

Index

- @ (“at-sign”), 422
 - & (ampersand), 277
 - &= (ampersand equal sign), 279
 - * (asterisk), 35, 680–681
 - \ (backslash), 62
 - ^ (caret), 278, 678, 680
 - \$ (dollar sign), 678, 679
 - . (dot), 678, 689–690
 - ** (double asterisk), 35, 133, 140
 - `` (double single quotes), 104
 - (hyphen), 680
 - (minus sign), 278
 - . (period), 678, 689–690
 - |= (pipe equals), 277
 - | (pipe symbol), 277, 678, 689
 - + (plus sign), 35, 159, 176–177, 213–214, 233, 681
 - # (pound sign), 34, 62
 - / (slash), 35, 130–131, 569
 - { } (braces), 681
 - [] (brackets), 259, 679–680, 690
 - > , >= (greater than), 277
 - < , <= (less than), 277
 - <> , != (“not equals” comparison operators), 36
- A**
- \A (special character), 678
 - abs () built-in function, 143
 - absolute import** statement, 494–495
 - abstract methods, 519
 - abstraction, 382, 516
 - access models, 115–116
 - access modes, 326, 327
 - ActiveX, 989. *See also* COM (Component Object Model)
 - adapters (database)
 - about, 932–933
 - definition of, 923
 - examples of, 934–945
 - Gadfly, 939–945

- adapters (*continued*)
 - MySQL, 934–936, 939–945
 - ORMs, 946
 - PostgreSQL, 936–937
 - related modules, 958–959
 - SQLite, 937–945
- add() method, 281, 282, 284
- addition (+) operator, 35
- AF_INET (socket family name), 716, 718, 730
- AF_UNIX (socket family name), 715, 718, 730
- aggregation, 517
- all() built-in function, 310
- alternation (|), 678
- American Standard Code for Information Interchange. *See* ASCII
- ancestors, 545, 546
- and keyword, 36, 100, 101
- anonymous functions, 439–441
- any() built-in function, 310
- AOP (aspect-oriented programming), 424
- API (Application Programming Interface), 489, 922–923, 985–989
- apilevel, 924–925
- Application Programming Interface. *See* API
- apply() built-in function, 414, 441, 442
- arguments
 - class methods, 543
 - command-line, 338–339
 - for exceptions, 372–375
- arguments (default)
 - functions, 49, 413, 429–442
 - GUI development, 824
 - Tkinter widgets, 824
 - using with instantiation, 532–533
- arguments (function)
 - decorators with/without, 423–424
 - default, 49, 413, 429–442
 - dictionary, keyword variable, 434–436
 - formal, 428–432
 - grouped, 413–414
 - keyword, 412–413, 434–436
 - optional, 48
 - positional, 428–429
 - variable argument objects, 436–439
 - variable-length, 433–439
- arguments (method), 532–533
- arithmetic game (example), 415–417
- arithmetic operators. *See* mathematical operators
- arrays. *See* lists; tuples
- ASCII, 144–145, 197–198, 497, 698
- aspect-oriented programming (AOP), 424
- assert** statement, 389–390
- AssertionError exceptions, 390, 391
- assigning/assignment
 - augmented, 37, 65–66
 - dictionaries, 255
 - lists, 209
 - multiple, 66
 - “multiple,” 66–67
 - numbers, 121
 - operators, 64–67
 - set types, 273, 274
 - strings, 168–169
 - tuples, 231
 - variables, 37, 64–67
- association (classes), 517
- asterisk operator (*), 680–681
- async* module, 741
- atomic storage, 112
- “at-sign” (@), 422
- attr() built-in functions, 560–561
- attribute(s)
 - built-in functions, 629
 - built-in methods, 632
 - built-in types, 536–537
 - class, 520–525
 - complex number, built-in, 126–127
 - definition of, 46
 - double underscore (__), 586
 - file, 336–337
 - functions, 420–421, 629, 630
 - importing, 496
 - interfaces to data, 516
 - __metaclass__, 610–611
 - methods, 632, 633
 - module, 333
 - multi-platform development, 333
 - naming, 478, 513
 - object, 90
 - privacy, 585–586
 - Queue module, 811
 - simple lookup example, 554–555
 - socket module, 730–731
 - special class, 523–525
 - special methods for customizing classes, 566
 - user-defined function, 630

- user-defined methods, 633
- using files to store, 604
- using local to substitute for module, 81

`AttributeError` exception, 363–364, 391, 590, 600

augmented assignment, 37, 65–66, 569, 574–575

authentication handler for HTTP, 867–869

auto-loaded modules, 496

B

`\b`, `\B` (special characters), 679

backbone (network), 858

backslash (`\`), 62

bank tellers server example, 713–714

base class, 50. *See also* parent/parent class

base representation, 143–144

`BaseException`, 371, 372, 391, 394

`BaseHTTPServer` module, 907–909, 911

bases argument, 504

`__bases__` attribute, 524, 525, 548

BASIC language, 10

Beazley, David, 981

BIFs. *See* functions (built-in)

BIMs. *See* methods (built-in)

binary function, 447–448

binding

- methods, 540–541
- namespaces, 480

bit operators (integer-only), 135–136

Boa Constructor, 849

boilerplate, 968–974

`bool()` factory function, 137, 138

Boolean operators (and, or, not), 36, 100–101, 292

Boolean types, 93, 123, 145–147

bound methods, 522, 541

brace operators (`{ }`), 681

bracket symbols (`[]`), 679–680, 690

break statement, 304–305, 308

BSD Unix, 12, 14, 16–17, 715

buffering (file), 327, 328

building Python, 13

`__builtin__`, 480–481, 496

built-in attributes, 126–127, 336–337

built-in exceptions, 391–393, 1040–1042

built-in functions. *See* functions (built-in)

built-in methods (BIMs). *See* methods (built-in)

“built-in” names, 69

built-in types, 91, 536–537

`__builtins__`, 69, 480–481, 496

Button widget (Tk), 825–831

byte type, 116

C

C language

- conversion to/from, 969, 970
- “dangling `else`” statements, 293
- extensions written in, 8
- `fopen()`, 326–327
- Python and, 6, 8–9, 23, 26
- `varargs`, 433

C# language, 27, 969. *See also* IronPython

C++ language, 8, 23, 25, 26, 969, 970

`__call__()` special method, 634–635

“call by reference,” 48

`callable()` built-in function, 636, 637, 831

callable class, 803–805

callbacks, 823, 824

calling functions, 49, 412–417

- arguments, 412–414, 436–439
- built-in, 427
- default arguments, 413
- example, 415–417
- function operator, 412
- grouped arguments, 413–414
- keyword arguments, 412–413
- logging with closures, 461–463
- with variable argument objects, 436–439

calling modules, 52–53

Canvas widget (Tk), 825

caret symbol (`^`), 678

case statement, proxy for, 294–295

case-insensitive import, 496–497

case-sensitive identifiers, 68

casting of sequences, 165–166

CGI (Common Gateway Interface)

- about, 875–877
- advanced example of, 896–906
- applications, 877–892
- building applications, 878–892
- `cgi` module, 878
- cookies, 895–897, 900–901, 906
- creating static form Web page, 879–881
- file uploading, 894
- generating form page, 882–886
- generating results page, 881–886
- HTTP headers, 882
- multipart form submission, 894

- CGI (*continued*)
 - multivalued fields, 895
 - setting up a Web server, 878–879
 - Unicode, 892–893
 - user inputs/error processing, 886–892
 - Web servers, 878–879, 906–909
- cgi module, 878
- CGIHTTPServer module, 907, 908, 911
- char type (unsupported), 116, 207
- character(s)
 - accessing, in strings, 169
 - matching any single (.), 678, 689–690
 - removing, 169–170
- character classes ([]), 679–680, 690
- characters, special, 208
 - and ASCII characters, 698
 - escaping using triple quotes, 38, 192–193
 - regular expressions, 676–677, 682, 690–693
 - repetition/grouping and, 690–693
 - representing character sets, 682
 - strings, 192–193
- Checkbutton widget (Tk), 825
- child/child class, 545–548, 554, 555, 586, 823
- chr () built-in function, 144, 145, 187
- class attributes, 520–525
 - accessing, 537–539
 - __bases__, 524, 525, 548
 - __class__, 524, 525
 - data, 520–521
 - determining, 522–523
 - __dict__, 522–525
 - __doc__, 524, 525
 - instance attributes vs., 537–540
 - methods, 521–522
 - modifying, 540
 - __module__ attribute, 524, 525
 - persistence, 539–540
 - privacy of, 586
 - __slots__, 597
 - special, 523–525
- class definition, 510–511
- class keyword, 50, 504
- class variables, 521
- classes, 50–52
 - about, 504–507, 518–520
 - built-in functions, 558–564
 - callable objects, 634
 - composition, 544–545
 - creating, 51–52, 510–511, 519
 - customizing with special methods, 564–585
 - declaring, 50–51, 72, 518–520
 - definition vs. declaration of, 519–520
 - methods, 507–512, 521–522
 - mix-in, 556
 - as namespace containers, 506
 - naming, 513
 - related modules, 615–617
 - Web server, 909
 - wrapping, 587
- classic classes, 503, 504, 542–543, 554–558.
 - See also* new-style classes
- classic division operator (/), 35, 130–131, 569
- classmethod () built-in function, 543
- clause, 63
- clear () method, 281, 282, 284
- client data, 875–878
- clients
 - FTP, 752–754
 - GUI applications as, 821
 - Internet, 747–748
 - NNTP, 760–765
 - POP3, 775–777
 - SMTP, 775–777
 - TCP, 723–726, 736, 738–740
 - twisted reactor TCP, 738–740
 - UDP, 728–729
 - Web, 855–857, 859–875
 - Windows, 713
- client/server architecture, 711–715, 821, 855–856
- client-side programming, 989–991
- close () built-in method, 332, 335
- closures, 422, 456–463, 680–681
- cmp () built-in function, 102, 103, 136, 137, 184–185, 215–216, 260–262
- code
 - commenting, 34–35
 - indenting, 41
 - integration of, 963
 - interpreted/byte-compiled, 11
 - profiling of, 8, 84–85, 965
 - running example, 22
 - runtime generation/execution of, 642–649
 - skipping, 365
 - wrapping, in boilerplate, 968–974
- code objects, 94, 635–636
- code reuse, 7, 964
- codecs, 199, 205

