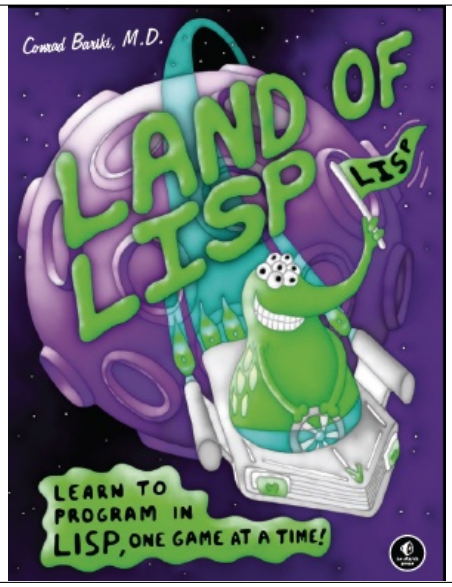
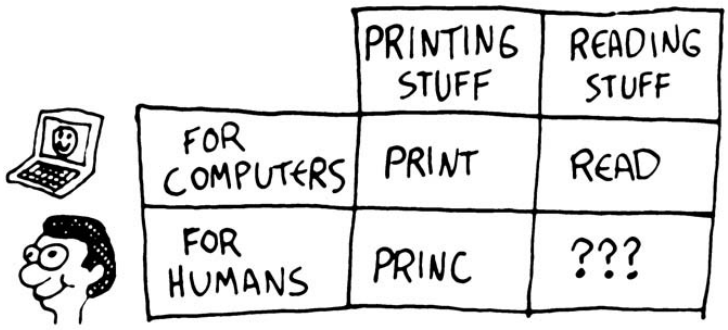


Chapter 6

Interacting with the World: Reading and Printing in Lisp

- Creating a better, more robust interface for the wizard's game.
- Using the most basic of all user interfaces, the *command-line interface*.



PRINT Examples

```
>(print '3)
3                ; An integer
>(print '3.4)
3.4              ; A float
>(print 'foo)
FOO              ; A symbol

;;  Symbols may be printed in ALL CAPS,
;;  since the reader in Common Lisp
;;  can convert to upper case (maybe default).

>(print '"foo") =>
"foo"            ; A string
>(print '#\a)
#\a              ; A character
```


PRINT and PRIN1

```
> (progn    (print "this")          ; with newlines
            (print "is")
            (print "a")
            (print "test"))
"this"
"is"
"a"
"test"

> (progn    (prin1 "this")          ; no newlines
            (prin1 "is")
            (prin1 "a")
            (prin1 "test"))
"this""is""a""test"
```


Print - Uses Keyword Parameters

```
<function> foo &key :stream :escape :radix :base :circle
:pretty :level :length :case :gensym :arr

;default stream      *standard-output*

Print foo to stream
prin1     - readably,
print     - readably between a newline and a space,
pprint    - readably after a newline, or
princ     - human-readably without extra characters

prin1,  print and  princ return foo.

Print foo
prin1-to-string   string readably or
princ-to-string   human-readably

terpri     - print a new line
fresh-line - print a new line unless already at the beginning
of one
```

PRINC – Human Readable

```
>(princ '3)
3
>(princ '3.4)
3.4
>(princ 'foo)
FOO
>(princ "foo")
foo
>(princ '#\a)
a

;;;  Omits quote marks and codes for
;;;     characters, etc.
```


Hello – with PRINT and READ

```
> (defun say-hello ()
    (print "Please type your name:")
    (let ((name (read)))
        (print "Nice to meet you, ")
        (print name)))
SAY-HELLO.
> (say-hello)
"Please type your name:" "bob"
"Nice to meet you, "
"bob"
```


