INDEX

Symbols & Numbers

@ (at), in control sequence parameters, 223
* (asterisk), in variable names, 23
*board-scale* variable, 406
*dice-scale* variable, 403
*from-tile* variable, 411
*num-players* variable, 418
*print-circle* variable, 111
*standard-output* variable, 364
*top-offset* variable, 403
@ (at), in control sequence
   parameters, 223
' (backquote), 344
   for enabling switching from data
to code mode, 73
\ (backslash), for escaped
   characters, 35
: (colon), for keyword parameters,
   81, 122
:@ flag, for columns in tables, 230–231
:if-exists keyword parameter, 243
:initial-value keyword
   parameter, 168
:junk-allowed parameter, 260
:pretty parameter, 117
:radix parameter, 260
:test keyword parameter, 141
   to use equal, 204
. (dot), for representing cons cells, 39
" (double quotes), for strings, 35
= (equal sign) function, 65
# (hash mark), for array, 154
#newline, 89
#space, 89
#tab, 89
#' (function) operator, 75
#$ prefix, for structures, 164
< (less-than) function, with sort, 170

() parentheses
   for calling commands and
   functions, 22, 24
empty lists, 25
   symmetry of nil and, 49–52
for list of declared variables
   in let, 28
for organizing code into lists, 33
' (single quote), as data indicator, 37
~ (tilde), for control sequences, 223
~& control sequence, 227
~< control sequence, 229
~> control sequence, 229
~; control sequence, 232
~{ control sequence, 231
~} control sequence, 231
~$ control sequence, 223, 226
~% control sequence, 227–228
~a control sequence, 223–224
~b control sequence, 225
~d control sequence, 225
~f control sequence, 226
~t control sequence, 228–229
~x control sequence, 225
| (vertical pipe), for case-sensitive
   symbols, 89
404 error page, 265

A

ab-get-ratings-max function, 395–396
ab-get-ratings-min function, 395–396
ab-rate-position function, 397
academic research, 8
accum function, 459
accumulator, 332
~a control sequence, 223–224
across in loop macro, 201, 320
add-cops function, 140, 141–142
add function, predicates in, 171
add-new-dice function, 316–317, 333–334, 425
add-passing-move function, 312, 384–385
add-plants function, 204, 212
add-two function, 299–300
add-widget function, 296–297, 298
AI (artificial intelligence), 8
alphanumericp function, 117
always in loop macro, 201
Amazon S3, 160
anaphoric macros, 347
and in loop macro, 201
and operator, 58
announce-winner function, 320
ANSI Common Lisp (CL), 15–16, 17–18. See also Common Lisp (CL)
append function, 75, 76, 143
append in loop macro, 201
apply function, 76
apt-get install clisp, 18
ARC assembly, 5
Arc Lisp dialect, 17, 359, 459
aref function, 154
and performance, 156
arrayp function, 170
arrays, 153–157
as in loop macro, 201
assembler, 5
assembly languages, 5
assoc function, 71, 83, 112
association lists (alists), 111–112, 141
attributes for print-tag, 339
for nodes in city, 142
for scenery description, 70–71
web request parameters in, 261
writing to file, 243
asterisk (*), in variable names, 23
at (@), in control sequence parameters, 223
at-loc-p function, 78
attacking-moves function, 313–314, 385, 419
Attack of the Robots! game, 233–234
Autocode, 5
backquote (’), 344
for enabling switching from data to code mode, 73
backslash (\), for escaped characters, 35
~b control sequence, 225
being in loop macro, 200
tbelow in loop macro, 196
bidirectional stream, 247
bigger function, 27
binary, number display as, 225
binary search, 23, 26
blocking operation, 247
board-array function, 308
board-attack-fail function, 419–420
board-attack function, 315–316
*board-scale* variable, 406
&body keyword, 344
Boolean values, manipulating, 58
branching, 56–57
with case form, 57–58
breaking out of loop, 198
brevity of code, 459
brightness function, 361
bug fighters
Clojure Lisp, 461
comic book, 429–463
Common Lisp Object System (CLOS), 451
continuations, 454
domain-specific language, 450
exception handling, 444–445
functional programming, 441
generic setters, 447
lazy evaluation, 462
macros, 443
bugs, functional programming to reduce, 301
by in loop macro, 201

C

C++ language, 9, 10, 32
#define directive, 340
cached results, clearing, 398
cache misses, performance impact, 160
cadadar function, 42
cadadr function, 42
cadr function, 40–41
calc-pt function, 403
capitalized text, converting all caps to, 97
capturing console output, 123
car function, 40–41, 75
case form, branching with, 57–58
case-insensitivity, of symbols, 33
case of text, adjusting, 97
case-sensitive symbols, 89
cdr function, 40, 143–144
cells, retrieving item from first slot, 40
centered columns, 230
chain of cons cells, 40, 108
chance nodes, in game tree, 418–420
characterp function, 170
characters
  comparison, 65
  literal, 89
  for padding numbers, 225
char-downcase function, 99
char-equal function, 65
charge function, 151
char-upcase function, 99
class parameter, 407
Church, Alonzo, 293
circle function, 362
circular lists, 110–111
CISC (complex instruction set computer), 8
city.dot.png picture, 145
CL (Common Lisp), 15, 17–18. See also Lisp
basics, 441
tail call optimization support, 333
client, for socket connection, 246