- `coerce()` built-in function, 139, 143
 - collisions (key), 268–269
 - colocated servers, 857, 858
 - columns (database), 920
 - COM (Component Object Model), 989–991
 - command line
 - arguments, 338–339, 490
 - FTP client program, 755
 - options, 15
 - running Python from, 14–17
 - switches, 653
 - commands (database), 920
 - comments, 34–35, 62, 69–70
 - Common Gateway Interface. *See* CGI
 - `compile()` function, 94, 636, 637–638, 684–686
 - compiling, 11, 685–686, 964, 965, 974–975
 - `complex()` factory function, 137, 138
 - complex numbers, 37, 38, 126–127
 - complex statements, 62–63
 - Component Object Model. *See* COM
 - composite objects, 112
 - composition, 516–517, 544–545
 - compound objects, 112
 - compound statements, 62–63
 - concatenation (+) sequence operator, 159, 176–177, 213–214, 233
 - conditional code execution, 647–649
 - conditional statements, 291–296. *See also* **while** statement (loops). *See also* loops
 - auxiliary statements for, 308
 - conditional expressions, 295–296
 - elif** (aka **else-if**) statement, 294–295, 308
 - else** statement, 292–294, 307–308, 378
 - if statement, 41–42
 - if** statement, 291–292
 - multiple, 292
 - and **pass** statement, 306–307
 - single statements suites, 292
 - switch/case statement proxy, 294–295
 - `connect()` function, 925–927
 - connection objects, 927–928
 - connectionless sockets, 717–718
 - connection-oriented sockets, 716–717
 - constructors, 510, 511, 527–529, 532–533, 930–931
 - container storage, 112
 - context expression (context_expr), 384
 - context management (for exceptions), 382–385
 - continuation (exception handling), 361
 - continuation (\) statements, 62
 - continue** statement, 305–306, 308
 - conversion
 - ASCII functions for, 144–145
 - codes, 969, 970
 - factory functions, 137–138
 - sequences, 165–166
 - symbols, 178–181, 1029
 - cookies, 856, 895–897, 900–901, 906
 - `copy()` function/method, 240–241, 267, 283
 - counting loops, 297
 - couroutines, 467–468
 - `cPickle`, 349, 350
 - `cProfile` module, 85
 - `CPython`, 26, 27
 - credit card transaction data example, 375–378, 380–381
 - cross-product generator expressions example, 317
 - currying, 450
 - cursor (database), 920
 - cursor objects, 929–930
 - customizing and extensions, 964
 - CWI (Centrum voor Wiskunde en Informatica), 6
 - cycles (import), 497–500
 - cyclic reference, 79
- ## D
- daemon threads, 801
 - “dangling else”, avoiding, 63, 293–294
 - data attributes. *See* attribute(s)
 - data descriptors, 599
 - data hiding, 516
 - data structures, 223–230
 - database programming, 919–959. *See also* DB-API (Python Database Application Programmer’s Interface)
 - about, 920–922
 - adapter examples, 934–945
 - components of databases, 920
 - Gadfly, 923, 939–945
 - MySQL, 920, 934–936, 939–945
 - operations/SQL, 920–922
 - ORMs, 947–957, 959
 - persistent storage, 919
 - PostgreSQL adapter example, 936–937
 - Python interfaces for relational databases, 931–932
 - related modules, 958–959
 - SQL, 920–922, 946

- database programming (*continued*)
 - SQLAlchemy, 946, 947–950, 953–955
 - SQLite, 937–945
 - SQLObject, 946, 951–953, 955–957
 - storage mechanisms, 919
 - supported, 932
 - Unicode, 202
- database servers, 713
- datagrams, 717, 727
- DB-API (Python Database Application Programmer's Interface), 924–945
 - about, 922–923
 - apilevel, 924–925
 - changes in versions of, 931
 - connect(), 925–927
 - connection objects, 927–928
 - cursor objects, 929–930
 - exceptions, 927
 - function attributes, 925–927
 - layers in, 923
 - module attributes, 924–926
 - paramstyle, 925, 926
 - thread safety, 925
 - type objects and constructors, 930–931
- DBM-style modules, 349
- debugging, 10, 84, 181, 437–439
- decimal floating point numbers, 147–148
- decode() built-in method, 200–201, 204
- decorators, 422–426, 460–463, 544
- deep copy, 240–241
- def keyword (statement), 48, 418, 421
- default arguments. *See* arguments (default)
- __del__() method, 528–530, 565
- del statement, 78–79, 122, 210, 258, 276
- delattr() built-in function, 560, 561
- delegation, 587–595
- __delete__() special method, 598, 599
- derivation, 517, 545–546
- descriptors, 598–610
- destructor (class), 528–530
- developer tools, 84–85
- diamond shape inheritance hierarchy, 556–558
- __dict__ attribute
 - built-in types, 537
 - class attributes, 522–525
 - instance attributes, 534–536
 - modifying, 535–536
- __slots__ attribute vs., 597
- vars(), 563
- dict() factory function, 263–265
- dictionaries, 40–41
 - accessing values in, 255–257
 - assigning, 255
 - built-in functions, 264, 265, 267
 - built-in methods, 254, 255, 265–268, 1034, 1035
 - cmp(), 260–262
 - comparing, 260–262
 - copy(), 267
 - creating, 255, 263
 - dict(), 263–265
 - exact match of, 262
 - fromkeys(), 266, 268
 - functions for, 263–265
 - hash(), 264, 265
 - and hash tables, 254
 - items(), 265, 266
 - iteration, 311–312
 - key-lookup operator ([]), 259
 - keys, 237–238, 265–269
 - keyword variable arguments, 434–436
 - len(), 264, 265
 - login/password programming example, 269–272
 - as mapping type, 253–258
 - membership operator, 259
 - operators for, 258–259
 - as Python feature, 7
 - related functions, 263–268
 - removing elements of/dictionaries, 258
 - setdefault(), 266–268
 - sorted(), 267
 - str(), 260
 - type(), 260
 - updating, 257–258
 - values(), 265, 266
- difference (-) operator, 278
- difference update (-=) operator, 279
- dir() built-in function
 - built-in types, 536
 - class attributes, 522, 523
 - classes, 54, 561
 - instance attributes, 534
 - instances, 561
 - lists, 220
 - local variables, 562

- modules, 561
 - objects, 561–562
 - direct access model type, 115–116
 - directory structure, 493–494
 - `discard()` method, 281, 282, 284
 - disk files, 315–317
 - dispatch, static vs. dynamic, 992
 - `distutils` package, 974
 - division operators, 35, 130–133, 569
 - `divmod()` built-in function, 140, 143
 - `__doc__` attribute, 524, 525, 629, 630, 632, 633
 - document “doc” strings, 34–35
 - documentation
 - classes, 617
 - extended import, 486
 - extensions to Python, 982
 - file-access related modules, 353
 - FTP, 756
 - generator expressions, 319, 471
 - GUI programming, 851
 - list comprehensions, 314
 - method resolution order, 558
 - module, 71, 72
 - NNTP, 765–766
 - OOP, 617
 - Python, 22–23
 - set types, 284
 - SMTP, 771
 - style guidelines for, 70
 - dollar sign symbol (`$`), 678, 679
 - DOS window, 14–16, 22
 - dot symbol (`.`), 678, 689–690
 - double precision floating point numbers, 125
 - double quotation mark (`"`), 207
 - double type, 117
 - double underscore (`__`) attributes, 586
 - downloading
 - IMAP, 772
 - POP3, 773–777
 - protocols for, 772–777
 - Python, 11, 13, 18–19
 - SMTP, 773–777
 - Yahoo! stock quote data, 986
 - dropping (dropped) databases, 920, 921
 - `dumbdbm` module, 349
 - dummy functions, 974
 - `dump()` function, 350
 - dynamic dispatch, 992
 - dynamic typing, 76
- E**
- EasyGUI, 849
 - electronic mail. *See* e-mail
 - elif** (aka **else-if**) statement, 294–295, 308
 - ellipsis objects, 95
 - else statement
 - exceptions, 378
 - try-except** statement, 378
 - else** statement, 292–294. *See also* **if** statement
 - “dangling else”, 293–294
 - for** loops, 307–308
 - usage of, loops/conditionals, 308
 - e-mail (electronic mail), 766–777
 - components/protocols, 766–767
 - IMAP, 772
 - POP3, 772–777
 - receiving, 771–772
 - related modules, 778
 - sending, 767–768
 - SMTP, 768–771, 775–777
 - embedding, extensions vs., 982
 - encapsulation, 516, 585–586
 - `encode()` built-in methods, 200–201, 204
 - encoding, 205, 497
 - `__enter__()` method, 385
 - Entry widget (Tk), 825
 - `enumerate()` built-in function
 - iterators, 310
 - lists, 218
 - for loops, 45, 300–301, 303, 304
 - sequences, 167
 - strings, 185–186
 - equal sign (`=`), 64–65
 - equality for sets, 276
 - `eric3`, 850
 - errors/error processing, 47–48
 - about, 359–360
 - “cleaner” approach, 663
 - DB-API, 927
 - definition of, 360
 - hiding, 372
 - runtime, 361
 - “safe and sane,” 10
 - standard error, 337
 - try-except** statement, 372
 - using CGI for, 886–892
 - `eval()` built-in function, 636, 638–639
 - events, 823

- Excel, 991–993, 1000–1002
- except** statement, 368–370
- Exception (root class), 370, 371, 373, 391, 394, 489
- exception condition, 360
- exceptions/exception handling. *See also specific headings, e.g., TryError exception*
 - about, 10, 360–361
 - arguments for, 372–375
 - assert** statement, 389–390
 - built-in, 391–393, 1040–1042
 - catching all, 370–372
 - context management for, 382–385
 - creating, 393–401
 - DB-API, 927
 - detecting/handling, 47–48, 364–382, 401–403
 - else clause, 378
 - examples of, 375–378, 395–401
 - except** statement with multiple, 369–370
 - finally clause, 378–379
 - and `os.path.exists()`, 81–82, 84
 - in Python, 361–364
 - raise** statement, 48, 386–389
 - related modules, 404
 - robustness of, 10
 - skipping code, 365
 - standard, 391–394
 - with** statement, 382–384
 - strings, 386
 - sys module, 403–404
 - try** statement with multiple, 368–369
 - try-except** statement, 364–365, 372, 378
 - try-except-else-finally** statement, 381–382
 - try-finally** statement, 379–381
 - unhandled, 365
 - Unicode, 204
 - upward propagation, 365
 - warnings, 489
 - Web servers, 401–402
 - and wrapping a built-in function, 365–368, 375–378
- `exec()` built-in function, 640–641
- exec** statement, 636
- `execfile()` built-in function, 651–652
- executable object** statement
 - built-in functions, 636–642
 - `callable()`, 637
 - `compile()`, 637–638
 - conditional code execution, 647–649
 - `eval()`, 638–639
 - `exec()`, 640–641
 - generating/executing code at runtime, 642–649
 - `input()`, 641–642
- executing/execution
 - callable objects, 628–635
 - code at runtime, 642–649
 - code conditionally, 647–649
 - code objects, 635–636
 - current process/user related functions, 666–667
 - file, 348, 662
 - imported modules, 73–74, 486, 487, 650–651
 - non-Python programs, 653–654
 - operating system interface functions, 666–667
 - other Python programs, 649–653
 - related modules, 668
 - restricted, 663
 - TCP server/clients, 725–726, 736, 740
 - terminating, 663–666
 - UDP server/clients, 729–730
 - `__exit__()` method, 385
- exiting, 791
- exponential notation output, 180
- exponentiation operator (`**`), 35, 133, 140
- `extend()` method, 214, 222
- extended import** statement (as), 485–486
- extended slicing, 162–164
- extensions to Python, 963–982
 - about, 8, 963–964
 - compilation of, 964, 974–975
 - creating application code, 966–968
 - documentation, 982
 - embedding vs., 982
 - Global Interpreter Lock, 980
 - importing, 976
 - main steps of, 965
 - multithreaded, 980
 - Psyco, 981–982
 - Pyrex, 981
 - reasons for creating, 964–965
 - reference counting, 977, 979–980
 - SWIG, 981
 - testing of, 976–979
 - Win32, 989, 990
 - wrapping code, 968–975
 - writing, 965–980

