

# Learn You Some Erlang for Great Good!

A Beginner's Guide



**Fred Hébert**

*Foreword by Joe Armstrong*



# INDEX

## Symbols and Numbers

- \* (multiplication), 10
- ! (bang), 144–145. *See also* message passing
- { } (empty tuple). *See* tuples
- = (match operator)
  - binding/pattern matching, 12
  - for comparison, 11
- == (equal to), 14–15
- := (exactly equal to), 14–15
- != (exactly not equal to), 14–15
- <= (less than or equal to), 15
- / (floating-point division), 10
- /= (not equal to), 14–15
- > (greater than), 15
- >= (greater than or equal to), 15
- < (less than), 15
- <= (binary generator), 28
- <- (list generator), 22
- <<>> (empty binary). *See* binaries
- <<">> (empty binary string).  
*See* strings
- (subtraction), 10
- (list subtraction), 19
- + (addition), 10
- ++ (list concatenation), 19
- '\$1', '\$2', ..., '\$\_'. *See* match specifications
- '\$end\_of\_table', 427. *See also* ETS
- "" (empty string). *See* strings
- ? (question mark). *See* macros
- ?MODULE, 152
- [] (empty list). *See* lists
- \_ (don't care variable), 17

- | (cons operator), 20–21
- || (list comprehension operator), 22

## A

- abstraction module,
  - API/Interface module, 295–296
  - example with `gen_event` behavior, 256
- accumulator, 64–65
- ACID, 424–425
- actor model, 3
- addition (+), 10
- administrator (there is only one), 449
- alias (type alias), 553
- alive (node), 452
- Amdahl's law, 141–142
- analyzing Erlang code, 318. *See also* Dialyzer
- andalso operator, 14, 48–49
- and operator, 14
- angry programmers, 332
- anonymous functions (funs), 78–82
- .app file
  - description, 305–306
  - examples
    - with missing fields, 308, 317
    - with proper fields, 336, 480, 522
  - registered field details, 317
- applications (OTP)
  - behavior, 304, 309
  - callback module example, 311
  - complex terminations, 333
  - configuration, 306–307

- applications (OTP), *continued*
  - currently running, 312
  - dependencies, 307, 317, 337
  - distributed, 473
    - example, 480
  - finding versions, 338–339
  - included applications, 333
  - library applications
    - (process-free), 314
  - loading, 331
  - named processes, 468
  - overriding, 317, 332, 341, 361, 481–482
  - restart strategy (temporary, transient, permanent), 313
    - with reltool, 343
  - statuses, 474
  - start, 310, 311–312, 313
  - stop, 312, 333
- application controller, 309, 474
- application master, 309–310
- appups
  - definition, 357
  - file definition, 365–366
  - file example, 366–367
- arithmetic operations, 10
- arity, 33
- arrays, 129
- assertion macros, 400–401.
  - See also* EUnit
- asynchronous message passing, 139, 445–446
- atoms, 12–13
- atom table, 13
- availability, 453–454. *See also* CAP theorem

**B**

- backward compatibility notice
  - recursive types, 553–554
  - simple\_one\_for\_one child restart strategy, 270
  - stack traces, 101
  - supervisor:terminate\_child function, 280
  - typed behaviors, 566, 567
- bad argument error, 91
- bad arithmetic error, 92
- bad arity error, 92
- bad function error, 92
- band operator, 26–27
- band supervisor, 271, 274
- bandwidth, 446–447
- bang (!), 144–145. *See also* message passing
- BEAM (virtual machine), 37
- .beam files, 36. *See also* compiling code
- behavior. *See also* applications (OTP); gen\_event behavior; gen\_fsm behavior; gen\_server behavior; supervisor (OTP)
  - defining behaviors, 214
  - principles, 200–201
  - typed behaviors, 566–567
- Berard, Edward V., 176
- BERT and BERT-RPC, 450
- BIFs. *See* built-in functions
- binaries
  - bit packing, 24
  - bit syntax, 23–24
  - compared to atoms, 28
  - pattern matching, 24–26
  - strings, 27–28
  - TCP segment example, 27
  - type specifiers, size, endianness, 25–26
- binary comprehension, 28–29
- binary generator (<=), 28
- binary tree, 72–75, 103–104
- binding, 46, 91
- bnot operator, 26
- Boolean, 14
- boot shell argument, 340–341
- boot file, 339, 340–341, 369
- boot script, 339–340
- bor operator, 26–27
- bottlenecks. *See* sequential bottlenecks
- bsl operator, 26–27
- bsr operator, 26–27

