# Homework 1: Overview and Picking Languages

See Webcourses2 and the syllabus for due dates.
In this homework you will get an overview of programming languages [Concepts] [EvaluateModels]. You will also become prepared to participate in a class discussion to decide on what we will study this semester.

## General Directions

This homework must be done in a group of 4 to 6 people; ideally the group would contain 5 people. Due to time limitations (for the presentations), we will not accept groups with fewer than 4 people.
Answers to English questions should be in your own words; don't just quote from articles or books.

## What to turn in

For problems that require an English answer, enter your answer into Webcourses2 (by pasting it into the assignment's answer box) for the "assignment" corresponding to that problem.

## Problems

1. [Concepts] [EvaluateLanguages] In a group, you are to choose one of the topics below, and prepare a report and a brief (4 minute) classroom presentation on it. (See details after the list of topics.) You must sign up for a group that was created by the instructor on Webcourses2.

   The topics all concern what programming models and languages we should study this semester. You can pick one of the following, or after consulting with the instructor, a topic of your own choosing. Note that in the list of topics there are both general topics (programming models as a whole) and specific programming languages. Your group can choose either kind of topic.

   **Multi-paradigm Languages** Multi-paradigm languages attempt to support many different programming models (paradigms) in a single programming language. See Floyd's paper "The Paradigms of Programming" [Flo79] for some background on paradigms. The introductory material in the book *Concepts Techniques and Methods of Computer Programming* [VH04] has a modern discussion of these ideas. (See `http://mitpress.mit.edu/books/chapters/0262220695chap1.pdf` for an online copy of chapter 1.)

   - **Lua** Lua [Lua] is a multi-paradigm scripting language. It is dynamically typed.
   - **Oz** Oz [Moz, VH04] is a multi-paradigm programming language. It is dynamically typed and has support for functional programming, lazy execution, message passing, relational programming, and other programming models such as object-oriented programming.
   - **Scala** Scala [Sca] is a statically-typed object-oriented language that also has many features of functional programming languages. It has libraries that provide support for the message passing (actor) model.

   **Functional Programming Model** This is programming with immutable data and without variable assignment which uses functions as data [BW88, Gra04, Hud89, Hug89, SF89].

   - **F#** F# [FSh] is a functional (and object-oriented) programming language that works in Microsoft's .NET programming environment.
   - **Haskell** Haskell [Has] is a statically-typed, lazy, functional programming language.
   - **JavaScript** JavaScript, or rather ECMAScript [ECM], is a scripting language for the web. However, it has features that allow it to be used as a dynamically-typed functional programming language.
   - **OCaml** OCaml [OCa] is a statically-typed higher-order functional programming language that also has object-oriented features.

**R** R [RLa] is a language for statistical analysis, which can also be seen as a functional programming language.

**Racket** Racket [Rac] is a variant of Scheme with many additional extensions. It has an extensive programming environment.

**Scheme** Scheme [ASS96, CE91, Sch] is a statically-scoped variant of Lisp, and can be used as a functional programming language. It has an advanced macro system.

**Message Passing Model** In the message passing model, also known as the actor model [Agh91, HBS73], programs are composed of actors that have local state and can send and receive messages. This model directly supports client-server programming. Both Scala and Oz support this model to some extent.

**Erlang** Erlang [Erl] is a language for building parallel, distributed, and fault-tolerant systems. It is organized around the message passing model and also has some features of the functional and relational programming models.

**Distributed Programming Model** In the distributed programming model [BST89, Cri91], programs are spread over physically distinct computers, and do not communicate by shared memory. Erlang also supports this model.

**Orc** Orc [Orc] is a coordination (or scripting) language for distributed and concurrent systems.

**Declarative Concurrent Programming Models** The declarative concurrent programming model [VH04, Chapter 4] adds parallelism (threads) to the declarative programming model; that is, programs use immutable data. Oz, some Haskell extensions, and dataflow languages also support this model.

**Clojure** Clojure [Clo] is a dynamically-typed, functional language that runs on the JVM. If features support for multi-threading.

**Relational Programming Model** In this programming model, also known as logic programming, one specifies relations and lets the programming language implementation search for answers to queries. It generalizes database queries and allows for rapid prototyping. Oz also supports this model.

**$\lambda$Prolog** $\lambda$Prolog [Lam] is a higher-order logic programming language. It has a static type system and a module system.

**XSB** XSB [XSB] is a logic programming language with tabling features, which are useful for memoization and modeling deductive databases.

**Object-Oriented Programming Model** The widely-used object-oriented programming model [AC96, Coo91, GHJV95] features mechanisms to encapsulate data (classes and objects) as well as mechanisms to implement code by stating how it differs from other code. Java, C#, and C++ are well-known examples of this model which we will *not* be studying.

