Unit-5

Logic Programming Language Functional Programming Languages Scripting Language

R.JEEVITHA
Asst Professor CSE Department
Narsimha Reddy Engineering College (Autonomous)
Secunderabad, Telangana State, India-500100.
Ph.No: 09959845657

CONCEPTS

Logic Programming Language

Introduction Fundamentals of

FPL LISP

ML HASKELL

Applications of FPL Scripting

languages

FUNTIONAL PROGRAMMING LANGUAGE

The design of the imperative languages is based directly on Von Nuemann Architechture.

The design of the functional language is based on mathematical functions.

MATHEMATICAL FUNCTION

Def: A mathematical function is a mapping of members of one set, called the domain set, to another set, called the range set.

A lambda expression specifies the parameter(s) and the mapping of a function in the following form l(x) x * x * x

For the function cube (x) = x * x * x

Lambda expressions describe nameless functions

Mathematical function(cont..)

Lambda expressions are applied to parameter(s) by placing the parameter(s) after the expression

e.g. (l(x) x * x * x)(3) which evaluates to 27

A Function for Constructing Functions

DEFINE - Two forms:

To bind a symbol to an expression

e.g.,

(DEFINE pi 3.141593)
(DEFINE two_pi (* 2 pi))

Fundamentals of Functional Programming Languages

The objective of the design of a FPL is to mimic mathematical functions to the greatest extent possible.

The basic process of computation is fundamentally different in a FPL than in an imperative language.

In an imperative language, operations are done and the results are stored in variables for later use

Fundamentals of FPL(cont..)

Management of variables is a constant concern and source of complexity for imperative programming.

In an FPL, variables are not necessary, as is the case in mathematics.

In an FPL, the evaluation of a function always produces the same result given the same parameters.

This is called *referential transparency*.

LISP

The first functional programming language.

Data object types: originally only atoms and lists.

List form: parenthesized collections of sublists and/or atoms

E.g., (A B (C D) E)

A Bit of LISP

Originally, LISP was a typeless language.

There were only two data types, atom and list.

LISP lists are stored internally as single-linked lists.

Lambda notation is used to specify functions and function definitions, function applications,and data all have the same form.

INTRODUCTION TO 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

Primitive Functions:

Arithmetic: +, -, *, /, ABS, SQRT
e.g., (+ 5 2) yields 7

QUOTE -takes one parameter; returns the parameter without evaluation.

QUOTE is required because the Scheme interpreter, named EVAL, always evaluates parameters to function applications before applying the function. QUOTE is used to avoid parameter evaluation when it is not appropriate.

QUOTE

QUOTE can be abbreviated with the apostrophe prefix operator

e.g., '(A B) is equivalent to (QUOTE (A B))

CAR takes a list parameter; returns the first element of that list

e.g., (CAR '(A B C)) yields A (CAR '((A B) C D)) yields (A B)

CDR takes a list parameter; returns the list after removing its first element

e.g., (CDR '(A B C)) yields
(B C)

(CDR '((A B) C D)) yields
(C D)

- CONS takes two parameters, the first of which can be either an atom or a list and the second of which is a list; returns a new list that includes the first parameter as its first element and the second parameter as the remainder of its result

e.g., (CONS 'A '(B C)) returns (A B C)

LIST - takes any number of parameters; returns a list with the parameters as elements.

Predicate Functions: (#T and () are true and false)

1. EQ? takes two symbolic parameters; it returns #T if both parameters are atoms and the two are the same.

e.g., (EQ? 'A 'A) yields #T
(EQ? 'A '(A B)) yields ()

LIST? takes one parameter; it returns #T if the
parameter is an list; otherwise ()

NULL? takes one parameter; it returns #T if the parameter is the empty list; otherwise ()

Note that NULL? returns #T if the parameter is ()

Numeric Predicate Functions =, <>, >, <, >=, <=, EVEN?, ODD?, ZERO?

5. *Output Utility Functions:*

(DISPLAY expression)
(NEWLINE)

Lambda Expressions:

Form is based on l notation e.g.,
(LAMBDA (L) (CAR (CAR L))) L is

called a bound variable Lambda

expressions can be applied

e.g., ((LAMBDA (L) (CAR (CAR L))) '((A B) C D))

To bind names to lambda expressions e.g.,(DEFINE (cube x) (* x x x))

Example use:(cube 4)

- Evaluation process (for normal functions):

Parameters are evaluated, in no particular order.

The values of the parameters are substituted into the function body.

The function body is evaluated.

The value of the last expression in the body is the value of the function.

Control Flow:

Selection- the special form, IF (IF predicate then_exp else_exp) e.g.,(IF (<> count 0)

(/ sum
count) 0
)

Multiple Selection - the special form, COND

- General form:
- (COND

  (predicate_1 expr {expr})
  (predicate_1 expr {expr})

  …

  (predicate_1 expr {expr})
  (ELSE expr {expr})

)

  Returns the value of the last expr in the first pair whose predicate evaluates to true

COMMON LISP

- A combination of many of the features of the popular dialects of LISP around in the early 1980s.
- A large and complex language--the opposite of Scheme.
- *Includes*: records, arrays, Complex numbers, character strings, powerful i/o capabilities, packages with access control, imperative features like those of Scheme ,iterative control statements.

ML

- A static-scoped functional language with syntax that is closer to Pascal than to LISP
- Uses type declarations, but also does type inferencing to determine the types of undeclared
- It is strongly typed (whereas Scheme is essentially

ML(cont..)

- Includes exception handling and a module facility for implementing abstract data types
- Includes lists and list operations
- The val statement binds a name to a value (similar to DEFINE in Scheme)
- Function declaration form:
- fun function_name (formal_parameters)

ML(cont..)