- built-in functions (BIFs), 20. *See also*
  - erlang module
  - in guards, 59
  - tracing, 574
  - type-test, 59
- bxor operator, 26–27

## C

- calculator example, 107
- Candea, George, 138
- CAP theorem, 453–454
  - zombie survivors example, 454
- case clause (error), 91
- case ... end expression, 52–53
- catch keyword, 100–103
- circular dependencies, 41
- client/server, 201, 204–207
- cluster (of Erlang nodes), 458, 461
- C Nodes, reference, 450
- code\_change functions, 363–365.
  - See also specific behaviors*
- code clashes, 196–197
- code path, 194–195, 331
- code purge, 191
- code server, 191
- comments, 34
- Common Test
  - all() function, 488
  - basic suite example, 488
  - configuration parameter, 392, 488–489
  - data\_dir value, 489
  - directories (not found error), 528
  - directory structure, 487
  - distributed, 504
    - logs, 506–507
    - running tests, 507–508
    - slave nodes, 505
    - test specifications, 505–506
  - fixtures (equivalent of), 491–492
  - groups, 493–496
  - instantiation, 491–492, 500
  - logs, test reports, 489–490, 501
  - Mnesia example, 519, 528–529
  - priv\_dir value, 487
  - running tests, 489, 503–504
  - skipping tests, 502
  - SSH, 504

- suites, 500–501
- test control, 495–496
- test specifications, 501–502, 503–504
- compiler flags, 37–38
- compile-time errors, 88–89
- compiling code
  - Emakefiles, 194–195, 308
  - with erlc (executable), c()
    - (function), and compile (module), 36
  - export\_all option, 182
  - with hipec, 38
- complex data structures, 60
- concurrency. *See also* processes
  - implementation, 140
  - vs. parallelism, 136
  - read and write with ETS, 424
- config shell argument, 482, 484
- configuration
  - AppName shell argument, 332
  - config shell argument, 482, 484
  - OTP applications, 306–307
    - (*see also .app* file)
  - overriding, 317, 332, 341, 361, 481–482
- cons operator (|), 20–21
- consistency, 453. *See also* CAP theorem
- continuation-passing style (CPS), 320–322
- cookies, 462–464
- cowboy server, 389
- crashing. *See* let it crash
- CT master, 507–508
- ct\_run executable, 489, 503–504
- ct:run function, 489, 503–504
- ct\_slave function, 504–505
- Cunningham, Ward, 54
- curling, 252

## D

- data structures
  - arrays, 129
  - binary tree, 72–75, 103–104
  - benchmark of key/value data structures, 128
  - complex, 60

- data structures, *continued*
  - dictionaries, 128–129
  - directed graphs, 131–132
  - general balanced, 128–129
  - IO lists, 375–377
  - ordered dictionaries, 127–128
  - priority queue, 159
  - property lists, 127
  - queues, 132–133, 293–295
  - sets, 130–131
    - ETS tables, 422
  - stacks, 108, 560–561
- databases
  - on disk (DETS), 433
  - full-featured (*see* Mnesia)
  - in-memory (*see* ETS)
  - table viewer, 536
- datagrams, 377
- dead (node), 452
- deadlock
  - FSM event deprivation
    - (hanging), 327–328
  - message collision
    - (asynchronous), 229
  - message collision
    - (synchronous), 226
  - messaging a dead process, 153–154
  - shared state, 173
  - supervisors and children
    - spawning, 390
- debug\_info compiler option, 37–38
- demonitor, 169–170
- dependencies
  - applications, 307, 317, 337
  - modules, 41
  - sibling processes, 288–291
- DETS, 433
- Dialyzer, 543
  - error messages, 548, 558, 562–563
  - exporting types, 564–565
  - foreign types, 565
  - function specifications, 557, 559
  - is never wrong, 572
  - none() type, 561
  - opaque types, 565–566
  - persistent lookup table (PLT), 543–544
  - polymorphic types, 567–570
  - recursive types, 554
  - running, 547, 550
  - singleton types, 550–551
  - typed behaviors (-callback attributes), 566–567
  - type definition, 551, 553, 554–557
  - type inference, 547, 548–549, 558
  - union and built-in types, 551–554
- dict module, 128–129
- directed graphs, 131–132
- directory structure
  - basic Erlang application, 180–181
  - multiple OTP applications, 331
  - OTP applications, 304–305, 307–308
  - releases (process quest example), 360–361
  - sysutils release, 340
- disconnecting nodes, 461
- disconnect\_node(Node) function, 461
- distributed application
  - controller, 474
- distributed applications, 473
- distributed Erlang
  - carrier protocol, 448
  - connecting nodes, 459, 465–466
  - cross-node communication, 459–460
  - heartbeats, 467
  - hidden nodes, 465–466
  - node names (long, short), 458
  - port range (configuration), 446
  - principles, 442–445
  - security model, 447–448
  - over SSL, 448
  - starting/stopping
    - programmatically, 467
- div operator, 10
- domains
  - function types, 562
  - node names, 458
  - socket communication, 380

