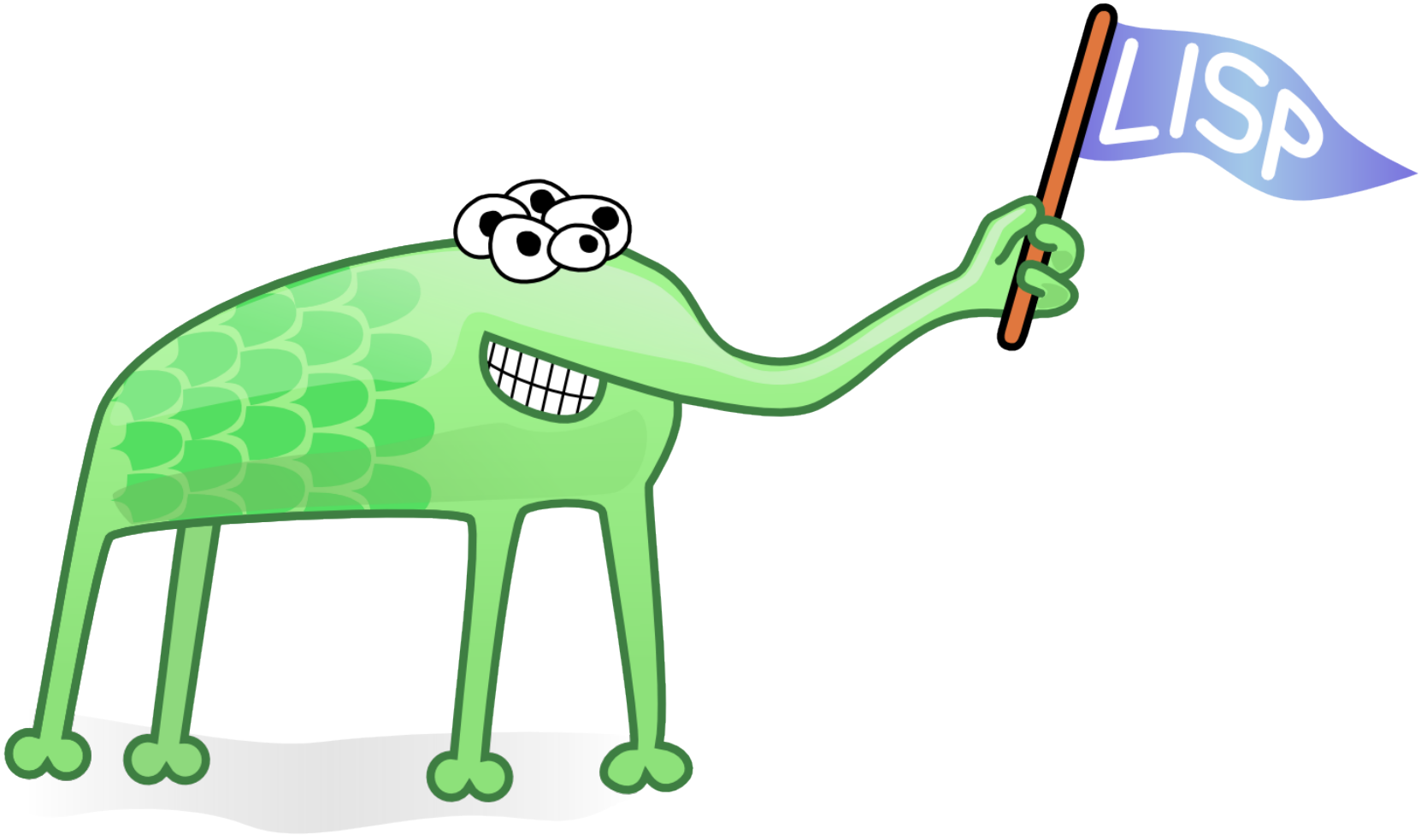
Sideeffekter og makroer i Lisp



PVV-kurs 18. mars 2010

Oversikt over kurset

(**Del 4**: Imperativ programmering (utsatt fra forrige kurs))

