

Symbols

- :about method 335
- :as option 87
- :body key 229
- :constructors option 119
- :decode key 209
- :doc key 89
- :exposes option 120
- :exposes-methods option 120
- :extends option 119
- :factory option 120
- :file key 89
- :find command 214
- :format option 213
- :gen-class 116
- :headers key 229
- :id-gen 290
- :implementors key 350
- :implements option 119
- :impl-ns option 120
- :init option 119
- :io tag 88
- :load-impl-ns option 120
- :macro key 89
- :main option 120
- :method message 330
- :methods option 119
- :name command 219
- :name option 119
- :new clause 327
- :post key 63
- :post-init option 120
- :pre key 63
- :prefix option 120
- :query-params map 231
- :query-string key 229
- :request-method key 229
- :safe tag 88
- :status key 229
- :status keyword 365
- :strs keyword 87
- :syms keyword 87
- :tasks-atom 290
- :total key 284
- :value keyword 365
- :when option 49
- .. macro 110–111
- .toUpperCase function 79
- *compile-path* var 117
- *eval-me* var 76
- *mysql-host* var 143
- *rabbit-connection* 244–245, 253, 256, 261–262, 264–265
- *session* function 387, 389–390, 393
- & symbol, as keyword 84–85
- #() reader macro 71, 152
- + function 67
- < function 45
- = function 45
- > macro 51
- > macro 50
- >> macro 51
- ~ character 154
- \$ prefixes 388
- \$attribute syntax 393
- \$search-terms 387, 389, 391, 393–395
- \$url-referrer 387, 390–391, 393–395
- \$user-agent 387–389, 391, 393–394

A

- abracadabra 374, 376, 378
- abstract syntax tree. *See* AST
- abstractions, macro-syntactic 17
- AbstractJavaClass 118
- accept method 95
- accept(NodeVisitor) 95
- access member 109
- access models unified 143–144
- accessor methods 190
- account-balance variable 12
- ACI (atomic, consistent, isolated) properties 135
- active record design pattern 190–191
- ActiveMQ server 242
- ActiveRecord library 190
- ad hoc hierarchies 100–103
 - Java language classes 101
 - resolving conflicts 102–103
 - visitor pattern 102
- add function 6
- add-5-to-triple function 319
- adder function 321–322
- add-pair function 316–317, 320
- add-quadruple-curried function 318–319
- add-user function 133
- add-watch function 144
- Advanced Message Queuing Protocol. *See* AMQP
- affiliate-fee 97–101

affiliate-id 100
 after-load callback 195
 agent function 42, 136–140, 143–144
 agent type 131
 agent-errors function 139
 agents 136–140

- await and await-for
 - functions 138
- errors and 138–139
- mutating 136–137
 - send function 136–137
 - send-off function 137
- side effects in STM
 - transactions 139–140
- validations 139

 aget tokens 2 function 114
 ahead of time. *See* AOT
 alength tokens function 114
 all-complete? function 365
 all-greater-than function 311–312
 all-lesser-than function 311–312
 all-ns function, and find-ns function 83
 all-users ref 47, 131–133
 alphabetically function 70
 alter function 39, 75, 117, 132–134, 140, 144
 alter-var-root function 349–350
 amap function 114
 AMQP (Advanced Message Queuing Protocol) 241

- exchange 243
- message queue 243
- routing key 243

 anaphora 368–370, 374
 anaphoric macros, if

- construct 369–371
- generalizing 370–371
- implementing 370

 anaphoric-if 370–371
 an-argument symbol 14
 and keyword 84–85
 and macro 160
 Animal class 94
 Animal interface 94
 Animal object 95
 Animal type 92
 anonymous functions 70–71
 another-argument symbol 14
 anya function 324
 AOT (ahead of time) 115
 apply-doc function 233
 are macro 188

areduce function 114
 args function 76–77, 109–110, 136–137, 140–141
 args-seq parameter 277–278, 292–293, 304
 arities 64
 arity

- multiple 64
- variable 37

 arrays 113–114
 aset tokens 2 actionable function 114
 aspect-oriented logging 76
 AssertionError 63
 assert-true macro 164
 AssignmentNode 95
 assoc function 56–58, 88, 133, 361
 associations namespace 191
 associations, in MySQL 193–194
 assoc-in function 58
 AST (abstract syntax tree) 6, 93
 asynchronous changes 136
 atom type 131
 atomic, consistent, isolated. *See* ACI
 atomicity 135
 atoms 140–142
 auto-gensym 157, 369
 await function 138
 await-for function 138

B

backquote (') reader macro 154
 back-quote-test macro 380–381
 bad-agent function 138–139
 balance function 70
 basicAck method 245, 264
 basicConsume 245, 263, 265
 basic-item-total function 64
 batch-size 290, 292–293, 304
 batch-wait-time 290, 292–293, 304
 bean macro 113
 before-save callback 195
 before-update callback 195
 belongs-to relationship 193–195
 benefits, of macros 157–158
 big data 189
 BigIntegers 53
 BigTable paper 199
 binding

- maps 86–87
- vars and 74–78

dynamic scope 75–77
 special variables 75, 77–78
 thread-local state 77
 vectors 84–86

- & symbol 84–85
- nested vectors 85–86

 binding form 180, 332
 buried updates 123
 business rules 99
 byte code, Java 115–120
 Bytes/toBytes column 201–202

C

calculate-total function 12
 calculators example 115–117

- *compile-path* var 117
- generated classes 116
- new files 116–117

 calculators namespace 116
 calculators_init.class 116
 calculators.class file 116
 calculators.clj 115
 calculators\$present_value__xx.class 116
 Calendar class 110
 calendar-obj symbol 111
 callbacks, for MySQL 195
 Cardelli, Luca 91
 cascading-clojure library 289
 Casalog 289
 case sensitivity 36
 catch clause 41
 category-is function 341, 343–344, 352, 356, 361–362
 catt function 66
 chapter-data.dist-fact namespace 295–297, 299
 chapter-data.rails-log namespace 285
 chapter-protocols.expense-protocol namespace 355–356
 chapter-protocols.expense-record.NewExpense class 360
 charges table 191
 chars, strings and numbers and 52–53
 checkValidity 95, 102
 class hierarchies, Java language 101
 classes