- downloads
  - code for the book, 6
  - code for Process Quest, 353
- 'DOWN' messages, 169. *See also*
  - monitors
- dynamic types, vs. static types, 56

## E

- eaddrinuse error, 379–380
- EDoc, 305, 575
- Emakefile, 194–195, 308
- embarrassingly parallel, 140–141
- env shell argument, 331
- EPMD (Erlang Port Mapper Daemon), 443
- Ericsson
  - AXD 301, 56
  - PLEX/AXE, 137
- .erlang.cookie file. *See* cookies
- erlang module
  - disconnect\_node(Node), 461
  - exit(Pid), 94–95, 97–98
  - exit(Pid, Reason), 94–95, 163
  - get\_cookie(), 463
  - is\_alive(), 470
  - is\_process\_alive(Pid), 411
  - monitor\_node(Node, Bool), 460
  - node(), 459
  - node(PidPortOrRef), 461
  - nodes(), 459
  - nodes(Flag), 465
  - registered(), 171
  - register(Name, Pid), 171
  - send(Dest, Msg, Opts), 465
  - set\_cookie(Node, Cookie), 463
  - spawn/1-3, spawn\_link/1-3, 142–143, 146, 164
  - spawn/2-4, spawn\_link/2-4, 462
  - unregister(Name), 171
  - whereis/1, 172
- Erlang Port Mapper Daemon (EPMD), 443
- Erlang Run Time System (ERTS), 338, 351
- Erlang shell. *See* shell
- Erlang Term Storage. *See* ETS
- erlc, 36. *See also* compiling code

- erlcount
  - file counter, 330
  - OTP application example, 316
- ERL\_LIBS, 331, 351, 368
- ERL\_MAX\_ETS\_TABLES, 421
- erl\_syntax module, 318
- erroneous entry. *See* rhubarb
- error kernel, 284
- errors, 88–104
  - compile time, 88–89
    - let it crash, 4
    - logical, 89–90
    - runtime, 90–93
- ERTS (Erlang Run Time System), 338, 351
- ETS (Erlang Term Storage), 419
  - compression, 425
  - controlling process, heir, 424
  - iteration, 426–427
  - lookup and insertion, 425–426
  - matching, 427–428
  - primary key, 422
  - selecting, 428–433
  - table access, 423
  - table options, 423–425
  - table types, 422–423
- EUnit
  - examples, 409
  - execution control, 406–407
  - fixtures, 404–406
  - generators, 402
  - test description, 407
  - test representation, 403
  - test sets, 402
  - time-sensitive tests, 413–414
  - running tests, 398–399, 401
  - verbose output, 417
- event manager. *See* gen\_event
  - behavior
- eventual consistency, 457
- exceptions
  - dealing with, 96–103
  - raising, 93–96
- exercises left to the reader
  - fixing sockserv for relups, 371
  - ppool pool limits, 296
  - regis code\_change
    - implementation, 439

- exercises left to the reader,
  - continued*
  - RPN calculator extension, 111
  - sockserv enhancements, 396
- exit/1-2, 94–95, 97–98, 163
- exit exceptions, internal, 94–95
- exit(Pid), 94–95, 97–98
- exit(Pid, Reason), 94–95, 163
- exit signal ('EXIT')
  - definition, 94–95
  - exit/2 function, 163
  - kill vs. killed (exit reasons),
    - 167–168
  - trapped errors and exit/1,
    - 165–166
  - trapped errors and exit/2,
    - 166–168
  - trapping exits, 164–165
- export\_all, 38, 182
- expressions, 578
- external term format, 450
- .ez files, 345

## F

- failover and takeover, 475–476
  - boot procedure, 484
  - configuration, 481–482
  - examples, 483–484
    - client operations, 381
    - close socket, 383
    - connect, 383
    - connection-based
      - protocol, 377
    - connection closed by
      - client, 395
    - keepalive, 382
    - listen to connections, 382
    - server, parallel, 389–390
    - server-mode operations, 382
    - simple (sequential), 387–388
    - TCP, accept connection,
      - 382–383
- fallacies of distributed
  - computing, 445
  - short list, 451
- false Boolean value, 14
- fault tolerance, 138–139
- file:consult/1 function, 501