CLISP, 18–19
  installing, 18
  printing of circular lists, 111
  shutting down, 19
  starting, 19
Clojure Lisp, 17, 461
  and lazy evaluation, 377, 462
  lazy sequences, 380
CLOS (Common Lisp Object System), 166, 451
closings predicate, 358
closing tag in XML, 358
closures, 326–328, 379
Clozure CL, 18
cl-sockets, 245
clusters, finding in Dice of Doom, 424–425
cmd variable, 95
CMUCL, 18
COBOL, 8
code
  brevity, 459
  vs. data, 35–37
  symmetry between data and, 91–92
code-char function, 260, 308
code composition, 298
code mode, 35, 36
  backquote (’), for enabling switching to, 73
coerce function, 98, 260
collect clause in loop, 137, 198
colon (:), for keyword parameters, 81, 122
color
  for dice, 407
  manipulating, 361
columns in table, centered, 230
comic book, 4
  bug fighters, 429–463
  on functional programming, 269–287
command-line interface, 85
  printing to screen, 86–87
commands, adding to permitted list, 368
Common Lisp (CL), 15, 17–18. See also Lisp
  basics, 441
  tail call optimization support, 333
Common Lisp HyperSpec, 170
  on control sequences, 233
Common Lisp Object System (CLOS), 166, 451
communication, with other network computers, 245
comparison, 62–65
  eql for numbers and characters, 65
  of symbols, 63
compiler, 5
  versions of function for, 172
complex instruction set computer (CISC), 8
computation, delayed, 124
computer, as game opponent, 321–326
concatenate command, 95
cond command, 56, 208
conditions, tricks with, 58–62
Congestion City, 131, 132. See also Grand Theft Wumpus game
  building final edges, 139–142
defining edges, 135–142
drawing map, 145–149
  from partial knowledge, 146–148
nodes for, 142–144
preventing islands, 137–139
walking around town, 148–149
connect-all-islands function, 139
connect-edge-list function, 140
connect-with-bridges function, 139
Conrad’s Rule of Thumb for Comparing Stuff, 62–63
cons cells, 37, 38, 107
  in nested lists, 42
cones, eq for comparing, 63
cons function, 38–40
consing, 39
console output, capturing, 123
console streams, 238
consf function, 170
constants, for game board dimensions, 402
continuations, 454
control sequences, 222–223
  Common Lisp HyperSpec on, 233
  for formatting numbers, 225–226
  iterating through lists with, 231–232
  for new lines, 227–228
control string parameter, for format function, 222–223
copy-list function, 211
copy-structure function, problems from, 211
count function, 167
counting from starting point to ending point, 197
count in loop macro, 201
currencies, formatting, 226