F

- factorial functions, 807–810
- factory functions, 136–145
 - built-in functions conversion to, 595–596
 - conversion, 137–138
 - definition of, 111
 - dict(), 263–265
 - list(), 218–219
 - numeric types, 1023–1025
 - sequence type operators, 1025–1028
 - set types, 280, 283–284, 1036–1037
 - standard type, 1022–1023
 - str(), 186–187
 - super(), 562–563
 - tuple(), 218–219
 - type, 111
 - type() as, 102
- fetching rows, 920
- Fibonacci functions, 807–810
- FIFO (first-in-first-out) data structure, 227
- file() built-in function, 47, 326, 328
- file extensions, 11
- file objects, 325–326
 - access modes for, 327
 - built-in attributes, 336–337, 1038–1039
 - built-in functions, 326–329
 - built-in methods, 329–336
 - close(), 332
 - file(), 47, 326, 328
 - fileinput(), 351, 352
 - fileno(), 332
 - flush(), 332
 - fnmatch(), 351, 352
 - glob(), 351, 352
 - input, 329–336
 - intra-file motion, 331
 - isatty(), 332
 - iterating through, 312, 315, 331–332
 - line separators, 330, 333
 - methods, 335–336, 1038–1039
 - modules, 351–353
 - multi-platform development, 333
 - open(), 46–47, 326–328
 - output, 330
 - read(), 329
 - readline(), 329
 - readlines(), 329
 - related modules, 351–353
 - seek(), 331
 - standard, 337–338
 - truncate(), 332
 - Universal NEWLINE Support, 328–329
 - using to store attributes, 604
 - wrapping a, 594–595
 - write(), 330
 - writelines(), 330
- file system, accessing, 339–347
- File Transfer Protocol. *See* FTP
- fileinput() module, 351, 352
- “file-like” objects, 325–326, 353
- files. *See also* persistent storage
 - command-line arguments, 338–339
 - example code for reading, 332–335
 - execution of, 348
 - as storage mechanism, 919
 - text file manipulation example programs, 79–85
 - transferring, 748–755, 759–766
 - uploading, 894
 - using to store attributes, 604
- filter (warnings), 489–490
- filter() built-in function, 313, 314, 441–445
- finally clause, 378–379
- findall() function, 684, 694
- first-in-first-out (FIFO) data structure, 227
- flattening, 348
- float() built-in function, 366–368, 374–375
- float() factory function, 137, 138
- float type, 117
- floating point numbers, 37, 38, 125, 147–148, 180
- floor() built-in function, 141, 142
- floor division (//), 35, 130–132, 569
- flush() built-in method, 332, 335
- fnmatch() module, 351, 352
- folding, 448
- for statement (loops), 43–45, 298–304
 - break** statement, 304–305, 308
 - continue** statement, 305–306
 - else** statement for, 307–308
 - file iteration, 331–332
 - with iterator types, 301
 - and **pass** statement, 306–307
 - range(), 301–303
 - with sequence types, 299–301
 - sequence-related built-in functions for, 303–304
 - syntax for, 298–299
 - xrange(), 303

- forking processes, 789
- form Web page, 879–881, 883–886
- formal arguments (functions), 428–432
- format operator, 178–181, 205, 208, 1029
- forms, 894
- forward references, 418–419
- FP. *See* functional programming
- frame objects, 94
- Frame widget (Tk), 825
- free variables, 457, 458
- “**from** module **import** *”, 487, 586
- from-import** statement, 485, 487, 489, 494–496
- fromkeys() method, 266, 268
- frozenset() factory function, 280, 283
- FTP (File Transfer Protocol), 748–756, 858
- function pointers, 426
- functional nesting, 456
- functional programming (FP), 439–453
 - anonymous functions/lambda, 439–441
 - apply(), 441, 442
 - built-in functions, 441–449
 - constructs, 24
 - debugging/performance measurement example, 437–439
 - examples, 437–439, 450–453
 - filter(), 441–445
 - map(), 441, 442, 445–447
 - partial function application, 450–453
 - reduce(), 441, 442, 447–449
- functions, 48–49, 409–439. *See also* functional programming; methods; scope
 - about, 409–410
 - accessing pathname, 342–347
 - arguments, 48, 49, 412–414, 423–424, 428–442
 - attributes, 420–421
 - callable objects, 628–629
 - calling, 49, 412–417, 436–439
 - classes vs., 518–519
 - closures, 456–463
 - creating, 418–426
 - declaring, 48, 72, 73, 418–419
 - declaring vs. definition, 418
 - decorators, 422–426
 - def** statement, 418
 - default arguments, 49, 413, 429–442
 - descriptors, 606
 - directory access, 340–341
 - examples, 415–417, 425–427, 431–432, 456–463
 - file access, 340–341
 - formal arguments, 428–432
 - forward references, 418–419
 - global vs. local variables, 453–455
 - grabbing Web pages example, 431–432
 - grouped arguments, 413–414
 - inner/nested, 421–422
 - integer-only, 143–145
 - keyword arguments, 412–413
 - lambda, 463–465
 - logging calls to, with closures, 461–463
 - numeric type, 137–143, 1023–1025
 - operator, 412
 - passing, 426–427
 - positional arguments, 428–429
 - procedures vs., 410
 - for **re** module, 684–685
 - return values, 410–412
 - standard type, 136–137, 1022–1023
 - variable argument objects, 436–439
 - variable-length arguments, 433–439
- functions (built-in) (BIFs)
 - attributes, 629
 - callable, 628–629, 636–642
 - classes, 558–564
 - conversion to factory functions, 595–596
 - executable objects, 636–642
 - file objects, 326–329
 - functional programming, 441–449
 - instances, 558–564
 - integer-only, 143–145
 - lists, 165, 166, 215–219, 242–245
 - mapping types, 260–262
 - module, 491–493
 - new-style classes, 595–596
 - numbers, 136–145
 - numeric types, 137–143, 150–151, 1023–1025
 - objects, 558–564
 - operational, 139–143, 166–168
 - sequence types, 165–168, 185–186, 216–219, 242–245, 1025–1028
 - sequence-related, 303–304
 - set types, 280, 283–284, 1036–1037
 - standard types, 101–110, 136–137, 215–216, 1022–1023
 - string types, 184–187, 242–245
 - tuples, 165, 166, 232–234, 242–245
 - wrapping, for exceptions, 365–368

functions (user-defined) (UDFs), 629–630
 __future__ directives, 489
 FXPy, 850

G

Gadfly database, 923, 939–945
 garbage collection, 79
 generalization, definition of, 517
 generator expressions, 315–319
 cross-product example, 317
 disk file example, 316–317
 refactoring example, 318–319
 generators, 467–471
 enhanced, 470–471
 simple, 468–470
 yield statement, 468
 geometry managers, 823
 __get__() method, 598–600, 602
 GET method, 880, 882, 895
 __getattr__() method, 588, 589, 597–598, 600
 getattr() built-in function, 560, 561, 588–590
 __getattrattribute__() special method, 597–600
 getopt module, 339
 GIL. *See* Global Interpreter Lock
 Glade, 850
 glob() module, 351, 352
 Global Interpreter Lock (GIL), 790–791, 980
global statement, 455
 global variables, 453–455
 globals() built-in function, 492
 GNOME-Python, 850
 Gopher, 858
 grandchild class, 554, 555
 graphical user interface programming. *See* GUI (graphical user interface) programming
 greater than symbols (> , >=), 277
 Greenlets, 27
 Grid (geometry manager), 823
 group() method, 685, 686
 grouped arguments (functions), 413–414
 grouping, 682–683, 692–693
 groups() method, 686
 GTK+, 840, 846–848, 850
 GUI (graphical user interface) programming, 819–851
 about, 822–824
 class example, 452–453

documentation, 851
 file system traversal example, 834–840
 FTP client program, 755
 GTK+/PyGTK, 840, 846–848
 other GUIs for, 840–849
 partial function application example, 831–834
 Python MegaWidgets, 840, 843
 related modules, 848–849
 Swing, 1003–1006
 Tel/Tk/Tkinter, 819–820
 Tix, 840, 842–843
 Windows clients as, 713
 wxWidgets/wxPython, 840, 843–846
 gzip module, 351, 352

H

hardware, 712
 hasattr() built-in function, 560, 561
 __hash__() method, 269
 hash() built-in function, 264, 265
 hash tables (dictionaries), 254
 hashable objects, 237, 253, 264, 269–272
 Haskell language, 24
 has_key() method, 256
 header file, including, 968
 heavyweight processes, 789
 “Hello World!” program, 32–34, 483, 826–831, 1003–1006
 help() built-in function, 34, 54
 hex() built-in function, 143–145
 hexadecimal output, 180
 hierarchy, definition of, 517
 host-port pairs, 716
 hotshot module, 85
 HTML (Hyper-Text Markup Language), 875–877
 HTTP (HyperText Transfer Protocol), 856, 867–869, 882
 HTTP_COOKIE environment variable, 896
 Hyper-Text Markup Language. *See* HTML
 HyperText Transfer Protocol. *See* HTTP

I

id() built-in function, 113, 114, 117
 identifiers, 67–68
 names, 70
 objects, 90
 scope, 453, 454

