Language Choice and Object-First Approaches in Teaching CS1

Presented by David Riley
Language choice papers

- Fleck: Prolog as the first programming language
- Loui: In praise of scripting: Real Programming Pragmatism
- Sanders & Langford: Students’ perceptions of Python as a first programming language at Wits
- Radenski: “Python First”: a lab-based digital introduction to computer science
Fleck (Prolog as first)

Contends that Prolog is a good language for starting CS students off

- Designed by Dijkstra, based on solution of constraints; students learn to generate provably correct programs and learn the principles of
- Is interpreted, so the compile-swear-edit-recompile-run-swear-debug cycle of C and Java is shortened somewhat; “live” examples are possible in interpreter
- Good environments already exist for free
- Recursion is simple and natural
- Decent built-in simple I/O facilities (line-based prompts)
Fleck, cont’d.

- Also cites a number of problems with Prolog
  - Not particularly procedurally-oriented; this is cited as a mixed benefit, as students don’t become focused on the raw mechanics, but it leaves them somewhat less prepared for procedural languages
  - Prolog has a number of aspects that are challenging for novices, such as rather arcane substitutes for negation (“not identical” and “not unifiable”)
- Nonetheless, claims good results over 6 years
- Paper seems overly focused on provable correctness, which while laudable, is dubiously practical
- Python satisfies most of Prolog’s proclaimed benefits over C/Java (interpreted, simple I/O, etc.)
Loui (In praise of scripting)

- Argues that scripting languages are superior to Java for teaching CS1
  - “Scripting” languages are somewhat ambiguously defined, but his general criteria are “low I/O specification costs” (i.e. simple I/O), use of implicit coercion and weak typing, widespread use of associative arrays, regexes and other heuristics
- Argument based on ease of rapid prototyping and simpler syntax
  - Enables simpler algorithmic design without mumbo-jumbo
  - Some might argue that these are problematic aspects of scripting languages as well
- Recommends Gawk (really?), JavaScript, PHP or ASP, but acknowledges that Python and Ruby are popular
Sanders & Langford (students’ perceptions of Python)

- Wits started with Scheme, did partial move to Python in ‘05, used Python exclusively in ‘06/’07
  - New programmers took well to Scheme, but existing programmers generally did not like the idea of learning a new language, especially one they felt was not a “real world” language
  - Scheme did not effectively close performance gap
- Most students generally thought Python was a good first year language
  - Has the benefit of being a “real world” language
- Some students with experience still resent having to learn a new language
Radenski ("Python First")

- Many students with preliminary skills in Java/C++ succeed in Java-based classes, but novices tend to fare poorly
  - Cites retention rates as low as 50% for Java-based CS1
- Java necessarily starts students off with the rather complex Java object model (start with classes, then move quickly to methods, static members, etc.)
  - Earlier CS1 classes used “teaching” languages, such as Basic, Pascal, LOGO
  - Made learning “fun”
  - Not appropriate languages for commercial development
Advantages of Python

- Good hybrid of teaching and commercial language
- Easy to start procedurally to introduce generic programming topics, then gently introduce objects
- Handily supports imperative, functional and object-oriented programming

Author contends that object-first teaching is generally difficult for both instructor and student
- Some object-first books postpone control statements (if, while, etc) to the second half of the book

Still recommends Java for CS2, after groundwork is laid for objects in Python CS1
Radenski, cont’d.

- The author’s lab-based course in Python is presented
  - Includes mainly self-guided labs available on www.studypack.com
  - Example: Lab B in Graphics and Interfaces has students generate fractals
  - 37 required labs, 25 optionals
    - For a 14-week semester, averages 2.6 required labs per week, 1.8 optional
Object-First papers

- Hu: Just say “A class defines a data type”
- Gries: A principled approach to teaching objects first
- Goldwasser and Letscher: Teaching an object-oriented CS1 - with Python
- Raymond et al, Lister: Research perspective on the objects-early debate
Hu (a class defines a data type)

- Author contends that introductory CS classes do students a disservice by inadequately defining what a data type is
  - **Author asserts that a correct definition should be “consistent with the following: a data type characterizes how a set of entities is internally represented and algorithmically manipulated”**
  - **ADTs and objects extend naturally from this definition**
    - For example, + is intuitive on primitives and on string objects, but many students do not stop to think about the differences in the underlying operation

- Author also contends that given this definition of a data type, a class simply defines a data type
  - **Terms like “template” and “blueprint” suggest how a class can be viewed, but not what it is or does**
Hu, cont’d.