data
  vs. code, 35–37
generic process for handling, 166–172
  symmetry between code and, 91–92
tree-like, 113
data mode, 35, 37
  backquote (’’) for enabling switching to, 73
data structures, self-referential, 111
  ~d control sequence, 225
dead animals, in evolving environment, 212
dead monsters, checking for, 179
Debian-based Linux machine, CLISP on, 18
debugging
  in functional programming, 441
  string streams and, 250–251
def function, 180
decimal number, value
  displayed as, 225
decimal point, and number type, 34
declaration, of function, 29
decode-param function, 259–260
default, code mode as, 36
def- condition function, 254–255
defmacro command, 341, 342–344
defmethod command, 171–172, 180
defparameter command, 23, 24, 135
defstruct command, 163, 164, 172, 173, 180, 208
  for brigand, 185–186
  for hydra, 183
to include monster type fields, 181
for slime mold, 185
defun command, 25, 27
defvar command, 24
delayed computation, 124
deprecated function, 117
depth-first search, 394
describe-location function, 71
describe-objects function, 78
describe-obj function, 78
describe-path function, 72–73
describe-paths function, 73–74, 75, 77
destination parameter, for format function, 222
Dewdney, A.K., "Simulated evolution; wherein bugs learn to hunt bacteria," 202
Dice of Doom game, 303–336
attacking, 315–316
calculating attacking moves, 313–314
calculating passing moves, 312–313
computer opponent, 321–326
game loop with AI player, 324–325
minimax algorithm, 323
minimax algorithm code, 323–324
decoupling rules from rest of game, 309–310
finding neighbors, 314–315
game board, 307–309
3-by-3 sample game, 334–336
5-by-5, 398–400
constants for dimensions, 402
using SVG format, 402–408
generating game tree, 311–312
new game-tree function, 317–318
performance improvement, 326–336
playing against another human, 318–321
input from human players, 319
main loop, 318
state of game information, 318–319
winner determination, 319–320
playing first human vs. computer game, 325–326
reinforcements, 316–317
rules, 304
sample game, 304–306
tail call optimization, 333–334
version 1, 306–321
global variables, 306–307
version 2, 384–386
  alpha beta pruning, 393–400
  lazy lists for game tree, 384
  score-board function, 390
  starting game on 4-by-4 board, 386
  winning by a lot vs. winning by a little, 389–393
version 3 (web-based), 401
  announcing winner, 410
drawing die, 403–405
drawing tile, 405–406
game board, 406–408
game board in HTML, 412
handling computer player, 412
handling human player, 410–411
initializing new game, 410
playing, 413–414
web server interface, 408–412
version 4
  calling dice rolling code from game engine, 420–421
  improving reinforcement rules, 423–425
  increasing number of players, 417–418
  rolling dice, 418–423
  updating AI, 422–423
dice_of_doom.v2.lisp file, 402
*dice-scale* variable, 403
digit-char-p function, 116
digraph command (Graphviz), 115
direct-edges function, 138
directed graph, 124
dirty code, 294, 296
dividing by zero, 53
division function, 34
DOCTYPE declaration, 258
dod-request-handler function, 408–409
domain, explained, 355–356
domain of function, 292
domain-specific language (DSL), 231, 355, 450. See also macros
dot (.), for representing cons cells, 39
dot->png function, 123
dotimes function, 161, 175
DOT information generation, 115–120
   edges conversion, 119
   labels for graph nodes, 117–118
   node identifiers conversion, 116–117
   for nodes, 118
   turning DOT file into picture, 120–123
dot-name function, 116
do token, 197, 200
dotted lists, 108–109
double quotes ("), for strings, 35
downfrom in loop macro, 201
downto in loop macro, 201
draw-board function, 309
draw-board-svg function, 407
draw-city function, 145
draw-die-svg function, 403
draw-dod-page function, 409, 412
draw-known-city function, 147, 149
draw-tile-svg function, 405
draw-world function, 212–213
DSL (domain-specific language), 231, 355, 450. See also macros
dunk function, 368–369, 371
dynamic variable, 24
dynamic website, 265–267
   testing request handler, 265–266
EDSAC Initial Orders, 5
else in loop macro, 201
Emacs Lisp, 17
empty lists (), 39
   as false value, 50–51
   other expressions as disguises for, 51–52
end in loop macro, 201
energy, in plants, 203
eq function, 33, 57, 63
eql function, 65, 331
equal function, 63, 331
egalp function, 65, 330
= (equal sign) function, 65
error command, 254
escaped characters, in strings, 35
eval command, 92
danger of, 101
improving, 96
every function, 167, 179
evolution function, 213–214
evolving environment game, 202–218
   animals, 205–212
      anatomy, 205–207
      eating process, 209
      energy, 206
      motion, 207–208
      properties, 206
      reproduction, 210–212
      starting point, 207
      tracking genes, 206
   turn function, 208–209
   bimodal distribution in, 217–218
drawing world, 212–213
plants
environment, 203
   growth, 204
   simulating day, 212
   starting simulation, 214–218
   user interface, 213–214
exception handling, 95, 253–256, 444–445
   custom conditions, 254–255
   intercepting conditions, 255
   resources protected against unexpected conditions, 255–256
   signaling condition, 254
   for web server, 265
exponent, 36
expressive language, 10
expt function, 34, 36

F
false value, empty list () as, 50–51
`false` control sequence, 226
files
    streams to write and read, 242–243
    writing information to, 121
file streams, 238
finally in loop macro, 200
find-empty-node function, 144–145
find-if function, 61, 167
find-island function, 139
find-islands function, 139
Firefox, for Dice of Doom game, 413–414
Firefox 3.7 alpha, for SVG support, 402
first-class values, functions as, 104
flet function, 29
    for local function definition, 95
floating-point numbers, 34
    control sequences for formatting, 226
force command, 378–380
for in loop macro, 196, 201
format function, 193. See also printing
    anatomy, 221–223
    control string parameter, 222–223
    destination parameter, 222
    and text justification, 228
formatting numbers, control sequences for, 225–226
forms, 36
    nested, 36
FORTRAN, 5
freeing of variables, 327
fresh-line command, 227
from in loop macro, 201
    *from-tile* variable, 411
funcall function, 178, 327
functional programming, 54, 71, 441
    anatomy of program, 295–298
    benefits, 301–302
    comic book, 269–287
    higher-order, 105
    and loops, 315
    problems from, 375–376
    reduce function, 352–353
    side effects, 294, 300–301
    using, 299–300
    what it is, 292–295
function operator, shorthand for, 75
functionp function, 170
function pipeline, 309
functions
    calling in Lisp, 22
    call to itself, 30
    comprehensive list of sequence, 170
    creating with lambda, 103–105
    deprecated, 117
    generic, 116
    higher-order, 75
    names available in defined functions, 29–30
    namespaces for, 75
    nullary, 120
    parentheses for, 22
    sending string streams to, 249