- creating instances 326–327
- defining 325–326

- classes (*continued*)
 - extending, and implementing interfaces 114–115
 - generated 116
 - inheritance of 332–337
 - Java, importing 107–108
- classes/com/curry/utils
 - directory 116
- classify function 386, 392–393, 395
- class-name function 326–333, 335–337
- Classname-symbol method 109
- CLASSPATH variable 119
- clear-agent-errors 139
- clj.script.examples
 - namespace 120
- clj-html 221, 236, 238–239
- clj-record library 190–197
- clj-record.boot namespace 196
- clj-record.core namespace 196–197
- clj-str 213, 217, 219
- Clojure function 116, 120, 316–319
- Clojure language 3–59
 - calling from Java Interop functionality 120–121
 - code, compiling to Java byte code 115–120
 - data structures 52–58
 - chars, strings, and numbers 52–53
 - keywords and symbols 53
 - nil, truth, and falsehood values 52
 - sequences 53–58
 - description of 4–6
 - doc macro and find-doc function 33–34
 - expression problem in 340–341
 - Hello, world program 32–33
 - installing 31
 - program flow 42–52
 - conditional forms 43–45
 - functional iteration 45–50
 - threading macros 50–52
 - REPL 31–32
 - sources of power of 10–26
 - advantages of Clojure language 18
 - functional language 18–23
 - functional programming 11
 - JVM 11
 - JVM-based language 23–26
 - Lisp 10–17
 - structure of 36–42
 - functions 36–37
 - let form 37–38
 - reader macros 41–42
 - side effects with do functions 39
 - try/catch/finally blocks and throw form 40–41
 - support for object orientation 26–29
 - syntax 6–10
 - lists, vectors, and hashes 9–10
 - XML and parentheses 7–9
- clojure.test table 200–202, 208–210
- clojure.test library 341, 352–353
- clojure.walk namespace 388, 390
- clojure.xml namespace 81
- clojure-hadoop 288
- clojure-json library 81, 227
- closures 321–338
 - and delayed computation 322–323
 - and free variables 321–322
 - and objects 323–325
 - lexical 79–80
 - object system for 325–338
 - class inheritance 332–337
 - creating instances 326–327
 - data abstraction 337
 - defining classes 325–326
 - defining methods 328–330
 - invoking methods 330–331
 - referring to this 331–332
 - state of 327–328
- collecting results, of higher-order functions 308–310
- column oriented, HBase 198–199
- com.curry.utils.calculators namespace 116
- com.gentest.AbstractJavaClass 119
- com.gentest.ConcreteClore-Class 119
- combinability, increasing 394
- combine function 277
- comment character 41
- comment macro 36, 159
- comments, whitespace and 35–36
- commute function 132–134, 144
- compare-and-set! function 141–142, 144
- compile function 116
- compile time, shifting computation to 374–379
- completion function 68
- Compojure web-application framework 232–236
 - handling parameters 233
 - Hello, world! application 232–233
 - response from 234–235
 - sessions handling with 235–236
 - with-request-bindings macro 234
- computation, shifting to compile time 374–379
- compute-across function 311–312, 315
- compute-averages-from function 372–373
- concurrency
 - and multicore future 21
 - immutability and 128–129
- cond form 162
- conditional forms 43–45
- config-for function 205–206, 209
- config-keys function 205
- conflicts, resolving 102–103
- conj function 349–350
- connecting, to RabbitMQ server 243–244
- cons cell 53
- cons sequence 54
- consistency 135
- constantly function 68, 181
- consumer type 213
- consumer-for 245, 247, 253, 257, 262, 264–265
- consumer-id 386–387, 393
- control characters 103

cookie handling 227, 233
 cookies module 231
 cookies, for HTTP
 interface 227–229
 coordinated changes 131
 core namespace 191
 counter variable 20
 counting words 274–276
 create, read, update, and delete.
 See CRUD
 create-ns function 83
 create-response function
 234–235
 CRUD (create, read, update,
 and delete) 190
 cube-all function 308–309
 curry function 317–320
 currying functions 316–321

D

daily-report function 156
 dash prefix 119
 data abstraction, for
 closures 337
 data elements 386
 data mapper
 for HBase 202–204
 reading from HBase
 208–210
 writing to HBase 204–208
 for Redis 212–220
 implementing 215–219
 persistence 214–215
 using Redis objects 214
 data processing 273–306
 map/reduce paradigm
 274–289
 analyzing rails
 sessions 285–288
 counting words 274–276
 generalizing 276–279
 large-scale data
 processing 288–289
 parsing logs 279–285
 master/slave
 parallelization 289–306
 defining job 290
 defining slave 293–295
 dispatching job 292–293
 maintaining status
 290–292
 rerunning job 300–306
 running job 296–298
 seeing task errors 298–300

 using master-slave
 framework 295–296
 data storage 189–220
 HBase 197–210
 column oriented 198–199
 data mapper for 202–210
 direct access to 200–202
 scalability 198
 schema design 199
 versions 199
 MySQL 190–197
 associations in 193–194
 callbacks 195
 clj-record.boot
 namespace 196
 clj-record.core
 namespace 196–197
 code organization of clj-
 record 197
 user model 191–193
 using active record design
 pattern 190–191
 validations 195
 Redis 210–220
 data mapper for 212–220
 installing 210
 lists in 211–212
 sets in 212
 strings in 211
 data structures 52–58
 chars, strings, and
 numbers 52–53
 keywords and symbols 53
 nil, truth, and falsehood
 values 52
 sequences 53–58
 lists 54
 maps 56–58
 vectors 55–56
 data types
 defrecord 360–363
 deftype 363–364
 Java language support for 363
 reify 364–366
 date_operations_spec.clj
 file 170
 date-operations
 namespace 171–172
 day-from function 178
 db-query function 142–143
 dcf.clj file 117
 declare macro 66, 159
 decoding 205
 decrement function 177
 DefaultSelenium 24