(**Del 5**: Makroer)

(**Del 6**: Eksempel: Postmodern)

Lisp 4: Imperativ programmering

Eirik Alderslyst Nygaard Øystein Ingmar Skartsæterhagen

Programvareverkstedet

18. mars 2010

(Funksjonell programmering)

(Alle eksempler vi har vist til nå har vært skrevet i *funksjonell* stil)

(Alt en funksjon gjør er å produsere en returverdi, og denne verdien avhenger kun av argumentene til funksjonen)

(Hvis en funksjon kalles flere ganger med samme argumenter, returnerer den det samme hver gang)

(Dette gjør programmene ryddige, men enkelte ting blir vanskelige å implementere)

(SETF)

(setf *ting verdi*)

(SETF erstatter verdien i en ting med en ny verdi)

(SETF-eksempel)

```
CL-USER> (defvar *x* 5)
*X*
CL-USER> *x*
5
CL-USER> (setf *x* (+ *x* 3))
8
CL-USER> *x*
8
CL-USER> (setf *x* (list 1 2 3))
(1 2 3)
CL-USER> (setf (first *x*) 5)
5
CL-USER> *x*
(5 2 3)
CL-USER>
```

(READ og PRINT)

(READ brukes til å lese inn et lisputtrykk. Uttrykket vil bli parset og returnert som vanlige lisp-objekter)

(PRINT skriver ut lisp-objekter i en leselig form (den samme representasjonen som READ leser inn)

```
CL-USER> (first (read))
(1 2 3)
1
CL-USER> (+ (read) (read))
40 2
42
CL-USER> (print (list 1 2 3))
(1 2 3)
(1 2 3)
CL-USER>
```

(FORMAT)

(FORMAT brukes til å skrive ut formatert tekst)

(format *destinasjon formatstreng argumenter*)

(Destinasjonen kan være en strøm, T eller NIL)

```
CL-USER> (format t "Hello, world!")
Hello, world!
NIL
CL-USER> (format nil "Hello, world!)
"Hello, world!"
CL-USER> (format *standard-output* "Hello, world!")
Hello, world!
NIL
CL-USER>
```

(FORMAT med enkle argumenter)

```
CL-USER> (format t "Dagens tall er: ~D~%" (random 10))
Dagens tall er: 4
NIL
CL-USER> (format t "Jeg heter ~S og er ~D år gammel~%"
"John" 82)
Jeg heter "John" og er 82 år gammel
NIL
CL-USER> (format t "Dagens gullkorn er: ~A~%"
"Sometimes the appropriate response to reality is to go
insane.")
Dagens gullkorn er: Sometimes the appropriate response to
reality is to go insane.
NIL
CL-USER>
```

(FORMAT kan skrive ut tall på forskjellige måter)

```
CL-USER> (format t "~R~%" 1501205)
one million five hundred one thousand two hundred five
NIL
CL-USER> (format t "~:R ~:R ~:R~%" 1 2 5)
first second fifth
NIL
CL-USER> (format t "~@R ~@R ~@R~%" 1 130 1259)
I CXXX MCCLIX
NIL
CL-USER>
```

(FORMAT lar deg skrive ut tall med sine engelske navn eller som det romerske tallsystemet)

(FORMAT og flyttall)

```
CL-USER> (format t "~F~%" 3.14159)
3.14159
CL-USER> (format t "~,2F~%" 3.14159)
3.14
CL-USER> (format t "~,2E~%" 123456)
1.23e+5
CL-USER> (format t "~,2,3F" 2.3)
2300.00
CL-USER>
```