G
game-action macro, 369–371
game board
    AI adjustments for larger, 387–400
    for Dice of Doom, 307–309
    3-by-3 sample game, 334–336
    5-by-5, 398–400
    constants for dimensions, 402
    using SVG format, 402–408
game-eval function
    approved list of commands for, 101
    limiting commands called, 96
    game-loop function, 174–175
game-print function, 96–99
game-read function, 94–95
game-repl function, 93–94, 365
games. See also Dice of Doom game;
    evolving environment game;
    Grand Theft Wumpus game; Orc Battle game;
    Wizard’s Adventure Game
Game-Board: Attack of the Robots! game, 233–234
Guess-My-Number, 21–23
loading code from REPL, 365–366
winning by a lot vs. winning by a little, 389–393
game tree
    branches hidden in clouds, 376–377
    chance nodes in, 418–420
    generating, 311–312
    memoizing, 330
    trimming, 387–389
game-tree function, 311, 317–318
garbage collection, 9, 327
Garret, Ron, 257
gen-board function, 308
genersonalized reference, 155
generic functions, 116
    creating with type predicates, 170–172
generic setters, 154–156, 447
gensym function, 349
get-connected function, 138, 161, 162, 424, 425
get-connected-hash function, 163
gen-content-params function, 263
gethash function, 155, 158, 160, 162
get-header function, 262
    testing, with string stream, 262–263
get-ratings function, 324, 391, 422
    new versions, 395
GET request, 257
    request parameters for, 259
global functions, defining, 25–28
global variables
    changing value, 27
    defining, 23–24
    in look function, 80
    macros and, 370
    for player and monsters, 173–174
    setting inside conditional branch, 54
Google BigTable, 160
Graham, Paul, 17
    Arc Lisp dialect, 359
Grand Theft Wumpus game. See also Congestion City
    basics, 131–135
    clues, 142
    drawing map, 145–149
    from partial knowledge, 146–148
    with hash tables, 161–163
    initializing new game, 144–145
    playing game, 149–151
    police roadblocks, 139
graph->dot function, 124
graphs
    creating, 114–124
    creating picture of, 123–124
    directed, 124
    labels for nodes, 117–118
    undirected, 124–127
    visualizing, 114
graph utilities, loading, 135
graph-util.lisp file, 127
Graphviz, 114–124
Graphviz DOT file
    edges conversion, 119
    for graph drawing library, 115–120
    labels for graph nodes, 117–118
    node identifiers conversion, 116–117
    for nodes, 118
    turning DOT file into picture, 120–123
guess-my-number function, 25–27
Guess-My-Number game, 21–23
Guile Scheme, 17

H
hackers
    and dangerous commands, 101
    and read command, 262
handle-computer function, 324, 388, 397, 412, 421
handle-direction function, 148
handle-human function, 319, 385–386, 421
handle-new-place function, 149
handler-case function, 254
hash collisions, 160
hash-edges function, 162
hash-key in loop macro, 200
hash-keys in loop macro, 200
hash mark (#), for array, 154
hash-table-p function, 170
hash tables, 155, 157–163
   Grand Theft Wumpus game
      with, 161–163
   inefficiency for small tables, 160
   performance, 160–161
   for plants, 204
   returning multiple values, 159–160
hash-value in loop macro, 200
hash-values in loop macro, 200
Haskell, 17, 296
   and lazy evaluation, 377
have function, 367
health meter, for monsters, 180
hello-request-handler function, 265
heuristics, 389
hexadecimal, number display as, 225
Hickey, Rich, 17
hidden state, 299
hierarchical data, 113
higher-order functions, 75
higher-order programming, 105, 298–300
homoiconic programming code, 91
HTML5 standard, 402
HTML code, 97
   embedding SVG pictures, 402
   page skeleton, 265
   tag macro to generate, 360–361
html tags, 258
HTTP (Hypertext Transfer Protocol), 256
http-char function, 260
HTTP escape codes, 259
http.lisp file, 257
Hughes, John, “Why Functional Programming Matters,” 310
Hunt the Wumpus, 129
hyperlinks, in SVG image, 361
Hypertext Transfer Protocol (HTTP), 256