defclass function 328–334
 defcurried macro 319, 321
 defhtml macro 236
 define construct 8
 defmacro 370–371, 377–382
 defmethod 97–102, 104
 defmocktest construct 186–187
 def-modus-operandi
 macro 346–349, 351,
 355–356
 defmulti 97–98, 100–104
 defn macro 61–62, 197
 defonce macro 160
 defprotocol protocol 355–360
 and nil argument 359
 defining new 356–357
 extend function 358
 extend-type macro 357–358
 participating in 357
 reflecting on 359–360
 defrecord function 339, 360–
 364, 366
 def-redis-type macro 213, 217,
 219
 def-rot-encrypter macro
 377–378
 defsegment 387–390, 392–395
 deftest macro 170
 deftype data type 363–364
 defwebmethod macro 163–164
 defworker 249–250, 255, 257,
 260, 269
 delayed computation, and
 closures 322–323
 deliver function 115, 146
 delivery-from function 245, 247,
 253, 257, 262, 264–265, 268
 deref function 131–132, 136–
 138, 140, 143, 146
 derive function 100
 design, driven by DSLs 383–385
 destroy-record function 193,
 197
 destructuring 83–87
 map bindings 86–87
 vector bindings 84–86
 & symbol 84–85
 nested vectors 85–86
 detail-modus-operandi
 macro 348–353
 direct access, to HBase 200–202
 dirty reads, and unrepeatable
 reads 123
 discounted-cash-flow
 function 117

- DISPATCHED status 291
 - dispatch-fn 97–98
 - dispatching job 292–293
 - dispatch-reporting-jobs
 - function 47
 - dispatch-value 98
 - dispatch-worker 252
 - dissoc function 57, 361
 - dist_fact.clj file 295
 - dist-fact namespace 299
 - distributed parallel
 - programming 249–272
 - from-swarm macro 260–261
 - handling exceptions 271
 - multicasting messages 261–266
 - message-seq macro 263–265
 - next-message-from macro 262–263
 - send-message macro 261–262
 - parameterizing wait time 272
 - rebinding variables 272
 - remote workers 249–253
 - calling all 266–271
 - defining new 250–251
 - handling work
 - requests 251–252
 - sending work requests 252–253
 - request priorities 271
 - testing framework 258–260, 271
 - work requests 253–256
 - fire and forget 255–256
 - handling 253–254
 - message-handling loop 255
 - processing 254
 - do form 155
 - do functions, side effects
 - with 39
 - doall function 78
 - do-another-thing function 39
 - doc macro, and find-doc function 33–34
 - do-first-thing function 39
 - domain-specific languages. *See* DSLs
 - do-many-things function 39
 - doseq macro 45, 47
 - dosync 150
 - dosync block 22
 - dosync function 39, 118, 132–133, 139–140, 144
 - dosync macro 134
 - dot dot macro 110–111
 - dot operator 110
 - dot special form 109–110
 - dotimes macro 45, 47
 - doto macro 111–112
 - do-to-all function 309–310, 312
 - dots 108
 - double method dispatch 93–95
 - DSLs (domain-specific languages) 15, 383–395
 - and macros 158
 - design driven by 383–385
 - user classification 385–395
 - adding primitives to execution engine 391–394
 - data element 386
 - dynamic updates 395
 - increasing
 - combinability 394
 - power of DSL 391
 - segmenting 387–391
 - session persistence 386–387
 - dsl-store namespace 390
 - duck typing 92
 - dynamic languages, Clojure
 - as 18
 - dynamic polymorphism 92
 - dynamic scope 75–77
 - dynamic updates 395
- E**
-
- echo-app function 231
 - else expression 151
 - encoding 205
 - encrypt13 function 378
 - enter password string 12
 - equals method 362
 - error handling 354
 - errors
 - and agents 138–139
 - task 298–300
 - every? function 68
 - exchange, for AMQP 243
 - exchange-name parameter 261–265
 - exchange-type parameter 262–265
 - execution engines, adding primitives to 391–394
 - exhibits-oddity 151, 155
 - expand-method function 347–350
 - Expense class 341–346, 350
 - Expense data type 357
 - expense namespace 340, 351, 355, 360, 362
 - ExpenseCalculations 350–359
 - expense-report function 75
 - ExpensesConnection object 16
 - expenses-greater-than function 181
 - expense-test namespace 341, 352–353
 - expensive-audit-log 249, 259–260, 266–267, 270–271
 - expression problem 339–366
 - data types
 - defrecord 360–363
 - deftype 363–364
 - Java language support
 - for 363
 - reify 364–366
 - if-then-else constructs 344
 - in Clojure language 340–341
 - in Java language 341–343
 - modus operandi 346–354
 - def-modus-operandi macro 346–347
 - detail-modus-operandi macro 347
 - error handling 354
 - tracking of 348–354
 - monkey patching 344
 - multimethods solution 344–346
 - protocols 355–360
 - wrappers 343
 - extend function 358
 - extend-protocol protocol 355–360
 - and nil argument 359
 - defining new 356–357
 - extend function 358
 - extend-type macro 357–358
 - participating in 357
 - reflecting on 359–360
 - extend-type macro 357–358
 - extensibility 25–26
 - eXtensible Markup Language. *See* XML
 - external DSLs 388
 - external libraries 81
 - extreme programming. *See* XP

F

fact function 295
 factorial function 48, 295–297, 299–300, 303
 falsehood value, nil and truth values and 52
 fanex 263, 265–267
 fast-calc function 69
 favorite-movies set 126–127
 fcf.clj file 117
 fetch functions 179
 fetch-all-expenses function 179–180, 186
 fetch-expenses-greater-than function 179, 181
 Fibonacci numbers 20
 fields, methods and 108–112
 file module 231
 file-info module 231
 files, adding to calculators example 116–117
 filter function 48, 312, 316, 329, 333, 336–338
 filtering lists 311–312
 final-amount-> function 51
 finally clause 41
 find-by-sql function 197
 find-doc function 33–34
 find-method 330, 334–336
 find-ns function 83
 find-records 193, 196
 find-total-expenses function 47
 fire and forget, work requests 255–256
 first-expr 373
 flatten-content function 239
 fn form 37
 for form 49–50
 for loop 20
 format method 93
 forms 12, 109–110
 free variables, and closures 321–322
 free-cash-flow function 117
 from-proxies function 296, 298, 306
 from-swarm macro 256–257, 259–261, 268, 272
 function composition 69
 functional iteration 45–50
 doseq and dotimes macros 47
 filter function 48
 for form 49–50
 loop/recur function 45–47

