Lisp

30th October 2019

Table Of Contents

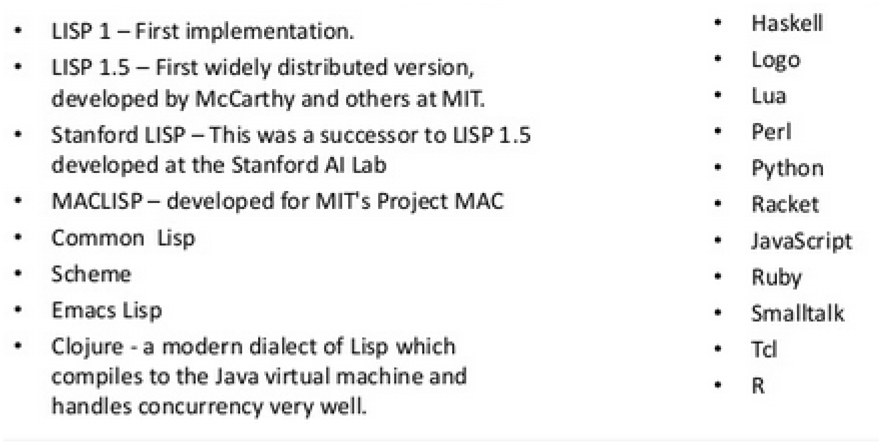
What is Lisp?

- Lisp (historically LISP) is a family of computer programming languages with a long history and a distinctive, fully parenthesized prefix notation.
- Originally specified in 1958, Lisp is the second-oldest high-level programming language in widespread use today.
- Only FORTRAN is older, by one year. Lisp has changed since its early days, and many dialects have existed over its history.
- Today, the best-known general-purpose Lisp dialects are Clojure, Common Lisp, and Scheme.
- The name LISP comes from LISt Processor.
- Linked lists are one of Lisp's major data structures, and Lisp source code is made of lists.

History

- Invented by John McCarthy in 1958 while he was in MIT.
- First implemented by Steve Russel on a IBM 704 computer.
- Lisp was used as the implementation of the programming language Micro Planner, which was used in the famous AI system SHRDLU.
- In the 1970s, as AI research spawned commercial offshoots, the performance of existing Lisp systems became a growing issue.

LISP Dialects



- LISP 1 – First implementation.
- LISP 1.5 – First widely distributed version, developed by McCarthy and others at MIT.
- Stanford LISP – This was a successor to LISP 1.5 developed at the Stanford AI Lab
- MACLISP – developed for MIT's Project MAC
- Common Lisp
- Scheme
- Emacs Lisp
- Clojure - a modern dialect of Lisp which compiles to the Java virtual machine and handles concurrency very well.

- Haskell
- Logo
- Lua
- Perl
- Python
- Racket
- JavaScript
- Ruby
- Smalltalk
- Tcl
- R

Lisp Program Structure

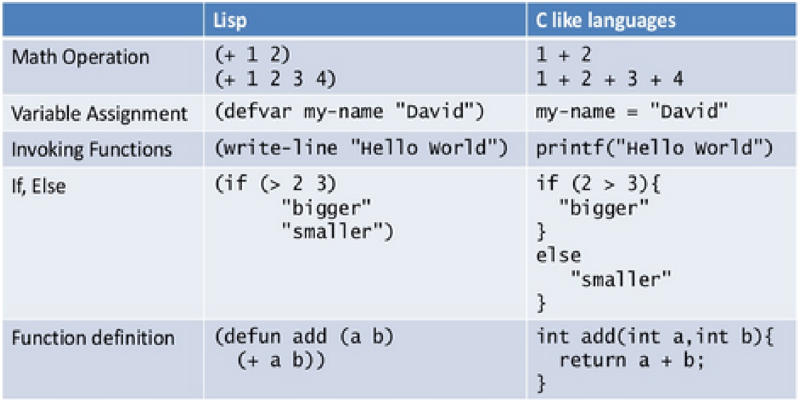
- Symbolic Expressions or S-Expressions.
- The S-Expressions are composed of three valid objects, atoms, lists and strings.
- Lisp programs run either on **interpreter** or as a **compiled code.**
- Lisp statements are case-insensitive, so the below lines are same:

(*write-line "Hello World.!"*)
(*WRITE-LINE "Hello World.!"*)

- Lisp represents a function call f(x) as (f x).