I
if command, 50, 52–54
   :if-exists keyword parameter, 243
if in loop macro, 201
imperative code, 294
   code composition with, 298–299
imperative game engine, 310
implicit progrn, 55
incf function, 179
indentation of code, 28
infinite loop
   getting out of, 93
   preventing, 111
infinity, positive and negative, 397
Information Processing Language, 5
in in loop macro, 201
initially in loop macro, 200
:initial-value keyword parameter, 168
init-monsters function, 178
input-stream-p command, 240
input streams, 238, 240–241
installing CLISP, 18
instruction set of processor, 5
integers, 34
   control sequences for formatting, 225
intern command, 262
interpreter, 5
   versions of function for, 172
intersection function, 141
into in loop macro, 201
inventory function, 83
IP address, in socket address, 245
islands, preventing, 137–139
isomorphic item, 63
iterating
   across sequence, 167–170
   through lists, with format control sequences, 231–232
   through list values, 197
J
Java language, 10
Jones, Simon Peyton, 300
   :junk-allowed parameter, 260
justified text, 228–231
K
key/value pair
   returning for alist, 112
   storage, 160
keyword parameter, 117–118, 122
   for find function, 81
known-city.dot-png file, 148
known-city-edges function, 146–147
known-city-nodes function, 146

L
labels, for graph nodes, 117–118
labels function, 29–30, 78
for local function definition, 95
lambda calculus, 6, 105, 293
lambda function, 178, 179, 255, 314
and closures, 326–327
importance, 105
purpose, 105–105
largest-cluster-size function, 424–425
launching website, 266–267
lazy-car command, 380
lazy-cdr command, 380
lazy command, 378–380
lazy-cons command, 380
lazy evaluation, 376–384, 423, 462
lazy-find-if function, 383
lazy game tree, 310
lazy lists
adjusting AI functions to use, 387–400
converting between regular lists and, 381–382
converting to regular lists, 382
for Dice of Doom game tree, 384
library for, 380
mapping and searching, 383–384
lazy-mapcan function, 383, 385
lazy-mapcar function, 383
lazy-nil function, 381, 385
lazy-nth function, 383
lazy-null function, 381
legality of game move, 148
legal-tiles parameter, 407
length function, 166–167
less-than (c) function, with sort, 170
let* command, 140
let command, 28, 123, 140, 327, 340
prog command and, 344
lexical variable, 123, 327–328
library, for lazy lists, 380
limit-tree-depth function, 388, 395, 423
line breaks, 28
linking data pieces, cons function for, 38–40
Lisp. See also Common Lisp (CL)
basic etiquette, 24–25
dialects, 15–18
for scripting, 17
features, 2–3
Guess-My-Number game, 21–23
origins, 4–9
source of power, 10–11
technologies supporting, comic book, 429–463
up-and-coming dialects, 17
valid expression example, 3
LispWorks, 18
list function, 41, 359
list-length function, 167
listp function, 170, 240
lists, 33, 37–42. See also association list
(alist); lazy lists
vs. arrays, 156–157
association, 111–112
benefits of using, 71
calculating length, 51
checking for membership, 60–61
circular, 110–111
control sequences for iterating through, 231–232
dotted, 108–109
empty, 39
as false value, 50–51
other expressions as disguises for, 51–52
functions, 38–42
iterating through with loop, 197
joining multiple into one, 76
macro for splitting, 346–347
nested, 41–42
of objects, 77–78
pairs, 109–110
sequence functions for, 166
vs. structures, 165–166
sum function for, 169
literal characters, 89
lit variable, and capitalization rules, 99
load command, 135
local functions, defining, 29–30
local variables
   defining, 28
   for value returned by read
function, 88
log information, streams for, 249
long strings, 250
look function, 80, 93
lookup key, of hash table, 158
loop macro, 93, 136–137, 193, 195–202
   breaking out, 198
   collect clause, 198
   counting from starting point to
   ending point, 197
do token, 197
   iterating through list values, 197
   with multiple for clauses, 198–199
   nested, 199
   periodic table of, 200–201
when token, 197
loops
   with doimes function, 175
   for evolving environment, 202–218
   and functional programming, 315
   getting out of infinite, 93
   preventing infinite, 111
  