(FORMAT med lister og litt mer)

```
CL-USER> (format t "~{~A, ~}" '(1 2 3))
1, 2, 3,
NIL
CL-USER> (format t "~{~A~^, ~}" '(1 2 3))
1, 2, 3
NIL
CL-USER> (format t "~{~{~10@<~:(~r~)~>~}~%~}"
                   '((1 2) (3 4) (5 6)))
One        Two
Three      Four
Five       Six
NIL
```

(FORMAT lar deg skrive ut og formatere lister som du selv vil)

(Iterasjon)

(Alt som kan uttrykkes iterativt kan også uttrykkes rekursivt)

(Har sett: Minne- og hastighetsfordelene med iterasjon kan også oppnås med halerekursjon)

(Iterasjon er altså egentlig ikke nødvendig)

(Men det finnes likevel iterasjonsoperatorer)

(Gjenta noe et antall ganger: DOTIMES)

```
CL-USER> (dotimes (i 10) (format t "~D" i))
0123456789
NIL
CL-USER>
```

(dotimes (*variabel ganger*) *ting-å-gjøre*)

(Iterer over en liste: DOLIST)

```
CL-USER> (dolist (a '(foo bar baz)) (format t "~A " a))
FOO BAR BAZ
NIL
CL-USER>
```

(dolist (*variabel liste*) *ting-å-gjøre*)

(Generell iterasjon: LOOP)

(LOOP: sveitsisk arméiterasjonsform)

(LOOP kan brukes til å skrive for-løkker, while-løkker, foreach-løkker og andre rare løkker)

(LOOP er et eget «lite» programmeringsspråk)

(LOOP FOR)

```
CL-USER> (loop for num in '(1 2 3)
               do (format t "Item: ~A~%" num))
Item: 1
Item: 2
Item: 3
NIL
CL-USER>
```

```
CL-USER> (loop for num from 1 to 3
               do (format t "Item: ~A~%" num))
Item: 1
Item: 2
Item: 3
NIL
CL-USER>
```

(Generering av verdier med LOOP)

```
CL-USER> (loop for num in '(1 2 3)
               sum num)
6
CL-USER>
```

```
CL-USER> (loop for num in '(1 5 2)
               maximize num)
5
CL-USER>
```

(Generering av lister med LOOP)

(Loop kan gjøre noe med hvert element i en liste og returnere en ny liste med resultatene)

```
(loop for num in '(1 2 3)
  collect (* 2 num))
⇝ (2 4 6)
```

(Det er også mulig å traversere flere lister samtidig)

```
(loop
  for x in '(1 2 3)
  for y in '(10 20 30)
  collect (+ x y))
⇝ (11 22 33)
```

(Betingelser i LOOP)

```
CL-USER> (loop for x from 1 to 5
           if (evenp x)
             do (format t "~A is even~%" x)
           else
             do (format t "~A is odd~%" x))
1 is odd
2 is even
3 is odd
4 is even
5 is odd
NIL
CL-USER>
```

(Vi kan få deler av løkken til å bli utført kun på visse betingelser)

(Og mye mye mye mer)

(Oppgaver)

1. Skriv en ikkerekursiv funksjon som beregner fakultet av et heltall, ved hjelp av `LOOP`.
2. Skriv en funksjon `PRINT-LIST` som tar inn en liste og skriver den ut med ett element på hver linje, nummerert med romertall:

```
CL-USER> (print-list '(foo bar baz))
       I. FOO
      II. BAR
     III. BAZ
NIL
CL-USER>
```

(Løsningsforslag: !)

```
(defun ! (n)
  (let ((n! 1))
    (loop for i from 2 to n
       do (setf n! (* n! i)))
    n!))
```

```
(defun ! (n)
  (reduce #'* (loop for i from 2 to n
      collect i)))
```

(Løsningforslag: PRINT-LIST)

```
(defun print-list (list &optional (stream t))
  (loop
     for i from 1
     for elem in list
     do (format stream "~8<~@R~>. ~A~%" i elem)))
```