Thomas Finley, tfinley@gmail.com, http://tfinley.net/

■ Built-in Types

Numerics int, float, long, complex

`int()`, `float()`, `long()`, `comple()` zero
`int`(x), `float`(x), `long`(x), `complex`(x) parse string or cast number
`int`(x, b), `long`(x, b) parse b-base string x
`complex`($[a, [b]]$) complex number $a + bj$

n.`conjugate`() complex conjugate
n.`numerator` the a in $\frac{a}{b}$
n.`denominator` the b in $\frac{a}{b}$
n.`real` the a in $a + bj$
n.`imag` the b in $a + bj$
n.`bit_length`() bits to hold n (int/long)
n.`as_integer_ratio`() (a, b) with $n = \frac{a}{b}$

Operators: the uniquitous arithmetic `+,-,*,/,\%`, bitwise `|,^,&,<<,>>,~`, as well as `//` (floored quotient) and `**` (power).

Booleans bool

Booleans can be treated like numbers, where `False`==0 and `True`==1.

`bool`($[x]$) `False` iff x==`None`, `False`, 0, empty

Mutable Sequences

`append`(x) add single item x to end
`extend`(it) add it's items to end
`count`(x) number of items equal to x
`index`(x,$[i$,$[j]]$) lowest index $k \in [i, j)$ for x
`insert`(i, x) insert so item at index i is x
`pop`($[i]$) remove, return last item or item i
`remove`(x) delete first element equal to x
`reverse`() reverse sequence in place
`sort`($[cmp$, $[key$, $[reverse]]]$) sort in place
Both `index` and `remove` raise `ValueError` if element unfound.

Strings str, unicode

Both types derived from `basestring`.

Below, st is the string instance. For methods, when b, e appear, assume the operation is limited to the substring from b inclusive to e exclusive.

`capitalize`() only first char uppercase
`center`(w, $[fchar]$) $fchar$ padded to w len
`count`(s, $[b$, $[e]]$) num of non-overlap s
`decode`($[enc$, $[err]]$) decode encoded string
`encode`($[enc$, $[err]]$) encode string
  enc often `utf8`; see codecs
  err `strict`, raise `UnicodeError`
    `ignore`, `xmlcharrefreplace`
    `replace`, `backslashreplace`
`endswith`(s, $[b$, $[e]]$) suffix is s
`expandtabs`($[tabsize]$) subs tab w/spaces
`find`(s, $[b$, $[e]]$) position of s in st, or -1
`format`($*a, **kw$) new {} style formatting
`index`(s,$[b$,$[e]]$) if unfound raise `ValueError`
`isalnum`() consists of alphanumerics
`isalpha`() consists of alphabetics
`isdigit`() consists of digits
`islower`() has alphabetics, all lowercase
`isspace`() consists of whitespace
`istitle`() has alphabetics, titlecased
`isupper`() has alphabetics, all uppercase
`join`(it) it's items delimited with string
`ljust`(w, $[fchar]$) right-padded to w-len
`lower`() lowercase
`lstrip`($[chars]$) remove chars from start
`partition`(sep) (pre, sep, suf) or $(st$,″,″$)$
`replace`(old, new, $[c]$) replace all or c old
`rfind`(s, $[b$, $[e]]$) like `find` but from end
`rindex`(s, $[b$, $[e]]$) like `index` but from end
`rjust`(w, $[fchar]$) left-padded to w-len
`rpartition`(sep) like `partition` from end
`rsplit`($[s$, $[m]]$) like `split` but from end
`rstrip`($[cs]$) remove chars from end
`split`($[s$, $[m]]$) s-delimited substrings
  s if `None`, whitesp. delim., no ″ result
  m get m substrings, plus remainder
`splitlines`($[keepends]$) list of lines,
`startswith`(p, $[b$, $[e]]$) prefix is p
`strip`($[cs]$) remove chars from start/end
`swapcase`() lower to upper, upper to lower
`title`() titlecased (first letters upcased)
`translate`(tbl, $[delchrs]$) who uses this?!
`upper`() uppercase
`zfill`($width$) leftfills with `0`, handles ±

unicode

`isnumeric`() unicode numeric test
`isdecimal`() unciode decimal test

Sets set, frozenset

In this method table, s is our set instance. The input t can be any iterable. Editing methods (starting with `update`) exist for `set`, not `frozenset`.