Basic Syntax

- Lisp's syntax is a lot different when compared to C like syntax (C, C++, Java..).
- Lisp uses parenthesis () extensively.
- Lisp Uses prefix Notation.



	Lisp	C like languages
Math Operation	`(+ 1 2)` `(+ 1 2 3 4)`	`1 + 2` `1 + 2 + 3 + 4`
Variable Assignment	`(defvar my-name "David")`	`my-name = "David"`
Invoking Functions	`(write-line "Hello World")`	`printf("Hello World")`
If, Else	`(if (> 2 3)` `"bigger"` `"smaller")`	`if (2 > 3){` `"bigger"` `}` `else` `"smaller"` `}`
Function definition	`(defun add (a b)` `(+ a b))`	`int add(int a,int b){` `return a + b;` `}`

Building Blocks of Lisp

- Lisp programs are made up of three basic buliding blocks.
  1. **Atom**: An Atom is a number or string of contiguous characters. It includes numbers and special characters.

     Hello
     Lisp_is_Nice
     1234543321
     password@!23

  2. **List**: Sequence of atoms and/or other lists encoded in parenthesis.

     (or (and "zero" nil "never") "James" 'task 'time)

  3. **String**: Group of characters encoded in double quotation marks.

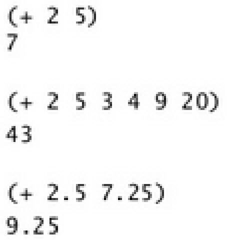
     "Hello from the Other siiide...."
     "Don't Leak this password."
     "1234567890"

Hello World

```
(write-line "Hello World!")
```

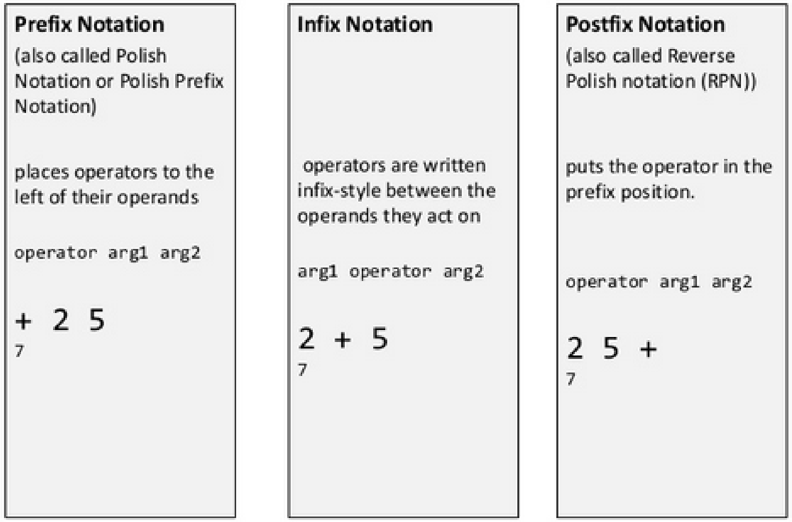
Polish Notations

- Lisp uses prefix notations to evaluate arithmetic operations.



```
(+ 2 5)
7

(+ 2 5 3 4 9 20)
43

(+ 2.5 7.25)
9.25
```

Polish Notations



Prefix Notation

(also called Polish Notation or Polish Prefix Notation)

places operators to the left of their operands

```
operator arg1 arg2
```

```
+ 2 5
7
```

Infix Notation

operators are written infix-style between the operands they act on

```
arg1 operator arg2
```

```
2 + 5
7
```