**Ruby** Ruby [Rub] is a language that is similar to Smalltalk, but with a more standard syntax. It also has features to support functional programming.

**Smalltalk** Smalltalk [Sma] is a dynamically-typed, pure object-oriented language that also has some features reminiscent of functional programming. The language has an unusual syntax and programming revolves around an interactive workspace.

Once you have selected your topic and formed your group, you and the group should answer the following questions.

(a) (10 points) What are the claimed advantages or benefits of the language or paradigm?

(b) (5 points) What are the ways in which the language or paradigm differs from languages that you are familiar with (such as C, C++, and Java) or paradigms that you are familiar with (such as object-oriented programming)?

(c) (10 points) What documentation and implementations are available for the language and paradigm? Are they free? Do they work on both PCs and Macs?

(d) (10 points) What would you and the class learn by studying this language or paradigm?

(e) (5 points) Your group's 4-minute presentation on the above items is worth 5 points.

# Points

This homework's total points: 40.

# References


[AC96] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer-Verlag, New York, NY, 1996.

[Agh91] Gul Agha. The structure and semantics of actor languages. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, volume 489 of *Lecture Notes in Computer Science*, pages 1–59. Springer-Verlag, New York, NY, 1991.

[ASS96] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. McGraw Hill, Cambridge, Mass., second edition, 1996.

[BST89] Henri E. Bal, Jennifer G. Steiner, and Andrew S. Tanenbaum. Programming languages for distributed computing systems. *ACM Computing Surveys*, 21(3):261–322, September 1989.

[BW88] Richard J. Bird and Philip Wadler. *Introduction to Functional Programming*. International Series in Computer Science. Prentice-Hall, New York, NY, 1988.

[CE91] William Clinger and Jonathan Rees (Editors). Revised$^4$ report on the algorithmic language Scheme. Available from `ftp://ftp.cs.indiana.edu/pub/scheme-repository/doc/standards/r4rs.ps.gz`., November 1991.

[Clo] `http://clojure.org/`. Accessed January 2013.

[Coo91] William R. Cook. Object-oriented programming versus abstract data types. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, volume 489 of *Lecture Notes in Computer Science*, pages 151–178. Springer-Verlag, New York, NY, 1991.

[Cri91] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.

[ECM] Ecmascript. `http://www.ecmascript.org/`. Accessed January 2013.

[Erl] Erlang programming language. `http://www.erlang.org/`. Accessed January 2013.

[Flo79] Robert W. Floyd. The paradigms of programming. *Communications of the ACM*, 22(8):455–460, August 1979.

[FSh] F# at microsoft research. `http://research.microsoft.com/en-us/projects/fsharp/`. Accessed January 2013.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Mass., 1995.

[Gra04] Paul Graham. *Hackers and Painters*, chapter Beating the Averages. O'Reilly, 2004. Available at `http://www.paulgraham.com/avg.html`.

[Has] Haskell - haskellwiki. `http://www.haskell.org/haskellwiki/Haskell`. Accessed January 2013.

[HBS73] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *1973 International Joint Conference on Artificial Intelligence*, pages 235–245. IJCAI, 1973.

[Hud89] Paul Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411, September 1989.

[Hug89] J. Hughes. Why functional programming matters. *Computer Journal*, 32(2):98–107, 1989.

[Lam] `http://www.lix.polytechnique.fr/Labo/Dale.Miller/lProlog/`. Accessed January 2013.

[Lua] The programming language lua web site. `http://www.lua.org/`. Accessed January 2013.

[Moz] Mozart-oz web site. `http://www.mozart-oz.org/`. Accessed January 2013.

[OCa] Ocaml. `http://caml.inria.fr/ocaml/index.en.html`. Access January 2013.

[Orc] `http://orc.csres.utexas.edu/`. Accessed January 2013.

[Rac] The racket language. `http://racket-lang.org/`. Accessed January 2013.

[RLa] `http://www.r-project.org/`. Accessed January 2013.

[Rub] `http://www.ruby-lang.org/en/`. Accessed January 2013.

[Sca] The scala programming language web site. `http://www.scala-lang.org/`. Accessed January 2013.

[Sch] Welcome to schemers.org. `http://schemers.org/`. Accessed January 2013.

[SF89] George Springer and Daniel P. Friedman. *Scheme and the Art of Programming*. McGraw-Hill, New York, NY, 1989.

[Sma] `http://www.smalltalk.org/main/`. Accessed January 2013.

[VH04] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. The MIT Press, Cambridge, Mass., 2004.

[XSB] `http://xsb.sourceforge.net/`. Accessed January 2013.