`isdisjoint`(t) if no item also in t
`issubset`(t) if all items also in t
`issuperset`(t) if all of t's items also in s
`union`(t, ...) set with the items in t or s
`intersection`(t, ...) ...in both t and s
`difference`(t, ...) ...in s but not t
`symmetric_difference`(t) either, not both
`copy`() give shallow copy
`update`(t, ...) add t's items
`intersection_update`(t, ...)
  keep only items also in t
`difference_update`(t, ...)
  discard any items also in t
`symmetric_difference_update`(t)
  keep items in either set, but not both
`add`(x) add x to set
`remove`(x) remove x, `KeyError` if $x \notin s$
`discard`(x) same, doesn't raise `KeyError`
`pop`() remove, return arbitrary item
  raises `KeyError` if empty
`clear`() remove all items

When using operators, unlike the methods, t must be another set. Inplace operators (`|=`, `&=`, `-=`, `^=`) exist for `set`s.

s<=t	$s \subseteq t$	s>t	$s \supset t$	s-t	$s \setminus t$
s>=t	$s \supseteq t$	s|t	$s \cup t$	s^t	$s \cup t \setminus s \cap t$
s<t	$s \subset t$	s&t	$s \cap t$		

Maps dict

In this method table, d is our map instance

`iter`(d) iterator over keys
`clear`() empties mapping
`copy`() returns copy of map
ℂ.`fromkeys`(seq, $[val]$) make map d so
  $d[seq[i]]$=$val[i]$, or ==`None`
`get`(k, $[def]$) $d[k]$ if exists, or def/`None`
`has_key`(k) if $d[k]$ exists, k `in` d preferred
`items`() list of key, value tuples
`keys`() list of keys
`values`() list of values
`iteritems`(), `iterkeys`(), `itervalues`()
  same, but iterators, doesn't copy a list
`viewitems`(), `viewkeys`(), `viewvalues`()
  same, but dynamic "view" objects
`pop`(k, $[def]$) remove, return $d[k]$ (or def)
  `KeyError` if not default, k `not in` d
`popitem`() remove, return arb. key, value
  raises `KeyError` if empty
`setdefault`(k, $[def]$) if $k \notin d$, $d[k] = def$
  returns $d[k]$
`update`($[o]$) add o's mappings

File Objects

Aside from actual files, file-like objects implementing a subset of methods/members are commonly used in Python code:

`close`() close the file
`flush`() flush file's internal buffer
`fileno`() integer file descriptor
`isatty`() file connected to tty-like device
`next`() next input line
`read`($[s]$) read s bytes, or till EOF
`readline`($[s]$) read s bytes, or till EOL
`readlines`($[s]$) read about s byes of lines
`seek`(o, $[w]$) set file position o bytes from
  start/current/end ($w = 0/1/2$)
`tell`() file's current position
`truncate`($[s]$) truncate to current pos
`write`(str) write str to file
`writelines`(it) write strs (no `\n` added)
`closed` if file closed
`encoding` encoding when writing `unicode`
`errors` unicode error handler
`mode` I/O mode for file
`name` filename
`newlines` detected newline if `U` in mode
`softspace` next `print` should add space

■ Special Method Names

Some behaviors (e.g., acting like number/sequence/map) implementable with special methods. *Every* method is prefixed/suffixed with `__`, and accept $self$ as first argument (except `__new__`), so these are omitted, so `foo`(a) is really `__foo__`($self, a$).