- identifiers (*continued*)
 - special with underscores, 69
 - style guidelines for names, 70
- identity comparison of objects, 96–100, 108
- IDEs (Integrated Development Environments), 17–22, 338
- IDLE (Integrated DeveLopment Environment) (Unix IDE), 18, 20–21
- if** statement, 41–42, 291–292, *See also* **elif** statement; **else** statement
- IMAP (Internet Mail Access Protocol), 772
- immutability, 65–66, 113, 194–196, 234–235, 273, 552
- implementation
 - of an abstraction, 516
 - hiding, 585
- `__import__()` built-in function, 491
- import** statement, 484
 - absolute import** statement, 494–495
 - at end of modules, 499
 - extended import** statement (*as*), 485–486
 - from-import** statement, 485, 487, 489, 494–496
 - loading of, 650–651
- importing attributes, 496
- importing modules, 52, 484–486
 - attributes for, 496
 - built-in functions, 491–493
 - case-insensitive, 496–497
 - definition of, 477
 - execution on, 73–74, 650–651
 - extension module, 976
 - features of, 486–491
 - `__future__`, 489
 - `globals()`, 492
 - `__import__()`, 491
 - import cycles, 497–500
 - loading vs., 487
 - `locals()`, 492
 - multi-line import, 485
 - names imported into current namespace, 487
 - names imported into importer’s scope, 488–489
 - new import hooks, 490–491
 - packages, 493–495
 - path search/search path, 478–480
 - related modules, 500–501
 - relative import, 495
 - `reload()`, 492–493
 - search path/path search, 478–480
 - style guidelines for, 72
 - Tkinter module, 821–822
 - from zip files, 490
- indentation
 - for code blocks, 41
 - “dangling `else`” statements, 293
 - style guidelines for, 70
 - of suites, 63–64
- `IndexError` exception, 363
- indexing slices, 164–165
- inequality sets, 276
- infinite loops, 297–298
- inheritance, 547–558
 - `__bases__` class attribute, 548
 - definition of, 517
 - diamond shape hierarchy, 556–558
 - multiple, 553–558
 - overriding methods via, 549–551
 - subclassing with, 512
- `__init__()` method
 - as constructor, 51, 510–511, 527–529
 - customizing classes with, 565, 634
 - instantiation, 510, 527–530, 533–534
 - overriding, 549–551
 - return value, 533–534
 - setting instance attributes in, 531–533
 - and tracking instances, 530
- `initModule()` module initializer function, 973–974
- inner/nested functions, 421–422, 456–458
- “in-place” operations, 569, 574–575
- input
 - file built-in methods, 329–336
 - `raw_input()`, 32–33, 54, 186, 641–642
 - standard, 337
 - user, 875–877, 886–892
- `input()` built-in function, 636, 641–642
- inserting rows, 920–922
- installing Python, 11–13
- instance attributes, 531–540
 - accessing, 511
 - class attributes vs., 537–540
 - definition of, 506–507
 - determining, 534
 - instantiation of, 531–534
 - “on-the-fly,” 531
 - setting, 531, 532
 - special, 534–536

- instances, 526–530
 - about, 504–507
 - binding methods to, 522
 - built-in functions, 558–564
 - callable objects, 51, 634–635
 - creating, 51–52, 511, 526–527
 - default values for, 532–533
 - definition of, 50
 - `__del__()`, 528–530
 - invoking methods via, 511–512
 - keeping track of, 530
 - instantiation
 - creating instances, 511, 526–527
 - default arguments with, 532–533
 - definition of, 506
 - `__init__()`, 510, 527–530, 533–534
 - instance attributes, 531–534
 - `__new__()`, 528
 - Thread class, 802–807
 - `int()` built-in function, 54, 142
 - `int()` factory function, 137, 138, 141
 - int type (unsupported), 117
 - integer-only functions, 143–145
 - integers, 122–125
 - bit operators, 135–136
 - Boolean, 123
 - format operator output, 180–181
 - long, 37, 38, 123–125
 - standard, 37, 123
 - unification of long integers/integers, 38, 124–125
 - unsupported types, 117
 - Integrated DeveLopment Environment. *See* IDLE
 - Integrated Development Environments. *See* IDEs
 - integration of code (Python/non-Python), 963
 - interfaces to data attributes, 516
 - internal types, 93–95
 - Internet, architecture of, 856–859
 - Internet client programming, 747–779.
 - See also* Web programming
 - about, 747–748
 - electronic mail, 766–778
 - FTP, 748–756, 858
 - newsgroups, 756
 - NNTP, 756–766, 858
 - related modules, 778–779
 - SMTP, 767–772, 775–777, 858
 - transferring files, 748–756
 - Usenet, 756
 - Web programming vs., 858–859
 - Internet Mail Access Protocol (IMAP), 772
 - Internet protocols, 858, 912. *See also specific headings, e.g.,* NNTP (Network News Transfer Protocol)
 - Internet Service Provider. *See* ISP
 - interning, 100
 - interpreted languages, 6, 11, 23, 965
 - interprocess communications (IPC), 715
 - intersection (`&`) operator, 277
 - intersection update (`&=`) operator, 279
 - Intranet, 857
 - introspection, 518
 - `IOError` exception, 326, 363, 365, 373, 392, 394, 399
 - IPC (interprocess communications), 715
 - IronPython, 12, 27, 989
 - is keyword, 99
 - `isatty()` built-in method, 332, 335
 - `isinstance()` built-in function, 106–109, 559–560, 596–597
 - ISP (Internet Service Provider), 857, 858
 - `issubclass()` built-in function, 558–559
 - `items()` built-in methods (dictionaries), 265, 266
 - `__iter__()` built-in method, 576–579
 - `iter()` function, 313
 - iterating/iteration
 - file, 331–332
 - mechanics of, 309–310
 - by sequence item/index, 299–301
 - through a matrix, 314–315
 - through files, 315
 - iterators, 309–313
 - about, 309–310
 - any number of terms example, 577–579
 - creating, 313
 - dictionaries, 311–312
 - files, 312
 - for loops, 301, 309–313, 331–332
 - mutable objects, 312
 - related modules, 319–320
 - sequences, 309–311
 - `itertools` module, 319–320
- J**
- Java, 8, 12, 23–26, 969, 1002–1003
 - JavaScript, 23, 24
 - JIT (just-in-time) compiler, 982
 - `join()` method, 159, 176, 189, 191

just-in-time (JIT) compiler, 982
 Jython, 12, 24, 26, 1002–1006

K

Kanter, Brian, 756
 KDE desktop environment, 850
 KeyboardInterrupt exception, 371, 391, 394
 KeyError, 257, 363
 key-lookup operator ([]), 259
 keys (dictionary)
 collisions, 268–269
 comparing, 261–262
 as hashable object, 253, 264, 269–272
 mapping type, 253, 254
 restrictions on, 268–269
 keys () built-in method, 254, 255, 265, 267
 key-value pairs, 40, 254, 257
 keyword arguments (functions), 412–413
 keyword variable arguments (dictionary), 434–436
 keywords
 and, or, not, 36, 99–101
 class, 50, 504
 def, 48, 418, 421
 identifiers, 67
 is, 99
 partial function application, 451–452
 tables of, 68, 1022

L

Label widget (Tk), 825–831
 lambda
 anonymous functions, 439–441
 callable objects, 630–631
 inner functions, 422
 and Lisp, 24
 list comprehensions, 313–314
 map (), 446–447
 reduce (), 448, 449
 return value, 439
 scope, 463–465
 LAN (Local Area Network), 857, 858
 languages
 comparisons of, 23–26
 compiles, 965
 high-level, 6–7
 interpreted, 6, 11, 23, 965
 Lapsley, Phil, 756
 last-in-first-out (LIFO) data structure, 223

len () built-in function, 54
 dictionaries, 264, 265
 lists, 216–217
 and range (), 45
 sequences, 166, 167, 185
 set types, 280, 283
 strings, 185
 less than symbols (< , <=), 277
 lexical variables, 458–463
 LIFO (last-in-first-out) data structure, 223
 lightweight processes, 789. *See also* threads
 line separators (terminators), 330, 333
 linking wrapper extensions, 975
 Linux, 12, 14, 16–17
 Lisp, 24, 25
 list () built-in function, 165, 166
 list comprehensions, 24, 45, 215, 313–316.
 See also generator expressions
 list () factory function, 218–219
 Listbox widget (Tk), 825
 lists, 39–40, 208–210
 accessing values in, 209
 assigning, 209
 as building blocks, 7
 building data structures with, 223–230
 built-in functions, 165, 166, 215–219,
 242–245
 built-in methods, 219–223, 242–245, 1034
 concatenation of, 213–214
 creating, 209, 223–230
 membership operators, 213
 operators for, 211–215, 242–245
 other data structures, creating with, 223–230
 queues using, 227–230
 removing elements/lists, 210–211
 repetition in, 214
 sequence type functions, 216–219, 242–245
 sequence type operators, 211–215, 242–245
 slices of, 211–213
 special features of, 223–230
 stacks using, 223–227
 standard type functions, 215–216
 standard type operators, 211
 tables of, 242–245
 tuples vs., 237–238
 updating, 210
 load () function, 350
 loading modules, 486, 487, 496

- Local Area Network. *See* LAN
- local variables, 81, 453–455, 562
- localhost, 720
- locals() built-in function, 492
- lock objects, 795
- logging function calls (with closures), 461–463
- logging module, 84
- logical errors, 360
- login/password programming example, 269–272
- long() factory function, 137, 138
- long integers, 37, 38, 123–125
- long type (unsupported), 117
- loops, 296–308. *See also* iterators
- auxiliary statements, 308
 - break** statement, 304–305, 308
 - continue** statement, 305–306
 - counting, 297
 - else** statement, 307–308
 - infinite, 297–298
 - and iterators, 301, 309–313, 331–332
 - and **pass** statement, 306–307
 - performance enhancement for, 175
 - range(), 44–45
 - for** statement, 43–45, 298–308, 331–332
 - while** statement, 42, 296–298, 304–308
- ## M
- MacOS X, 12, 14–19, 333
- macros (for reference counting), 980
- mail user agent (MUA), 771–772
- main(), 73
- maintainability, 9–10
- makefiles, 13
- mangled names. *See* name-mangling
- map() built-in function, 313, 314, 441, 442, 445–447
- mapping types
- access model category, 115–116
 - built-in functions/factory functions, 260–265
 - built-in methods, 265–268
 - dictionaries, 253–258
 - keys for, 253
 - operators for, 258–259
 - related functions, 263–265
 - special methods for customizing classes, 568
- marshal module, 348–350
- “Mash-ups,” 985
- match objects, 686
- matching. *See also* regular expressions
- any single character (.), 678, 689–690
 - beginning/end of strings, 678–679
 - closure operators, 680–681
 - “greedy” operators, 703–705
 - grouping, 682–683, 690–693
 - more than one pattern, with alternation (|), 678
 - more than one string, 689
 - multiple occurrence/repetition using closure operators, 680–681
 - negation (^), 680
 - parentheses (()), 682–683
 - ranges (-), denoting, 680
 - re module, 206–207
 - regular expressions, 680–681
 - repetition, 680–681, 690–693
 - searching vs., 675, 688, 703–705
 - special characters/symbols, 676–677, 682, 690–693
 - strings, 678–679, 687–689, 693, 701–703
 - word boundaries, 693
- match() re module function, 684, 687–688
- mathematical operators, 35–36, 130–135
- matrix iteration, 314–315
- max() built-in function, 166, 167, 185, 217
- membership (in, not in) operators, 158–159
- dictionaries, 259
 - lists, 213
 - for loops, 299
 - sequences, 158–159, 172–175
 - set types, 273, 276
 - strings, 172–175
 - tuples, 233
- memory management, 75–79
- dynamic typing, 76
 - garbage collection, 79
 - interpreter performing, 11
 - memory allocation, 76
 - reference counting, 76–79, 99, 977, 979–980
 - variables declarations, 75
- Menu widget (Tk), 825
- message transport agent (MTA), 767, 768, 772
- message transport system (MTS), 767
- Message widget (Tk), 825
- metacharacters, 676–677
- __metaclass__, 610–615
- metaclasses, 610–615
- method descriptors, 599

