

## Motivations for Studying Programming Languages

The syllabus discusses direct motivation for taking CS 334 and our goals for the semester. Here is a short essay on two of my personal motivations for studying programming languages. One is an extremely practical reason, and one is much more philosophical.

### 1. The Right Tool for the Job

When I was in graduate school, one of the qualifying exams to be admitted to candidacy as a Ph.D. student was a programming exam. Each year's class of incoming students received a detailed problem specification. We then each individually had to design and implement software solution, and demonstrate its application to the problem. A major requirement for the exam was documenting the software through a user manual, architectural design documentation, and inline documentation within the source code.

The examination was unpleasant—due to time constraints and the significance of the outcome, it was something like the hardest final project I'd ever worked on combined with crunch week at a software company. Actually, it was more like *twice* as unpleasant as those experiences combined.

What was much more entertaining was comparing solutions with the other students after the exam was over. The year that I took the exam<sup>1</sup>, the problem involved searching a large database for certain kinds of patterns. Parsing the database from the file format was a significant systems problem. Recognizing the patterns, which stretched across multiple records, then presented a hard design and algorithmic challenge. The table below summarizes the solutions by four of the students.

Developer	Language	Lines	Dev. Time	Execution Time
Greg	Prolog + Perl	253	3 days	10 hours
Matt	Scheme	1730	4 days	2 hours
Lisa	Java	10881	6 days	1.5 hours
Tom	C++	6208	7 days	20 min

Greg used Perl to reformat the database file and then a tiny Prolog program to search for the patterns. Because these languages are well-suited to those problems, he needed very little code and spent little time writing and debugging his program. His program ran very slowly, but since it only had to run once, that wasn't a problem.

Matt spend three days writing a special purpose programming language in Scheme and then about half a day writing a program in that new language to solve the actual problem. His program ran five times faster than Greg's, but took longer to develop because he had to debug the entire interpreter first. Lisa and Tom were well versed in Java and C++ from their other CS courses, so they used those languages. It took them about the same amount

---

<sup>1</sup> Since many of the people who took this test are now professors, I've changed the names to protect the innocent (and guilty) and combined stories from two different years the exam was given to make a point.

of time to implement their solutions. Tom was almost done at 5 days, but spent two additional days debugging a memory corruption problem. His solution was substantially faster than everyone else's and he was able to use a feature called templates to eliminate redundant code from his project, making the final source code much shorter than Lisa's.

I think Greg made the best choice here. That doesn't mean that Prolog and Perl are better languages than the others, or that you should use them for most projects. What he did was choose the right tool for the specific job at hand. Had performance been the primary concern, Tom's C++ would have clearly been the best. Had portability, concurrency, and working on a large team been the major concerns, Lisa's Java might have been the right tool.

## 2. Philosophy

If I go anywhere in time and space, in my role as a computer scientist I would choose to Princeton University and its Institute for Advanced Study from 1930–1950. Take that trip as a thought experiment and look around. In the lecture hall, Prof. Einstein is speaking on his theory of relativity. Researcher Claude Shannon is a theory of information in his office, and in the next building the beautiful mind of PhD student John Nash is developing game theory. Most important to us, a new field called computer science is emerging as group of mathematicians including Church, Turing, Gödel, and von Neumann explore *languages* for expressing *computation*. When modern computers are later invented, these will be called programming languages.

Today, on the first day of CS 334, you already know several languages for computation, including English, algebra, calculus, geometry, and Java. The Princeton group didn't have Java, but they did have all of the others, as well as a few more mathematical languages. Considering these different languages, you might ask some of the same questions as the Princeton group and those who followed in their footsteps:

- Are all languages equal? How small can a programming language be? [*Church, Turing, Felleisen, Sussman, Steele*]
- Can we prove that a program is correct before we run it? [*Gödel, Curry, Howard, Dijkstra*]
- What is the difference between a program and its data? Can programs write programs? [*von Neumann, Perlis, Sussman*]
- How does the structure of a machine affect the computations it can perform? [*Turing, von Neumann*]
- Are there functions beyond mechanical computation? [*Turing, Gödel*]

Thinking about these fundamental ideas will enhance your abilities as a computer scientist, a software designer, and a programmer. I hope that they will also give you a glimpse of the awesome frontier of knowledge at the intersection of computer science, mathematics, cognitive science, philosophy, and physics. Although it seems far from our everyday experience of writing Java and C++ programs, the nature of reality is tangled with the principles of programming languages. It is no coincidence that, for example, von Neumann's and Gödel's works also contribute to quantum mechanics and general relativity, or that the early computer scientists were rubbing elbows with Einstein, Nash, and Shannon.