map function 47–48
 reduce function 48–49
 while macro 45
 functional languages, Clojure as 18–23
 concurrency and multicore future 21
 higher-order functions 19
 immutability 19–20
 lazy and infinite sequences 20–21
 software transactional memory 22–23
 functional programming, for Clojure language 11
 functions 36–37, 61–73
 anonymous 70–71
 calling 67
 currying 316–321
 defining 61–67
 multiple arity 64
 mutually recursive functions 65–67
 recursive functions 65
 variadic functions 64–65
 definition of 37
 higher-order 67–70
 complement function 68
 constantly function 68
 every? and some functions 68
 memoize function 69
 partial function 68
 writing 69–70
 keywords and symbols 71–73
 partially applying 312–315
 adapting functions 314–315
 defining new functions 315–316
 variable arity 37
 future? function 146
 future-cancel? function 146
 future-cancelled? function 146
 future-done? function 146
 futures 145–146

G

gen-class macro 117–120
 generalizing map/reduce paradigm 276–279
 generateASM 95, 102
 generated classes 116
 gen-interface macro 117–120

gensym macro 234, 239
 getCurrentStatus method 119
 get-record 192–194, 196
 getSecret method 119
 GMT-FORMAT 282–283
 Goetz, Brian 123
 googling-clojurians 387, 391–395
 greatest-of function 310–311
 greet method 328
 GregorianCalendar class 171, 173, 175
 Grizzly project, for HTTP interface 222–224
 GrizzlyAdapter class 114
 GrizzlyAdapter interface 115, 222
 GrizzlyWebServer class 222–226
 group-by function 275–276, 278, 285–286

H

Hadoop Distributed File System. *See* HDFS
 Hadoop map/reduce framework 288
 handle-multiple-messages function 247–249
 handle-request-message 253–255, 258, 267, 269–270
 hashCode method 362
 hashes 9–10
 has-many relationship 193–194, 203–204
 hatt function 66
 HBase 197–210
 column oriented 198–199
 data mapper for 202–204
 reading from HBase 208–210
 writing to HBase 204–208
 direct access to 200–202
 scalability 198
 schema design 199
 versions 199
 HBase column 198
 HBase row 208
 HBase table 198–201, 203–204, 207–208, 210
 HBaseConfiguration table 201, 208
 hbase-site.xml 200
 HDFS (Hadoop Distributed File System) 198

hello function 229
 Hello, world program 32–33
 Hickey, Rich 23
 hierarchies, ad hoc. *See* ad hoc hierarchies
 higher-order functions 19, 67, 69–70, 308–312
 collecting results of 308–310
 complement 68
 constantly 68
 every? and some 68
 filtering lists 311–312
 memoize 69
 partial 68
 reducing lists 310–311
 writing 69–70
 highest-expense-during function 81
 homoiconic 14
 html macro 236, 238–239
 HTTP interface 222–229
 adding JSONP support to 224–227
 supporting cookies 227–229
 using Grizzly project 222–224

I

id field 192
 identities, and values 125–129
 immutability 126–129
 objects and time 127
 if construct, anaphoric 369–371
 generalizing 370–371
 implementing 370
 if form 43, 150
 if-let form 369–371, 374
 if-then-else constructs 344
 IGNORE string 279
 immutability 19–20
 and concurrency 128–129
 of values 126–127
 requirements for 130
 import form 24
 import macro 107
 Incanter project 25
 increment function 177
 increment-day 175
 increment-month 176
 increment-year 176
 independent changes 136, 140
 infinite data structures 20
 infinite sequences, lazy
 sequences and 20–21
 infix macro 161

init-model function 191–192, 194–197
 in-ns function 83
 insert-into-hbase function 208
 in-session 387, 393–394
 installing, Redis 210
 instances, creating 108–112
 interfaces
 implementing 114–115
 Java classes and 117–120
 internal DSLs 388
 Interop functionality, Java. *See* Java Interop functionality
 in-transaction values 135
 investigate-sessions function 287
 IPersistentMap 345, 347, 350–351, 355, 357, 361
 is_category_QMARK protocol 356
 is-category function 350–353, 355–358
 ISeq interface 25, 54, 56, 85
 ISeq sequence 53
 is-long? function 19
 it symbol 372
 item-total function 63
 iteration, functional. *See* functional iteration

J

jabberwocky.txt file 274–275, 278–279
 JAR file 31, 81, 200, 211
 Java classes
 and interfaces 117–120
 importing 107–108
 Java interface 93, 356, 358, 363
 Java Interop functionality 106–121
 calling 107–115
 arrays 113–114
 bean macro 113
 Clojure language from 120–121
 creating instances and accessing methods and fields 108–112
 implementing interfaces and extending classes 114–115
 importing Java classes 107–108
 memfn macro 112
 compiling Clojure code to Java byte code 115–120
 calculators example 115–117
 gen-class and gen-interface macros 117–120
 Java language
 byte code 115–120
 calling Clojure language from 24–25
 calling from Clojure language 24
 class hierarchy 101
 expression problem in 341–343
 support for data types 363
 Java Messaging Service. *See* JMS
 Java method 52, 112, 121, 139
 Java object 79, 108, 110–111, 114
 Java Virtual Machine. *See* JVM
 java.lang.Long 109
 javax.swing.JComb 101
 javax.swing.JComponent 101
 javax.swing.JFileChooser 101
 JComboBox 101
 JComponent 101
 JDBC-redis library 210
 JFileChooser 101
 JIT (just in time) compiler 13
 JMS (Java Messaging Service) 241
 job-id 290–294, 296–302, 304–306
 jobs
 defining 290
 dispatching 292–293
 rerunning 300–306
 running 296–298
 JRedis library 210
 JSONP callbacks 225
 jsonp parameter 225–227
 JSONP support, for HTTP interface 224–227
 JSR 914 241
 judge-credentials function 224, 226, 228
 just in time. *See* JIT
 JVM (Java Virtual Machine) 4
 as source of power for Clojure language 11
 Clojure as 23–26
 calling Clojure language from Java language 24–25

