

Symbolic Programming

Joachim De Beule
& Joris Bleys

Artificial Intelligence Laboratory
Brussels

Outline

- session 1:
 - Introduction
 - Installing+learning Lisp
- session 2:
 - Eliza (matching and dialog with a machine)
- session 3:
 - Extensions to basic matching

Introduction

- Program: takes input and uses it to produce some output
- e.g. calculator: 4-key input: "1 + 1 ="
output: "2"
- Lisp is somewhat like a pocket calculator:
- it takes input: "(+ 1 1)"
- and produces output: "2"
- written as:

(+ 1 1) => 2

- Lisp uses parenthesized prefix notation
- evaluation rules:
 - the first element of a list is normally seen as an operation (like “-” or “+”)
 - first all arguments (the rest of the list) are evaluated
 - then the operation is applied
- $(+ 1 2 3 4 5 6 7 8 9 10) \Rightarrow 55$
- $(- (+ 9000 900 90 9) (+ 5000 500 50 5)) \Rightarrow 4444$
- evaluation of a quoted object results in the object itself: $'(+ 1 2) \Rightarrow (+ 1 2)$

Symbolic Computation

- Lisp is more powerful than a pocket calculator
- First, it allows also to manipulate other things besides numbers, e.g. symbols:
- (append '(Joris) '(Bleys))
=> (JORIS BLEYS)

Symbolic Computation

- Second: it allows to define new operations and data structures with which something useful can be done:
- `(defun last-name (name)
 (first (last name)))`
`=> LAST-NAME`
- `(last-name '(Joris Bleys))`
`=> BLEYS`

Some Notes on Symbols

- Lisp does not attach any 'meaning' to symbols or other symbols it manipulates
- to do computations, you need to know about the (over 700!) built-in functions `+`, `append`,...
- symbols are not case sensitive (but...)
- almost any character is allowed in the name of symbols:
`'1+1=2 => 1+1=2`
- `(symbolp 'x) => T` ; `(symbolp 1) => NIL`

Some Notes on Lists

- a list is actually a chain of `cons' cells with the cdr of the last cons cell equal to the empty list '()' or NIL (these are the same):
- $(\text{cons } 1 \ 2) \Rightarrow (1 \ . \ 2)$
- $(\text{cons } 1 \ (\text{cons } 2 \ 3))$
 $\Rightarrow (1 \ . \ (2 \ . \ 3)) = (1 \ 2 \ . \ 3)$
- $(\text{cons } 1 \ (\text{cons } 2 \ (\text{cons } 3 \ \text{NIL})))$
 $\Rightarrow (1 \ . \ (2 \ . \ (3 \ . \ \text{NIL}))) = (1 \ 2 \ 3)$
- NIL (or '()) is also lisp's value for false