- file handling
  - consult files, 501
  - directories, 320–322
  - filenames and extensions, 322
  - metadata, 320–321
  - reading, 114, 330–331
  - writing, 114
- filters
  - higher-order functions, 83–84
  - list comprehension, 23
- finite-state machine, 220–223.
  - See also* gen\_fsm behavior
- firewall (ports required), 466
- floating-point division (/), 10
- flush command
  - implementation, 155–156
  - monitor messages, 169–170
  - shell command, 145
- folds
  - advanced example, 116–117
  - concept, 114
  - example with lists and dicts, 534
  - higher-order functions, 84–85
- food poisoning, 571
- forms, 32, 578
- Fox, Armando, 138
- Fredricksen, Eric, 358
- functional data structures, 73
- functions. *See also* built-in functions;
  - higher-order functions
- calls, 32
  - fully qualified, 191
  - local, 34
- clauses, 44–45
  - errors, 91
- declaring, 32, 34
- exporting, 33
- importing, 35

- funcs (anonymous functions), 78–82

## G

- gb\_sets module, 130–131, 294
- gb\_trees module, 128–129
- gen\_event behavior
  - callback module example, 254
  - code\_change callback, 251
  - concepts, 248–250
  - handle\_call callback example, 261

- handle\_event callback, 250
- handle\_info callback, 251
- init callback, 250
- named event handlers, 255
- remove/swap/add handlers (externally), 255
- remove/swap handler (from callback), 250–251
- request/response (handle\_call), 251
- supervised event handler, 258–259
- synchronous event handling, 251
- gen\_event\_EXIT message. *See* gen\_event behavior: supervised event handler
- gen\_fsm behavior
  - callback module example, 234–245
  - code\_change callback, 225
  - custom state callback, 224
  - handle\_event callback (global asynchronous event), 225
  - handle\_sync\_event callback (global synchronous event), 225
  - init callback, 223
  - request/response example, 240
  - state, 223
  - terminate callback, 225
  - trading system specification example, 225
- gen\_server behavior
  - asynchronous initialization, 291–292, 325–326
  - call and callbacks, 215
  - callback module, 213–217
  - code\_change callback, 213
  - handle\_call callback (synchronous call), 211–212
  - handle\_cast callback, 212
  - handle\_info callback, 212
  - init callback, 210–211
  - terminate, 212–213
  - timeout, 210, 215, 298
  - unexpected messages, 216
- get\_cookie() function, 463
- genproc library, 193

- group leaders
  - cross-node message forwarding, 462
  - I/O protocol, 309
- guard patterns. *See* if ... end expression
- guards, 48. *See also* if ... end expression; case ... end expression
  - and also and otherwise, vs. , and ;, 48–49
  - functions allowed in guards, 49, 59

- GUI, 575
- Gustavsson, Björn, 131

## H

- handling errors. *See* try ... catch ... end expression; supervisors
- handshake (TCP), 378
- hardware errors, 139
- header files, 126
- heartbeats, 467
- help
  - coding practices, 6
  - documentation, 6
  - shell help, 8
- hibernation, of processes, 211
- higher-order functions
  - concepts, 77–80
  - examples, 81–86
  - fold, universal, 86
- higher-order pattern matching. *See* match specifications
- hipe module, 38, 341
- Hitchhiker's Guide to the Galaxy*, 135–136
- homogenous networks, 450
- hot code loading
  - mechanisms, 191–192
  - pitfalls, 353–355

## I

- if ... end expression
  - else, as catchall, 50–52
  - error, 91
  - syntax and style, 49–52
  - vs. case ... end, 54



- image, of a function, 562
- imported functions, 35
- improper lists, 21
- included applications, 333
- inet module
  - active once, 384–386
  - vs. inets application, 384
  - socket options, 385
- inet:setopts function, 385
- installing Erlang, 5–6
- integers. *See* numbers
- IO lists, 375–377
- is\_alive() function, 470
- is\_process\_alive(Pid) function, 411

## J

- jobs management, 9, 155, 464

## K

- kernel (OTP application configuration), 482
- key/value storage, 127–130, 306, 419
- killing a process
  - exit/2 function, 163
  - kill signal, 167

