

Sec. III  
Ch. 10

## SECTION III

### LISP IS HACKING

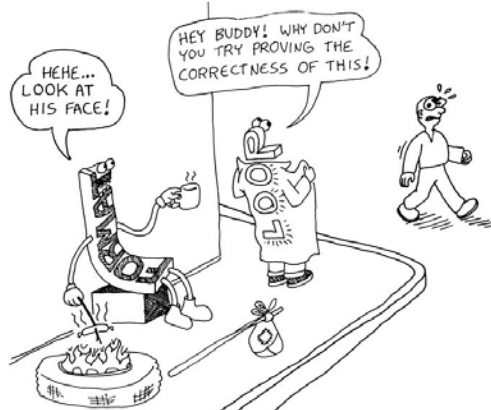
Loop –  
Ch. 10

Format –  
Ch. 11



Copyright © 2011 by Conrad Barski, M.D.

### Loop and Format: the Seedy Underbelly of Lisp



Copyright © 2011 by Conrad Barski, M.D.

### Looping with the Loop Command Ch. 10

- Example: the LOOP Macro

```
> (loop for i
      below 5
      sum i)
→ 10
```



Copyright © 2011 by Conrad Barski, M.D.

### Loop Magic Tokens

- for** allows you to declare a variable that iterates through a range of values.
- By *default*, **for** will count through the integers starting at zero.
- below** tells the **for** construct to halt when it reaches the specified value, excluding the value itself.
- sum** adds together all values of a given expression and makes the loop return that number.

### Exiting a Loop Early

```
> (loop for i
      from 0
      do (print i)
      when (= i 5)
      return 'falafel)
```

```
0
1
2
3
4
5
FALAFEL
```

## Periodic Table of the Loop Macro

<b>simple loop</b> Loop do end	AN ASTERISK* MEANS A WORD CAN HAVE AN "ING" FORM (SO "SUM" AND "SUNNING" ARE BOTH LEGAL) CREATE A LOCAL		
<b>do*</b> Loop do x below y do end	NOTHING & BREAKING OUT OF LOOPS HASH TABLES		
<b>repeat</b> Loop repeat n do end	<b>named</b> Loop named name do end	<b>using</b> Loop using name do end	<b>being</b> Loop being name do end
<b>return</b> Loop return do end	<b>return-from</b> Loop return-from do end	<b>the</b> Loop the do end	<b>each</b> Loop each do end
<b>initially</b> Loop initially do end	<b>while</b> Loop while do end	<b>hash-keys</b> Loop hash-keys do end	<b>hash-key</b> Loop hash-key do end
<b>finally</b> Loop finally do end	<b>until</b> Loop until do end	<b>hash-values</b> Loop hash-values do end	<b>hash-value</b> Loop hash-value do end

Copyright © 2011 by Conrad Barski, M.D.

"FOR" LOOPING				BUILDING CONDITIONS		EXTRACTING A RESULT	
<b>for</b> Loop for do end	<b>as</b> Loop as do end	<b>in</b> Loop in do end	<b>on</b> Loop on do end	<b>across</b> Loop across do end	<b>into</b> Loop into do end	<b>if</b> Loop if do end	<b>count*</b> Loop count do end
<b>by</b> Loop by do end	<b>from</b> Loop from do end	<b>to</b> Loop to do end	<b>always</b> Loop always do end	<b>and</b> Loop and do end	<b>maximize*</b> Loop maximize do end	<b>unless</b> Loop unless do end	<b>sum*</b> Loop sum do end
<b>then</b> Loop then do end	<b>upfrom</b> Loop upfrom do end	<b>upto</b> Loop upto do end	<b>never</b> Loop never do end	<b>else</b> Loop else do end	<b>append*</b> Loop append do end	<b>end</b> Loop end do end	<b>nconc*</b> Loop nconc do end
<b>downfrom</b> Loop downfrom do end	<b>downto</b> Loop downto do end	<b>thereis</b> Loop thereis do end	<b>end</b> Loop end do end	<b>nconc*</b> Loop nconc do end	<b>nconc*</b> Loop nconc do end	<b>nconc*</b> Loop nconc do end	<b>nconc*</b> Loop nconc do end

Copyright © 2011 by Conrad Barski, M.D.

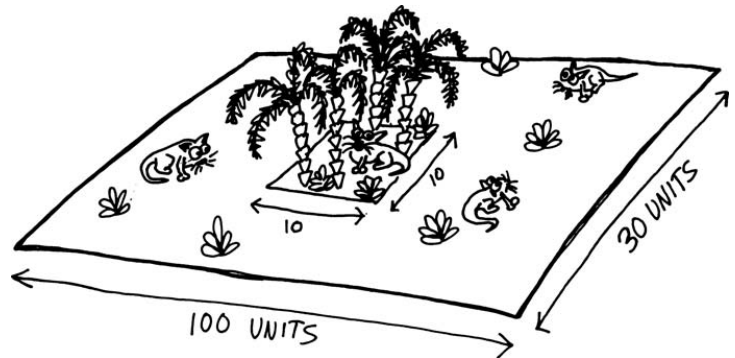
## Collect in Loops

```
> (loop for x below 10
      for y
      below 10
      collect (+ x y))
→ (0 2 4 6 8 10 12 14 16 18)
```

```
> (loop for i
      from 0
      for day
      in '(monday tuesday
           wednesday thursday
           friday saturday
           sunday)
      collect (cons i day))
→ ((0 . MONDAY) (1 . TUESDAY) (2 . WEDNESDAY)
   (3 . THURSDAY) (4 . FRI DAY) (5 . SATURDAY)
   (6 . SUNDAY))
```

## Evolution Game

Create a world with plants and animals, and let them evolve for millions of time units!  
Yes – using **Loop**!