Hello v. 2, PRINC and READ-LINE

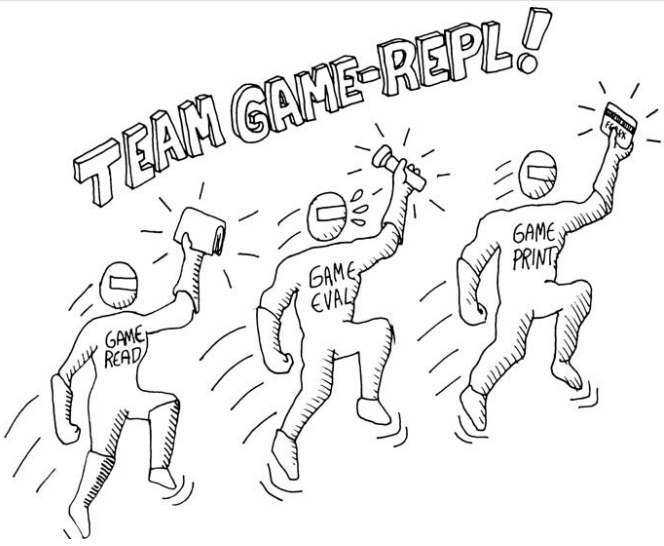
```
> (defun say-hello ()
    (princ "Please type your name:")
    (let ((name (read-line)))
        (princ "Nice to meet you, ")
        (princ name)))
SAY-HELLO

> (say-hello)
Please type your name: Bob O'Malley
Nice to meet you, Bob O'Malley
```


Adding a Custom Interface to Our Game Engine

```
> (defun game-repl ()
    (loop (print (eval (read)))) ;look familiar?
GAME-REPL

> (game-repl)
(look)
(YOU ARE IN THE LIVING-ROOM. A WIZARD
  IS SNORING LOUDLY ON THE COUCH. THERE
  IS A DOOR GOING WEST FROM HERE. THERE
  IS A LADDER GOING UPSTAIRS FROM HERE.
  YOU SEE A WHISKEY ON THE FLOOR.)
```

Custom REPL Functions

Wizard's Game REPL Code

- http://www2.hawaii.edu/~nreed/ics313/assignments/wizards_part2.lisp

- Common Lisp, the Language, 2nd ed.

http://www.cs.cmu.edu/Groups/AI/html/cltl/clm/index.html


```
;  wizards_game part 2
(defun game-repl ()
    (let ((cmd (game-read)))
        (unless (eq (car cmd) 'quit)
            (game-print (game-eval cmd))
            (game-repl))))

(defun game-read ()
    (let ((cmd (read-from-string
                  (concatenate 'string "("
                                       (read-line) ")"))))
         (flet ((quote-it (x)
                   (list 'quote x)))
             (cons (car cmd)
                   (mapcar #'quote-it (cdr cmd))))))

(defparameter *allowed-commands*
      '(look walk pickup inventory))

(defun game-eval (sexp)    ; s-expression
    (if (member (car sexp) *allowed-commands*)
        (eval sexp)
        '(i do not know that command.)))
```



```
(defun tweak-text (lst caps lit)
  (when lst
    (let ((item (car lst))
          (rest (cdr lst)))
      (cond ((eql item #\space)
                  (cons item (tweak-text rest caps lit)))
            ((member item '(#\! #\? #\.))
                  (cons item (tweak-text rest t lit)))
            ((eql item #\") (tweak-text rest caps (not lit)))
            (lit (cons item (tweak-text rest nil lit)))
            (caps (cons (char-upcase item)
                        (tweak-text rest nil lit)))
            (t (cons (char-downcase item)
                        (tweak-text rest nil nil)))))))

(defun game-print (lst)
    (princ (coerce (tweak-text (coerce (string-trim "() "
             (prin1-to-string lst)) 'list) t nil) 'string))
    (fresh-line))

; [Function]      coerce   object   result-type
; [Function]      string-trim   character-bag   string
```


Summary

- There you have it! We now have a basic engine for a text adventure game.
- We can `look` around the world
- `walk` between places
- Pick up objects with `pickup`;
- and check our inventory with `inventory`.