`new`(cls, [..]) new instance of cls (usually)
`init`([..]) initialize instance
`del`() called when about to be destroyed
`repr`() "formal" representation
`str`() "informal" representation
`lt`/`le`/`eq`/`ne`/`gt`/`ge`($other$) rich compare
`cmp`($other$) alternative; neg if $self < other$
`hash`() integer used in hashes
`nonzero`() if implemented, used in `bool`()
`unicode`() should return unicode
`getattr`(a) called *only* if $o.a$ unfound
`setattr`(a, v) called on $o.a = v$ attempt
`delattr`(a) called on `del`$o.a$ attempt
`getattribute`(a) called on $o.a$ attempt

Emulating Functions

`call`($[args]$) called when used as function

Emulating Containers

`len`() length of object
`getitem`(k) k is int/slice (seq), key (map)
`setitem`(k, v) like `getitem`; sets $o[k] = v$
`delitem`(k) removes item at k
  should raise `Type`/`Index`/`KeyError` if k bad type/seq index/unfound map index
`iter`() iterator on items (keys for map)
`reversed`() same, but reverse iterator
`contains`($item$) supports `in`/`not in` test

Emulating Numerics

`add`/`sub`/`mul`/`div`/`truediv`/`floordiv`/`mod`
`divmod`/`pow`/`lshift`/`rshift`/`and`/`xor`/`or`(b)
  supports $a + b$, etc.; `NotImplemented`
  raised for unsupported types
`radd`/`rsub`/$\cdots$(b) supports $b + a$, etc.
  called if b's non-`r` operator inapplicable
`iadd`/`isub`/$\cdots$(b) supports a+=b, etc.
`pow`(y, $[z]$) supporting `pow` builtin function
`neg`/`pos`/`abs`/`invert`() supports unary
  operators `-`, `+`, `abs`(), `~`
`complex`/`int`/`long`/`float`() obj as number
`oct`/`hex`() string octal/hexidecimal repr
`index`() integer, if used as index
`coerce`(b) (a, b) as common num. type

Implement Context Manager

`enter`() enter context, `as` target gets retval
`exit`($exc\_type, exc\_value, traceback$)
  args not `None` when exception raised
  `True` retval suppresses; never reraise

■ Built-in Functions

`abs`(x) absolute value of x
`all`(it) every x in it has `bool`(x)==`True`
`any`(it) any x in it has `bool`(x)==`True`
`bin`(x) format number as binary
`callable`(obj) if obj callable
`chr`(i) ASCII to character
`classmethod`($func$) first arg to decorated
  method is class instead of instance
`cmp`(x, y) negative if $x < y$
`compile`($src, fname, mode$, [..])
  src source code to compile
  $fname$ fake filename
  $mode$ ′[`exec`|`eval`|`single`]′ if src
  block/expression/interactive
`delattr`(obj, a) like `del` $obj.a$
`dir`() current scope's variable names
`dir`(obj) obj's attribute names
`divmod`(a, b) (quotient, remainder) of $\frac{a}{b}$
`enumerate`(it, [s=0]) yield $(index + s, x)$
`eval`($expr$, $[glob$, $[loc]]$) interpret Python
  expression with scope variables
`execfile`($fname$, $[glob$, $[loc]]$)
`file`(...) `open` preferred
`filter`($func, it$) x in it with true $func(x)$
`format`(val, $[fspec]$) format val by
  format specification mini-language
`getattr`(obj,a,$[def]$) obj.a if exists, or def
`globals`() name-value dict of global vars
`hasattr`(obj, a) if $obj.a$ exists
`hash`(obj) hashcode of obj
`help`($[obj]$) launch pydoc
`hex`(x) format number as hexidecimal
`id`(obj) object identifier (C pointer val)
`input`($[prompt]$) evaluate stdin input
`isinstance`(obj, cls) obj instance of cls
`issubclass`(cls, sup) cls subclass of sup
`iter`(obj) iterator over iterable obj
  over items (seqs) keys (maps) lines (files)
`iter`(f, s) yield $f()$ till s returned
`len`(s) sequence length
`locals`() name-value dict of local vars
`map`($func, it$, ...) list of $func(x$, ...$)$
`max`(it, $[key]$) x in it with x/$key(x)$ max
`max`($arg1, arg2$, ..., $[key]$)
  the arg with arg/$key(arg)$ max
`min`(...) analogous to `max`, but minimum
`next`(it, $[def]$) next item, or def if done
`oct`(x) format number as octal
`open`($fname$, $[mode$, $[bufsize]]$)
  $fname$ file name
  $mode$ `rwab+U` (read, write, append, binary, read+write, univ. newline)
  $bufsize$ 0 none, 1 line buf., >1 size
`ord`(c) ASCII code for character
`pow`(x, y, $[z]$) x^y mod z
`print`($[obj, \ldots]$, $[sep = ' ']$, $[end = '\backslash n']$, $[file = sys.stdout]$)
  write sep-delimited objs, end, to $file$
`property`($[fget$, $[fset$, $[fdel$, $[doc]]]]$)
  as member a of obj, ops on $obj.a$ handled by appropriate method
`range`($[b], e$, $[s]$) list [b,b+s,...] to e noninc.
`raw_input`($[prompt]$) terminal input
`reduce`($func, it$, $[init]$) $func$ 1st arg $init$
`reload`($module$) re-read module code
`repr`(obj) eval-able string for obj
`reversed`(seq) reverse iterator
`round`(x, $[n$=0$]$) round x to n places
`setattr`(obj, a, v) like $obj.a = v$
`sorted`(it, $[cmp$, $[key$, $[reverse]]]$)
  sorted list of items x in it
  cmp define ordering like `cmp`
  key order $key(x)$ instead of x in it
  $reverse$ if true, decending order
`staticmethod`($func$) first arg to decorated method is not instance
`sum`(it, $[s = 0]$) sum of x in it, plus s
`super`($type$, $[obj\_or\_type]$) proxy object
  delegates calls to parent/sibling of $type$
`type`(obj) get obj's type
`type`($name, bases, dict$) make new type
`unichr`(i) unicode character given
`vars`($[obj]$) obj.`__dict__` if obj else `locals`()
`xrange`($[b], e$, $[s]$) nonmaterialized `range`
`zip`($[it$, ...$]$) list of tuples, i^{th} tuple
  contains each iterator's i^{th} item