- method resolution order (MRO), 554–558, 562
- methods, 521–522
 - about, 507–512
 - arguments, 532–533
 - binding, 522, 540–541
 - callable objects, 631–633
 - class, 543
 - connection objects, 928
 - decorators, 422–426
 - file object, 335–336, 1038–1039
 - group(s), 686
 - invoking, 511–512, 540–541
 - naming, 513
 - for new-style classes, 597–600
 - overriding, via inheritance, 549–551
 - privacy, 585–586
 - for `re` module, 684–685
 - static, 542–544
 - strings, 188–191
- methods (built-in)
 - attributes, 632
 - dictionaries, 254, 255, 265–268, 1034, 1035
 - files, 329–336
 - lists, 219–223, 242–245, 1034
 - mapping types, 265–268
 - sequences, 242–245, 1025–1028
 - set types, 281–284, 1036–1037
 - strings, 188–191, 242–245, 1030–1033
 - tuples, 242–245
- methods (special) (for customizing classes), 564–585
 - any number of terms iterator example, 577–579
 - iterators, 576–579
 - multi-type example, 579–585
 - numeric customization (Time62) example, 572–576
 - Random Sequence iterator example, 576–577
 - RoundFloat2 simple example, 569–572
 - special, 564–585, 1043–1046
 - tables of, 565–568, 1043–1046
- methods (special) (new-style classes), 597–600
- methods (user-defined) (UDMs), 632–633
- Microsoft Office
 - Excel, 991–993, 1000–1002
 - Outlook, 996–1000
 - PowerPoint, 994–997
 - programming, with Win32 COM, 989–1002
 - Word, 993–994
- MIME (Multipurpose Internet Mail Extension)
 - headers, 863, 877
- `min()` built-in function, 166, 167, 185, 217
- mixed mode operations, 127–129, 204, 278
- mix-in classes, 556
- modeling (OOD), 515
- `__module__` attribute, 524, 525, 629, 633
- modulefinder, 500
- `ModuleMethods` [] array, 973
- modules, 52–53. *See also* importing modules
 - about, 64, 477
 - accessing module variables, 52–53
 - auto-loaded, 496
 - built-in functions, 491–493
 - `__builtins__` vs. `__builtin__`, 480–481
 - calling, 52–53
 - case-insensitive import, 496–497
 - cProfile, 85
 - debugging, 84
 - developer tools, 84–85
 - “executed” when loaded, 486
 - executing as scripts, 652–653
 - executing on import, 650–651
 - extended import** statement (as), 485–486
 - and files, 478–480
 - `__future__`, 489
 - hotshot, 85
 - import cycles, 497–500
 - importing vs. loading, 487
 - logging, 84
 - multi-line import, 485
 - names imported into importer’s scope, 488–489
 - namespaces, 478, 480–483
 - new import hooks, 490–491
 - numeric types, 148–149
 - packages, 493–495
 - pdb, 84
 - persistent storage, 348–350
 - preventing attribute import, 496
 - profile, 85
 - search path/path search, 478–480
 - separating, 64
 - sequence types, 238–239
 - source code encoding, 497
 - standard library, 148–149, 205–207
 - strings, 205–207
 - structure/layout for, 71–74
 - subprocess, 660–662

- warning framework, 489–490
 - Web server, 909
 - modulus operator, 132–133
 - MRO. *See* method resolution order
 - MTA. *See* message transport agent
 - MTS (message transport system), 767
 - MUA. *See* mail user agent
 - multi-line import, 485
 - multipart form submission, 894
 - multi-platform development, 12, 333, 964
 - multiple assignment, 66
 - multiple inheritance, 553–558
 - multiplication (*) operator, 35
 - Multipurpose Internet Mail Extension headers. *See* MIME (Multipurpose Internet Mail Extension) headers
 - multithreaded programming (MT), 787–814
 - about, 787–789
 - accessing threads, 792
 - examples, 792–793
 - exiting threads, 791
 - extensions to Python, 980
 - global interpreter lock, 790–791, 980
 - processes, 789
 - related modules, 813–814
 - thread module, 794–799, 814
 - threading module, 793, 794, 800–814
 - threads, 721, 789–813, 925
 - multi-type customization example, 579–585
 - “multiple” assignment, 66–67
 - mutable hash tables, 253
 - mutable objects, 65, 90, 113, 222, 312
 - mutable sets, 273, 279, 281–282, 284
 - mutable types, subclassing, 552–553
 - mutex module, 814
 - MySQL, 920, 934–936, 939–945
- N**
- __name__ attribute, 524, 629, 630, 632, 633
 - name lookup, 482–483
 - __name__ system variable, 73, 74
 - NameError exception, 362, 454, 455, 522
 - name-mangling, 516, 586
 - namespace(s), 480–483
 - __builtins__, 480–481
 - classes as containers for, 506
 - “Hello World!” example, 483
 - importing names into current, 487
 - modules, 478, 480–483
 - name lookup/scoping/overriding, 482–483
 - overriding, 482–483
 - scope, 454
 - types of, 480–481
 - useful features of, 483
 - variable scope, 465–466, 481–482
 - negation symbol (^), 680
 - .NET/Mono implementation, 27
 - network location components, 860
 - Network News Transfer Protocol. *See* NNTP
 - network programming, 711–742
 - client/server architecture, 711–715
 - functions/modules for, 718–731
 - related modules, 741–742
 - sockets, 715–720, 730–731
 - SocketServer module, 732–736
 - TCP clients/servers, 720–726, 732–740
 - twisted framework, 737–740
 - UDP clients/servers, 726–730
 - __new__() method, 528, 565
 - new import hooks, 490–491
 - NEWLINE character(s)
 - continuation (\), 62
 - escaping, 192
 - POSIX systems, 333
 - print** statement, 43
 - suppression of, 333–334
 - universal support for, 326, 328–329
 - write(), 53
 - newsgroups, 756
 - new-style classes. *See also* classic classes
 - advanced features of, 595–615
 - classic classes vs., 504
 - descriptors, 598–610
 - documentation, 617
 - general features, 595–597
 - __getattr__(), 597–600
 - metaclasses, 610–615
 - method resolution order, 554–558
 - and OOP, 503
 - privacy, 586
 - __slots__ class attributes, 597
 - super(), 562–563
 - next() built-in method, 335, 468–471, 576–579
 - NNTP (Network News Transfer Protocol), 756–757
 - client program, 760–765
 - documentation, 765–766

- NNTP (*continued*)
 - examples, 759–765
 - interactive, 759–760
 - nntplib>NNTP class methods, 758–759
 - object methods, 759–760
 - as original Internet protocol, 858
 - Python and, 758–759
 - non-data descriptors, 599
 - None type, 410–412, 533–534
 - non-keyword variable-length arguments (tuple), 433–434
 - non-Python programs, 653–654
 - “not equals” comparison operators (`!=` , `<>`), 36
 - not keyword, 36, 99–101
 - Not ImplementedError exception, 520
 - NUL characters, 192, 208
 - Null object, 92–93
 - numbers
 - assignment, 121
 - bit operators, 135–136
 - Boolean, 145–147
 - built-in/factory functions, 136–145
 - complex, 126–127
 - creating, 121
 - double precision, 125
 - floating point, 37, 38, 125, 147–148, 180
 - integers, 122–125
 - introduction to, 121–122
 - mathematical operators, 35–36, 130–135
 - mixed-mode operations, 127–129
 - numeric type functions, 137–143, 150–151, 1023–1025
 - numeric type operators, 127–136, 150–151, 1023–1025
 - operators, 35–36, 127–136, 150–151, 1023–1025
 - removing, 122
 - standard type functions, 136–137
 - standard type operators, 129–130
 - types, 37–38
 - updating, 122
 - numeric coercion, 128, 129
 - numeric customization (Time62) customization
 - example, 572–576
 - numeric type(s), 37–38
 - Boolean “numbers,” 145–147
 - functions, 137–143, 150–151, 1023–1025
 - operators, 127–136, 150–151, 1023–1025
 - related modules for, 148–149
 - special methods for customizing classes, 566–567, 569
- O**
- object(s), 89–95. *See also specific types*
 - assignment of, 65
 - attributes, 90
 - Boolean operators, 100–101
 - Boolean values of, 93, 96–97
 - built-in functions, 558–564
 - built-in types, 91
 - calling functions with variable arguments, 436–439
 - characteristics of, 90
 - classes/instances, 504–507
 - code, 94, 635–636
 - composite/compound, 112
 - connection, 927–928
 - copying, 239–241
 - cursor, 929–930
 - as default class, 504
 - ellipsis, 95
 - executable, 636–642
 - “file-like,” 325–326, 353
 - frame, 94
 - hashable, 237, 253, 264, 269–272
 - identity comparison of, 97–100, 108
 - internal types, 93–95
 - invoking, 526–527
 - mutable, 65, 90, 113, 222, 312
 - Null, 92–93
 - removing single reference to, 78–79
 - slice, 95
 - standard type operators, 96–101
 - standard types, 91, 93
 - traceback, 94
 - value comparison of, 93, 96–97, 108
 - wrapping, 588–595
 - XRange, 95
 - object-oriented design. *See* OOD
 - object-oriented language, 7
 - object-oriented programming. *See* OOP
 - object-relational managers. *See* ORMs
 - objects (callable), 628–635
 - class instances, 634–635
 - classes, 634
 - functions, 628–629