M
machine language, 4
macroexpand command, 345, 348,
   349–350
macro expansion, 341–342
macros, 54, 82, 104–105, 339, 443. See
   also domain-specific
   language (DSL);
   programming language
   avoiding repeated execution,
   347–348
   avoiding variable capture,
   348–350
dangers and alternatives, 352–353
   for defining new function, 370
   helper function, 358
   to implement lazy command, 379
reader, 101
recursive, 350–352
   simple example, 340–345
   for splitting lists, 346–347
   svg, 361–362
   transformation, 342–344
main-loop function, 296, 298
make-array command, 154
make-city-edges function, 139, 140
make-city-nodes function, 143
make-edge-list function, 136, 140
make-hash-table command,
   157–158, 161
make-lazy function, 381
make-orc function, 181
make-person function, 164, 165
make-string-input-stream function,
   249, 263
make-string-output-stream
   command, 249
mapcan function, 146, 147, 314, 363
mapcar function, 74, 141, 359
mapc function, 118, 138, 162
map function, 169–170
maplist function, 126
map of city
drawing, 145–149
   showing only visited nodes,
   146–148
mapping lazy lists, 383–384
mathematical functions,
   properties, 293
mathematical sets, hash tables for, 204
mathematical syntax, languages
   using, 6
max function, 212
maximize in loop macro, 201
McCarthy, John, 6–7
   "Recursive Functions of Symbolic
   Expressions and Their
   Computation by
   Machine," 7
member function, 60–61, 96
memoization, 328–331
memory, 5, 156
   software transactional, 461
Metaobject Protocol (MOP), 451
minimax algorithm code, game tree
   analysis with, 394
minimize in loop macro, 201
mod (remainder) function, 208
monetary floating-point value, 223
monster-attack function, 181
  for orcs, 182
  for slime mold, 185
monster-hit function, 176, 184
monsters. See Orc Battle game
monster-show function, for orcs, 182
MOP (Metaobject Protocol), 451
most-negative-fixnum, 397
most-positive-fixnum, 397
move function, 207–208
move in game, checking legality, 148
multiple dispatch, 452
multiple-value-bind command, 159
mutations, 165, 210
  with reproduce function, 211
my-length function, 331–332
  custom, 345
  improving, 350–352

N
named in loop macro, 200
names of functions, available in
defined functions, 29–30
namespaces, for variables and
  functions, 75
nconc in loop macro, 201
neato command (Graphviz), 115
negative infinity, 397
neighbors function, 142, 314–315,
  329–330
nested alists, 142
nested lists, 41–42
nested loop macro, 199
nested tags, in XML, 357
network computers, communication
  between, 245
never in loop macro, 201
new-game function, 144
  to draw known city, 147
newline, 89
new line
  control sequences for, 227–228
  in printed output, 226
  before printing, 87
nil, 38, 39, 52, 107
  lists not ending with, 109
  symmetry of () and, 49–52
nodes, 118
  for Congestion City, 142–144
  identifiers, converting, 116–117
nodes-dot function, 118, 119
non-deterministic programming, 454
non-visible characters, literals for, 89
nth function, 156
nullary functions, 120
null function, 61–62
numberp function, 170
numbers, 34–35
  comparison, 65
  control sequences for formatting,
    225–226
*num-players* variable, 418

O
object-oriented programming (OOP)
  languages, 9, 163, 451
  vs. Lisp, 165
objects
  descriptions
    at specific location, 77–78
    visible, 78–79
  inventory check, 83–84
  picking up, 82–83
objects-at function, 78, 82, 83
on in loop macro, 201
OOP (object-oriented programming)
  languages, 9, 163, 451
  vs. Lisp, 165
optimizing functional code, 326
  closures, 326–328
  memoization, 328–331
  tail call optimization, 331–334
orc-battle function, 174, 187–188
Orc Battle game, 172–188
  global variables for player and
  monsters, 173–174
  helper functions for player
  attacks, 177–178
  main game functions, 174–175
  monster management functions,
monsters, 179–186
  checking for dead, 179
  Cunning Brigand, 185–186
  functions for building, 174
  generic, 180–181
  hydra, 183–184
  Slimy Slime Mold, 184–185
  Wicked Orc, 181–182
player management functions, 175–177
  starting game, 187–188
orc datatype, 181
or operator, 58
orthogonal issues, 387
output-stream-p function, 240
output streams, 238, 239–240
  with-open-file command for, 242

P
padded value, for format function, 223
padding parameter, for number width, 225
pairs, 109–110
pairs function, 351, 359
parallel games, web server for multiple, 410
parameters, quoting, 95
parametric polymorphism, 9
paranoid strategy, 418
parentheses ()
  for calling commands and functions, 22, 24
  empty lists, 25
    symmetry of nil and, 49–52
  for list of declared variables in let, 28
  for organizing code into lists, 33
parse-integer function, 260
parse-params function, 261
parse-url function, 261–262
path descriptions in game, 72–77
  multiple at once, 73–77
performance
  arrays vs. lists, 156–157
  cons cells and, 113
  for Dice of Doom game, 326–336
  functional programming and, 300
hash tables and, 160–161, 163
tail calls and, 333
periodic table of loop macro, 200–201
permitted commands, adding to list, 368
person-age function, 164
pick-chance-branch function, 420–421
pick-monster function, 176
pickup function, 82
π constant, 226
picture, from DOT file, 120–123
player-attack function, 176, 177
player function, 314
play-vs-computer function, 324–325, 389
play-vs-human function, 386
police roadblocks, 139
polygon function, 362–363
polygons, for die, 403
port
  number in socket address, 245
    taking control of, 246
port 80, 264
port 8080, 264
position function, 167, 261
positive infinity, 397
POST request, 258
power, 193
predicates, 78, 116
:pretty parameter, 117
print function, 87
print-to-string function, 98, 116
princ function, 35, 90–91, 222, 223–224
*print-circle* variable, 111
printed representation, creating object from, 164
print function, 86–87
  priority use, 88
printing. See also format function
  creating stream for functions, 121
  multiple lines of output, 226–228
to screen, 86–87
  text justification, 228–231
print-tag function, 358
problem solving, 20
progn command, 54
programming
  heuristic techniques, 389
  nondeterministic, 454
programming language. See also
macros
higher-order, 298–300
learning, 2
properties in structures, 163
push function, 82–83, 112, 138, 240
for hash table values, 162
push-new command, 368, 370
Python, 9

