

Functional Languages

Lisp, Scheme and ML are functional in nature.

Programs are expressions to be evaluated.

Language design aims to *minimize* side-effects, including assignment.

Alternative evaluation mechanisms are possible, including

- Lazy (Demand Driven)

- Eager (Data Driven or Speculative)

Object-Oriented Languages

C++, Java, Smalltalk, Pizza and Python are object-oriented.

Data and functions are encapsulated into Objects.

Objects are active, have persistent state, and uniform interfaces (messages or methods).

Notions of inheritance and common interfaces are central.

All objects that provide the same interface can be treated uniformly. In Java you can print any object that provides the `toString` method. You can iterate through the elements of any Java object that implements the **Enumeration** interface.

Subclassing allows to you extend or redefine part of an object's behavior without reprogramming all of the object's definition. Thus in Java, you can take a `Hashtable` class (which is fairly elaborate) and create a subclass in which an existing method (like `toString`) is redefined, or new operations are added.

Logic Programming Languages

Prolog notes that most programming languages address both the logic of a program (what is to be done) and the control flow of a program (how to do what you want).

A logic programming language, like Prolog, lets programmers focus on a program's logic without concern for control issue.

These languages have no real control structures, and little notion of "flow of control."

What results are programs that are unusually succinct and focused.

Example:

```
inOrder( [] ).  
inOrder( [ _ ] ).  
inOrder([a,b|c]) :- (a<b),  
    inOrder([b|c]).
```

This is a *complete, executable* function that determines if a list is in order. It is naturally polymorphic, and is not cluttered with declarations, variables or explicit loops.

Review of Concepts from Procedural Programming Languages

Declarations/Scope/Lifetime/Binding
Static/Dynamic

- Identifiers are *declared*, either explicitly or implicitly (from context of first use).
- Declarations *bind* type and kind information to an identifier. Kind specifies the grouping of an identifier (variable, label, function, type name, etc.)
- Each identifier has a *scope* (or range) in a program—that part of the program in which the identifier is visible (i.e., may be used).