Python e Lisp:

Como unir dois mundos

DISCLAIMER

Python?

Everyone Nows

Lisp

Lisp?

LISt Processor

John McCarthy

MIT: IA Research Group

60’s

Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

JOHN MCCARTHY, *Massachusetts Institute of Technology, Cambridge, Mass.*

1. Introduction

A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit "common sense" in carrying out its instructions. The original proposal [1] for the Advice Taker was made in November 1958. The main requirement was a programming system for manipulating expressions representing formalized declarative and imperative sentences so that the Advice Taker system could make deductions.

In the course of its development the LISP system went through several stages of simplification and eventually came to be based on a scheme for representing the partial recursive functions of a certain class of symbolic expressions. This representation is independent of the IBM 704 computer, or of any other electronic computer, and it now seems expedient to expound the system by starting with the class of expressions called S-expressions and the functions called S-functions.

In this article, we first describe a formalism for defining functions recursively. We believe this formalism has ad-

2. Functions and Function Definitions

We shall need a number of mathematical ideas and notations concerning functions in general. Most of the ideas are well known, but the notion of *conditional expression* is believed to be new, and the use of conditional expressions permits functions to be defined recursively in a new and convenient way.

a. *Partial Functions.* A partial function is a function that is defined only on part of its domain. Partial functions necessarily arise when functions are defined by computations because for some values of the arguments the computation defining the value of the function may not terminate. However, some of our elementary functions will be defined as partial functions.

b. *Propositional Expressions and Predicates.* A propositional expression is an expression whose possible values are T (for truth) and F (for falsity). We shall assume that the reader is familiar with the propositional connectives $\wedge$ ("and"), $\vee$ ("or"), and $\sim$ ("not"). Typical propositional expressions are:

$$x < y$$

$$(x < y) \wedge (b = c)$$

x is prime

59’s

PROGRAMS WITH COMMON SENSE

John McCarthy
Computer Science Department
Stanford University
Stanford, CA 94305
jmc@cs.stanford.edu
http://www-formal.stanford.edu/jmc/

1959

1 Introduction

Interesting work is being done in programming computers to solve problems which require a high degree of intelligence in humans. However, certain elementary verbal reasoning processes so simple that they can be carried

Influences

IBM 704

IPL 2

Lambda Calculos

Alonzo Church

((parentheses))

(((Why so many parentheses?!)))

Syntax

- Atom
- Symbol
- List

Atom?

> 5

5

> 42

42

Symbol?

```
> 'somethingelsewhere
somethingelsewhere
> 'a
a
> '42
42
```

List?

```
> '(1 2 3 4)
(1 2 3 4)
> (list 15 48 'b 32)
(15 48 B 32)
```

Operators?

S-expression

(<operand> <arg1>...<argN>)

=> (+ 2 3 1)

6

=> (* 4 2 6)

48

Hy

Lisp dialect

Embedded in Python

Python abstract syntax tree

Datatypes

History

Introduction

> Datatypes

Comparation

Same Data Types

Array

=> [1 2 3 4]

[1 2 3 4]

Tuples

=> (, 1 2 3 4)

(1, 2, 3, 4)

Dictionary

=> {"dog" "bark"

... "cat" "meow"}

{’dog’: ’bark’, ’cat’: ’meow’}

Functions?

In Python

```
>>> def hello():
...     print "Hello World!"
Hello World!
```

In Hy

```
=> (defn hello [ ]
...    (print "Hello World!"))
=> (hello)
Hello World!
```

Conditional?

In Python

```
>>> if 2 > 3:
...     True
...  else:
...     False
False
```

In Hy

```
=> (if True 1)
1
=> (cond [(= 2 3) 0]
...      [(> 3 2) 2]
...      [True print("Ok")])
2
```

Lambda?

(fn [arg] body)

```
=> ( ( fn [ x ] x ) "Hy From Her!" )
Hy From Her
=> ( ( fn [ y n ]
...  ( + y n ) ) 4 5 )
9
```

(fn [x] x) === lambda x: x

Objects?

```
=> (defclass Cat [ ]
...    [age None
...     colour None]
...    (defn speak [ ]
...      (print "Meow")))
```

```
=> (setv spok (Cat))
=> (setv spok.age 5)
=> (.speak spok)
Meow
```

Recursion?

```
=> (defn fact [n]
...   (defn fact-aps [n acc]
...    (cond
...      [(= n 1) acc]
...      [True
...       (fact-aps (- n 1) (* acc n))]))
...  (fact-aps n 1))
```

=> (fact 5)

120

=> (fact 30)

265252859812191O5...

Diferences from other Lisps

No car, cdr, Lambda

But first, rest, second, fn

No Fully Recursion

From Python ?

import Works?

Yes!

Bytecode compatible inter python files

Meta Programing in Python

Macros

Macros?

```
=> (defmacro simplemacro [reps]
...  `(print (* ~reps ~reps)))
=> (simplemacro 4)
16
```

```
=> (defmacro ap-map [form lst]
... (setv v (gensym 'v)  f (gensym 'f))
... `((fn []
...         (defn ~f [it] ~form)
...         (for [~v ~lst]
...           (yield (~f ~v))))))
```

https://github.com/haller218/HylangPresentation

https://github.com/haller218