Lecture 1: Overview

CSC 131
Fall, 2012

Kim Bruce

Do Languages Matter?

- Why choose C vs C++ vs Java vs Python ...
- What criteria to decide?
- Impact on programming practice
- SIGPLAN Education Board documents

Why Article

- Learn widely-applicable design & implementation techniques
- Creating new domain specific languages or virtual machines
- Learning new computational models and speeding language learning
- Choosing the right language

Provide Abstractions

- Data Abstractions:
  - Basic data types: ints, reals, bools, chars, pointers
  - Structured: arrays, structs (records), objects
  - Units: Support for ADT's, modules, packages
- Control Abstractions:
  - Basic: assignment, goto, sequencing
  - Structured: if...then...else, loops, functions
  - Parallel: concurrent tasks, threads, message-passing

PL's & Software Development

- Development process:
  - requirements
  - specification
  - implementation
  - certification or validation
  - maintenance
- Evaluate languages based on goals

Goals of Some older PL's

- Languages & their goals:
  - BASIC - quick development of interactive programs
  - Pascal - instruction
  - C - low-level systems programming
  - FORTRAN, Matlab - number-crunching scientific
- What about large-scale programs?
  - Ada, Modula-2, object-oriented languages

PL Choice

- Languages designed to support specific software methodologies.
- Language affect way people think about programming process.
- Hard for people to change languages if requires different way of thinking about process.

Minimum Requirements

- Natural
- Implementable
- Efficient
- Reliable
- Maintainable

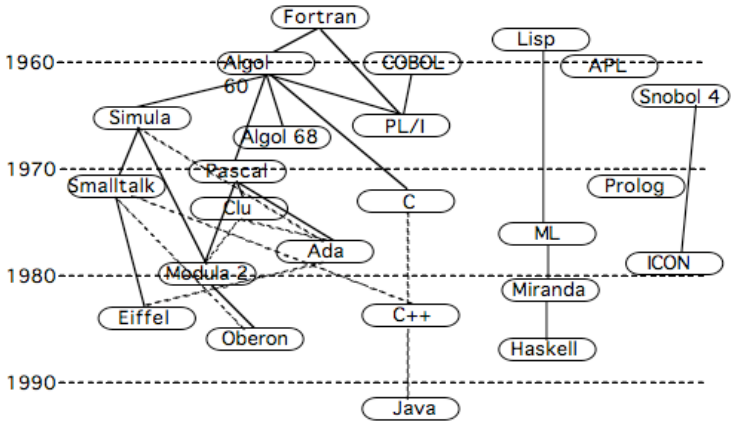
Paradigms
or whatever you want to call them

- Not crisp boundaries
  - Procedural
  - Functional
  - Logic or Constraint-programming
  - Object-oriented

History of PL's

- Machine language
  ⇒ Assembly language
  ⇒ High-level language
- Single highly-trained programmer
  ⇒ Teams of programmers

History of PLs



Course Goals

- Upon completion of course should be able to:
  - Quickly learn programming languages, & how to apply them to effectively solve programming problems.
  - Rigorously specify, analyze, & reason about the behavior of a software system using a formally defined model of the system's behavior.
  - Realize a precisely specified model by correctly implementing it as a program, set of program components, or a programming language.

Administrivia

- Web page at
  - http://www.cs.pomona.edu/classes/cs131/
- Text by Mitchell: being revised now
- If needed, get account at
  - https://www.dci.pomona.edu/account-bin/account_request.php
  - Do it early!! Currently no systems manager.

Administrivia

- Prerequisite:
  - CS 81, Computability and Logic
    - Use computability extensively, esp. at beginning
    - Formal grammars when talk about parsing
- Homework
  - Generally due every week on Thursday night.
    - Posted on Friday
  - All homework must be turned in electronically
    - Prefer Tex'ed, but can scan in if want to write up by hand
    - *... but must be legible!!*

On-Line Discussions

- Will be on Piazza
- You will receive an invitation later this week.
  - Do not throw it away!
- You can ask and answer questions on-line.
  - TA's and I will monitor and respond.