Postfix Notation

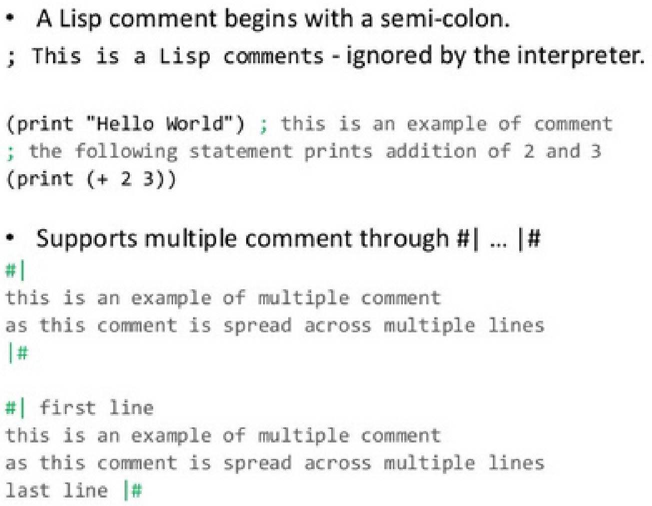
(also called Reverse Polish notation (RPN))

puts the operator in the prefix position.

```
operator arg1 arg2
```

```
2 5 +
7
```

Comments in LISP



- A Lisp comment begins with a semi-colon.

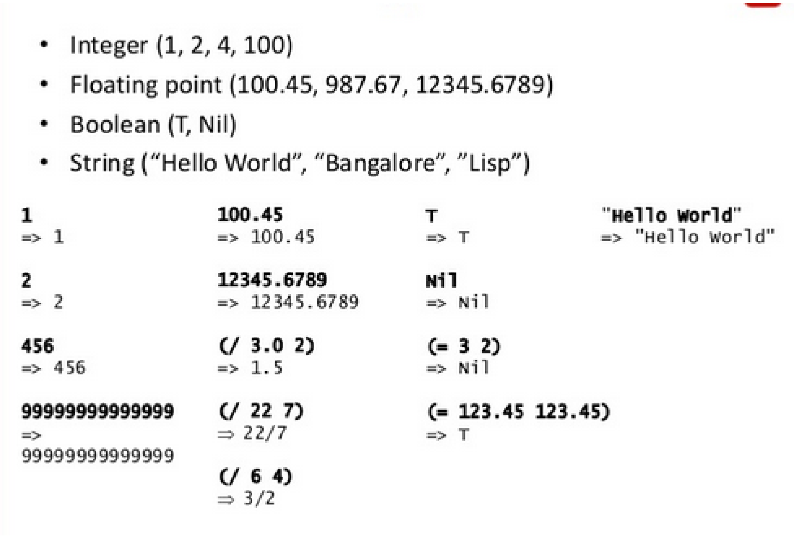
; This is a Lisp comments - ignored by the interpreter.

```
(print "Hello World") ; this is an example of comment
; the following statement prints addition of 2 and 3
(print (+ 2 3))
```

- Supports multiple comment through #| ... |#

```
#|
this is an example of multiple comment
as this comment is spread across multiple lines
|#

#| first line
this is an example of multiple comment
as this comment is spread across multiple lines
last line |#
```

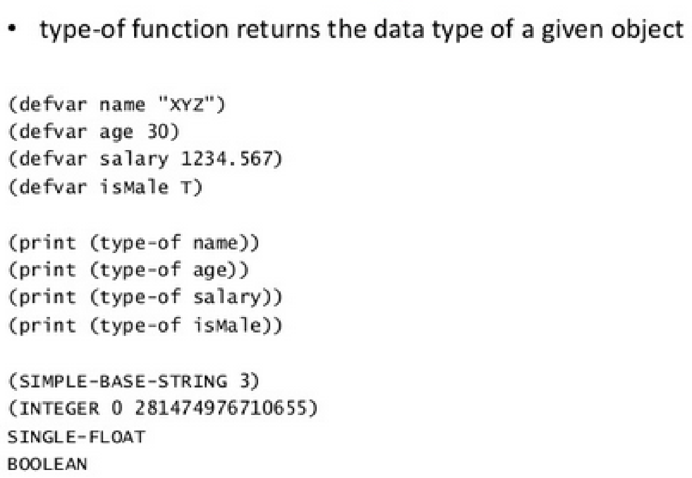
Data Types



- Integer (1, 2, 4, 100)
- Floating point (100.45, 987.67, 12345.6789)
- Boolean (T, Nil)
- String ("Hello World", "Bangalore", "Lisp")

```
1                  100.45             T                  "Hello World"
=> 1               => 100.45          => T               => "Hello World"

2                  12345.6789         Nil
=> 2               => 12345.6789      => Nil

456                (/ 3.0 2)          (= 3 2)
=> 456             => 1.5             => Nil

99999999999999     (/ 22 7)           (= 123.45 123.45)
=>                 ⇒ 22/7             => T
99999999999999
                   (/ 6 4)
                   ⇒ 3/2
```