■ Regular Expressions, re

Module has these flag constants and functions. In these, p is a pattern, s is the string we search, f are or-ed flags, c is number of ops.

`IGNORECASE/I` case insensitive-matching
`LOCALE/L` `\w\W\b\B\s\S` locale dependent
`MULTILINE/M` `^$` match begin/end lines
`DOTALL/S` . matches `\n`
`UNICODE/U` Uni. charprop for `\w\W\b\B\d\D\s\S`
`VERBOSE/X` space ignored, `#` comments
`compile`(p, $[f]$) `RegexObject` on p, flags f
`search`(p, s, $[f]$) find match in s
`match`(p, s, $[f]$) match start of s
`split`(p, s, $[c$, $[f]]$) p splits s up to c times
  capturing groups in p included in list
`findall`(p, s, $[f]$) s substrs matching p
  if one group in p, item is group string
  if multiple groups, item is tuple of groups
`finditer`(p, s, $[f]$) yields `MatchObjects`
`sub`(p, r, s, $[c$, $[f]]$) matches replaced with r
  if r func, gets `MatchObject`, returns string
`subn`(p, r, s, $[c$, $[f]]$) same, but returns tuple
  (`replaced_string, numsubs`)
`escape`(s) escape out RE special strings

RegexObject Methods/Attrs

`search`/`match`/`findall`/`finditer`(s,$[b$,$[e]]$)
`sub`/`subn`(r, s, $[c]$) similar to module funcs
`split`(s, $[c]$) but compiled, and b,e substr
`flags` which flags it was compiled with
`groups` number of capturing groups
`groupindex` `(?P<id>)` names to nums
`pattern` the pattern that was compiled

MatchObject Methods/Attrs

`expand`(t) backslash subs on t using match
`group`($[group1$, ...$]$) get indicated group(s)
`groups`($[def$=`None`$]$) get all groups,
  replacing empties with def
`groupdict`($[def]$) map named groups to grp
`start`($[grp]$) start index of match/group
`end`($[grp]$) end index of match/group
`span`($[grp]$) tuple of start, end
`pos` b passed to `RegexObject` method
`endpos` e passed to `RegexObject` method
`lastindex` index of last matched group
`lastgroup` name of last matched group
`re` `RegexObject` that produced this
`string` string this match came from

