Last modified: **2024-08-22**

Interpreter Command-Line Options

`-h` / `--help`	Show command-line options	`--help-xoptions`	Show help on `-X` options
`--help-env`	Show help on environment vars	`-V` / `--version`	Show interpreter version

`-b`	Issue warning on `bytes` to `str` conversion without encoding, or comparing `bytes` to `str` or `int`
`-bb`	As `-b` but issue an error instead of a warning
`-B`	Don't write `.pyc` files for imported modules
`--check-hash-based-pycs` (`default` | `always` | `never`)	Controls hash-based `.pyc` validation with source files
`-d`	Enable parse debugging output (expert only)
`-E`	Ignore all `PYTHON*` environment vars
`-i`	Enter interactive mode after running script
`-I`	Isolated mode: sets `-E -P -s`
`-O`	Remove asserts and set `__debug__` to `False`
`-OO`	As `-O` plus discard docstrings
`-P`	Don't prepend potentially unsafe path to `sys.path` (CWD for `-m` *module* or `-c` *code*, script dir for scripts)
`-q`	Don't show copyright/version, even in interactive
`-R`	Override `PYTHONHASHSEED=0`
`-s`	Don't add user site packages directory to `sys.path`
`-S`	Disable `site` import and its `sys.path` updates
`-u`	Force `stdout` and `stderr` streams to be unbuffered
`-v`	Verbose trace of module imports and cleanup
`-v`	As `-v` plus show every file checked locating modules
`-W`	Warning control, see below
`-x`	Skip first line of source (DOS hack to skip shebang)
`-X`	Implementation specific options, see below

`-W` *Options*

The simple form applies a default behaviour to all warnings

`-Wdefault`	Warn once per location called from
`-Werror`	Convert warnings to exceptions
`-Walways`	Warn on every call
`-Wall`	Alias for `-Walways`
`-Wmodule`	Warn once per calling module
`-Wonce`	Warn once per Python process
`-Wignore`	Never warn

The full form allows specific cases to be targeted

action `:` *message* `:` *category* `:` *module* `:` *lineno*

Action applied to warnings matching other fields specified
Overrides earlier specifications, empty fields match all values

action	One of the actions listed above
message	Match substring of message (ignores case)
category	e.g. `DeprecationWarning`
module	Match fully-qualified module name
lineno	Matches line number, zero matches all lines

`-X` *Options*

`-X faulthandler`	Enable `faulthandler` module
`-X showrefcount`	In debug builds, show total refcount
`-X tracemalloc`	Enable `tracemalloc` module
`-X int_max_str_digits=`*x*	Set `int` to `str` conv. limit
`-X importtime`	Show how long each import takes
`-X dev`	Development mode Enables checks too expensive to enable by default
`-X utf-8`	Enable UTF-8 mode (ignore locale)
`-X pycache_prefix=`*path*	Set root dir for `.pyc` files
`-X warn_default_encoding`	Issue `EncodingWarning` if default encoding is used
`-X no_debug_ranges`	Disable inclusion of extra location information in bytecode instructions, reduces size
`-X frozen_modules=off`	Ignore frozen modules
`-X perf`	Enable Linux `perf` profiler support