Data Types

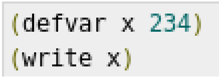


- type-of function returns the data type of a given object

```
(defvar name "XYZ")
(defvar age 30)
(defvar salary 1234.567)
(defvar isMale T)

(print (type-of name))
(print (type-of age))
(print (type-of salary))
(print (type-of isMale))

(SIMPLE-BASE-STRING 3)
(INTEGER 0 281474976710655)
SINGLE-FLOAT
BOOLEAN
```

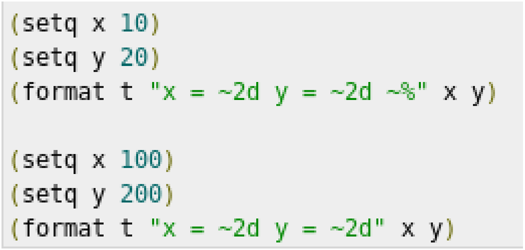
Variables

- **Global Variables**
  Global variables are declared using defvar keyword.



```
(defvar x 234)
(write x)
```

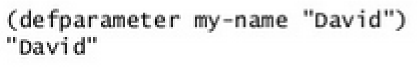
- **Local Variables**
  Global variables are declared using setq keyword.



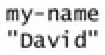
```
(setq x 10)
(setq y 20)
(format t "x = ~2d y = ~2d ~%" x y)

(setq x 100)
(setq y 200)
(format t "x = ~2d y = ~2d" x y)
```

Variables

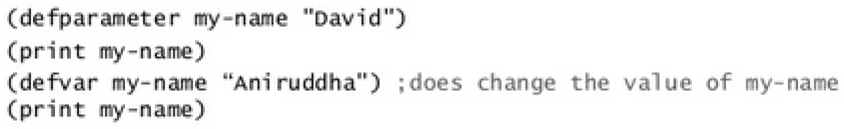
Dynamic Global variables are declared using defparameter keyword.



```
(defparameter my-name "David")
"David"
```

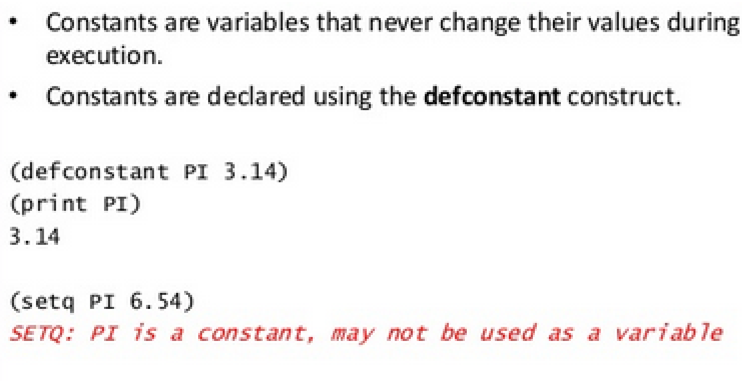


```
my-name
"David"
```



```
(defparameter my-name "David")
(print my-name)
(defvar my-name “Aniruddha") ;does change the value of my-name
(print my-name)
```

Constants

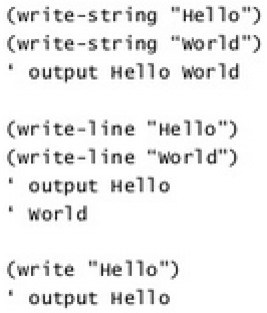


- Constants are variables that never change their values during program execution.
- Constants are declared using the **defconstant** construct.

```
(defconstant PI 3.14)
(print PI)
3.14

(setq PI 6.54)
SETQ: PI is a constant, may not be used as a variable
```

Print

Printing in LISP:



```
(write-string "Hello")
(write-string "World")
' output Hello World

(write-line "Hello")
(write-line "World")
' output Hello
' World

(write "Hello")
' output Hello
```

Thank You