RE Pattern Syntax

`.` any char but `\n` `\b` word boundary
`^` match str start `\d` mass decimal
`$` match str end `\s` whitespace char
`\` escape spec. chars `\w` alphanum or _
`\n` match group n `\Z` match str end
`\A` match str start

`\B`,`\D`,`\S`,`\W` compliment of `\b`,`\d`,`\s`,`\w`
`\a`,`\b`,`\f`,`\n`,`\r`,`\t`,`\v`,`\x`,`\\` reg. escapes
`?` match $0, 1$ reps of preceding
`*` match ≥ 0 reps of preceding
`+` match ≥ 1 reps of preceding
`{m}` match m reps of preceding
`{m,n}` match m to n reps of preceding
`*?`, `+?`, `??`, `{m,n}?` non-greedy variants
`|` for $A|B$ match A or B
`[`$chars$`]` match set of chars
`[^`$chars$`]` match anything but chars
`(...)` beginning and end of group
`(?iLmsux)` set flags `ILMSUX` in the pattern
`(?:...)` non-grouping regular parens
`(?P<`n`>...)` named group n
`(?P=`n`)` match previously named group n
`(?#...)` a comment, contents ignored
`(?=...)` lookahead, match, don't consume
`(?!...)` negative lookahead assertion
`(?<=...)` positive lookbehind assertion
`(?<!...)` negative lookbehind assertion
`(?(`id/$name$`)`y`|`n`)`
  match y if group id/$name$ exists, else n

■ datetime

`MINYEAR` smallest allowed year, 1
`MAXYEAR` largest allowed year, 9999

Shared Methods/Members

ℂ.`min` most negative/earliest instance
ℂ.`max` most positive/earliest instance
ℂ.`resolution` smallest possible difference
`initarg` init args usually members, e.g., `date` have `year`, `month`, `day` members

timedelta

ℂ($days, seconds, microseconds$,
  $milliseconds, minutes, hours, weeks$)
  all args optional, can be float, default 0
  only first three args become members
`total_seconds`() total seconds in delta

datetime

ℂ($year, month, day$, $[hour, minute, second$,

microsecond,*tzinfo*]) date/time comb.
ℂ.`today`() current local date/time
ℂ.`now`([*tz*]) similar, but with `tzinfo`
ℂ.`utcnow`() current UTC date/time
ℂ.`fromtimestamp`(*ts*, [*tz*])
local date/time from POSIX timestamp
ℂ.`utcfromtimestamp`(*ts*)
UTC date/time from POSIX timestamp
ℂ.`fromordinal`(*ord*) *ord*==1 is 1-Jan-1
ℂ.`combine`(*d*, *t*) mix `date` and `time` inst.
ℂ.`strptime`(*s*, *f*) parse *s* formatted as *f*
`date`() `date` instance with the same date
`time`() `time` inst. time, `tzinfo`==`None`
`timetz`() `time` inst. with same `tzinfo`
`replace`(*year*, *month*, *day*, *hour*, *minute*, *second*, *microsecond*, *tzinfo*)
new instance, indicated fields replaced
`astimezone`(*tz*) adjusted to timezome
`utcoffset`()/`dst`()/`tzname`()
calls same method on `tzinfo` with `self`
`timetuple`() convert to `time.struct_time`
`utctimetuple`() tzone adjusted to UTC
`toordinal`() day num if 1-Jan-1 is day 1
`weekday`() weekday, Monday==0, etc.
`isoweekday`() weekday, Monday==1, etc.
`isocalendar`() ISO year,weeknum,weekday
`isoformat`([*sep*]) msecs if ≠0, UTC offset
`YYYY-MM-DD`[*sep*]`HH:MM:SS[.mmmmmm][+HH:MM]`
`ctime`() e.g., `Tue Jun 22 21:45:35 2010`
`strftime`(*f*) string formatted by *f*

