Functional Programming Notes

Igor Dimitrov

2024-12-18

Table of contents

Preface

This is a Quarto book.

1 Introduction

This is a book created from markdown and executable code.

See for additional discussion of literate programming.

2 Reading List

2.1 Logo

- Computer Science Logo Style, Vol 1 - 3. Harvey.

2.2 Lisp

- Common Lisp - A Gentle Introduction to Symbolic Computation. Touretzky
- Common Lisp - An Interactive Approach. Shapiro
- Land of Lisp - Learn to Program in Lisp, One Game at a Time. Barski
- Common Lisp - Paul Graham
- On Lisp - Advanced Techniques for Common Lisp - Paul Graham
- Lisp - Let Over Lambda. Hoyte
- Lisp in Small Pieces - Queinnec
- The Anatomy of Lisp - John Allen

2.3 Scheme

- Simply Scheme. Harvey
- Concrete Abstractions - an Introduction to Computer Science using Scheme. Hailperin
- An Introduction to Scheme. Jerry Smith
- Exploring Computer Science with Scheme. Oliver Grillmeyer
- Programming in Scheme. Eisenberg
- Programming and Metaprogramming in Scheme. Jon Pearce
- Structure and Interpretation of Computer Programs
- Algorithms for Functional Programming. John David Stone

2.4 Racket

- How to Design Programs - an Introduction to Programming and Computing. Felleisen
- Realm of Racket - Learn to Program, One Game at a Time!. Van Horn, Felleisen

- Racket Programming the Fun Way - From Strings to Turing Machines. James Stelly
- Die Macht der Abstraktion. Klaeren

2.5 Haskell

- Functional Programming. Fokker
- Introduction to Functional Programming using Haskell. Bird
- Introduction to Functional Programming Systems Using Haskell. Antony Davie
- Thinking Functionally with Haskell - Richard Bird
- Haskell - The Craft of Functional Programming. Thompson
- Programming in Haskell. Graham Hutton
- Reasoned Programming. Broda
- Introduction to Computation, Haskell, Logic, and Automata. Wadler
- Algorithm Design in Haskell. Bird
- The Fun of Programming - Gibbons
- Algebra of Programming. Bird, de Moor

2.6 SML

- Elements of Fucntional Programming with SML. Reade
- Introduction to Programming Using SML. Hansel, Rischel
- Functional Programming Using SML. Wikstroem
- Elements of ML Programming. Ullman
- Purely Functional Data Structures. Okasaki

2.7 OCAML

- Think OCAML - How to Think Like a Computer Scientist. Monje, Downey
- OCaml from the very Beginning. Whitington
- More OCaml. Whitignton
- OCaml - Functional Programming for the Masses. Madhavapeddy

2.8 Implementation and Practical Aspects

- Functional Programming - Application and Implementation. Henderson
- Functional Programming. Field, Harrison
- The Architecture of Symbolic Computers

2.9 General

- Functional Programming - Practice and Theory. Bruce J. MacLennan
- Equations, Models, and Programs - a Mathematical Introduction to Computer Science. Myers.

2.10 Theory

- An Introduction to Functional Programmming Through Lambda Calculus. Michaelson
- Introduction to Functional Programming. Gordon
- Introduction to Functional Programming. Harrison
- Type Theory & Functional Programming. Thompson
- Foundations of Functional Programming. Lawrence C Paulson

2.11 Python

- A Functional Start to Computing with Python. Herman
- SICP with Python

2.12 C

- Functional C. Muller

2.13 Articles

- Transformational Programming and the Paragraph Problem. Bird
- From Informal Requirements to a Running Program: A Case Study in Aglebrain Specification and Transformational Programming. Partsch
- Recursion Equation as a Programming Language. Turner
- Combining Algebraic and Algorithmic Reasoning: An Approach to the Schorr-Waite Algorithm. Broy, Pepper
- Algebraic Calculation of Graph and Sorting Algorithms. Moeller
- Law and Order in Algorithmics. Fokkinga
- Functional Programming with Bananas, Lenses, Envelopes, and Barbed Wire. Fokkinga, Paterson
- An Introduction to the Theory of Lists. Bird

Part I

Logo

3 Introduction to Logo

- word-oriented (as opposed to number-oriented) lisp-like functional programming language.