- lambda, 439
- methods, 631–633
- oct () built-in function, 143–145
- OOD (object-oriented design), 514–516, 545
- OOP (object-oriented programming), 504–518, 520–617
 - about, 504–514
 - buzzwords, 516–518
 - classes, 518–520
 - languages, 585–586
 - new-style classes, 503
 - and real-world problems, 514–515
 - relationship OOD and, 514–515
- open () built-in function, 46–47, 54, 326–328
- operational built-in functions, 139–143, 166–168
- operations (database), 920
- operator(s)
 - assignment (=), 64–65
 - asterisk (*), 680–681
 - augmented assignment, 65–66
 - bit, 135–136
 - Boolean, 36, 100–101
 - brace ({ }), 681
 - closure, 680–681
 - dictionaries, 258–259
 - difference (-), 278
 - difference update (-=), 279
 - division, 35, 130–133, 569
 - exponentiation (**), 35, 133, 140
 - format, 178–181, 205, 208, 1029
 - function, 412
 - “greedy,” 703–705
 - intersection (&), 277
 - intersection update (&=), 279
 - key-lookup ([]), 259
 - lists, 211–215, 242–245
 - mapping types, 258–259
 - mathematical operators, 35–36, 130–135
 - membership, 158–159, 172–175, 213, 233, 259, 273, 276, 299
 - and mixed mode operations, 127–129
 - modulus, 132–133
 - multiple assignment, 66
 - “multiple” assignment, 66–67
 - “not equals” comparison (!=, <>), 36
 - numeric type, 127–136, 150–151, 1023–1025
 - overloading addition, 573–575
 - parentheses and, 36
 - plus (+), 681
 - question mark (?), 681
 - raw string, 168, 182–184
 - repetition (*) sequence, 159–160
 - retention update (&=), 279
 - reverse quote (` `), 103–104
 - sequence type, 158–159, 170–178, 211–215, 232–233, 242–245, 1025–1028
 - set type, 273, 274, 276–279, 282–284, 1036–1037
 - slices ([], [:], [: :]), 39, 160–165, 170–172, 211–213, 233
 - standard type, 129–130, 1022–1023
 - string format, 178–181, 205, 208, 1029, 1030
 - strings, 168, 170–184, 208, 242–245
 - symmetric difference (^), 278
 - symmetric difference update (^=), 279
 - table of, 1046–1048
 - ternary, 295–296
 - tuples, 232–234, 242–245
 - Unicode string (u/U), 184
 - union (|), 277, 279
 - update (|=), 279
- operator module, 616–617
- optparse module, 339
- or keyword, 36, 100, 101
- ord () built-in function, 144, 145, 187, 204
- ORMs (object-relational managers)
 - employee role database example, 947–957
 - related modules, 959
 - and SQL, 946
 - SQLAlchemy, 946–950, 953–955
 - SQLObject, 946, 951–953, 955–957
 - as storage mechanism, 919
- os module
 - additional functionality of, 347
 - attributes, 667
 - examples, 343–347
 - external program execution functions, 654–662
 - file/directory access functions, 339–341
 - os.exec* (), 654–655, 659
 - os.wait* (), 655, 659–660
 - os._exit (), 666
 - os.fork (), 654, 658–659
 - os.kill (), 666
 - os.popen (), 655, 657, 661–662
 - os.spawn* (), 655, 660
 - os.system (), 654, 656–657, 661
- OSError exception, 365
- os.path module, 81–82, 84, 342–347, 352
- Outlook, 996–1000

- output (program), 32–33
 - output (standard), 337
 - output built-in methods, 330
 - overriding (overloading)
 - built-in names, 69
 - global variable, 454
 - methods via inheritance, 549–551
 - and mixed-mode operation, 128
 - namespaces, 482–483
- P**
- packages, 493–495
 - packer (geometry manager), 823
 - Pango, 846
 - paramstyle, 925, 926
 - parentheses (()), 36, 673–682
 - parent/parent class, 512, 545–548, 554, 555, 586, 822, 823
 - partial function application (PFA), 450–453, 831–834
 - pass** statement, 306–308, 504
 - passing functions, 426–427
 - path search, 478–480
 - pathname, 14, 16, 342–347
 - pattern matching. *See* regular expressions (REs)
 - pdb debugging module, 84
 - PEPs (Python Enhancement Proposals), 53, 71, 617
 - performance enhancement, 8, 175, 965
 - performance measurement example, 437–439
 - period symbol (.), 678, 689–690
 - Perl, 23, 25, 26
 - persistent storage, 348–350, 919, 920
 - PFA. *See* partial function application
 - PHP, 25
 - pickle module, 202, 348–350
 - pipe symbol (|), 678, 689
 - pkgutil, 500
 - plain integers, 123
 - plus operator (+), 681
 - PMW. *See* Python MegaWidgets
 - Pmw (Python MegaWidgets), 849
 - pointer type, 117
 - polymorphism, 517–518
 - POP (Post Office Protocols), 772
 - pop() method, 281, 282, 284
 - POP3, 772–777
 - port numbers, 716
 - portability, 8–9
 - positional arguments (functions), 428–429
 - POSIX systems, 333, 792
 - Post Office Protocols. *See* POP; POP3
 - Postel, Jonathan, 768
 - PostgreSQL, 936–937
 - pound sign (#) (hash symbol), 34, 62
 - pow() built-in function, 140, 143
 - PowerPoint, 994–997
 - precedence, 35, 600, 602
 - precision, 117
 - print** statement, 32–33
 - printf()-like functionality, 208
 - privacy, 585–586, 965
 - procedures, functions vs., 410
 - processes, definition of, 789
 - producer-consumer problem, 810–813
 - profile module, 85
 - profiling of code, 8, 84–85, 965
 - programmers, 402–403
 - programs
 - executing other non-Python, 653–654
 - executing other Python, 649–653
 - prompts (primary/secondary), 31
 - property() built-in function, 606–610
 - proprietary source code, 965
 - protocols (Internet), 858
 - Psyco, 981–982
 - “public” attributes, 585
 - pure virtual functions, 519
 - .py file extension, 11, 478
 - PyArg_Parse*() function, 969
 - PyArg_ParseTuple() function, 971
 - Py_BuildValue() function, 969, 971, 972
 - .pyc files, 11, 965
 - pyFLTK, 850
 - PyGTK, 840, 846–848, 850
 - PyGUI, 850
 - Py_InitModule(), 973–974
 - .pyo file extension, 11
 - PyObject, 969–973
 - PyOpenGL, 850
 - PyQt, 850
 - PyQtGPL, 850
 - Pyrex, 981
 - Python Enhancement Proposals. *See* PEPs
 - Python FAQ, 617
 - Python Library and Language Reference manual, 617

Python MegaWidgets (PMW), 840, 843
 Python version 2.0, 313
 Python version 2.2, 526–527, 617
 Python version 2.4, 316
 Python Virtual Machine, 790–791
 PythonCard, 849
 PYTHONCASEOK environment variable, 497
 PYTHONPATH environment variable, 479, 500
 PythonWin IDE, 19–21

Q

Qt GUI, 850
 querying (databases), 920
 question mark operator (?), 681
 Queue module, 793, 810–814
 queue, using lists to build, 227–230
 quotation marks, 207
 quote* () functions, 865–866

R

race condition, 790
 Radiobutton widget (Tk), 825
raise statement, 48, 386–389
 raising an exception, 360, 361
 random module, 149
 Random Sequence iterator example, 576–577
 range () built-in function, 44–45, 54, 219, 300, 301–303
 range symbol (-), 680
 ranges (-), 680, 682
 rapid prototyping, 10
 raw strings, 168, 182–184, 208, 698
 raw_input () built-in function, 33–34, 54, 186, 641–642
 RDBMS (relational database systems), 919, 920, 931–932
 re module, 206–207, 684–698
 read () built-in method, 329, 335
 readinto () method, 330, 335
 readline () built-in method, 312, 329, 335
 readlines () built-in method, 329, 336
 realm, 867
 reason (exceptions), 373
 rebinding, 480
 receiving e-mail, 771–772
 recursion, 466–467
 redirecting output, 33
 reduce () built-in function, 441, 442, 447–449

refactoring, 318–319
 refcount, 76
 reference, 65, 79, 239
 reference counting, 76–79, 99, 977, 979–980
 reflection, definition of, 518
 regular expression engine, 23, 204
 regular expressions (REs)
 about, 673–676
 any single character (.), 678, 689–690
 ASCII characters, 698
 beginning/end of strings, 678–679
 character classes ([]), creating, 679–680, 690
 compiling, 685–686
 example, 698–705
 finding every occurrence, 694
 and “greedy” operators, 703–705
 grouping, 682–683, 690–693
 match (), 684, 687–688
 match objects/group(s) methods, 686
 matching more than one pattern, with alternation (|), 678
 matching more than one string, 689
 matching strings, 701–703
 matching word boundaries, 693
 multiple occurrence/repetition using closure operators, 680–681
 negation (^), 680
 parentheses (()), 682–683
 ranges (-), denoting, 680
 raw strings, 183, 698
 re module, 206–207, 684–698
 repetition, 680–681, 690–693
 searching vs. matching, 675, 688, 703–705
 searching/replacing, 694–695
 search () re module, 688
 special characters/symbols, 676–677, 682, 690–693
 splitting on delimiting pattern with, 695–698
 strings, 701–703
 sub ()/subn (), 694–695
 word boundaries, 693
 regular integers, 123
 relational database systems. *See* RDBMS
 relative complement (-) operator. *See* difference (-) operator
 relative import (packages), 495
 reload () built-in function, 492–493
 remove () method, 281, 282, 284