`date`

ℂ(*year*, *month*, *day*) init with date
ℂ.`today`() current local date
ℂ.`fromtimestamp`(*ts*) from POSIX t.s.
ℂ.`fromordinal`(*ord*) *ord*==1 is 1-Jan-1
`replace`(*year*, *month*, *day*) get repl. date
`timetuple`() convert to `time.struct_time`
`toordinal`() day num if 1-Jan-1 is day 1
`weekday`() weekday, Monday==0, etc.
`isoweekday`() weekday, Monday==1, etc.
`isocalendar`() ISO year,weeknum,weekday
`isoformat`() in format `YYYY-MM-DD`
`ctime`() e.g., `Tue Jun 22 00:00:00 2010`
`strftime`(*f*) date string formatted by *f*

`time`

ℂ(*hour*,*minute*,*second*,*microsecond*,*tzinfo*)
all args optional, default to 0/`None`
`replace`(*hour*,*minute*,*second*,*microsecond*, *tzinfo*) new inst. with fields replaced
`isoformat`() msecs if ≠0, UTC offset if tz
`HH:MM:SS[.mmmmmm][+HH:MM]`
`strftime`(*f*) time string formatted by *f*
`utcoffset`()/`dst`()/`tzname`()
calls same method on `tzinfo` with `self`

`tzinfo`

`utcoffset`(*dt*) delta from UTC inc. DST
`dst`(*dt*) DST timedelta offset (e.g. 0/1 hrs)
`tzname`(*dt*) description of timezone
`fromutc`(*dt*) UTC time to local time
subclasses rarely override default

`strftime`/`strptime` Format Ops

`a`	abr wkday name	`p`	`AM`, `PM`
`A`	full wkday name	`S`	sec as `00`-`61`
`b`	abr month name	`U`	week num `00`-`53`
`B`	full month name		wk 0 before 1st Su
`c`	date & time repr	`w`	wkdy as `0`-`6` (Su-Sa)
`d`	day as `01`-`31`	`W`	∼`U`, but from Mo
`f`	zero padded μsec	`x`	date repr
`H`	hour as `00`-`23`	`X`	time repr
`I`	hour as `01`-`12`	`y`	year as $00-99$
`j`	yr day as `001`-`366`	`Y`	year with century
`m`	month as `01`-`12`	`z`	UTC off as ±`HHMM`
`M`	min as `00`-`59`	`Z`	timezone name

■ `collections`

`Counter`

Maps elements to count, like a multiset. Is `dict` subclass, but `fromkeys` inapplicable.
ℂ() empty counter
ℂ(*it*) count elements in *it*
ℂ(*map*) element maps to counts
`elements`() iter, elems repeat count times
`most_common`([*n*]) *n* top elem/count pairs
`update`([*it_or_map*]) increments counts
`subtract`([*it_or_map*]) decrements counts
ℙ+ℙ add counters together
ℙ-ℙ subtract, keeps only positives
ℙ&ℙ intersect, keeps minimum count
ℙ|ℙ union, keeps maximum count

`deque`

Generalizes stacks and queues.
ℂ([*it*,[*maxlen*]]) inits with *it*'s items
len capped at *maxlen*, items discarded from opposite end
`append`(*x*) add *x* to right side
`clear`() empty the deque
`count`(*x*) count items equal to *x*
`extend`(*it*) add *it*'s items to right side
`pop`() remove, return rightmost item
`appendleft`/`extendleft`/`popleft`
similar, but ops on the left side
`remove`(*x*) remove first *x*, or `ValueError`
`reverse`() reverse element order in place
`rotate`(*n*) rotate *n* step right (left if neg)
`maxlen` max deque size, `None` if unbound

`defaultdict`

ℂ([*fact*,[..]]) like `dict`; missing keys get *fact*()
`default_factory` callable for default vals

`namedtuple`

Fixed len tuple type with named fields.
ℂ(*name*, *fields*, [*verbose*, [*rename*]])
name the type's name
fields space/comma delim `str`, or seq
verbose if `True`, prints class def
rename bad fieldnames replaced with _*d* positional names for index *d*
`_make`(*it*) make instance from sequence
`_asdict`() map of field names to values
`_replace`(*kwargs*) copy tuple, use k.w. args to replace values
`_fields` the tuple of string fieldnames

`OrderedDict`

Like `dict`, but remembers insertion order.
ℂ(..) acts like `dict` constructor
`popitem`(*last*=`True`) remove and return key/value pair, LIFO if *last* else FIFO

Abstract Base Classes

Subclasses implement 𝔸bstract methods, 𝕄ixin methods provided.