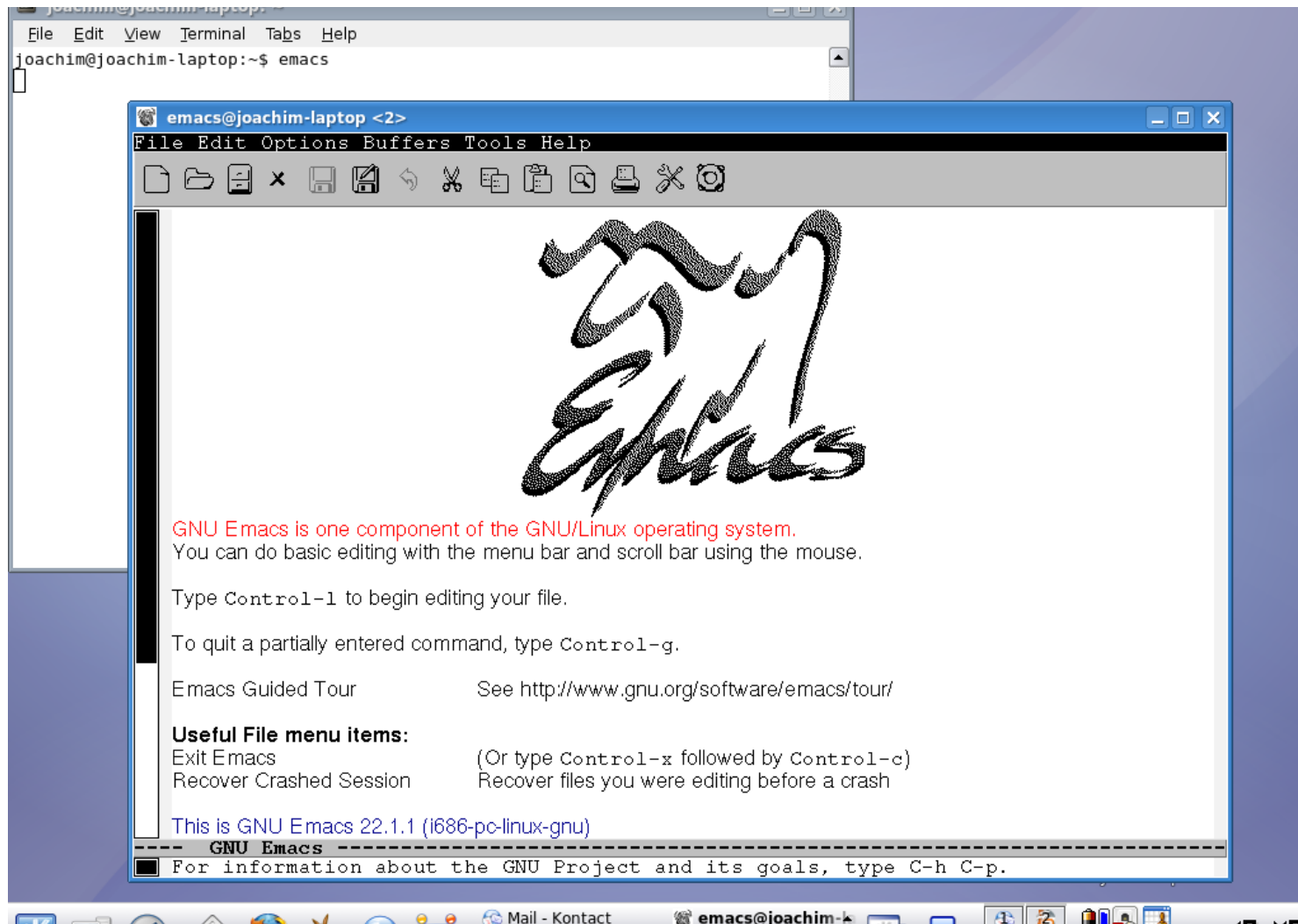
Some Notes on Functions

- `last-name => error`
- `#'last-name => #<function LAST-NAME>`
- `(setf last-name 'BLEYS) => BLEYS`
- `last-name => BLEYS`
- `#'last-name => #<function LAST-NAME>`
- `#'(lambda (x) (+ 1 x))`
`=> #<Interpreted Function ...>`