- removing
 - dictionary elements/dictionaries, 258
 - lists/list elements, 210–211
 - numbers, 122
 - set members/sets, 276
 - single object reference, 78–79
 - strings/characters, 169–170
 - tuple elements/tuples, 232
 - repetition
 - lists, 214
 - regular expressions, 680–681, 690–693
 - special characters/grouping and, 690–693
 - strings, 177–178
 - tuples, 233
 - repetition (*) sequence operator, 159–160
 - repr () built-in function, 102–104
 - REs. *See* regular expressions
 - restricted execution, 663
 - retention update (&=) operator, 279
 - return value(s), 222, 410–412, 439, 533–534
 - reverse quote operator (' '), 103–104
 - reversed () built-in function, 166, 167, 217, 222, 303, 304, 310
 - Rexx, 26
 - robustness, 10
 - root window, 822
 - round () built-in function, 140–143
 - RoundFloat2 customization example, 569–572
 - rows, 920–922
 - Ruby, 24, 25
 - running Python, 13–22
 - in an IDE, 17–21
 - code examples, 22
 - interactive interpreter from command line, 14–16
 - as a script from command line, 16–17
 - runtime errors, 361
 - runtime generation/execution of code, 642–649
 - RuntimeError exception, 392
- S**
- scability, 7–8
 - scalar storage, 112
 - Scale widget (Tk), 825, 829–831
 - scope, 453–466
 - closures, 456–458
 - global statement, 455
 - global vs. local, 453–455
 - lambda, 463–465
 - name lookup, 482–483
 - names imported into importer's, 488–489
 - namespaces, 465–466, 481–482
 - number of, 456
 - overriding, 482–483
 - scripts
 - communicating with MS Office using, 990–1002
 - generating/executing code with, 642–649
 - as modules, 64
 - running Python as, 16–17
 - Scrollbar widget (Tk), 825
 - search path, 14–17, 478–480
 - searching, 454, 675, 688, 703–705
 - search () re module, 684, 688
 - Secure Socket Layer (SSL), 866
 - seek () built-in method, 331, 334, 335
 - select () function, 741
 - self argument, 51, 507, 510, 541, 550, 551
 - __self__ attribute, 629, 632
 - semicolon (;) (multiple statements), 64
 - sending e-mail, 767–768
 - sequence(s), 157–167. *See also* lists; strings; tuples
 - access model type, 115–116
 - built-in functions, 165–168, 185–186, 216–219, 242–245, 1025–1026
 - concatenation of, 159
 - conversion/casting of, 165–166
 - and iterators, 309–311
 - keys for, 253, 254
 - membership operators, 158–159, 172–175
 - methods, 242–245
 - operational built-in functions for, 166–168
 - operators, 158–159, 170–178, 211–215, 232–233, 242–245, 1025–1028
 - related modules for, 238–239
 - repetition of, 159–160
 - sequence-related built-in functions, 303–304
 - slicing, 160–165
 - special methods for customizing classes, 567–568
 - standard type operators, 158
 - for statement used with, 299–301
 - stride indices, 162–165
 - strings, 170–178
 - table of, 242–245
 - servers
 - bank tellers as example of, 713–714
 - infinite loop for, 823–824

- TCP, 720–723, 725–726, 732–738, 740
- twisted reactor TCP, 740
- UDP, 726–730
- Web, 401–402, 713, 855–858, 875–879, 906–909, 911
- window system as, 821
- server-side COM programming, 990
- `__set__`() special method, 598, 599
- `set`() factory function, 280
- set types, 273–276
 - accessing values in, 275
 - assigning, 273, 274
 - built-in functions, 280, 283–284, 1036–1037
 - built-in methods, 281–284, 1036–1037
 - creating, 273, 274
 - difference (-) operator, 278
 - difference update (-=) operator, 279
 - equality/inequality, 276
 - factory functions, 280, 283–284
 - `frozenset`() factory function, 280
 - intersection (&) operator, 277
 - intersection update (&=) operator, 279
 - `len`() built-in function, 280
 - membership operator, 273, 276
 - mixed operations, 278
 - operation/relation symbols, 274
 - operators, 276–279, 283–284, 1036–1037
 - related modules, 284
 - removing set members/sets, 276
 - retention update (&=) operator, 279
 - `set`() factory function, 280
 - subsets/supersets, 277
 - symmetric difference (^) operator, 278
 - symmetric difference update (^=) operator, 279
 - table of, 282–284
 - types of sets, 273
 - union (|) operator, 277, 279
 - update (|=) operator, 279
 - updating, 275
 - using operators vs. built-in methods, 282
- `setattr`() built-in function, 536, 560, 561
- `setdefault`() built-in method, 266–268
- shallow copy, 240
- shell scripting, 23
- `shelve` module, 349–350
- short type (unsupported), 117
- `showname`() method, 52
- `shutil` module, 351, 352
- Simple Mail Transfer Protocol. *See* SMTP
- `SimpleHTTPServer` module, 907, 908, 911
- Simplified Wrapper and Interface Generator (SWIG), 981
- single character (.), 678, 689–690
- single element tuples, 236–237
- single quotation mark ('), 207
- single underscore (_) attributes, 586
- `site` module, 500
- sizes, comparing dictionary, 261
- sleeping (threads), 789, 792–793, 796–799, 803–807
- slice objects, 95
- slices ([] , [:] , [: :]) sequence operators (slicing), 160–165
 - indexing, 164–165
 - lists, 211–213
 - stride indices, 162–165
 - strings, 39, 170–172
 - tuples, 39, 233
- `__slots__` class attributes, 597
- SMTP (Simple Mail Transfer Protocol)
 - about, 768
 - clients, 775–777
 - documentation, 771
 - e-mail, 767, 772
 - example, 770–771, 775–777
 - interactive, 770–771
 - object methods for, 769
 - as original Internet protocol, 858
 - Python and, 768–769
 - `smtplib.SMTP` class modules, 769
- `smtplib.SMTP` class modules, 769
- `socket(s)`, 715–718
 - about, 715–716
 - addresses, 716
 - built-in methods, 719–720
 - connection-oriented vs. connectionless, 716–718
 - creating, 721, 724
- `socket`() module function, 718, 730–731, 741–742
- `SocketServer` module, 721, 732–736, 741, 814
- `SOCK_STREAM` type, 717
- software (client/server architecture), 712–713
- Solaris, 12, 14, 16–17
- `sorted`() built-in function, 167, 217, 222, 267, 303, 304
- source code (encoding), 497

- spacing and block delimitation, 63
- spawning
 - processes, 789
 - threads, 721
- special symbols, 676–677
- specialization, 517
- `split()` method, 190, 191
- `split()` `re` module, 685, 695–698
- SQL (Structured Query Language), 920–922, 946
- SQLAlchemy, 946, 947–950, 953–955
- SQLite, 937–945
- SQLObject, 946, 951–953, 955–957
- SSL (Secure Socket Layer), 866
- stack, building using lists, 223–227
- “stack trace,” 10
- stackless Python implementation, 12, 27
- standard error, 337
- standard exceptions, 391–394
- standard files, 337–338
- standard input, 337
- standard integers, 37, 123
- standard output, 337
- standard type functions
 - built-in, 101–110
 - dictionaries, 260
 - lists, 215–216
 - mapping type, 260
 - numeric, 136–137
 - set types, 280
 - strings, 184–185
 - table of, 1022–1023
- standard type operators
 - Boolean, 100–101
 - dictionaries, 259
 - lists, 211
 - mapping type, 259
 - numeric, 129–130
 - objects, 96–101
 - sequence types, 158
 - set types, 276–277
 - strings, 170
 - tables of, 110, 1022–1023
 - tuples, 232–233
- standard types
 - about, 91
 - by access model, 115–116
 - categorizing, 111–116
 - deriving, 551–553
 - by storage model, 112–113
 - unsupported types, 116–117
 - by update model, 113–114
 - wrapping, 592–593
- `StandardError` exception, 373, 391
- star operator (`*`), 680–681
- “stateless” protocol, 856, 895
- statements (Python)
 - comments, 34–35, 62, 69–70
 - continuation, 62
 - grouping multiple, 62–63
 - multiple, on single line, 64
 - rules for, 61
 - suites, 62–63
- static data, 521
- static dispatch, 992
- static members, 521, 539
- static methods, 542–544
- `staticmethod()` built-in function, 543
- `stderr`, 337
- `stdin`, 337
- `stdout`, 337
- stock quote server, 986–989, 1001–1002
- `StopIteration` exception, 298, 301, 310
- storage (attributes), 604
- storage model, 112–113, 116
- `str()` built-in function, 54, 102–104, 136, 137, 165, 166, 260
- `str()` factory function, 186–187
- stride indices, 162–165
- string format operator, 178–181, 208, 1029, 1030
- string templates, 182
- `StringIO` module, 351, 352
- strings, 38–39, 168–170
 - accessing values of, 169
 - assigning, 168–169
 - built-in functions, 184–187, 242–245
 - built-in methods, 188–191, 242–245, 1030–1033
 - `chr()`, 187
 - concatenation, 39, 176–177
 - creating, 168–169
 - debugging, 181
 - ending of, 208
 - `enumerate()`, 185–186
 - exceptions, 386
 - identifiers, 67–68
 - immutability of, 194–196
 - `len()`, 185

- matching, 701–703
 - matching beginning/end of, 678–679, 693
 - matching more than one, 689
 - matching within, 688
 - max(), 185
 - membership, 172–175
 - min(), 185
 - no char type for, 207
 - non-NUL/ '0' ending of, 192, 208
 - NUL characters, 192, 208
 - operators, 170–184, 242–245
 - ord(), 187
 - quotation mark delimited, 207
 - raw strings, 168, 182–184, 208, 698
 - raw_input(), 186
 - regular expressions, 206–207, 678–679, 688, 689, 693, 701–703
 - removing, 169–170
 - repetition of, 39, 177–178
 - sequence operators, 170–178, 242–245
 - slices of, 170–172
 - special/control characters, 192–193, 208
 - standard library modules for, 205–207
 - standard type operators, 168
 - str(), 186–187
 - string templates, 182
 - string-only operators, 178–184
 - summary of, 207–208
 - tables of, 242–245
 - triple quotes, 38, 193–194, 208
 - unichr(), 187
 - Unicode, 184, 186–187, 196–205, 892–893
 - updating, 169
 - zip(), 186
- Structured Query Language. *See* SQL
- “stubs,” 974
- style guidelines, 69–75, 513
- sub() function/method, 685, 694–695
- subclasses/subclassing
- creating, 512–513, 546, 805–807
 - derivation, 545–546
 - multiple inheritance, 553–558
 - standard types, 551–553
 - using, 513
- subn() function/method, 694–695
- subprocess module, 660–662
- subsets, 277
- substrings, accessing, 169
- subtraction (-) operator, 35
- suites, 41–42, 62–64, 292
- sum() built-in function, 166, 167, 218
- summation functions, 807–810
- super() built-in function, 562–563
- supersets, 277
- SWIG (Simplified Wrapper and Interface Generator), 981
- swing, 851
- Swing GUI development, 1003–1006
- switch statement, 294–295
- symmetric difference (^) operator, 278
- symmetric difference update (^=) operator, 279
- syntax
- comments, 62
 - continuation of lines, 62
 - decorators, 422
 - dictionaries, 254
 - ease of reading, 9
 - errors, 47, 360
 - identifiers, 68
 - mandatory symbols, 9
 - statement, 61
- SyntaxError exception, 362, 392, 430
- sys module, 337–339, 403–404, 479, 491
- sys.argv, 338–339
- sys.exit function, 791
- sys.exit() function, 663–665
- sys.exitfunc() function, 665–666
- SystemError exception, 392
- SystemExit, 371, 391, 394, 663–664, 791
- ## T
- tables, 253, 920, 921
- tabs, 63
- Tcl, 24–26
- TCP (Transmission Control Protocol), 856
- clients, 723–726, 735–736, 738–740
 - servers, 720–723, 725–726, 732–738, 740
- TCP/IP (Transmission Control Protocol/Internet Protocol), 717
- tell() built-in method, 331, 334, 336, 641
- telnet protocol, 858
- tempfile module, 351, 352
- templates (string), 182
- terminating execution, 663–666
- os._exit(), 666
 - os.kill(), 666

