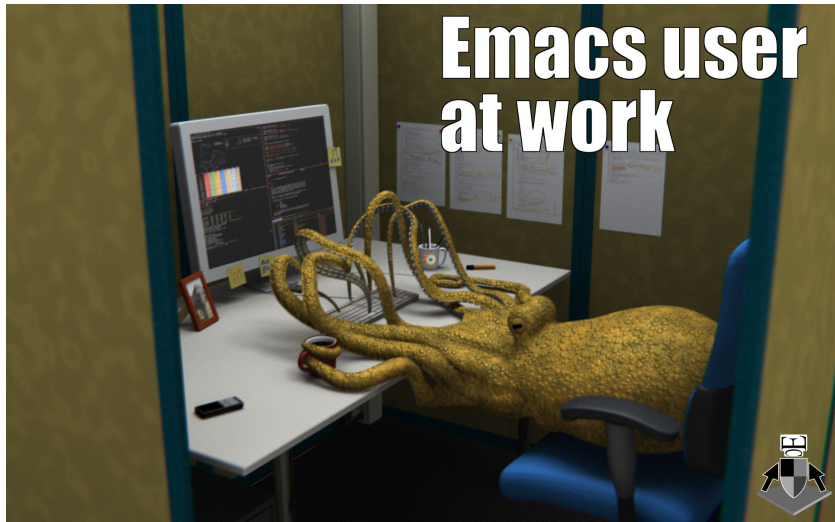


Lisp

'(Emacs Lisp)

Dimitri Fontaine

29 février 2012



Emacs inclue tout le nécessaire pour développer en Elisp, éditeur, documentation, compilateur, interpréteur, debugger interactif. . .

- C-M-x runs the command `eval-defun`
- C-x C-e runs the command `eval-last-sexp`
- L runs the command `dired-do-load`
- M-x `ielm` runs the command `ielm`
- C-j runs the command `eval-print-last-sexp`

La syntaxe Lisp est très simple, tellement simple qu'il faut s'y habituer:

Example (Syntaxe Emacs Lisp)

```
(defun dim:add-my-extra-load-paths (&optional paths)
  "define a list of paths to add to 'load-path'
  and add each of them"
  (let ((dim:paths '("~/dev/emacs/el-get"
                    "~/dev/emacs.d")))
    (dolist (path (or paths dim:paths))
      (setq load-path (cons path load-path))))
  (dim:add-my-extra-load-paths))
```

D'autres exemples:

Example (Syntaxe Emacs Lisp)

```
(defmacro until (cond &rest body)
  '(while (progn ,@body ,cond)))
```

```
(defmacro when-running-debian (&rest body)
  "eval body only when running under debian"
  '(when (equal
    (lsb-release "Distributor ID") "Debian")
    ,@body))
```

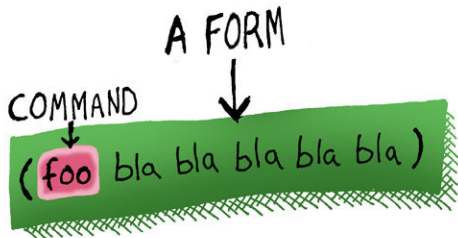
`http://www.lisperati.com/`

A LIST IN LISP



`(Bla bla bla bla bla)`

`http://www.lisperati.com/`

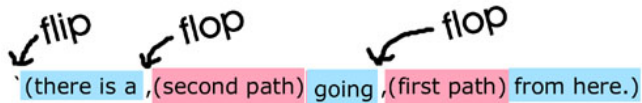


<http://www.lisperati.com/>



`http://www.lisperati.com/`

`(there is a ,(second path) going ,(first path) from here.)`



Pourquoi tant de parenthèses?

Est-ce vraiment spécifique à Lisp?

Example (java)

```
closeButton.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent event) {  
        System.exit(0);  
    }  
});
```

Example (C)

```
cmd->objectname =  
    pstrdup(NameStr(  
        ((Form_pg_conversion) GETSTRUCT(tup))->conname));
```

Lisp a été découvert en 1960 et continue d'évoluer.

Pour les curieux: <http://www.gigamonkeys.com/book/>

- Inclue les *DSL* via les macros (cf. `loop`)
- CLOS pour le développement object
- Grande famille (Common Lisp, Scheme, Emacs Lisp, ...)
- Variables dynamiques et closures lexicales
- Seule famille de langage à proposer les *macros*
- Typage dynamique
- De nombreux types disponibles, pas seulement les *listes*
- Gestion d'exception très avancée
- Continuations

Emacs inclue tout le nécessaire pour développer en Elisp, éditeur, documentation, compilateur, interprêteur, debugger interactif. . .

- Environnement interactif complet
- Documentation de référence
- Portabilité étendue (Linux, Windows, MacOSX, Solaris, etc. . .)
- Interprêteur et compilateur (byte code)
- Intégration système (synchrone et asynchrone avancée)
- API riche, des milliers d'extensions (c1.e1)
- 36 ans d'histoire avec rétro compatibilité