History

Table

Description automatically generated

**FUNCTIONAL PROGRAMMING**

Functional programming is a style of programming

First FP language 🡪 LISP

Imperative Programming:

* Program = Data + Algorithms
* g(f(2)) 🡪 apply f(2) first and get the result and apply it to g
* works very well with Von Neumann machine.

OO Programming:

* Program = Object. message (object)

Functional Programming:

* Program = Functions to Functions
* another possibility is getting g of f function

Computation is done by application of functions

Functional Programming Languages

A functional language supports and advocates for the style of FP

Important Features:

* Everything is function (input -> function -> output)
* No statement or expression
* No variables or assignments (only constant values, arguments, and returned values. Thus no notion of state, memory location)
  + If I have only constants and have values that are copied, we are talking about pass by value concept so I need to copy them. This brings inefficiency.
* No loops (only recursive functions)
  + because you don’t have variable concept
* No side-effect (Referential Transparency):
  + The value of a function depends only on the values of its parameters.
  + Evaluating a function with the same parameters gets the same results.
  + There is no state.
  + Evaluation order or execution path don’t matter. (random() and getchar() are not referentially transparent.)
* Functions are first-class values: functions are values, can be parameters and return values, can be composed.

FP in Imperative Languages

Graphical user interface, text

Description automatically generated

I don’t have assignments, variables and loops in functional style.

We use setq in LISP, isn’t it assignment?

* LISP is not purely functional.
* You can treat that local parameter setting as a functional thing.

Why does it matter, anyway?

The advantages of functional programming languages:

* Simple semantics, concise, flexible
* ``No’’ side effect
* Less bugs

It does have drawbacks:

* Execution efficiency
  + Implementing concept on imperative machine
  + mapping is not one-to-one, it is complex
* More abstract and mathematical, thus more difficult to learn and use

Even if we don’t use FP languages:

* Features of recursion and higher-order functions have gotten into most programming languages

Functional Programming Languages in Use

Popular in prototyping, mathematical proof systems, AI and logic applications, research and education

Scheme:

* Document Style Semantics and Specification Language (SGML stylesheets)
* GIMP
* Guile (GNU’s official scripting language)
* Emacs (Kind of editor)

Haskell

* Linspire (commerical Debian-based Linux distribution)
* xmonad (X Window Manager)

XSLT (Extensible Stylesheet Language Transformations)

**SCHEME**

Scheme: Lisp dialect

Syntax (slightly simplified):

Text

Description automatically generated

Everything is an expression: programs, data, …

* Thus programs are executed by evaluating expressions
* Expressions are application of functions built-in or not built-in

Only 2 basic kinds of expressions:

* atoms: unstructured
* lists: the only structure (a slight simplification)

Expressions

Graphical user interface, application

Description automatically generated

Expressions are either an atom or list of atoms.

There is no concept of variable. Identifier may be local argument for a function or sth like that.

Evaluation of Expressions

Programs are executed by evaluating expressions. Thus semantics are defined by evaluation rules of expressions.

Evaluation Rules:

* + number | string: evaluate to itself
  + Identifier: looked up in the environment, i.e., dynamically maintained symbol table
  + List: recursively evaluate the elements

Eager Evaluation

A list is evaluated by recursively evaluating each element:

* unspecified order
* first element must evaluate to a function

This function is then applied to the evaluated values of the rest of the list (prefix form)

A picture containing text

Description automatically generated

Most expressions use applicative order evaluation (eager evaluation): subexpressions are first evaluated, then the expression is evaluated

(correspondingly in imperative language: arguments are evaluated at a call site before they are passed to the called function)

Lazy Evaluation: Special Forms

if function (if a b c):

* a is always evaluated
* Either b or c (but not both) is evaluated and returned as result.
* c is optional. (if a is false and c is missing, the value of the expression is undefined.)

e.g., (if (= a 0) 0 (/ 1 a))

If this one (if) is eager evaluation, you would implement a b and c, then apply if to it

cond : (cond (e1 v1) (e2 v2) ... (else vn))

* The (ei vi) are considered in order
* ei is evaluated. If it is true, vi is then evaluated, and the value is the result of the cond expression.
* If no ei is evaluated to true, vn is then evaluated, and the value is the result of the cond expression.
* If no ei is evaluated to true, and vn is missing, the value of the expression is undefined.

(cond ((= a 0) 0) ((= a 1) 1) (else (/ 1 a)))

define function: declare identifiers for constants and function, and thus put them into dynamic symbol table.

Graphical user interface, text, application

Description automatically generated

(define a b) -----> I am gonna define identifier “a” which is gonna be a constant doing sth with b

Defining constant and it is not going to change so it is not a variable

Quote, or ' for short, has as its whole purpose to not evaluate its argument:

(quote (2 3 4)) or '(2 3 4) returns just (2 3 4).

(we need a list of numbers as a data structure)

lazy evaluation because (2 3 4) is not gonna be evaluated, just put in a list.

eval function: get evaluation back

(eval '(+ 2 3)) returns 5

Other Special Forms

**let function**: create a binding list (a list of name-value associations), then evaluate an expression (based on the values of the names)

A picture containing text

Description automatically generated

This is not an assignment.

let says that “I am giving you a parameter, I am defining a value for that parameter, you bind that, it’s a list of bindings, you can change that binding inside your code”. When you are outside the let function, it’s gone.

setq changes your binding list in that local context.

Lists

– Only data structure

– Used to construct other data structures

– Thus we must have functions to manipulate lists

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generatedData structures

Box diagrams

Chart, box and whisker chart

Description automatically generated

Other list manipulations: based on car, cdr, cons

Graphical user interface, text, application

Description automatically generated

We need mechanism to treat functions as first class values. For that we have lambda expressions.

They are used to dynamically create functions.

Graphical user interface, text, application

Description automatically generated

lazy evaluation, x\*x is not gonna be evaluated

Text

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated with low confidence

Text

Description automatically generated with medium confidence