Introduction to LISP

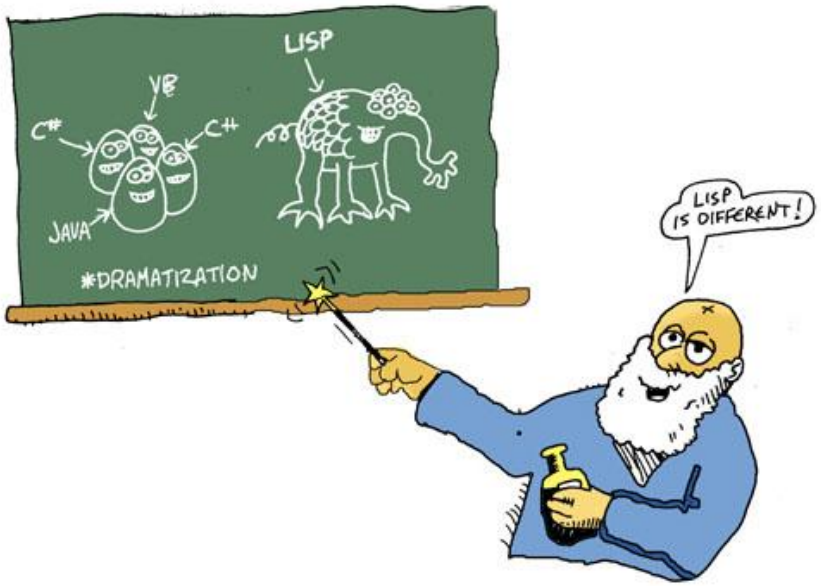
Lab 1

T-622-ARTI Introduction to AI, Spring 2011

Angelo Cafaro, PhD student at CADIA (V.2.16)

Contact: angelo08@ru.is

Mobile: +354 7744221



Getting started

- Great Lisp Tutorial: Part I, Part II and Part III.
- Lisp ➔ Long Incomprehensible String of **Parentheses** ☺ ➔ **Use an advanced editor to indent code using Lisp style**
- Editors:
  - Lispbox 0.7 (Stable Release), an Emacs + SLIME + Common Lisp easy to install package (Win32) or (MAC OS X).
    - **SLIME**: Superior Lisp Interaction Mode for Emacs;
  - Eclipse Lisp Plugin (Win32, MAC OS X and Linux): Dandelion.

Basic Lisp

- Lisp **interpreter**: read-eval-print loop (REPL);

- Everything is an **Expression**:
  - Something evaluated ➔ Submitted to REPL and executed ➔ Returns a Value;
  - An **Expression** is an **Atom** or a **List** of zero or more **expressions** separated by whitespace and enclosed in ();

Basic Lisp types (Atoms expressions)

- **Numbers** (integers, floating-point, complex, …):
  - `27  2  7.519  #C(3.2 2)` ➔ `3.2 +2i`

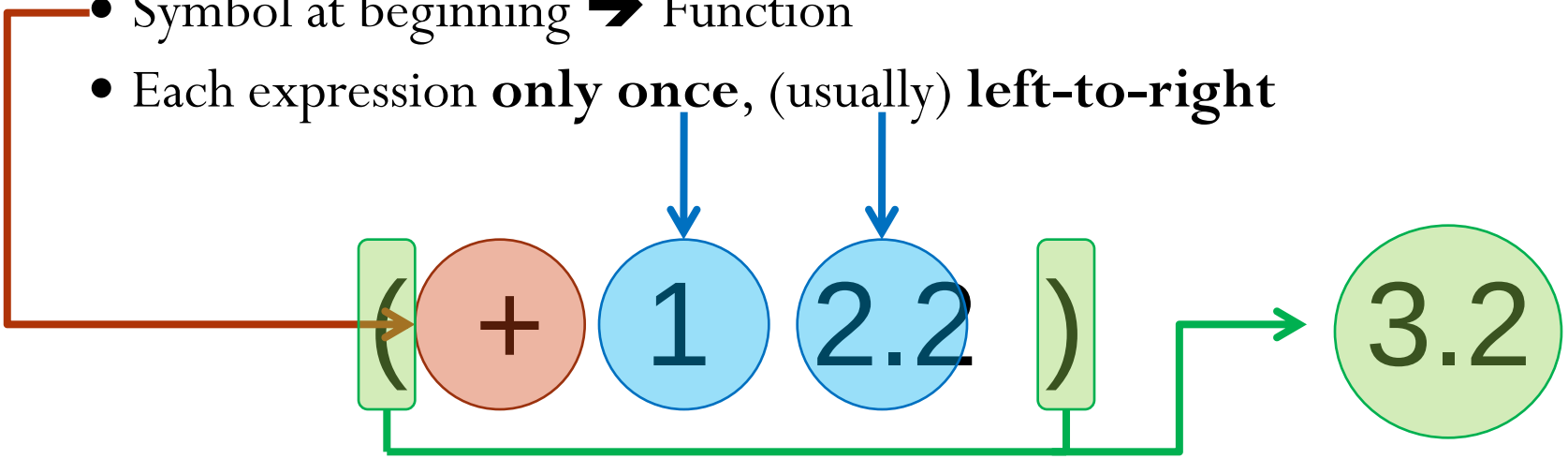
- **Characters** (arrays of chars):
  - `#\x  #\-  #\B`
  - `#\space  #\newline #\\`