- Author not particularly impressed with objects-first
  - Often results in “object monopoly”, where object orientation is taught at the expense of algorithmic design
- Advocates teaching good programming practices first (including the role of datatypes in problem solving) without explicitly addressing objects
  - Objects might be addressed as a type of user-defined data type, perhaps like an elaborate struct
Gries (a principled approach to OO first)

- Starts with important pedagogical principles for teaching CS1 before getting to teaching objects
  - **Reveal the programming process**
    - Allow resubmissions of the first few projects so students don’t get discouraged by mistakes
  - **Teach skills, not just knowledge**
    - Recommends mentoring for this, which is not particularly feasible for large classes
  - **Present concepts at appropriate abstraction levels**
    - Stop being so computer-centric: don’t talk about variables as bits of memory
    - Quotes Algol 60 spec (bonus!)
      - “The purpose of the algorithmic language is to describe computational processes...”
      - “A variable is a designation given to a single value.”
Gries, cont’d.

More principles

- **Order material to minimize introduction of topics without explanation (i.e. minimize “hand-waving”)**
  - “Teaching Java using either applications or applets violently violates [this principle]”
  - Suggests using an IDE for Java that allows for immediate execution of statements, not programs (a la an interpreted language)

- **Use unambiguous, clear, and precise terminology**
  - e.g. avoid talking about pointers and references, if possible, don’t confuse parameter and argument, etc.

- **Introduce names for your entities**
  - Instead of talking about pointers, talk about names; this is much more concrete for non-programmers and is semantically very close
Gries, cont’d

- Introduces “file folder” metaphor for objects
  - Filing cabinet for class: cabinet label denotes what’s inside
  - Manila folder for object; name on tab is “name” of (address of) object (e.g. this is folder a0)
    - Folder has class label in upper right
    - Contents (fields, methods) inside

Images from Gries, A Principled Approach to Teaching OO First, pp.33-34, SIGCSE 2008
Gries, cont’d

- Manila folder model extends for inheritance
- More details in paper for talking concretely about references to objects using names as references (i.e. “Mary” can be in folder a1 in the drawer)
- Paper also has useful course outline

Images from Gries, A Principled Approach to Teaching OO First, pp.33-34, SIGCSE 2008
Goldwasser & Letscher
( teaching OO CS1 with Python )

- Starts with covering the great debate over when to teach OO programming
  - Everyone agrees that teaching OO is challenging, wherever it is
- Java is complex and beats the student over the head right out of the gate with high-level OO concepts
  - Situation has necessitated development of many pedagogical tools (such as BlueJ, DrJava, etc.) to make the task reasonable
- Python bypasses most of the initial complexities of Java and C++ while allowing for good OO teaching
Goldwasser & Letscher, cont’d.

- Python allows for greater emphasis on core principles with less focus on pesky syntax issues
- One can make an awareness of objects clear from the beginning without jumping in head-first
- Get started by using objects rather than creating them
  - Python has lots of built-in classes which students can manipulate (strings, lists, dictionaries, etc)
- Python has a relatively low syntactic “magic” quotient
  - No “main” routine necessary; program just starts from the first line
  - Certainly none of Java’s “main” method which must be a static member of the class
- All that, and it’s interpreted live!
Example intro to OO in Python:

```python
>>> groceries = list()
>>> groceries.append('bread')
>>> groceries.append('milk')
>>> groceries
['bread', 'milk']
>>> groceries.append('cereal')
>>> groceries.sort()
>>> groceries
['bread', 'cereal', 'milk']
```

Consistent object model

- All data types are classes (no primitives)

Simple transition to Java/C++
Raymond et al, Lister (research perspectives on OO-early)

- This is much longer than I thought and I only got through 2 of 20 pages last night before resorting to skimming (so *caveat lector*)

- Claim 1: OO is more difficult to learn than imperative
  - *Students still have lots of trouble with imperative*
    - “Fragile” knowledge of basic concepts and poor problem-solving skills
  - *OO, however, does not seem to simplify things*
    - Poor conceptual understanding of objects
    - Poor understanding of “object” and “class”
Lister et al, Raymond, cont’d.

- Claim 2: Students who learn OO-first don’t learn algorithmic design well
  - Little evidence; studies suggest little difference
- Claim 3: Students who learn imperative first have trouble with OO
  - Very little research; anecdotal evidence seems to suggest this, but it’s thin
- Claim 4: Successful student learning requires match between language and teaching paradigm
  - Many teachers attempting to teach OO in a procedural way
  - No evidence that this is actually harmful; no empirical studies, either
Lister *et al*, Raymond, cont’d.

- Most of rest of paper focuses on some research, but mostly anecdotal evidence for benefits of various approaches
  - Lots of personal accounts from instructors
  - Mainly consists of emails from mailing lists
- Ultimately not that conclusive, but insightful as to reasons for the struggle over OO-first