`Container`	𝔸:`__contains__`
`Hashable`	𝔸:`__hash__`
`Iterable`	𝔸:`__iter__`
`Iterator`(`Iterable`)	𝔸:`next`, 𝕄:`__iter__`
`Sized`	𝔸:`__len__`
`Callable`	𝔸:`__call__`
`Sequence`(`Sized`, `Iterable`, `Container`)	𝔸:`__getitem__`, 𝕄:`__contains__`, `__iter__`, `__reversed__`, `index`, `count`
`MutableSequence`(`Sequence`)	𝔸:`__setitem__`, `__delitem__`, `insert` 𝕄 :`append`, `reverse`, `extend`, `pop`, `remove`, `__iadd__`
`Set`(`Sized`, `Iterable`, `Container`)	𝕄 :`__le`/`lt`/`eq`/`ne`/`gt`/`ge`/`and`/`or`/`sub`/`xor__`
`MutableSet`(`Set`)	𝔸 :`add`, `discard`
`Mapping`(`Sized`, `Iterable`, `Container`)	𝔸 :`__getitem__`, 𝕄:`__contains__`, `keys`, `items`, `values`, `get`, `__eq__`, `__ne__`
`MutableMapping`(`Mapping`)	𝔸 :`__setitem__`, `__delitem__`, 𝕄 :`pop`, `popitem`, `clear`, `update`, `setdefault`
`MappingView`(`Sized`)	𝕄 :`__len__`
`KeysView`(`MappingView`, `Set`)	𝕄 :`__contains__`, `__iter__`
`ItemsView`(`MappingView`, `Set`)	*same*
`ValuesView`(`MappingView`)	*same*

■ `heapq`

Heap queue algorithm. Many functions operate on a heapified list *h*.

`heappush`(*h*, *v*) add *v* to heap
`heappop`(*h*) remove, return smallest
`heappushpop`(*h*, *v*) like push then pop
`heapreplace`(*h*, *v*) like pop then push
`heapify`(*x*) make list *x* into heap
`merge`(∗*x*) given sorted inputs, single iterator over merged sorted items
`nlargest`(*n*, *it*, [*key*]) top *n* items from *it*, or with top values from *key*(*x*)
`nsmallest`(*n*, *it*, [*key*]) same, but *n* least

■ `bisect`

Binary search for *x* on sorted seqeucen *s*. Can operate on *b*:*e* sublist instead.

`bisect_left`(*s*,*x*,[*b*,[*e*]]) leftmost index to insert *x* in *s* to keep *s* sorted
`bisect_right`(*s*,*x*,[*b*,[*e*]]) rightmost, same
`bisect`(...) same as `bisect_right`
`insort_left`(*s*,*x*,[*b*,[*e*]]) same, but does
`insort_right`(*s*,*x*,[*b*,[*e*]]) the insertion
`insort`(...) same as `insort_right`

■ `os`

Operating system dependent module.

`name`

Process Parameters

`environ`
`chdir`(*path*)
`fchdir`(*fd*)
`getcwd`()
`ctermid`()
`getegid`()
`geteuid`()
`getgid`()
`getgroups`()
`initgroups`(*username*, *gid*)
`getlogin`()
`getpgrp`()
`getpid`()
`getppid`()
`getresuid`()
`getresgid`()
`getuid`()
`getenv`(*varname*, [*value*])
`putenv`(*varname*, *value*)
`setegid`(*egid*)
`seteuid`(*euid*)
`setgid`(*gid*)
`setgroups`(*groups*)
`setpgrp`()
^{g}`setpgid`(*pid*, *pgrp*)
`setregid`(*rgid*, *egid*)
`setresgid`(*rgid*, *egid*, *sgid*)
`setresuid`(*ruid*, *euid*, *suid*)
`setreuid`(*ruid*, *euid*)
`getsid`(*pid*)
`setsid`()
`setuid`(*uid*)
`strerror`(*code*)
`umask`(*mask*)
`uname`()
`unsetenv`(*varname*)