3.1 Basic Notions

Words and Lists

Basic building blocks of logo programs are *words*. Everything in logo is a word. By default logo interprets a word as a name of a procedure and tries to carry it out. Examples of procedures are `sum`, `print`. We will define procedures more carefully below.

In order for a word to be interpreted as itself, and not as a name of a procedure it must be *escaped*.

E.g.

```
print hello
```

will not print `hello`, but will try interpret `hello` as a name of a procedure, and output and error. A word is escaped by prepending it with a quote like this: `"hello`. For this reason escaping is also called quoting.

So

```
print "hello
```

will print `hello`.

Numeric words are automatically escaped. That is, `print 2` and `print "2` are the same. It makes sense since procedure names can not be numbers, so numbers are automatically understood to evaluate to themselves.

List is simply a combination of multiple words. All logo programs are lists. Logo evaluates a list by evaluating the procedures contained in the list. e.g.

```
print sum 3 2
```

Just as words can be escaped, so can be lists. In that case logo will not try to evaluate the list and carry out the computation, rather the list will be evaluated to itself. Lists are escaped by enclosing them in square brackets. e.g.

```
[print sum 3 2]
```

will not print 5. It is simply the list itself. Note that this is not a complete logo instruction. (We will carefully define what an instruction is below). The `print` procedure is flexible and accepts not only words but also (escaped) lists. So we can supply the above list to it.:

```
print [print sum 3 2]
```

This will print `print sum 3 2`

Escaped words or escaped lists that evaluate to themselves are called *data*.

Lists are very flexible and can contain different sorts of data, including other lists:

```
[[1 apple] 2 [banana 15 cherry]]
```

Above list contains two lists and one number. It is a nested list. A list that is not nested, like `[banana 15 cherry]` is called a *flat list* of a *sentence*.

This is different to many other languages, where lists usually must contain objects or data of the same type.

Procedures and Instructions

procedures are programs that carry out computations. e.g. `print`, `sum` are names of procedures. In conventional programming languages there are many forms of statements that achieve certain things, like assignment statement, `if` statement, `while` statement, all having their unique syntax rules. In lisp-like languages this is much simpler. There is only one type of statement: procedure invocation.

A procedure is a name of a program, not a concrete instance of it. To carry out a concrete computation, a procedure must be supplied with information (data). Doing this is called *invoking* a procedure. E.g. `sum 3 2`, `print 10`. Different procedures accept different number of data. E.g. `sum` expects two words, while `print` expects a single one.

There are two type of procedures:

1. *commands*: commands have *effects* when invoked. Effects change something in the *state* of the computer, e.g. `print`
2. *operations*: operations return *values / data* when invoked. They do not change the state of the computer. e.g. `sum`. Another way to say is that operations are *evaluated* when invoked.

since operations return values, they can be used in place of data as an input to a procedure. We saw it with `print`:

```
print sum 3 2
```

Since invoked operations are values, they can be supplied to operations as inputs. This can be done indefinitely and such combinations are called expressions. More formally expression are defined inductively as follows:

1. data are expressions.
2. if an operation `op` expects `n` inputs, and `e1,..., en` are expressions. Then `op e1 ... en` is an expression.

An *instruction* is invocation of one or more procedures. E.g. `print sum 3 2` or `print 10`. More formally an instruction is a list, where the first word is a command and the rest of the words are expressions that evaluate to inputs necessary to carry out the command. E.g.:

```
print sum 3 2
```

`print` is a command and it expects single datum. `sum 3 2` is an expression evaluating to `5`, which in turn is passed to `print` as input.

A procedure is described (specified) by the following:[see 1]

1. Is it a command or an operation?
2. How many inputs does it accept?
3. What are the types of each of the inputs? (word, list or array)?
4. If the procedure is an operation, what is the *output*? (Its description and type) If the procedure is a command what is the effect? (The description of the effect) .

Manipulating Words and Lists

References

1. Harvey, B. *Computer Science Logo Style: Volume 3*. (MIT Press, Cambridge, Mass, 1997).

Part II

Gofer

4 Intro to FP with Gofer

4.1