## L

- lambda calculus, 78
- last call optimization (elimination), 70
- last write wins, 457
- latency, 446
- leak, processes or memory, 265, 363, 413
- Learn You A Haskell for Great Good!* (Miran Lipovača), 1, 105
- let it crash, 4
  - crash-only software, 138
  - mechanisms for handling errors, 87–88
- libraries
  - community, 575
  - path, 331
- linear scaling, 4, 140–142
- links
  - cross-node, 446, 460–461
  - definition, 162–164

- to establish dependencies, 179
- unlinking, 162, 164

- Lipovača, Miran, 1, 105
- list comprehensions
  - for database queries, 539–540
  - filters, 23
  - generator expressions, 22
  - set theory origins, 21
  - syntax, 22–23
- list generator (<-), 22
- lists
  - basic operations, 19–20
  - improper, 21
  - modification costs, 367
  - operations, 53, 86, 328
  - order of evaluation, 415
  - pattern matching, 20
  - recursive definition, 20–21
  - strings, 18–19
  - syntax, 18, 20–21
- load balancing, processes by the VM, 140
- logical clocks (lampport, vector clocks), 457
- logic errors, 89–90
- loops, 61

## M

- macros
  - calling, 39
  - defining, 38–39
  - predefined, 39
- mafiaapp (Mnesia example), 513–514
- Magic 8 Ball example, 476–484
- mailbox
  - mechanisms, 151
  - rationale, 139
  - reading messages, 145 (*see also* receive expression)
- make module. *See* compiling code: Emakefiles
- make\_ref() function
  - request/response, 173
  - unique values, 415
- map (higher-order function), 79, 83
- match. *See* pattern matching

- match operator (=)
    - binding/pattern matching, 12
    - for comparison, 11
  - match specifications
    - ets:fun2ms function/parse transform, 430–432
    - fun2ms examples (Mnesia), 531, 533–534
    - structure, 427, 429–430
  - membership test, 53, 103–104
  - memory leak, 265, 363, 413
  - message passing
    - asynchronous messages, 139, 445–446
    - hiding messages, 152–153
    - metaphor, 3
    - receiving messages (*see* receive expression)
    - request/response, 147–148
    - sending messages, 144–145
  - Mnesia, 512
    - activity access context, 523–524
    - CAP approach, 513
    - changing types of tables and schemas, 518
    - database location, 517
    - distributed setup example, 519–521
    - indexes, 517–518
    - local content (not replicated), 537
    - operations, 524–525
    - primary key, 514–515
    - query examples, 526
    - schema, 516–517
    - system information, 535
    - tables
      - creating, 517–518
      - fragmentation, 512
      - limits of, 516
      - replicating with local content, 518
      - structure of, 514
      - types, 516, 518
      - waiting for, 522
    - transactions, 531
  - mnesia:activity/2 function, 523–524
  - mnesia:create\_schema function, 516–517
  - ?MODULE macro, 152
  - modules
    - attributes
      - compile, 38
      - custom, 40–41
      - export, 33
      - export\_type, 565
      - ifdef, -else, -endif, and -ifndef, 39–40
      - vsn, version, 41
    - concurrent versions, 191
    - definition, 31–33
    - interface
      - export attribute, 33
      - hiding messages, 152–153
      - filename, 33
      - metadata (*Module:module\_info/0-1* functions), 40
  - monitor\_node(Node, Bool)
    - function, 460
  - monitors
    - demonitor, 169–170
    - nodes, 460
    - of processes,
      - cross-node, 446, 460–461
      - defined in comparison to links, 168–169, 178
    - stacking, 168
  - monotonic time, 392
  - ms\_transform.hrl include file. *See* match specifications: ets:fun2ms function/parse transform
  - multicore. *See* SMP
  - multiplication (\*), 10
- ## N
- nagger, 296
  - named processes
    - alternative registry (gproc), 193
    - conflict resolution (global), 468
    - distributed, 359–360
    - dynamic names, 174
    - global registry, 467–468
    - OTP processes, 468
    - rationale, explanation, 170–174
  - named tables (ETS), 424
  - name shell argument, 458–459

- net\_kernel module, 467
- netsplits
  - definition, 452
  - zombie survivors example, 454–456
- network topology, 448–449
- network reliability, 445–446
- node() function, 459
- node(PidPortOrRef) function, 461
- nodes, 458
- nodes(Flag) function, 465
- node synchronization (OTP applications), 481–482
- nonlocal returns, 103–104
- not operator, 14
- now() function, 392
- number crunching, 27
- numbers
  - arithmetic, operators for, 10
  - bases, 11
  - floating-point, 10
  - integers, 10
  - in RPN, 109

