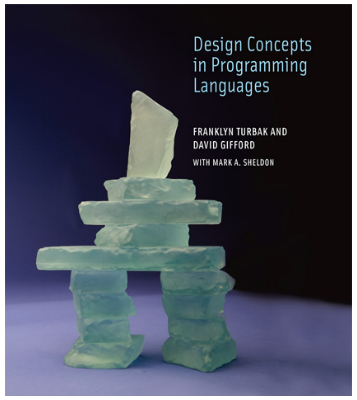
Our rough plan...

- Small scale: essential language dimensions
  - Racket/Lisp, ML, functional programming, historical context
  - core language features
  - interpreters
  - foundations
- Large scale: modularity, etc.
  - Different approaches to modularity, trade-offs
  - OOP vs. FP
- Parallelism and Concurrency
  - Scala

https://cs.wellesley.edu/~cs251

- syllabus
- schedule (still under construction)
- psets (PS1 will be posted by Fri)
- office hours poll
- visit me in office hours this week!
- Mercurial
- CS Linux machines
- wx appliance

PL is my passion!


Design Concepts in Programming Languages
FRANKLYN TURBAK AND DAVID GIFFORD
WITH MARK A. SHELDON

- First PL project in 1982 as intern at Xerox PARC
- Created visual PL for 1986 MIT masters thesis
- 1994 MIT PhD on PL feature (synchronized lazy aggregates)
- 1996 – 2006: worked on types as member of Church project
- 1988 – 2008: *Design Concepts in Programming Languages*
- 2011 – current: lead TinkerBlocks research team at Wellesley
- 2012 – current: member of App Inventor development team