Grace

- New language designed for teaching novices
  - Under development at Pomona, Portland State, and Victoria University, Wellington, NZ
  - Several published papers, partial implementations
- Goal: Integrate current ideas in programming languages into a simple, general-purpose language aimed at novices.

Why New Language for Novices?

- Most popular languages too complex & low-level.
- Complexity necessary for professionals, but ...
- "Accidental complexity" of language can overwhelm "inherent complexity".
- Minimize language complexity so can focus on programming/design complexity.

Existing Languages Woefully Out-of-date

- C (1972), C++ (1983), Python (1989), Java (1994)
- History of pedagogical languages:
  - Basic, Logo, Pascal
  - *... but not recently!*
  - *Miniworlds different: Alice, Karel the Robot, Greenfoot*

Java Problems

- **public static void** main(String [] args)
- Primitives *versus* objects, "==" *versus* "equals"
- Flawed implementation of generics
- Static *versus* instance – on variables & methods
- float *versus* double *versus* int *versus* long

Python Problems

```
>>> class aClass:
    """A simple example class"""
    val = 47
    def f(self):
        return 'hello world'

>>> x = aClass()
>>> x.value = 17
>>> x.val
47
>>> x.f()
'hello world'
```

disappearing `self`?

uncaught typos

no information hiding

Fine for scripting, but not large-scale software development

What if we could have:

- Low syntactic overhead of Python, *but with*
  - information hiding
  - consistent method declaration & use
  - required variable declarations
  - optional (& gradual) type-checking
  - direct definition of objects
  - first-class functions

21

Hello World in Grace:

```
print "hello world"
```


22

Objects

Consistent indenting is required! But no semicolons.

```
def mySquare = object {
      var side := 10
      method area {
           side * side
      }
      method stretchBy(n) {
           side := side + n
      }
}
```

Defaults: instance variables and constants are confidential (protected), methods are public

Annotations can override the defaults

23

Objects contain declarations

- definitions:
  - def x:Number = 17
- variables:
  - var y: String := “hello”
- methods:
  - method m(w:Number,z:String) -> Done {...}
- types:
  - type Point = {x -> Number
    y -> Number ...}

Typed Objects

```
type Square = {                                  like Void
      area -> Number
      stretchBy(n:Number) -> Done
}

def mySquare:Square = object {
      var side:Number := 10
      method area -> Number {
            side * side
      }
      method stretchBy(n:Number) -> Done {
            side := side + n
      }
}
```


25

Classes

- Classes take parameters and generate objects

```
class aSquare.withSide(s:Number) -> Square {
      var side:Number := s
      method area -> Number {
          side * side
      }
      method stretchBy(n:Number) -> Done {
           side := side + n
      }
      print "Created square with side {s}"
}
```

Type annotations can be omitted or included

26

Or Object w/Factory Method

```
def aSquare = object {
   method withSide(s:Number) -> Square {
      object{
         var side:Number := s
         method area -> Number {
            side * side
         }
         method stretchBy(n:Number)-> Done {
            side := side + n
         }
         print "Created square with side {side}"
      }
   }
}
```

What is type of aSquare?

27

Blocks

- Syntax for anonymous functions

```
def double = {n -> n * n}
double.apply(7)  // returns 49
// block is implicitly object w/apply method

def nums = aList.from(1)to(100)
def squares = nums.map {n -> n * n}
```

function

multipart method names

Blocks can take 0 or more parameters

28

Blocks

- Blocks make it simple to define ne structures" as methods

block, evaluated repeatedly

```
while {boolExp} do { someStuff }

squares.forEach {n ->
        if (n.isEven) then {print n}
}
```

boolean expression, evaluated once

Parentheses can be dropped if argument bounded by {} or ""
No parens needed for parameterless methods

29

Running Grace

- Compilers generating C or Javascript
  - Instructions at http://gracelang.org/applications/minigrace/
  - Choose web-based unless you are a systems hacker.

Grace on the Web

- Go to:
  - http://www.cs.pomona.edu/~kim/minigrace/
  - Either paste in program or (better) use button "add a file" then "Choose file" to upload
  - "Go" button will compile and execute code.
  - Error messages not great yet.
  - Simple test code in pop-up menu to far right of "go"