- JVM (*continued*)
 calling Java language from Clojure language 24
 extensibility 25–26
 type hints 25
- K**
-
- k arguments 314
 keys-config object 205–210
 keyword functions 71
 keywords, and symbols 53, 71–73
 klass function 326–336
- L**
-
- languages, domain-specific.
See DSLs
 large-scale data processing 288–289
 Cascading and cascading-clojure 289
 Cascalog 289
 Hadoop map/reduce framework and clojure-hadoop 288
 laziness, and special variables 77–78
 lazy sequences, and infinite sequences 20–21
 lazy-message-seq function 248
 lazy-request-seq 280–281, 283
 lazy-seq 21, 281, 283
 let bindings 53
 let block 322
 let form 37–38, 78–79, 156, 180
 let macro 368–369, 371
 lexical analyzer 12
 lexical closures 79–80
 lexical term 74
 lexically scoped variable 74
 Lisp
 as source of power for Clojure language 10–17
 Clojure language as reincarnation of 4–5
 list function 54, 150, 153
 list-of-expenses sequence 67
 lists 9–10
 filtering 311–312
 in Redis 211–212
 reducing 310–311
 uniqueness of 54–55
 load-code function 392–393, 395
 load-string 391–392
 localState method 119
 locking 124–125
 disadvantages of 124–125
 new problems with 125
 locks 124
 log-call function 179, 182–183
 log-file format 280
 log-function-call 15
 LoginController 280–282, 284
 login-user function 163
 log-message action 39, 140
 logs. *See* parsing logs
 long-computation-one function 249–250, 259–260, 270
 long-computation-two function 249, 259–260, 270–271
 long-run function 32, 145
 loop/recur construct 65
 loop/recur function 45–47
 lost updates 123
- M**
-
- macro system 14–15
 macro-removing boilerplate example 15–17
 macro-syntactic abstraction example 17
 metaprogramming with 15
 macroexpand 160, 368, 370, 378, 380–382
 macro-generating macros 379–383
 example template 379
 implementing make-synonym macro 380–382
 reasons to use 382–383
 macro-removing boilerplate 15–17
 macros 148, 154–166, 368–383
 anaphoric 369
 and macro 160
 assert-true macro 164
 bean 113
 benefits of 157–158
 and DSLs 158
 code generation 158
 convenience 158
 comment macro 159
 declare macro 159
 defonce macro 160
 defwebmethod macro 163–164
 doc, and find-doc function 33–34
 dot dot 110–111
 doto 111–112
 infix macro 161
 macro-generating 379–383
 example template 379
 implementing make-synonym macro 380–382
 reasons to use 382–383
 memfn 112
 randomly macro 162–163
 reader 41–42
 shifting computation to compile time 374–379
 templates for 153–157
 generating names in 156–157
 splicing 155–156
 textual substitution with 149–150
 threading 50–52
 thread-first 50–51
 thread-last 51–52
 thread-it 371–374
 time macro 160–161
 using unless function 150–153
 while 45
 macro-syntactic abstraction 17
 maintaining status 290–292
 makeSound 92–94
 make-synonym macro 379–382
 Man class 94
 managed references 131
 map argument 47
 map bindings 86–87
 map function 47–48, 288, 309–310
 map/reduce algorithm 278
 map/reduce paradigm 274–289
 analyzing rails sessions 285–288
 sessions analysis 286–288
 session-seq 285–286
 counting words 274–276
 generalizing 276–279
 large-scale data
 processing 288–289
 Cascading and cascading-clojure 289
 Cascalog 289
 Hadoop map/reduce framework and clojure-hadoop 288

- map/reduce paradigm
 - (*continued*)
 - parsing logs 279–285
 - log-file format 280
 - rails requests 284–285
 - requests sequences 280–284
- map-reduce function 277–279, 284–285, 287
- maps 56–58
- mark-dispatched function 291–293, 296–297, 304–305
- master/slave
 - parallelization 289–306
 - defining job 290
 - defining slave 293–295
 - dispatching job 292–293
 - maintaining status 290–292
 - rerunning job 300–306
 - running job 296–298
 - seeing task errors 298–300
 - using master-slave framework 295–296
- master-slave framework 295–296
- matches? function 387, 391–395
- MAX-CONNECTIONS var 74
- McGranaghan, Mark 229
- member access 109
- member-since data element 71
- memfn macro 112
- memoize function 69
- memory, software
 - transactional 22–23
- message queue, for AMQP 243
- message-object parameter 244, 261–262, 264
- message-seq 247–249, 263–265, 267, 272
- message-seq function 248
- message-seq macro, multicasting messages 263–265
- messaging 272
 - ActiveMQ server 242
 - AMQP 241, 243
 - and distributed parallel programming 249–272
 - from-swarm macro 260–261
 - handling exceptions 271
 - multicasting messages 261–266
 - parameterizing wait time 272
 - rebinding variables 272
 - remote workers 249–253, 266–271
 - request priorities 271
 - testing framework 258–260, 271
 - work requests 253–256
- JMS 241
- RabbitMQ server 242–248
 - connecting to 243–244
 - receiving messages 244–248
 - sending messages 244
- STOMP 241
- ZeroMQ server 242
- meta function 88
- metadata 88–89
- metaprogramming, with macro system 15
- method dispatch 92–97
 - multiple 96–97
 - single and double 93–95
 - visitor pattern 95–96
- methods
 - and fields 108–112
 - defining 328–330
 - invoking 330–331
- method-specs function 329–330, 333, 336–338
- middleware modules, Ring project 230–232
- mock-calls atom 184–186
- mocking functions 178, 181–182
 - clearing recorded calls 185–186
 - removing global state 186
 - verifying mocked calls 184–185
 - versus stubs 182–186
- mocking macro 183
- models, unified access 143–144
- modus operandi 346–354
 - def-modus-operandi macro 346–347
 - detail-modus-operandi macro 347
 - error handling 354
 - tracking of 348–354
 - during def-modus-operandi macro 348
 - during detail-modus-operandi macro 348–353
 - querying modus operandi 353–354
- mo-methods-registration function 348–349
- monkey patching 344
- month-from function 173, 178
- multicasting messages, distributed parallel
 - programming 261–266
 - message-seq macro 263–265
 - next-message-from macro 262–263
 - send-message macro 261–262
- multicore future, concurrency and 21
- multimethods 80, 97–105
 - ad hoc hierarchies 100–103
 - Java language classes 101
 - resolving conflicts 102–103
 - visitor pattern 102
 - as solution to expression problem 344–346
 - description of 97–99
 - functions without 97
 - multiple dispatch 99–100
 - Redis-clojure client 103–105
- multiple arity 64
- multiple dispatch 96–97
- multiversion concurrency control. *See* MVCC
- mutating
 - agents 136–137
 - send function 136–137
 - send-off function 137
 - atoms 141–142
 - compare-and-set! function 141–142
 - reset! function 141
 - swap! function 141
 - ref construct 132–134
 - alter function 132–133
 - commute function 133–134
- mutation
 - state and unified access model 143
 - watching for 144–145
- mutually recursive functions 65–67
- MVCC (multiversion concurrency control) 22, 135–136
- my-own-function 14–15
- MySQL 190–197
 - associations in 193–194
 - callbacks 195
 - clj-record.boot namespace 196
 - clj-record.core namespace 196–197