## O

- Okasaki, Chris, *Purely Functional Data Structures*, 133
- O’Keefe, Richard, 52
- one\_for\_all child restart strategy, 267
- one\_for\_one child restart strategy, 266–267
  - vs. simple\_one\_for\_one, 268
- onion-layered system, 283–284
- Open Telecom Platform (OTP), 199–200, 209–210. *See also individual behaviors; application (OTP)*
- operators
  - arithmetic, 10
  - bitwise, 26–27
  - Boolean, 14
  - comparison, 14–16
  - message passing, 144–145
- orddicts module, 127–128
- ordsets module, 130–131
- orElse operator, 14, 48–49

- or operator, 14
- output, printing with io:format/2-3, 45, 394

## P

- packages, applications
  - (.ez files), 345
- packaging (releases), 340
- packets, TCP/IP and UDP, 377
- parametrized types. *See Dialyzer: polymorphic types*
- parse transform, 430
- partition tolerance, 454. *See also CAP theorem*
- pa shell argument, 194–195
- pattern matching
  - binary tree example, 73–74
  - case ... end, 53
  - exceptions, 96–98
  - in functions, 43–44
  - greeting example, 44
  - lists, 20
  - lists in functions, 45–46
  - match operator (=), 12
  - records, 124
  - tuples, 17–18
  - variable bindings, 46–47
- persistent data structures, 73
- persistent lookup table (PLT), 543–544
- pid (process identifier)
  - definition, 143
  - global ordering, 242
  - internal representation, 461
  - pid/3 shell command, 153
  - self() function, 144
- PLT (persistent lookup table), 543–544
- polymorphic types, 567–570
- port. *See also UDP*
  - controlling process, 383–384
  - definitions, 379
  - inet module, 384
- portability, 341
- postfix notation (RPN), 106
- ppool application, 282–283, 325–326

- predicate, 84
- prefix notation, 106
- primary key, 422, 424, 514, 529
- printing with `io:format/2-3`, 45, 394
- priority queue, 159
- private tables, 423
- process dependencies, 288–291
- processes
  - as actors, 3
  - memory space, 140
  - rationale, 136, 137–138
  - storing state, 150–151
  - usage in parallelism, 4
- `process_flag` (`trap_exit`) function, 164–165
- process identifier. *See* `pid`
- process leak, 265, 363, 413
- process loop (storing state), 150–151
- Process Quest, 358–360
  - files to download, 353
  - gameplay, 361–363
  - socket server (`sockserv_serv`), 391
- process skeleton, 200
- `proc_lib` module, 412
- profiling, 574
- Progress Quest (RPG), 357–358
- `proplists` module, 127
- protected expression, 96, 100
- protected tables, 423
- protocol design
  - and event management, 253–254
  - finite-state machines, 225
  - between processes, 178–180
  - questions to ask about, 233
- public tables, 423
- Purely Functional Data Structures*  
(Chris Okasaki), 133

## Q

- query handle, 539
- query list comprehensions (QLC), 539–540
- queues, 132–133, 293–295
- quorum, 457

## R

- race conditions
  - behavior termination calls vs. clean-up, 437–438
  - message collision (asynchronous), 229, 231–232
  - provoking (with `Common Test`), 498–500
  - shared state, 172–173
  - solving by picking a master, 241–242
  - supervisor and children, 292
- random
  - bytes, 392
  - pick from a list or tuple, 479
  - seeding, 392
- range, 32. *See also* `image`
- rebar, 306, 352
- receive expression
  - after clause, 154
  - catchall, 146
  - defensive, 159
  - priority, 156
  - selective, 156–158
  - syntax, 145–146
- `record_info`(`Record`) function, 517
- records
  - `code_change` incompatibilities, 363–365
  - declaration in a module, 122
  - field position in, 124
  - hiding (in `Mnesia`), 527
  - nested, 124
  - pattern matching warts, 182
  - reading values, 123–124
  - sharing definitions, 126
  - in the shell, 122–123
  - syntax, 122–126
  - updating, 125–126
- recursion
  - base case, stopping condition, 63
  - duplicate example, 66
  - list length example, 63
  - list zipping example, 69–70
  - mathematical definition (factorial), 62