- **Strings (not ending with \0)**:
  - `"This is a string"`

Basic Lisp Types (List Expressions)

- Lisp **syntax**, parenthesized prefix notation:
  - `(+ 1 2.2)`
  - `(- 10 (* 4 2))`

- List Evaluation:
  - Symbol at beginning ➔ Function
  - Each expression **only once**, (usually) **left-to-right**



Other Atoms Types

- **Symbols:**
  - **Functions**, at the head of a list:
    - `(print "hello world")`
  - **Variables**, in all the other cases. When evaluated returns the contents of the variable;
    - Setting the value for the variable print to "hello": `(setf print "hello")`
    - Using the new symbol "print": `(print print)`

- **Special Constants:**
  - `nil`: represents false;
  - `t` (or everything but nil): represent true;

Control Structures

- Evaluable lists not observing the ("function rule" seen so far):
  - **Macros** or **Special Forms**.
- The **IF** Special Form:
  - `(if test-expression`
    `then-expression`
    `optional-else-expression)`
  - Example:

```
(if (<= 3 2) (* 3 9) (+ 4 2 3))
```

  - In Pseudocode

```
if (3 <= 2)
then return 3 * 9
else return 4 + 2 + 3
```

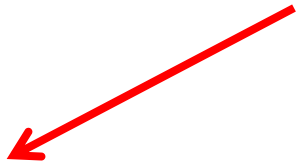
- **Blocks** of Expressions:
  - `(progn expr1 expr2 expr3 ...)`
  - Example:

```
(if (> 3 2)
    (progn
        (print "hello")
        (print "yo")
        (print "whassup?")))
```

Global and LocalVariables

- **Global** Variables:
  - **Syntax:** `(setf variable-symbol expression)`
  - SETF is a Macro and not a Function, evaluates only *expression;*
  - Example: `(setf x (* 3 2))`
  - Use: `x`
- **Local** Variables**:**
  - **Syntax:**

    (let (*declaration1 declaration2 …*)
    expr1
    expr2
    …)
  - **Example**: `( let ( (x 3) ) (print x) (setf x 9) (print x) )` Use **setf** to change the value of a declared one inside a let statement.



Declaration forms:

`var` - A symbol representing the variable. It is initialized to **nil**.
`(var expr)` - A list with the variable symbol and an expression.

Writing Functions

- Created using a the Macro **`defun`**:
- **Syntax**:

```
(defun function-name-symbol (param1 param2 ...)
    expr1
    expr2
    expr3 ... )
```

- Example: `(defun do-hello-world ( )`
  `"Hello, World!")`
- Use: `(do-hello-world)`

List and Symbols as Data

- Lists are Function or Macro calls;
- Symbols are Variable References;
- **Problem**: Skipping the Evaluation → Solution: **Quote**!
- Examples:
  - `(quote (hello world 1 2 3))` → (HELLO WORLD 1 2 3)
  - `(quote my-symbol)` → MY-SYMBOL
  - `(quote (hey yo yo))` → (HEY YO YO)
- Abbreviation:
  - `(quote my-symbol)` or `'my-symbol`
  - `(quote (hey yo yo))` or `'(hey yo yo)`

Lisp Style (cont.)

- **Bad Indentation Example:**
  - `(if  (< (* 3 4) 5) (sin (+ 3 x)) ( - x y ))`

- **Good Indentation Example:**
  - ```
    (if  (< (* 3 4) 5)
         (sin (+ 3 x))
      (- x y))
    ```
    ```

Lisp Style (2)

- **Bad Lisp Style (Declarative):**
  - **(defun do-the-math (x y z)**
    (setf w (+ x y))
    (setf n (* z w))
    (+ x n))
- **Good Lisp Style (Functional):**
  - **(defun do-the-math (x y z)**
    (+ x (* z (+ x y))))