Programming Language Design Principles & Goals

Programmers interact with program through language

- quickly learn programming language (tradeoff with language power)
- quickly express intent, model application domain
- read other people's code, understand intent
- reason about & ensure correctness, debug
- reason about performance trade-offs, ensure good performance
- modify/extend program

Tools interact with program through language

- compilers quickly analyze, optimize, and translate
- debuggers, program understanding tools
- machine program generators

Goals

Want language that makes these interactions easy as possible

W.r.t. programmers:

- learnable (simple, "natural", no special cases)
- expressive, easy to write
  - for particular application domain? or in general?
- readable, understandable; no "clever" encodings
- simple, clear feature interactions;
  no undefined combinations;
  no error-prone features;
  early, extensive automatic error checking
- simple, clear, efficient implementation model
- what helps extensibility, modifiability?

W.r.t. tools:

- easy to automatically reason about & optimize
- easy to quickly extract important implementation info
- unambiguous
- regular enough to generate by machine

Design principles

Simplicity, regularity, orthogonality

- identify a small number of key concepts
- let them be combined in all combinations w/o restrictions

Provide abstraction mechanisms that enable
writing & checking (parameterized) code once,
reusing/instantiating code many times in wide range of ways

Be able to represent model of application as directly as possible

Fully define the semantics of programs, independent of
underlying implementation environment

Avoid error-prone constructs,
provide automatic checks on consistency for allowed
constructs