- recursion, *continued*
  - reverse example, 66–68
  - sorting example, 70–71
  - sublist example, 68–69
  - tail recursion, 64–65
  - thinking recursively, 75–76
  - tree example, 72–75
  - while vs. tail recursion, 67
- reduce. *See* folds
- referential transparency, 2–3
- registered() function, 171
- register(Name, Pid) function, 171
- registers. *See* named processes
- regular expressions, 318, 325, 329–330
- releases
  - OTP release, 336
  - packaging, 369
  - profiles, 346, 351
  - starting, with a boot file, 340–341, 344–345
- RELEASES file, 369, 372. *See also* reltool; systools
- release updates (relups), 357
  - essential applications and configuration, 361
  - generating relup files, 368
  - manual work involved, 367–368
  - RELEASES file, 372
  - reltool vs. systools, 368
  - rolling back, 372
  - step-by-step list, 372–373
  - upgrading, 370
  - upgrading sockets, 370–371 (*see also* sys module)
- reltool
  - app file handling, 347–348
  - boot\_rel option, 343, 346, 349
  - building a release, 343–344
  - configuration file example, 341, 343
  - directory structure (output), 345
  - ERTS version, 344
  - filters, 347
  - lib\_dir option (library directories), 343
  - options, 342
    - application-wide, 347
    - module-specific, 348
    - release-wide, 346
  - recipes, 348–351
  - relocatable, self-contained release, 346
  - removing .ez files, 361
  - specifying application version, 345
  - target spec, 343–344
- reminder (concurrent application example), 176
- rem operator, 10
- remote procedure calls (RPC), 469. *See also* rpc module
- remote shell, 464
- REPL. *See* shell
- request/response
  - gen\_server behavior, 211–212
  - make\_ref(), 158
  - monitor example, 185
  - optimization, 158
- reserved words, 14
- rest\_for\_one child restart strategy, 267
- Reverse Polish notation (RPN), 106
- role-playing game (RPG). *See* Process Quest
- root directory (ROOT\_DIR variable), 344
- Rotem-Gal-Oz, Arnon, 445
- RPC (remote procedure calls), 469. *See also* rpc module
- rpc module
  - asynchronous call, 469–470
  - cast, 470
  - Mnesia example, 521
  - multicall, 470
  - multicast (eval\_everywhere function), 470
  - synchronous call, 469
- RPN (Reverse Polish notation), 106
- running code
  - escript, 120
  - outside of the shell, 118–119
  - self-executable code, 344 (*see also* releases)

run queue, 140  
run\_test executable, 489, 503–504  
runtime errors, 90–93

## S

SASL, 355, 361  
saving state after crash, 283  
scalability, 137–138  
scheduler, 140  
schema, 516–518  
security, 447–448  
selective receive, 156–158  
self() function, 144  
send(Dest, Msg, Opts) function, 465  
sequential bottlenecks  
    optimizing, 420  
    parallelism definition, 141–142  
serialization, 58  
set\_cookie(Node, Cookie)  
    function, 463  
sets, 130–131, 422  
shadowing, 82  
sharding, 421  
shared-nothing practice, 138  
shell, 7–8  
    arguments and flags  
        -AppName, 332  
        -boot, 340–341  
        -boot\_var, 346  
        -config, 482, 484  
        -env, 331  
        escaping, 332  
        evaluating at boot time, 484  
        -make, 194, 331  
        -man, 6  
        -name, 458–459  
        -noshell, 344  
        -pa, 194–195  
        -smp, 142, 144  
        -sname, 458–459  
    commands  
        autoimported functions, 32  
        c(), 36, 191  
        cd(), 36  
        exiting the shell, 8  
        f/0-1, 12  
        flush(), 145

1(), 191  
    for navigating, 8  
    pid/3, 153  
    records, 122–123  
    regs(), 171  
jobs management, 9, 155, 464  
short-circuit operators, 14, 48–49  
syntax, 9–10  
shutdown  
    init:stop function, 336  
    orderly shutdown, 265  
    OTP termination reason, 212  
    VM shutdown, 313  
        through releases, 336  
sibling dependencies, 288–291  
signal, 95, 163–165  
simple\_one\_for\_one child restart  
    strategy, 268, 280  
single assignment, 2, 11  
sleeping, 143, 155  
SMP (symmetric multiprocessing),  
    137, 142, 144  
-sname shell argument, 458–459  
sofs module, 130–131  
spawn/1-3, spawn\_link/1-3 functions,  
    142–143, 146, 164  
spawn/2-4, spawn\_link/2-4  
    functions, 462  
spawning processes  
    remotely, 461  
    safely with proc\_lib module, 412  
    spawn/1, spawn\_link/1 functions,  
        142–143, 164  
    spawn/2, spawn\_link/2  
        functions, 462  
    spawn/3, spawn\_link/3 functions,  
        146, 164  
    spawn/4, spawn\_link/4  
        functions, 462  
.spec file, 501–502, 503–504  
specific vs. generic code,  
    separating, 209  
stack, 108, 560–561  
stack traces, 101  
start\_link crash, shell error, 299  
static types, vs. dynamic types, 56