Copyright © 2011 by Conrad Barski, M.D.

## Loop Summary

- The **loop** macro can be simple
- The **loop** macro can be complex
- Creating a game using **Loop**!
- Next: Ch. 11: **Format**

## Printing Text with the Format Function - Ch. 11

THE CONTROL STRING

A CONTROL SEQUENCE

"Add onion rings for only (~\$) dollars more!" 1.5) A VALUE PARAMETER

(format t "Add onion rings for only (~\$) dollars more!" 1.5)

THE DESTINATION PARAMETER

nil = just create a string  
t = print to console  
stream = output to stream

Copyright © 2011 by Conrad Barski, M.D.

## The Destination Parameter

The first parameter to the format function is the *destination* parameter, which tells **format** *where to send the text* it generates.

Here are its possible values:

- **nil** - Don't print anything; just **return the value as a string**.
- **t** - Print the value **to the console**. In this case, the function just returns nil as a value
- **stream** - Write the data to an **output stream** (covered in Chapter 12).

## The Control String

- The second parameter to the **format** function is a *control string*, which controls the text formatting.
- The format function's power lies in the control string.
- By default, the text in this string is simply printed as output.
- **Control sequences** in this string affect the format of the output
- `(format nil "Add onion rings for only ~$ dollars more!" 1.5)`  
→ Add onion rings for only 1.50 dollars more!
- The control sequence `~$` indicates a *monetary floating-point value*.
- Every control sequence recognized by the format function begins with the **tilde (~)** character.

```
> (prin1 "foo")
→ "foo"
> (princ "foo")
→ foo
```

**format** with the `~s` and `~a` control sequences to produce the same behavior as `prin1` and `princ`.

## Control Sequences for Formatting Integers

- First, we can use format to display a number using a different base. For instance,
- hexadecimal (base-16) with the `~x` control sequence:  
▪ `> (format t "The number 1000 in hexadecimal is ~x" 1000)`  
The number 1000 in hexadecimal is 3E8
- binary (base-2) using the `~b` control sequence:  
▪ `> (format t "The number 1000 in binary is ~b" 1000)`  
The number 1000 in binary is 1111101000
- display as a decimal (base-10) number, using the `~d` control sequence:  
▪ `> (format t "The number 1000 in decimal is ~d" 1000)`  
The number 1000 in decimal is 1000

```
> (defun random-animal ()
  (nth (random 5)
    '("dog" "tick" "tiger"
      "walrus" "kangaroo")))
→ RANDOM-ANIMAL

> (random-animal)
→ "walrus"

> (loop repeat 10
  do
    (format t
      "~5t~a ~15t~a
~25t~a~%"
      (random-animal)
      (random-animal)
      (random-animal)))
```

kangaroo	tick	dog
dog	walrus	walrus
walrus	tiger	tiger
walrus	kangaroo	dog
Kangaroo	tiger	dog
tiger	walrus	kangaroo
tick	dog	tiger
kangaroo	tick	kangaroo
tiger	dog	walrus
kangaroo	kangaroo	tick

## Format Directives

- | Directive | Interpretation  | Directive | Interpretation                     |
|-----------|-----------------|-----------|------------------------------------|
| ~%        | new line        | ~X        | hexadecimal integer                |
| ~&        | fresh line      | ~bR       | base-b integer                     |
| ~         | page break      | ~R        | spell an integer                   |
| ~T        | tab stop        | ~P        | plural                             |
| ~<        | justification   | ~F        | floating point                     |
| ~>        | terminate ~<    | ~E        | scientific notation                |
| ~C        | character       | ~G        | ~F or ~E, depending upon magnitude |
| ~(        | case conversion | ~\$       | monetary                           |
| ~)        | terminate ~(    | ~A        | legibly, without escapes           |
| ~D        | decimal integer | ~S        | READably, with escapes             |
| ~B        | binary integer  | ~~        | ~                                  |
| ~O        | octal integer   |           |                                    |

## Format Examples

- <http://www.gigamonkeys.com/book/a-few-format-recipes.html>
- <http://psg.com/~dlamkins/sl/chapter24.html>
- see also CLQR

## Summary Printing with Format

### (format dest string args)

The `format` function normally returns `NIL`, but as a **secondary effect** it causes things to be **written on the display or to a file**.

- 1<sup>st</sup> parameter: **dest** the symbol `T` will print/write **to the display**, `NIL` will **return** the resulting output  
`STREAM` will send to a stream, e.g. a file
- 2<sup>nd</sup> parameter: **string** the format control string.
  - `~%`: move to a new line
  - `~&`: move to a new line unless it knows it's already at the beginning of a new line.
  - `~S`: inserts the printed representation of a Lisp object into the message that Format prints.
- 3<sup>rd</sup> parameter: **args** zero or more variables and/or expressions to be printed where specified in the format string.

```
> (format t "Hi, mom! ~% This is ~S" 'Tom)
→ Hi, mom!
   This is Tom
   Nil
```