MySQL (*continued*)
 code organization of clj-
 record 197
 user model 191–193
 creating records 192
 deleting records 193
 reading records 192–193
 updating records 193
 using active record design
 pattern 190–191
 validations 195
 MySQL database 190
 MySQL table 190

N

n arguments 314
 names, in macros 156–157
 namespaces 80–83
 all-ns and find-ns functions 83
 create-ns and in-ns
 functions 83
 ns var 80–82
 reload and reload-all
 references 82
 use and require
 references 81–82
 ns-interns and ns-publics
 functions 83
 ns-resolve and resolve
 functions 83
 ns-unmap and remove-ns
 functions 83
 nested vectors 85–86
 new form 108
 new IO. *See* NIO
 new operator 327
 new-class function 326–327,
 333–337
 new-connection 244, 246, 264
 NewExpense 360–363
 new-expense function 340–342,
 351–352, 359, 361–362
 new-job function 290, 292, 297,
 299, 301–302, 304
 new-redis-type 216–217, 219
 new-user function 323–325
 nextDelivery method 245, 264
 next-message-from 245–247,
 261–265
 next-message-from macro
 262–263
 next-terms 20–21
 nil argument, extend-protocol
 protocol and 359

nil value, truth and falsehood
 values and 52
 NIO (new IO) framework 222
 NodeVisitor 95–96
 noenpnqnoen 374, 376, 378
 notation, prefix 34–35
 ns macro 80
 ns var 80–82
 reload and reload-all
 references 82
 use and require
 references 81–82
 ns-interns function 83
 ns-publics function 83
 ns-resolve function 83
 ns-unmap function 83
 nth function 84
 NullPointerException 52, 155
 num1 variable 321–322
 num2 variable 321–322
 NUM-ALPHABETS 375, 377
 Number class 52
 numbers, chars and strings
 52–53

O

Object class 363
 object orientation, Clojure
 language support for
 26–29
 object system, for closures
 325–338
 class inheritance 332–337
 creating instances 326–327
 data abstraction 337
 defining classes 325–326
 defining methods 328–330
 invoking methods 330–331
 referring to this 331–332
 state of 327–328
 object-oriented. *See* OO
 object-relational mapping. *See*
 ORM
 objects
 and closures 323–325
 and time 127
 old-name symbol 380–382
 on-response 253, 257, 268
 on-swarm function 251, 257,
 268–269, 365
 OO (object-oriented)
 paradigm 26
 org.currylogic.damages.calcula-
 tors namespace 81

ORM (object-relational
 mapping) 212

P

painting object 207
 paintings table 202–203, 205
 parameters, handling with
 Compojure 233
 params module 231
 parent-class-spec function 333,
 337–338
 parentheses 7–9, 113
 Park class 94–95
 parse-line 274–276, 278–279
 parsing logs 279–285
 log-file format 280
 rails requests 284–285
 requests sequences 280–284
 partial function 307–308,
 315–321
 partially applying,
 functions 312–315
 adapting functions 314–315
 defining new functions
 315–316
 partially-applied function
 314–315
 persistence, of session
 386–387
 Person class 127
 Person function 326
 Person interface 94
 phantom reads 124
 Pmap function 143
 polish notation 34
 polymorphism 90–105
 duck typing 92
 method dispatch 92–97
 multiple 96–97
 single and double 93–96
 visitor pattern 95–96
 multimethods 97–105
 ad hoc hierarchies
 100–103
 description of 97–99
 functions without 97
 multiple dispatch 99–100
 Redis-clojure client
 103–105
 subtype 91–92
 poorest-first function 70
 postconditions 63
 postwalk function 388–390,
 393–394

power, sources of for Clojure
 language 10–26
 functional programming 11
 JVM 11
 Lisp 10–17
 prefer-method 103
 prefix notation 34–35
 present-value function 116
 price-calculator-for-tax
 function 313–314
 price-with-tax function
 313–314
 prime? function 49
 primes-less-than function 49
 primitives, adding to execution
 engine 391–394
 print-amounts function 84
 println function 368, 371, 379,
 382, 393
 print-the-var function 76
 private functions 81
 process-request function 254,
 258, 269–270
 profit-level 100
 program flow 42–52
 conditional forms 43–45
 functional iteration 45–50
 doseq and dotimes
 macros 47
 filter function 48
 for form 49–50
 loop/recur function
 45–47
 map function 47–48
 reduce function 48–49
 while macro 45
 programmable programming
 language 15
 programming, functional 11
 promises 146
 protocols, defprotocol and
 extend-protocol 355–360
 and nil argument 359
 defining new 356–357
 extend function 358
 extend-type macro
 357–358
 participating in 357
 reflecting on 359–360
 protocols.expense-record
 namespace 360–362
 proxy macro 114
 public functions 81
 Put class 202