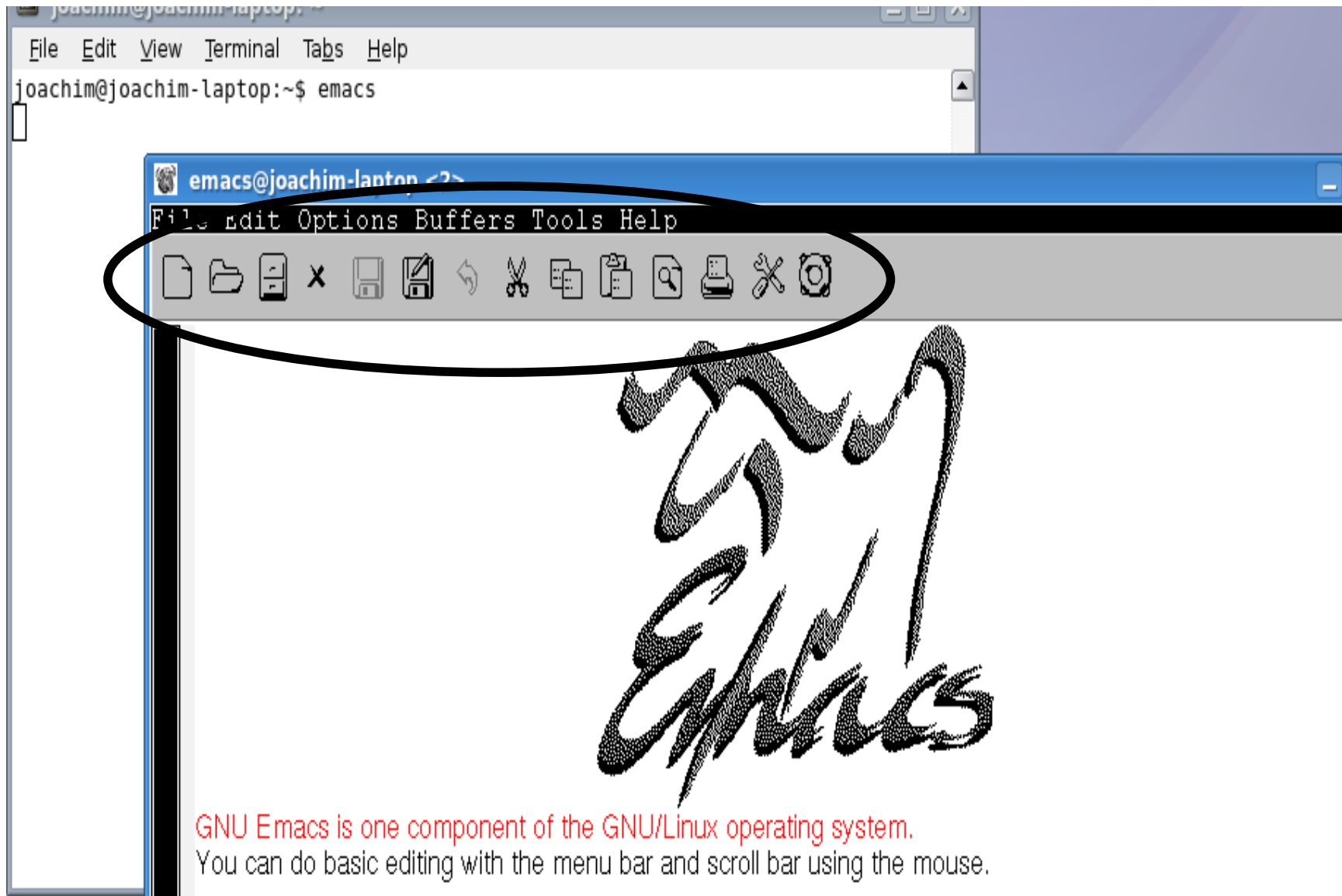
Some Notes on Functions

- `(mapcar #'(lambda (x y) (+ x y 1)) '(1 2 3) '(10 20 30))`
`=> (11 22 33)`
- `(mapcar #'(lambda (x y) (+ x y 1)) '(1 2 3) (10 20 30))`
`=> (12 23 34)`
- `(apply #'(lambda (x y z) (+ x y z)) '(1 2 3)) => 6`
- `(funcall #'(lambda (x y z) (+ x y z)) 1 2 3) => 6`
- `(defun make-n-adder (n) #'(lambda (x) (+ x n)))`
- `(funcall (make-n-adder 5) 3) => 8`

Emacs, The extensible self-documenting text editor.



Emacs Menus



GNU Emacs is one component of the GNU/Linux operating system.
You can do basic editing with the menu bar and scroll bar using the mouse.

Emacs ShortKeys

- Menu -> File -> Visit new file = `C-x C-f'
- `C-...' means Control-key + ...
- `M-...' means Alt-key + ...

Extensible: Emacs is written in a language for which it is itself an interpreter

- It can be programmed and changed almost without limits
- Example: `M-x doctor'
- It can also be programmed to serve as an interface to e.g. Lisp
- => SLIME or the
'Superior Lisp Interaction Mode for Emacs'
- start with `M-x slime' in Emacs

Slime ShortKeys

- C-M-x : evaluate expression
- C-c C-c : compile expression
- M-. : go to source
- M-Tab : completion
- C-c C-d h : lookup in hyperspec
- ...

Variables

- `(setf p '(John Q Public)) => p`
- `p => (JOHN Q PUBLIC)`
- `(let ((x 1))
 (+ x 1)) => 2`

Lists

- $p \Rightarrow (\text{JOHN Q PUBLIC})$
- $(\text{first } p) \Rightarrow \text{JOHN}$
- $(\text{second } p) \Rightarrow \text{Q}$
- $(\text{last } p) \Rightarrow (\text{PUBLIC})$
- $(\text{length } p) \Rightarrow 3$
- $(\text{elt } p \ 0) \Rightarrow \text{JOHN}$
- $(\text{find 'Q } p) \Rightarrow \text{Q}$
- $(\text{find 'R } p) \Rightarrow \text{NIL}$
- $(\text{member 'Q } p) \Rightarrow \text{'(Q PUBLIC)}$
- ...

Some forms needed in the following

- `(defconstant +e+ 2.7182817)`
- `(assoc 2 '((1 . a) (2 . b) (3 . c))) => (2 . b)`
- `(eq '() NIL) => T`
- `(and 1 2 3) => 3` ; `(and 1 NIL 3) => NIL`
- `(char (symbol-name 'Joris) 1) => #\o`

Conditional Forms

- (if test result-1 result-2)
- (cond ((test-1 result-1 ...)
 (test-2 result-2 ...)
 ...
 (t result-n)))

conditional	if form	cond form
(when test a b c)	(if test (progn a b c))	(cond (test a b c))
(unless test x y)	(if (not test) (progn x y))	(cond ((not test) x y))
(and a b c)	(if a (if b c))	(cond (a (cond (b c))))
(or a b c)	(if a a (if b b c))	(cond (a) (b) (c))
(case a (b c) (t x))	(if (eql a 'b) c x)	(cond ((eql a 'b) c) (t x))