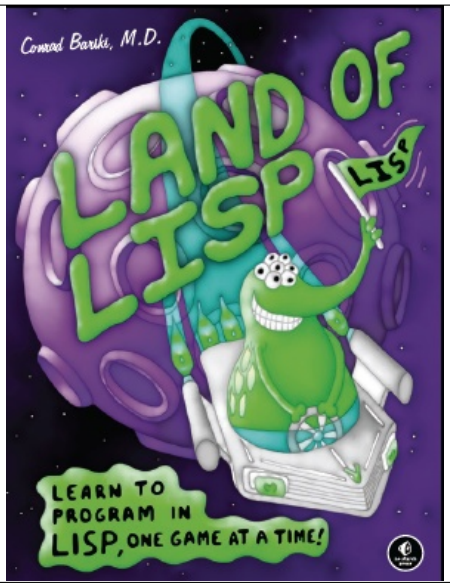


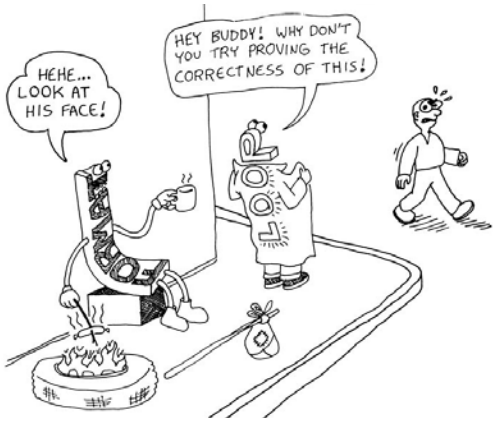
Sec. III
Ch. 10

SECTION III
LISP IS HACKING

Loop – Ch. 10

Format – Ch. 11

Loop and Format: the Seedy Underbelly of Lisp



Looping with the Loop Command Ch. 10

- Example: the LOOP Macro

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

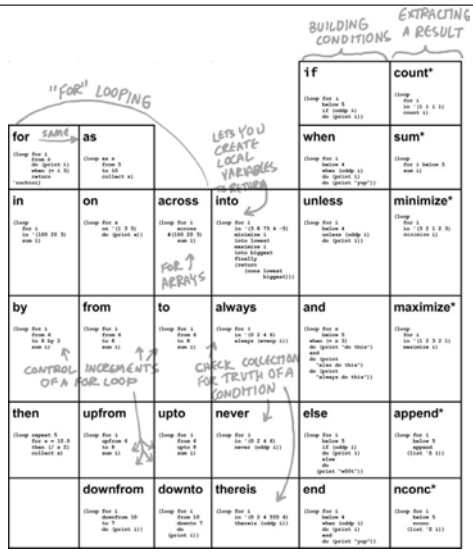
Loop Magic Tokens

- **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



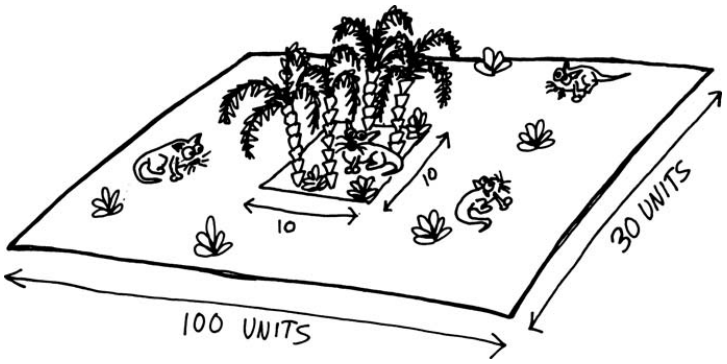
Collect in Loops

```
> (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 . FRIDAY) (5 . SATURDAY)
   (6 . SUNDAY))
```


Evolution Game

Create a world with plants and animals, and let them evolve for millions of time units!

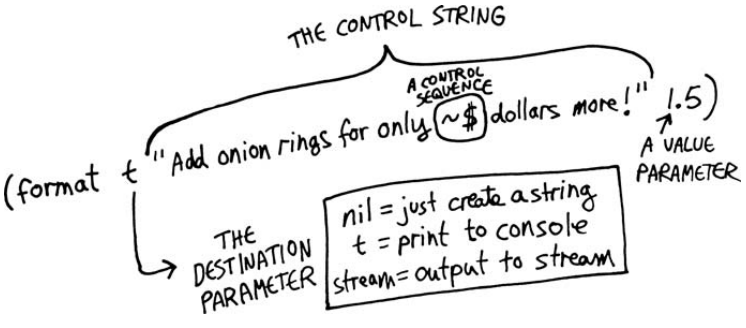
Yes – using **Loop**!



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 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
~%	new line
~&	fresh line
~|	page break
~T	tab stop
~<	justification
~>	terminate ~<
~C	character
~(	case conversion
~)	terminate ~(
~D	decimal integer
~B	binary integer
~O	octal integer

Directive	Interpretation
~X	hexadecimal integer
~bR	base-b integer
~R	spell an integer
~P	plural
~F	floating point
~E	scientific notation
~G	~F or ~E, depending upon magnitude
~$	monetary
~A	legibly, without escapes
~S	READably, with escapes
~~	~

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.

- 1st 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
- 2nd 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.
- 3rd 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
```