- strings
  - as binaries, 27–28
  - as lists, 18–19
  - to a number, 109
- strong types, vs. weak types, 57
- subtraction (-), 10
- success typing, 545–547
- supervision tree, 283–284, 285–286
- supervisor (OTP), 264–265. *See also individual child restart strategies*
  - asynchronous initialization, 392
  - child specifications, 268–271
  - definition, 266
  - dynamic supervision, 277
    - operations on children, 278
    - with `simple_one_for_one` child restart strategy, 279
  - init/1 callback, 266
  - killing/terminating supervisors, 297
  - restart, 269
  - restart limit, max time, 268
  - shutdown time, 270, 288
  - worker handling supervisor termination, 272–274
  - worker/supervisor distinction, 270
- supervisors
  - basic, 196–197
  - very basic, 171, 173
- symmetric multiprocessing (SMP), 137, 142, 144
- syntax
  - arithmetic precedence rules, 10
  - control flow (*see individual control flow expressions*)
  - functions, 44–45
  - tokens, 578
  - type specifications, 557, 559
  - ugly, 577, 579–580
- sys module
  - code updates, 356
  - OTP worker status, 417
- system limits
  - atom table, 13
  - DETS table size, 433

- ETS max tables, 421
- general limits, 93
- timer size, 183
- system processes, 164
- systools
  - boot file, 339–340
  - options, 340
  - packaging release, 340
  - `systools.rel` file, 338

## T

- takeover, and failover, 475–476
  - boot procedure, 484
  - configuration, 481–482
  - examples, 483–484
    - client operations, 381
    - close socket, 383
    - connect, 383
    - connection-based protocol, 377
    - connection closed by client, 395
    - keepalive, 382
    - listen to connections, 382
    - server, parallel, 389–390
    - server mode operations, 382
    - simple (sequential), 387–388
    - TCP, accept connection, 382–383
- TDD, 408
- terminations, complex, 333
- tests
  - ad hoc, 110
  - black box and white box, 485–486
  - Common Test (*see Common Test*)
  - EUnit (*see EUnit*)
  - integrating EUnit and Common Test, 508
  - synchronization, 413–414
  - system processes testing, 417–418
  - system testing, 485
- throws
  - nonlocal returns, 103–104
  - type of exception, 95–96
- time conversions, 182
- time handling, 186, 188–189

- timeout, 154, 210, 215, 298
- time travel, 491
- tokenizing, 108, 114
- top-level supervisor, 310
- tracing, 574
- tracking workers (termination), 324
- transaction, 512
- transport cost, 449–450
- trap\_exit option, in process\_flag function, 164–165
- true Boolean value, 14
- try ... catch ... end expression
  - after (finally), 99
  - catch-all clause, 98
  - example, 99–100
  - handling exceptions, 96–98
  - protected expressions, 96, 100
  - syntax, 96
- tuples
  - pattern matching, 17–18
  - storing tuples, 421–422
  - syntax, 16
  - tagged tuples, 17–18
- two-phase commit, 232–233, 241–243
- type alias, 553
- type inference. *See* success typing
- types. *See also* Dialyzer
  - comparisons, sorting order of types, 16
  - conversion (casting), 57–58
  - detection, 59
  - dynamic vs. static, 56
  - strong vs. weak, 57
- type signature. *See* Dialyzer:
  - function specifications

## U

- UDP
  - close socket, 380
  - connectionless protocol, 377
  - IPv4 and IPv6, 379–380
  - open socket, 379–380
  - recv (receive), 381
  - send, 380
  - socket, 378
  - socket operations, 379

- undefined error, 91–92
- unique values, 415
- University of Catania, Unict team
  - and IANO robot, 5
- unlink, 162, 164
- unregister, 171
- unregister(Name) function, 171
- user switch command. *See* jobs management

## V

- variables
  - don't care (`_`), 17
  - scope with closures, 81–82
  - single assignment, 11
  - syntax, 11–12
- version scheme, 363

## W

- waiting. *See* sleeping
- weak types, vs. strong types, 57
- Wheeler, David, 232
- whereis/1 function, 172
- worker, 270
- wx application, 575

## X

- xref module, 318

## Z

- zoo example, 568