Q

querying, modus operandi
 353–354
 QueueingConsumer 243, 245,
 262, 264–265
 queue-name parameter 245,
 247, 262–265
 quote form 41

R

RabbitMQ server 242–248
 connecting to 243–244
 receiving messages 244–248
 sending messages 244
 RABBITMQ-CONNECTION
 var 74
 rails requests 284–285
 rails sessions, analyzing 285–288
 sessions analysis 286–288
 session-seq 285–286
 rand-int function 162
 randomly macro 162–163
 random-queue-name
 function 262, 264–265
 reader macros 41–42
 readers 13–14
 read-eval-print loop. *See* REPL
 reading, from HBase 208–210
 read-lines function 275–276,
 278–281, 283, 285
 reads
 dirty and unrepeatable 123
 phantom 124
 read-string 253, 257–258, 268–
 269, 282–283
 receiving messages, with Rab-
 bitMQ server 244–248
 record function 192
 records, in MySQL
 creating 192
 deleting 193
 reading 192–193
 updating 193
 recordSound function 92
 recover-job 302–304
 recur form 46
 recursive functions, mutual
 65–67
 Redis 210–220
 data mapper for 212–220
 implementing 215–219
 persistence 214–215
 using Redis objects 214
 installing 210
 lists in 211–212
 sets in 212
 strings in 211
 redis namespace 211
 Redis protocol 104
 Redis server 103
 Redis type 219
 Redis-aware object 213, 219
 Redis-clojure client 103–105
 redis-clojure library 210–211,
 213, 291
 Redis-clojure project 211
 redis-config function 294, 296–
 297, 299, 304–305
 reduce functions 48–49, 114,
 288
 reducer function 276–279, 284–
 285, 287–288
 reducing lists 310–311
 ref construct 131–136
 mutating 132–134
 alter function 132–133
 commute function 133–134
 STM 134–136
 ACI properties of 135
 MVCC 135–136
 transactions 134–135
 ref type 131
 references, managed 131
 reflecting, on extend-protocol
 protocol 359–360
 ref-set function 118, 132–133,
 141, 144
 reify data type 364–366
 reify macro 339, 360, 363–365
 reload module 231
 reload reference, and reload-all
 reference 82
 reload-all reference, reload ref-
 erence and 82
 remote workers, distributed
 parallel programming
 249–253
 defining new 250–251
 handling work requests
 251–252
 sending work requests
 252–253
 RemoteWorker 365–366
 remove-ns function, ns-unmap
 function and 83
 remove-watch function 144
 REPL (read-eval-print loop) 18,
 31–32

request map 163
 request object 163
 requests sequences 280–284
 request-seq 280–287
 require function 82
 require namespace 107
 require reference, use reference
 and 81–82
 rerunning job 300–306
 reset function 141, 144
 resolve function, ns-resolve func-
 tion and 83
 response, from
 Compojure 234–235
 response-for function 254, 258,
 269
 rest-expr 373
 Result object 208
 return-final-value 39
 return-queue-name 252
 Ring framework 229–230
 Ring project 229–232
 middleware modules
 230–232
 overview 229
 root binding 74, 160
 root program tag 8
 ROT13 374–376
 routes* function 233
 routing key, for AMQP 243
 routing-key parameter 244,
 261–265
 RT class 121
 running job 296–298
 run-reports function 47
 run-task function 292–294, 300,
 302, 304
 runtime type 94
 run-worker-everywhere 266–
 267, 269–271, 395

S

scalability, of HBase 198
 scale variable 79
 Scan class 202
 schema design, of HBase 199
 scope 73–80
 dynamic 75–77
 let form 78–79
 lexical closures 79–80
 vars and binding 74–78
 dynamic scope 75–77
 laziness and special
 variables 77–78

special variables 75
 thread-local state 77
 SDK (software development
 kit) 222
 segmenting users 387–391
 select-if function 312, 315
 Selenium object 24
 send function 136–137
 sending messages with Rab-
 bitMQ server 244
 send-message function 244–246,
 254–255, 257–258, 268–269
 send-message macro 261–262
 send-message-on-queue
 action 140
 send-off function 136–137, 140,
 144
 seq-exprs vector 49
 sequences 53–58
 lazy and infinite 20–21
 lists uniqueness of 54–55
 maps 56–58
 vectors 55–56
 service method 114
 service-http-request
 function 223–224, 226–227
 service-name function 250–251,
 257, 269
 session persistence 386–387
 session-assoc 235
 session-id 283–287
 sessions analysis 286–288
 sessions handling with
 Compojure 235–236
 session-seq 285–286
 sets in Redis 212
 s-expressions 12, 36
 sharding 198
 shared state 123–124
 shift-by 375–378
 shifted-by 378
 shifted-tableau function
 375–379
 shortcuts for anonymous
 functions 71
 should-run? function 300
 side effects
 in STM transactions 139–140
 with do functions 39
 Simple Text-Oriented Messag-
 ing Protocol. *See* STOMP
 SimpleDateFormat 108–109,
 171
 single method dispatch 93–95
 slave, defining 293–295

slave-wrapper 294, 304
 software development kit. *See*
 SDK
 software transactional memory.
 See STM
 some function every? function
 and 68
 sorter-using function 70
 special forms dot 109–110
 special variables laziness and 75,
 77–78
 splice reader macro 156
 splicing for macros 155–156
 square brackets 9
 square-all function 308–309
 src directory 115, 211
 src/com/curry/utills folder 115
 StackOverflowError 65
 stacktrace module 231
 state 122–147
 agents 136–140
 await and await-for
 functions 138
 errors and 138–139
 mutating 136–137
 side effects in STM
 transactions 139–140
 validations 139
 and unified access
 model 143–144
 mutation 143
 reading 143
 transactions 143–144
 atoms 140–142
 for closures 327–328
 futures 145–146
 identities and values 125–129
 immutability 126–129
 objects and time 127
 managed references 131
 problems with 123–125
 locking as solution to
 124–125
 shared state 123–124
 promises 146
 ref construct 131–136
 mutating 132–134
 STM 134–136
 requirements for
 immutability 130
 vars 142–143
 watching for mutation
 144–145
 states thread-local 77
 static method 109