Functions that use arithmetic or relational operators cannot be polymorphic--those with only list operations can be polymorphic

Haskell

- Similar to ML (syntax, static scoped, strongly typed, type inferencing)
- Different from ML (and most other functional languages) in that it is PURELY functional (e.g., no variables, no assignment statements, and no side effects of any kind)
- Most Important Features
  - Uses lazy evaluation
  - Has “list comprehensions,” which allow it to deal with infinite lists

HASKELL(cont..)

Examples

Fibonacci numbers (illustrates function definitions with different parameter forms)

```
fib 0 = 1
fib 1 = 1
fib (n + 2) = fib (n + 1) + fib n
```

2.Lazy evaluation

Infinite lists

e.g., positives = [0..]

squares = [n * n | n ¨ [0..]]

(only compute those that are necessary)

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.

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

Scripting languages

- Pragmatics
  - *Scripting* is a paradigm characterized by:
    - -use of scripts to glue subsystems together;
    - -rapid development and evolution of scripts;
    - -modest efficiency requirements;
    - -very high-level functionality in

Scripting languages(cont.)

- A software system often consists of a number of subsystems controlled or connected by a script.
- In such a system, the script is said to glue the sub systems together.

Python

PYTHON was designed in the early 1990s by Guido van Rossum.

PYTHON borrows ideas from languages as diverse as PERL ,HASKELL ,and the object-oriented languages, skillfully integrating these ideas into a coherent whole.

PYTHON scripts are concise but readable, and highly expressive.

Values and types

PYTHON has a limited repertoire of primitive types: integer, real, and complex Numbers.

It has no specific character type; single-character strings are used instead.

its boolean values (named False and True) are just small integers.

PYTHON has a rich repertoire of composite types: tuples, strings, lists, dictionaries, and objects.

Variables, storage, and control

PYTHON supports global and local variables.

Variables are not explicitly declared,simply initialized by assignment.

PYTHON adopts reference semantics. This is especially significant for mutable values, which can be selectively updated.

Primitive values and strings are immutable; lists, dictionaries, and objects are mutable; tuples are mutable if any of their components are mutable.

PYTHON's repertoire of commands include assignments, procedure calls, con-ditional (if- but not case-) commands, iterative (while- and for-) commands, and exception-handling commands.

PYTHON if- and while-commands are conventional.

Pythons reserved words

and assert break class continue def del

elif

else except exec finally for from global if

import in is lambda not or pass

print

raise return try while yield

Dynamically typed language

Python is a dynamically typed language. Based on the value, type of the variable is during the execution of the program.

Python(dynamic)

C = 1

C = [1,2,3]

C(static)

Double c; c = 5.2;

Strongly typed python language:

Weakly vs strongly typed python language differ in their automatic conversions.

Perl(weak)

```
$b = `1.2`
$c = 5 * $b;
```

Python(strong)

```
=`1.2`
c= 5* b;
```

Bindings and scope

- A PYTHON program consists of a number of modules, which may be grouped into packages.
- Within a module we may initialize variables, define procedures, and declare classes
- Within a procedure we may initialize local variables and define local procedures.
- Within a class we may initialize variable components and define procedures (methods).
- PYTHON was originally a dynamically-scoped language, but it is now statically scoped.

Binding and scope

In python, variables defined inside the function are local to that function. In order to change them as global variables, they must be declared as global inside the function as given below.

```
S = 1
Def myfunc(x,y);
Z = 0

Global s;
S = 2
Return y-1 , z+1;
```

Procedural abstraction

- PYTHON supports function procedures and proper procedures.
- The only difference is that a function procedure returns a value, while a proper procedure returns nothing.
- Since PYTHON is dynamically

Python procedure

```
Eg :Def gcd (m, n):
    p,q=m,n
    while p%q!=0:
    p,q=q,p%q
    return q
```

Python procedure with Dynamic Typing

Eg:
```
def minimax (vals):
    min = max = vals[0]
    for val in vals:
    if val < min:
    min = val
    elif val > max:
    max =  val
    return min, max
```

Data Abstraction

- PYTHON has three different constructs relevant to data abstraction: packages
- ,modules , and classes
- Modules and classes support encapsulation, using a naming convention to distinguish between public and private components.

Data abstraction(cont..)

A Class is a group of components that may be class variables, class methods ,and instance methods.

A procedure defined in a class declaration acts as an instance method if its first formal parameter is named self and refers to an object of the class being declared. Otherwise the procedure acts as a class method.

Data abstraction(cont..)

- To achieve the effect of a constructor, we usually equip each class with an initialization method named "_init_"; this method is automatically called when an object of the class is constructed.
- PYTHON supports multiple inheritance: a class may designate any number of superclasses.

Separate Compilation

PYTHON modules are compiled separately.

Each module must explicitly import every other module on which it depends

Each module's source code is stored in a text file. Eg: program.py

When that module is first imported, it is compiled and its object code is stored in a file named program.pyc

Separate Compilation(cont..)

Compilation is completely automatic

The PYTHON compiler does not reject code that refers to undeclared identifiers.Such code simply fails if and when it is executed

The compiler will not reject code that might fail with a type error,nor even code that will certainly fail, such as:

```
def fail (x):
print x+1, x[0]
```

Module Library

- PYTHON is equipped with a very rich module library, which supports string handling ,markup , mathematics, cryptography, multimedia, GUIs, operating system services ,internet services, compilation, and so on.
- Unlike older scripting languages, PYTHON does not have built-in high-level string processing or GUI support , so module library provides it.

- THE
- END