

Class 01: Introduction to the Course

Held: Monday, January 22, 2007

Summary: Today we begin the class by considering the key issues we will discuss this semester.

Related Pages:

- EBoard.

Assignments

- Do Wednesday's readings.
- HW1: Intro Survey.

Notes:

- Upcoming talks: Prof. Meehan on Vocation, Thursday at 11:00 a.m. Dr. Stone on Life, Thursday at 4:15 p.m.

Overview:

- What is a programming language?
- How should we study programming languages?
- Why study programming languages?
- Course details.

Background

- I believe in beginning my courses with a discussion of the topic of study. In particular, what are we studying, why do we study that topic, and how do we study that topic.
- The following are my notes on the topic. I'd prefer if you waited to read them until we've had some discussion.
 - Those of you who've been in my classes before probably have a good sense of how to balance reading and discussing.

What is a programming language?

- Surprisingly, different people (even different programming language theorists) seem to have different definitions of "programming language".
- **What is your definition?**
- Here are some garnered from notes and books. Are they all the same? What accounts for the differences?
 - *Hoare* (in part): A tool to aid the programmer.

- *Louden*: A notational system for describing computation in machine-readable and human-readable form.
- *Reade* One, rather narrow, view is that a program is a sequence of instructions for a machine. We hope to show that there is much to be gained from taking the much broader view that programs are descriptions of values, properties, methods, problems, and solutions. The role of the machine is to speed up the manipulation of these descriptions to provide solutions to particular problems. A programming language is a convention for writing descriptions which can be evaluated.
- *Rebelsky*: A notation for expressing algorithms so that they may be understood by humans and processed by machines.
- *Stansifer*. The purpose of language is communication. Human beings use natural languages to communicate among themselves. Programming languages are used to communicate with literal-minded machines.

Why study programming languages?

- Different languages provide significantly different perspectives on how to express algorithms, data structures, and control. They may also provide different features for developing algorithms. By visiting these different perspectives, you will find new ways to express your ideas (and perhaps even find ways to express ideas that were previously too difficult to express).
- Most of us will need to write programs in the future. Exposure to a wider variety of paradigms, principles, and languages
 - gives you tools and methodologies for choosing which language to use for a project;
 - increases your ability to learn new languages;
 - often improves your programming ability in your language of choice.
- Most major software packages end up including some form of programming language. By studying languages, you will be better able to design your own languages.
 - The Tcl crowd would tell you that everyone should just include Tcl in their package.
 - The FSF crowd might tell you that Scheme would be even better.
 - These days, the Ruby and Python crowds might also argue for their languages, although both are a bit bigger than Tcl or Scheme.

How should we study programming languages?

- There are many different perspectives on how best to study programming languages. At times, it seems every course (or at least every text) promotes a different perspective.
 - Each perspective has certain advantages and disadvantages.
- Some perspectives are language-based.
 - One might cover a new language every week. This provides a wide introduction to languages and gives some basis for comparison. At the same time, it makes it difficult to cover languages and issues in any depth.
 - One might cover a new language ever month. This permits greater understanding of the particular languages covered, but often at the expense of higher-level understanding.
 - One might cover one language that is so complex and multifaceted that it introduces a wide variety of topics.
- Some perspectives are implementation-based.

- One might develop interpreters for a number of languages. This enhances understanding of many design issues and their advantages and disadvantages. At the same time, implementation is difficult and can detract from learning of more general issues.
- One might develop an interpreter or compiler for a single language that expresses concepts from multiple paradigms. Again, this helps illustrate a number of key concepts and allows you to experiment with variations on a theme, but has the potential to miss many important issues.
- Some perspectives are semantics-based.
 - One might investigate a single form of semantic notation and develop semantics for a number of languages (or for one large language).
 - One might investigate a number of forms of semantic notation and develop semantics for a small language.
- Some perspectives are concept-based.
 - One might study the various paradigms and design possibilities. This is the most general strategy, but runs the risk of ignoring more concrete details.
- We will use a variety of strategies. Our primary focus will be on general language issues, but we will also investigate some languages and some particular details of individual languages.
- We will ground our work on readings from the primary literature.

Course Organization

- Read the Course front door.
- Scan the Course at a Glance.