Q
quasiquoting, 73
quit command, 19
quote command, 95
quoting, 37
quote-it function, 95

R
:radix parameter, 260
raise-price function, 445
RAM, 156
random edges
generating, 135–136
and island prevention, 137–139
random function, 177, 308, 363
random-monster function, 177
random-node function, 136
random numbers, generating, 177
random-plant function, 204
random walk, 363
randval function, 177, 180
range of function, 292
rate-position function, 323–324,
330–331, 391
new versions, 397
rational number, function
returning, 34
RDF (Resource Description
Framework), 3
read-char command, 241
reader, 33
reader macros, 101
read-eval-print loop (REPL), 19, 22
loading game code from, 365–366
setting up custom, 93–94
testing, 99–100
read-from-string function, 95, 410
read function
danger of, 101
local variable for value
returned by, 88
reading data, input streams for,
240–241
read-line function, 91
recurse macro, 350–351
recursion, 30, 50, 332
in macros, 350–352
reduced instruction set computer
(RISC) hardware
architecture, 8
reduce function, 167–169
initial value for, 168
reference, generalized, 155
referential transparency, 293, 301
reinforcements, rules for choosing
number in Dice of
Doom, 425
remhash function, 209
remove-duplicates function, 141, 320
remove-if function, 320
remove-if-not function, 78, 138
repeat in loop macro, 200
REPL. See read-eval-print loop (REPL)
reproduce function, 210
mutations with, 211
request body, 257
parsing, 263
request handler, testing, 265–266
request-handler function, 264
request-handler parameter, 264
request header, 257
parsing, 261–262
request parameters
decoding lists of, 260–261
decoding values for HTTP,
259–260
for web server, 258–261
Resource Description Framework
(RDF), 3
resources, freeing up, 248–249
response body, 258
response header, 258
restarts, 444–445
return-from in loop macro, 200
return in loop macro, 200
return value, for command, 25
reverse function, 222
RISC (reduced instruction set computer) hardware architecture, 8
roll-dice function, 420
round function, 159
Ruby, 9
rule engine, 310
runtime, 342

say-hello function, 87–88
SBCL (Steel Bank Common Lisp), 18
scalable vector graphics (SVG). See SVG images
scenery description, association list for, 70–71
Scheme, 15
namespace for, 76
tail call optimization in, 333
score-board function, 390
screen, printing to, 86–87
Script-Fu Scheme, 17
scripting, Lisp dialects for, 17
searching
.lazy lists, 383–384
.sequence functions for, 167
security, eval function and, 92
self function, 351–352
self-referential data structures, 111
semantics, 31–32
Semantic Web, 3
sending message over socket, 246–248
sequence functions, 166
...for searching, 167
sequences, 166–170
...iterating across, 167–170
serve function, 263–265
server, for socket connection, 246
set-difference function, 139
setf function, 27, 83, 111, 329, 447
...for array, 154–155
...to change structure property, 164
shallow copy of structure, 211
Short Code, 5
shortcut Boolean evaluation, 59
show-monsters function, 179
shutting down CLISP, 19
side effects, 441
...of functional programming, 294, 300–301
signaling condition, for error handling, 254
sin function, 293
single quote (’), as data indicator, 37
slots, 163
smaller function, 27
socket, serve function creation of, 264
socket-accept command, 247
socket-connect command, 247
sockets, 244–249
...addresses, 245
...connections, 246
...sending message over, 246–248
socket-server-close command, 249
socket-server function, 246
socket streams, 238
software transactional memory, 461
same function, 167
sort function, 170
#\space, 89
special form
...if as, 53
...let command as, 340
special variable, 24
splash command, 371
split macro, 346–347
splitting lists, macro for, 346–347
#\$ prefix, for structures, 164
*standard-output* variable, 364
starting CLISP, 19
start-over function, 28
statistics, of dice rolls, 422
Steel Bank Common Lisp (SBCL), 18
Steele, Guy L., 16
streams, 121, 237–238
...bidirectional, 247
...closing on network computer, 248–249
...commands to interact with, 242
...for files, 242–243
...types, 238–241
string builders, 250
string datatype, 70
string-downcase function, 358
string-equal function, 65
stringp function, 170
strings, 35
  converting symbol list to, 98
  sequence functions for, 166
string streams, 238, 249–251
debugging and, 250–251
  get-header function testing with, 262–263
Stroustrup, Bjarne, 10
structures, 163–166
  vs. lists in Lisp code, 165–166
  when to use, 165–166
subseq function, 170
substitute-if function, 116–117
substitute-if-not function, 117
sum function, for arrays and lists, 169
sum in loop macro, 196, 201
suspension, 120. See also thunks
Sussman, Gerald Jay, 16
SVG images
  attributes for, 361
circles, 362
  Dice of Doom game board using, 402–408
polygons, 362–363
  writing, 356–364
svg macro, 361–362
svg Style function, 362
SVG Web, 356
symbol-function command, 329
symbolp function, 170
symbols, 33–34
  benefits of using, 71
  comparing, 63
  converting list to string, 98
symmetry
  of () and n1, 49–52
  between code and data, 91–92
syntax
  building blocks for Lisp, 32–35
  and semantics, 31–32
T
  #\tab, 89
tables
  output as, 228–229
  trick for creating pretty, 232–233
tab variable, 331
tag macro, 359–360
  to generate HTML, 360–361
tail call, 332
tail call optimization, 331–334
take-all function, 382
take function, 382
	 control sequence, 228–229
TCP/IP, 256
TCP packets, 245
technologies supporting Lisp, comic book, 429–463
terpri function, 226–227
test functions, 116
testing
  get-header function with string stream, 262–263
  user interface, 99–100
:test keyword parameter, 141
  to use equal, 204
text. See also strings
  breaking into equal length pieces, 232
  converting all caps to capitalized, 97
  justified, 228–231
  processing, 67
text game interface, 92–99
  testing, 99–100
the in loop macro, 200
then in loop macro, 201
thereis in loop macro, 201
threatened function, 391
threatened hex, in Dice of Doom, 390
three-way-if macro, 443
thunks, 120–121
  for creating graph picture, 123
  tilde (~), for control sequences, 223
time command, 161
to in loop macro, 201
top-level definition of variable, 23
*top-offset* variable, 403
tree-like data, 113
true/false functions, 78
turn function, for animals, 208–209	weak-text function, 98
type-checking, 166
  in generic functions, 167
type dispatching, 172
type-of function, 180–181

