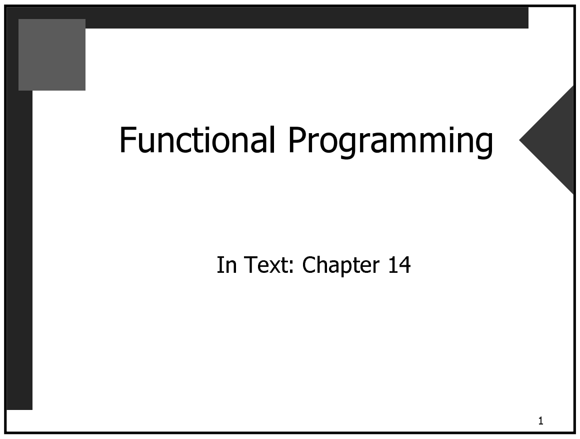
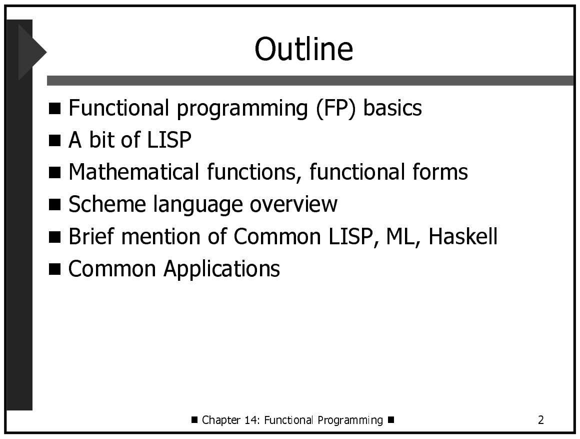


Functional Programming

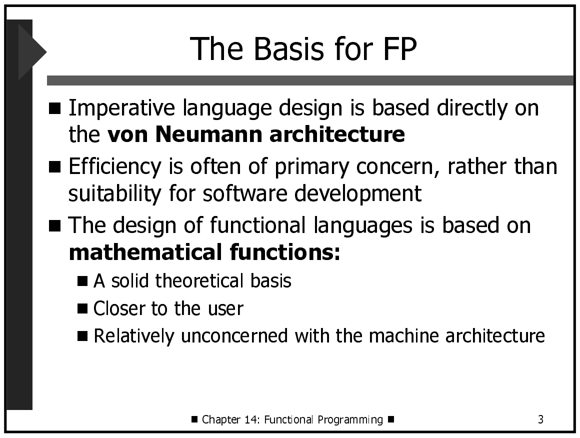
In Text: Chapter 14

1



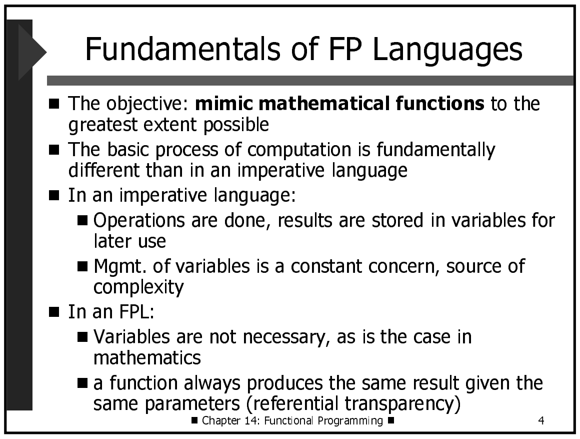
Outline

- Functional programming (FP) basics
- A bit of LISP
- Mathematical functions, functional forms
- Scheme language overview
- Brief mention of Common LISP, ML, Haskell
- Common Applications



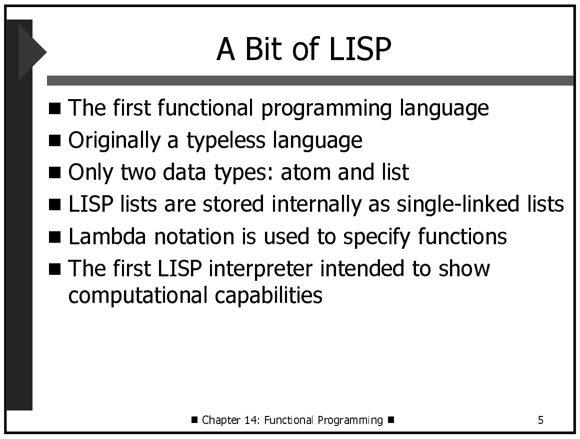
The Basis for FP

- Imperative language design is based directly on the **von Neumann architecture**
- Efficiency is often of primary concern, rather than suitability for software development
- The design of functional languages is based on **mathematical functions:**
  - A solid theoretical basis
  - Closer to the user
  - Relatively unconcerned with the machine architecture



