$$\mathcal{E}\mathcal{L} = \frac{mc^2}{\sqrt{1-\frac{q^2}{c^2}}}$$

FUNKCIONÁLNÍ A LOGICKÉ PROGRAMOVÁNÍ

4. LISP: PROMĚNNÉ, BLOKY, ZADÁNÍ SEMESTRÁLNÍ PRÁCE

2011 Jan Janoušek
MI-FLP

OPP A
PRA HA PRA GUE PRA GA PRA G
EVROPSKÁ UNIE

Evropský sociální fond
Praha & EU:
Investujeme do vaší budoucnosti

Variables and constants in Lisp

Global variables and constants

Variables:

```
>(defparameter *global* 17)
>(defvar *global* 17)
```

global variables are conventionally written enclosed in *

Constants:

```
> (defconstant glob-const 11)
```

Local variables

- Local variables are created in functions:

```
>(defun foo (x y z) (+ x y z))
```

- and in statements of cycles, etc… :

```
>(dotimes (x 10) (format t "~d " x))
```

- operator `let`: `(let (variable*) body-form*)`

```
>(let ((a 10) (b 12))
                   (- b a)
                   (* a b))
120
```

Local variables - example

```
(defun foo (x)
  (format t "Parameter: ~a~%" x) ; |<------ x is arg
  (let ((x 2))                        ; |
    (format t "Outer LET: ~a~%" x) ; | |<---- x is 2
    (let ((x 3))                      ; | |
      (format t "Inner LET: ~a~%" x)) ; | | |<--x=3
    (format t "Outer LET: ~a~%" x))   ; | |
  (format t "Parameter: ~a~%" x))     ; |
```

```
CL-USER> (foo 1)
Parameter: 1
Outer LET: 2
Inner LET: 3
Outer LET: 2
Parameter: 1
NIL
```

Assignment

- Macro `setf`:

```
(setf x 1)
(setf y 2)

(setf x 1 y 2)


(incf x)      === (setf x (+ x 1))
(decf x)      === (setf x (- x 1))
(incf x 10)   === (setf x (+ x 10))
```

More on functions - parameters

Optional arguments of functions

- Optional parameters `&optional`:

```
(defun foo (a b &optional c d) (list a b c d))

(foo 1 2)     ==> (1 2 NIL NIL)
(foo 1 2 3)   ==> (1 2 3 NIL)
(foo 1 2 3 4) ==> (1 2 3 4)


(defun foo (a &optional (b 10)) (list a b))

(foo 1 2) ==> (1 2)
(foo 1)   ==> (1 10)
```

Rest and keywords parameters

- Rest parameters as a list `&rest`:

```
(defun + (&rest numbers) ...)
```

- Optional keywords parameters `&keyword`:

```
(defun foo (&key a b c) (list a b c))

(foo)                   ==> (NIL NIL NIL)
(foo :a 1)              ==> (1 NIL NIL)
(foo :b 1)              ==> (NIL 1 NIL)
(foo :c 1)              ==> (NIL NIL 1)
(foo :a 1 :c 3)         ==> (1 NIL 3)
(foo :a 1 :b 2 :c 3) ==> (1 2 3)
(foo :a 1 :c 3 :b 2) ==> (1 2 3)
```

Function Return Values

- `return-from`:

```
(defun foo (n)
  (dotimes (i 10)
    (dotimes (j 10)
      (when (> (* i j) n)
        (return-from foo (list i j))))))
```

Note. **RETURN-FROM**s are used much less frequently in Lisp than return statements in C-derived languages

Blocks

Simple sequencing

block	returns
`(prog1` ***form-1 form-2*** **…** ***form-n*** `)`	***form-1***
`(prog2` ***form-1 form-2*** **…** ***form-n*** `)`	***form-2***
`(progn` ***form-1 form-2*** **…** ***form-n*** `)`	***form-n***

block and return-from once more

(block *name form-1 form-2* ... *form-n*)

(return-from *name val*)

Block is like progn, returning the last value

```
> (block test 1 2 3 4 5)
5
> (block test 1 2 (return-from test 33) 4 5)
33
> (block nil 1 2 (return 33) 4 5)
33
> (block a (+ (block b 1 2 (return-from b 33) 4)
              (block c 5 6 (return-from a 77) 8)))
77
```

Introducing semestral work follows…