static term 74
 status, maintaining 290–292
 STM (software transactional memory) 22, 129, 134–136
 ACI properties of 135
 MVCC 135–136
 transactions 134–135, 139–140
 STOMP (Simple Text-Oriented Messaging Protocol) 241
 streams 20
 String class 52
 String object 112–113
 StringBuilder 238–239
 strings
 chars and numbers and 52–53
 in Redis 211
 stubbing 179–181
 stub-fn function 181–183, 185
 subSequence function 112
 subtype polymorphism 91–92
 superclass methods 119
 swap! function 141, 144
 symbol object 72
 symbols, keywords and 53, 71–73
 synchronous changes 131, 140
 sync-set macro 150
 syntax 6–10
 lists, vectors, and hashes 9–10
 of Clojure language 34–36
 case sensitivity 36
 prefix notation 34–35
 whitespace and comments 35–36
 XML and parentheses 7–9
 syntax quote character 154
 SyntaxNodes 93, 95–96

T

take function 21
 task errors 298–300
 task-id 290–294, 304–306
 task-statuses function 300–301, 303, 306
 TDD (test-driven development) 169–188
 dates and string example 170–178
 expense finders example 178–179
 mocking functions 181–182
 clearing recorded calls 185–186
 removing global state 186
 verifying mocked calls 184–185
 versus stubs 182–186
 organizing tests 187–188
 stubbing 179–181
 templates
 for macros 153–157
 generating names in 156–157
 splicing 155–156
 macro-generating macros example 379
 test-driven development. *See* TDD
 test-filter-greater-than test 187
 test-form 370–371
 testing macro 187
 testing string 187
 test-is library 170, 178
 textual substitution, with macros 149–150
 this keyword 336
 thread-first macro 50–51
 threading macros 50–52
 thread-first 50–51
 thread-last 51–52
 thread-it macro 371–374
 thread-last macro 51–52, 372
 thread-local state 77
 threshold-length parameter 19
 throw form, try/catch/finally blocks and 40–41
 Thrown class 55–56, 155–156, 164–165
 time macro 160–161
 time, objects and 127
 tokens array 114
 total_cents protocol 356
 total-all-numbers function 64
 total-amount function 340–341, 343–345, 361–362
 total-cents function 340, 350–352, 361–364
 total-cost function 64
 total-cpu-time function 136–137
 total-of function 310–311
 total-rows function 140–141
 tracking, of modus operandi 348–354
 during def-modus-operandi macro 348
 during detail-modus-operandi macro 348–353
 querying 353–354
 trampoline function 66
 transactional memory, software 22–23
 transactions
 state and unified access model 143–144
 STM 134–135, 139–140
 truth value, nil and falsehood values and 52
 try/catch/finally blocks, and throw form 40–41
 try-catch block 322–323
 twice function 77
 type hints 25
 types, data. *See* data types

U

unary function 47
 underscore identifier 38
 unified access model, state and 143–144
 mutation 143
 reading 143
 transactions 143–144
 unless control 150
 unless function 150–153
 unquote splicing macro 156
 unquote-slice 381
 unquoting 155
 unrepeatable reads, dirty reads and 123
 up-case function 79
 update-in function 58, 349–350
 update-on-response function 252–253, 256–257, 268
 update-response function 234–235
 updates
 dynamic 395
 lost or buried 123
 update-status-as function 291–292, 305
 use clause 173
 use namespace 107
 use reference, and require reference 81–82
 user model, for MySQL 191–193
 creating records 192
 deleting records 193
 reading records 192–193
 updating records 193
 username function 70

users table 190, 192
 users, classification of 385–395
 adding primitives to execution engine 391–394
 data element 386
 dynamic updates 395
 increasing combinability 394
 power of DSL 391
 segmenting 387–391
 session persistence 386–387

V

validation namespace 191
 validations 139, 195
 validator function 135
 validator-fn function 139
 values, identities and 125–129
 immutability 126–129
 objects and time 127
 values-from 296–300, 306
 var function 378, 387, 389, 393–394
 Var object 121
 var type 131
 variable arity 37
 variable capture 157
 variables
 laziness and 77–78
 special 75
 variadic functions 64–65
 vars 142–143
 compile-path 117
 and binding 74–78
 dynamic scope 75–77
 special variables 75, 77–78
 thread-local state 77
 ns 80–82
 reload and reload-all references 82
 use and require references 81–82

vector bindings 84–86
 & symbol 84–85
 nested vectors 85–86
 vectors 9–10, 55–56, 85–86
 versions, of HBase 199
 visitor pattern
 ad hoc hierarchies 102
 method dispatch 95–96

W

wait-until-completion
 function 256–257, 260, 268
 watching, for mutation 144–145
 web services 221–239
 clj-html library 236–238
 Compojure web-application framework 232–236
 handling parameters 233
 Hello, world!
 application 232–233
 response from 234–235
 sessions handling with 235–236
 with-request-bindings macro 234
 HTTP interface 222–229
 adding JSONP support to 224–227
 supporting cookies 227–229
 using Grizzly project 222–224
 Ring project 229–232
 middleware modules 230–232
 overview 229
 while form 45
 while loop 45
 while macro 45
 whitespace, and comments 35–36
 with-audited-connection 16–17

with-california-taxes
 function 313–314
 with-cookies plug-in 233
 with-rabbit form 244
 with-rabbit macro 244
 with-request-bindings macro 234
 with-request-params plug-in 233
 with-session 235
 Woman class 94, 333–334, 336
 word-frequency function 275–276, 278
 work requests, distributed parallel programming 253–256
 fire and forget 255–256
 handling 253–254
 message-handling loop 255
 processing 254
 worker-data 251–252, 257, 268–269
 worker-init-value 251, 256–257, 267–269
 worker-runner 250–251, 257, 269
 wrappers 343
 writing, to HBase 204–208

X

XML (eXtensible Markup Language) 7–9
 XP (extreme programming) 177
 XSLT transforms 158

Y

year-from function 173, 178

Z

ZeroMQ server 242