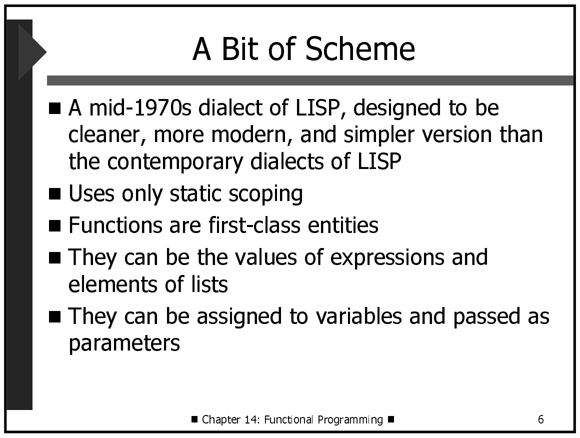
Fundamentals of FP Languages

- The objective: **mimic mathematical functions** to the greatest extent possible
- The basic process of computation is fundamentally different than in an imperative language
- In an imperative language:
  - Operations are done, results are stored in variables for later use
  - Mgmt. of variables is a constant concern, source of complexity
- In an FPL:
  - Variables are not necessary, as is the case in mathematics
  - a function always produces the same result given the same parameters (referential transparency)



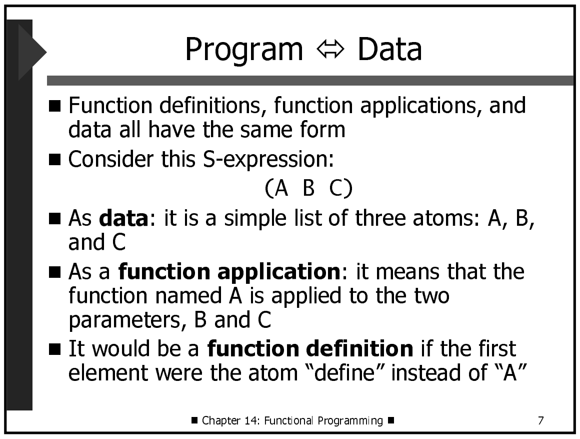
A Bit of LISP

- The first functional programming language
- Originally a typeless language
- Only two data types: atom and list
- LISP lists are stored internally as single-linked lists
- Lambda notation is used to specify functions
- The first LISP interpreter intended to show computational capabilities



A Bit of Scheme

- A mid-1970s dialect of LISP, designed to be cleaner, more modern, and simpler version than the contemporary dialects of LISP
- Uses only static scoping
- Functions are first-class entities
- They can be the values of expressions and elements of lists
- They can be assigned to variables and passed as parameters



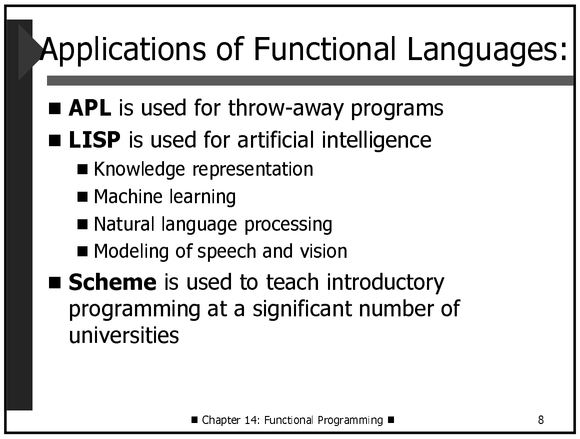
Program ⇔ Data

- Function definitions, function applications, and data all have the same form
- Consider this S-expression:

```
(A B C)
```

- As **data**: it is a simple list of three atoms: A, B, and C
- As a **function application**: it means that the function named A is applied to the two parameters, B and C
- It would be a **function definition** if the first element were the atom "define" instead of "A"

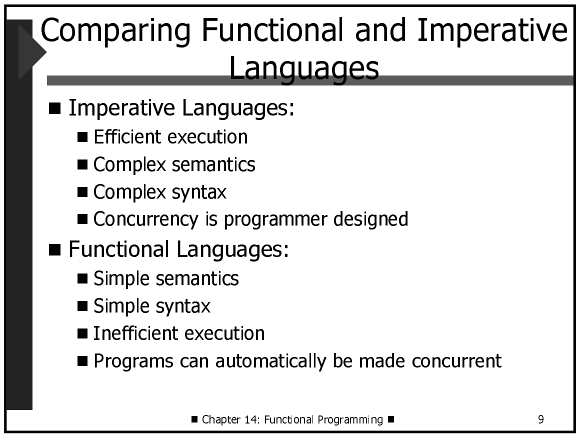
■ Chapter 14: Functional Programming ■ 7



Applications of Functional Languages:

- **APL** is used for throw-away programs
- **LISP** is used for artificial intelligence
  - Knowledge representation
  - Machine learning
  - Natural language processing
  - Modeling of speech and vision
- **Scheme** is used to teach introductory programming at a significant number of universities

■ Chapter 14: Functional Programming ■ 8



Comparing Functional and Imperative Languages

- Imperative Languages:
  - Efficient execution
  - Complex semantics
  - Complex syntax
  - Concurrency is programmer designed
- Functional Languages:
  - Simple semantics
  - Simple syntax
  - Inefficient execution
  - Programs can automatically be made concurrent

■ Chapter 14: Functional Programming ■ 9