- terminating execution (*continued*)
 - `sys.exit()`, 663–665
 - `sys.exitfunc()`, 665–666
 - `SystemExit`, 663–664
- ternary operator, 295–296
- testing, 74–75, 437–439, 976–979
- text file manipulation example programs, 79–85
- Text widget (Tk), 825
- thread module, 794–799, 814
- threading module, 794, 800–808, 814
 - daemon threads, 801
 - examples, 802–813
 - Fibonacci/factorial/summation functions, 807–810
 - objects, 800
 - other functions for, 809–810
 - producer-consumer problem, 810–813
 - Queue module, 810–813
 - thread class, 801–802
 - thread module vs., 794
- threads, 801–802
 - accessing, 792
 - creating, 802–805
 - definition of, 789–790
 - examples, 792–793
 - exiting, 791
 - global interpreter lock, 790–791
 - modules for, 794–813
 - passing in callable class instance, 803–805
 - passing in function, 802–803
 - safety, 925
 - spawning, 721
- TIDE + IDEStudio (Tix Integrated Development Environment), 849
- tilde (`~`) expansion, 352
- “timestamp decoration” example, 425–426
- Tix (Tk Interface eXtensions), 840, 842–843, 849
- Tk, 24, 819–820, 823–826, 849
- Tkinter
 - about, 819–820, 849
 - adding Tk to applications, 821–822
 - examples, 826–840
 - file system traversal GUI example, 834–840
 - installing/working with, 820–821
 - partial function application example, 831–834
 - as Tk port, 24
 - top-level window, 824
- TkZinc, 849
- Tool Command Language (Tcl), 819
- Toplevel widget (Tk), 824, 825
- “traceback” notice, 361
- traceback objects, 94
- transferring files
 - about, 748
 - client example, 760–765
 - documentation, 765–766
 - examples, 752–754, 759–765
 - FTP, 748–750
 - `ftplib.FTP` class methods, 750–752
 - interactive example, 759–760
 - Python FTP support, 750
 - typical FTP clients, 755
- Transmission Control Protocol. *See* TCP
- Transmission Control Protocol/Internet Protocol. *See* TCP/IP
- triple quotes, 38, 193–194, 208
- true division (`/`), 130, 131, 569
- `truncate()` built-in method, 332, 334, 336
- try** statement, 368–369, 378–379
- try-except** statement, 47, 364–365, 372, 378–379, 382, 383
- try-except-else-finally** statement, 381–382
- try-finally** statement, 379–383
- `tuple()` built-in/factory function, 165, 166, 218–219
- tuples, 39–40, 231–232
 - assessing values of, 231–232
 - assigning, 66–67, 231
 - built-in functions, 165, 166, 232–234, 242–245
 - built-in methods, 242–245
 - concatenation of, 233
 - creating, 231, 233
 - default collection type, 235–236
 - dictionary keys for, 237–238
 - flexibility of, 234–235
 - immutability of, 234–235
 - as keys, 269
 - lists vs., 237–238
 - membership, 233
 - non-keyword variable-length arguments, 433–434
 - operators, 232–234, 242–245
 - removing tuple elements/tuples, 232
 - repetition, 233
 - sequence operators, 232–233, 242–245

- single element, 236–237
 - slicing, 233
 - special features of, 234–238
 - standard operators, 232–233
 - tables of, 242–245
 - updating, 232
 - twisted framework, 737–740
 - twisted reactor TCP, 737–738
 - type(s). *See also* standard types
 - built-in, 91
 - built-in functions, 54, 92, 102–103, 105–110, 136, 137
 - categorizing standard, 111–116
 - function to check, 106–107
 - internal, 93–95
 - module, 615–617
 - None, 92–93
 - object, 90–92
 - return values and function, 410–412
 - Unicode, 204
 - unsupported, 116–117
 - “wrapping a type,” 587
 - type() built-in functions, 54, 102–103
 - checking types with, 105–109
 - finding object types with, 92
 - numbers, 136, 137
 - table of, 110
 - type() factory functions, 102, 111, 260
 - TypeError exception, 187, 264, 374, 375, 393, 434, 522
 - typing, dynamic, 76
- U**
- ‘U’ access mode, 326, 328–329
 - UDFs. *See* functions (user-defined)
 - UDMs. *See* methods (user-defined)
 - UDP (User Datagram Protocol), 717–718
 - clients, 728–729
 - servers, 726–730
 - unbinding, 480
 - unbound methods, 511, 522, 541–542
 - UnboundLocalError, 455
 - underscores (`_`, `__`), 69
 - unhandled exceptions, 365
 - unicr() function, 187, 203
 - Unicode
 - CGI, 892–893
 - codecs, 199
 - coercion for, 204
 - common codecs/encodings, 205
 - decoding, 200–201
 - definition of, 197–198
 - encoding, 200–201
 - exceptions, 204
 - ordinals, 204
 - regular expression engine, 204
 - rules for, 201–202
 - source code encoding, 497
 - standard encodings, 204, 205
 - strings, 184, 186–187, 196–205, 892–893
 - terminology for, 197
 - using, 198–199, 201–203
 - unicode() built-in function, 203
 - unicode() factory function, 186–187
 - UnicodeError exception, 204
 - Uniform Resource Locators. *See* URLs
 - union (`|`) operator, 277, 279. *See also* update (`|=`) operator
 - Universal NEWLINE Support (UNS), 326, 328–329
 - Unix, 697–698
 - availability of Python on, 12
 - compiling extensions on, 964
 - IDE for, 18–19
 - installing Python on, 12
 - Internet systems running, 858
 - line separators, 333
 - multithreaded programming, 792
 - running Python on, 14, 16–19
 - shell scripting languages, 7
 - UNS. *See* Universal NEWLINE Support
 - update model, 113–114, 116
 - update (`|=`) operator, 279
 - updating
 - dictionaries, 257–258
 - lists, 210
 - numbers, 122
 - rows, 920, 922
 - set types, 275
 - strings, 169
 - tuples, 232
 - uploading files, 894
 - upward propagation (of exceptions), 365
 - urllib module, 862–866
 - urllib2 module, 866–869
 - urlparse module, 861–862

- URLs (Uniform Resource Locators), 856, 859–861
 - Usenet, 756
 - User Datagram Protocol. *See* UDP
 - user input, 875–877, 886–892
 - user interface, 920
 - user-defined functions. *See* functions (user-defined)
 - user-defined methods. *See* methods (user-defined) (UDMs)
 - UserDict module, 615, 617
 - UserList module, 615, 617
 - users (application), 401–402
 - UserString module, 615, 617
 - UTF-8 encoding, 199, 205
 - UTF-16 encoding, 199, 205
- V**
- value(s)
 - accessing dictionary, 255–257
 - Boolean, 93
 - comparing dictionary, 262
 - comparison of object, 93, 96–97, 108
 - list, 209
 - object, 90, 93, 96–97, 108
 - set type, 275
 - string, 169
 - tuple, 231–232
 - ValueError exception, 374, 375, 393
 - values() built-in method, 265, 266
 - van Rossum, Guido, 6, 18, 23, 53, 295, 544
 - variable-length arguments (functions), 433–439
 - variables
 - accessing module, 52–53
 - assignment of, 37, 64–67
 - declarations for, 72, 75
 - global vs. local, 453–455
 - multiple assignment, 66
 - “multiple” assignment, 66–67
 - naming, 69
 - scope, 453–466
 - underscores in naming, 69
 - using local to substitute for module attributes, 81
 - vars() built-in function, 563
 - versions of Python, 12
 - VisualBasic.NET, 27, 969
- W**
- warning(s), 393, 489–490, 927
 - Watters, Aaron, 923
 - Web addresses, 860
 - Web applications, 910–911
 - Web browsers, 755, 855, 856
 - Web clients, 855–857, 859–875
 - Web crawlers, 860–875
 - Web pages, 432, 881–886
 - Web programming, 855–912. *See also* CGI (Common Gateway Interface); Internet client programming
 - advanced Web clients, 869–870
 - cookies, 856, 895–906
 - crawlers, 860–875
 - fully interactive sites, 886–892
 - Internet architecture, 856–859
 - Internet programming vs., 858–859
 - multipart form submission/file uploading, 894
 - multivalued fields, 895
 - related modules, 909–912
 - Unicode, 202, 892–893
 - urllib module, 862–866
 - urllib2 module, 866–869
 - urlparse module, 861–862
 - URLs, 859–861
 - user input/error processing, 886–892
 - Web clients, 859–875
- Web servers
- about, 855–858
 - CGI, 875–878, 906–909
 - as common software server, 713
 - exception handling, 401–402
 - processing client data with, 875–878
 - related modules, 911
 - setting up, 878–879
- Web services, 985–989, 1000–1002
- Web sites, fully interactive, 886–892
- Web surfing, 855–856, 859–869, 886–892
- while** statement (loops), 42, 296–298
- break** statement, 304–305
 - continue** statement, 305–306
 - counting loops, 297
 - else** statement, 307–308
 - infinite loops, 297–298
 - and **pass** statement, 306–307
 - syntax for, 296–297
- while_suite, 42
- who command (Unix), 697–698
- widgets, 822–826

Win32 platforms/systems
 availability of Python on, 12
 compiling extensions on, 964
 multithreaded programming, 792
win32ui, 851
Windows clients, 713
windows servers, 713
Windows/DOS platforms
 IDE for, 19–21
 installing Python on, 12–13
 line separators, 333
 running Python on, 14–17, 19–21
with statement, 382–384
 –without-universal-newlines switch, 329
with_suite (context object), 385
Word (Microsoft), 993–994
word boundaries, matching, 678–679, 693
wrapping/wrappers
 adding `initModule()` module initializer
 function, 973–974
 adding `MethodDef ModuleMethods []` array/
 table, 973
 an object with enhancements, 594–595
 any object example, 588–593
 built-in function (exceptions), 365–368
 compilation of, 974–975
 PyObject* function, 969–973
 try-except, 364–365

 using “stubs”/dummy functions, 974
 “wrapping a type,” 587
`write()` built-in method, 330, 336, 594–595
`writelines()` built-in method, 330, 336
wxGlade, 849
wxPython, 840, 843–846, 849
wxWidgets, 840, 843–846, 849

X

X Window System, 821
XML processing, 910–911
`xrange()` built-in function, 95, 303
`xreadlines()` method, 330, 335

Y

Yahoo! Finance Stock Quote server, 986–989,
 1001–1002
yield statement, 468, 470
yielding (threads), 789

Z

\Z (special character), 678
ZeroDivisionError exception, 362, 391
`zip()` built-in function, 167, 186, 218, 303, 304
zip files, 490
zipfile module, 351, 352
zipimport, 500, 501
zlib module, 351, 352