type predicates, for generic functions,
170–172

U

edges->dot function, 126
ugraph->dot function, 126
ugraph->png function, 126, 145
undirected graphs, 124–127

unless, 55

in loop macro, 201
until in loop macro, 200
unwind-protect function, 256, 264
update-world function, 212
upto in loop macro, 201

URLs for web pages, name/value
pairs in, 260

user interface, 85

command-line, 85
printing to screen, 86–87
for evolving environment game,
213–214
testing, 99–100
for Wizard’s Adventure Game,
92–99
using in loop macro, 200
usocket, 245

V

vacuum-tube computer systems, 4

values function, 159

variable capture, 348–350

variables. See also global variables;
local variables

asterisks (*) in names, 23
declaration in let command, 28
defining, 140
destruction, 327
in functional programming,
293, 301
function to create unique
name, 349
lexical, 123, 328
for location descriptions, 70
modifying value, 447
namespaces for, 75

variable shadowing, 333

versions of function, 172

vertical pipe (|), for case-sensitive
symbols, 89

virtual memory paging, performance
impact, 160

visible objects, describing, 78–79

visualizing graphs, 114

visual noise, 340

W

walk function, 81–82, 148

web-announce-winner function, 410

web forms, 258

web-handle-human function, 410–411

web-initialize function, 409, 410

web resources

downloading CLISP installer, 18
for Graphviz, 115
Lisp projects, 3

web server, 256–265

continuation-aware, 454
how it works, 256–258
interface for Dice of Doom,
408–412

for computer player, 412
for human player, 410–411
limitations, 409–410

parsing request body, 263
parsing request header, 261–262
request parameters, 258–261
serve function, 263–265

webserver.lisp file, 402

website

dynamic, 265–267
launching, 266–267

weld function, 367–368, 370–371

when in loop macro, 201
when token, 55, 197

while in loop macro, 200

winners function, 319–320

with in loop macro, 200

with-open-file command, 121, 122,
123, 242–244

with-open-stream macro, 264

with-output-to-string macro, 250–251
Wizard’s Adventure Game
basic requirements, 69–70
custom game commands, 365–373
dunk, 368–369
game-action macro, 369–371
welding, 366–368
custom interface, 92–99
DOT information for, 119–120
location descriptions, 71
look command, 79–80
map of house in alists, 114
object descriptions at specific
location, 77–79
object inventory check, 85–84
path descriptions, 72–77
picking up objects, 82–83
playing completed version,
371–373
scenery description with
association list, 70–71
walk function, 81–82
world for, 68–69
write-char command, 240

X
~x control sequence, 225
XML, 113
XML format
nested tags, 357
and SVG format, 357
xmlns attribute, 361

Z
zero, dividing by, 53