For the changes implemented in development mode, see page 14

Interpreter Environment Variables

These options are considered "set" if they are set and their value is a non-empty string, but their value is otherwise ignored	
PYTHONCASEOK	If set, ignore case in `import` statements *(Windows and MacOS only)*
PYTHONASYNCIODEBUG	If set, enable debug mode of `asyncio` module
PYTHONMALLOCSTATS	If set, print statistics on `pymalloc` allocator—ignored if `malloc()` allocator is being used
PYTHONLEGACYWINDOWSFSENCODING	If set, revert default filesystem encoding values revert to pre-3.6 values
PYTHONLEGACYWINDOWSSTDIO	If set, use old console reader/writer—unicode chars will be encoded per active code page
These options must be set to a particular value	
PYTHONHOME	Location of standard Python libraries—specify as *prefix*`:`*exec_prefix* to override separately
PYTHONPATH	Additional Python module search path—format is colon separated paths
PYTHONPLATLIBDIR	If non-empty, overrides `sys.platlibdir`—usually either `lib` or `lib64`
PYTHONSTARTUP	Path of file from which to read commands executed at startup in interactive mode
PYTHONBREAKPOINT	Names callable using dotted path syntax to be run within `sys.breakpoint`
PYTHONHASHSEED	Set to `0` to disable default hash randomisation, or positive integer up to $2^{32}-1$ to set seed
PYTHONIOENCODING	Override encoding of `stdin`/`stdout`/`stderr`—specify as *encodingname*`:`*errorhandler*
PYTHONUSERBASE	Set base path for user `site-packages` and the `pip --user` option
PYTHONEXECUTABLE	If set, `sys.argv[0]` will be this instead of the executable name *(MacOS only)*
PYTHONWARNINGS	Equivalent to `-W` option, comma-separated list is equivalent to multiple `-W` options
PYTHONMALLOC	One of: `default`, `malloc`, `pymalloc`, `debug`, `malloc_debug`, `pymalloc_debug`
PYTHONCORERCELOCALE	Set to `0` to skip coercing ASCII to UTF-8, set to `warn` to emit warnings of coercion used
PYTHONTZPATH	Search path for system timezone info, for `zoneinfo`—set to empty string to use `tzdata`

These may be set to a non-empty string to achieve the equivalent of a specified command-line option			
PYTHONSAFEPATH	Set `-P`	**PYTHONINSPECT**	Set `-i`
PYTHONUNBUFFERED	Set `-u`	**PYTHONDONTWRITEBYTECODE**	Set `-B`
PYTHONNOUSERSITE	Set `-s`		
These may be set to an integer to be equivalent to specifying an option N times, or any other non-empty string to specify once			
PYTHONOPTIMIZE	Count of `-O`	**PYTHONVERBOSE**	Count of `-v`
PYTHONDEBUG	Count of `-d`		
These are equivalent to setting some of the `-X` *options*			
PYTHONDEVMODE	Set `-X dev`	**PYTHONUTF8**	Set `-X utf-8`
PYTHONPERFSUPPORT	Set `-X perf`	**PYTHONTRACEMALLOC**	Set `-X tracemalloc`
PYTHONFAULTHANDLER	Set `-X faulthandler`	**PYTHONPYCACHEPREFIX**	Set `-X pycache_prefix`
PYTHONNODEBUGRANGES	Set `-X no_debug_ranges`	**PYTHONPROFILEIMPORTTIME**	Set `-X importtime`
PYTHONWARNDEFAULTENCODING	Set `-X warn_default_encoding`		
PYTHONINTMAXSTRDIGITS	Set `-X int_max_str_digits=`*x*		

Conditionals & Loops

```
if expr:
    ...
elif expr:
    ...
else:
    ...
```

```
while expr:
    ...
else:
    ... Run if no early exit
```

```
for target in iterable:
    ...
else:
    ... Run if no early exit
```

```
match subject:
    ...
    case pattern [| pattern [...]] [ as name]:
        ... Can use name to refer to value matched
    case [ name1, name2, *name3, name4 ]:
        ... name3 receives all unmatched items
    case { name1: expr, name2: expr, **name3 }:
        ... name3 receives all unmatched items
    case type( [expr, name=expr] ):
        ... See notes to right
    case pattern if expr:
        ...
    case ( _ | name):
        ... Can use name to refer to value matched
```

Only one irrefutable pattern (i.e. matches anything) per `match`

```
case type(name1=expr1,
          name2=expr2):
```

↓

Match if subject has both attrs name1 and name2, with values expr1 and expr2 respectively

↑

```
(name1, name2, name3)
```

↑

```
__match_args__
```
attribute fetched from subject

↑

```
case type(expr1,expr2):
```

Try and With

```
try:
    ...
except type as name:
    ...
except (type, type) as name:
    ...
except:
    ...
else:
    ... Run if no exceptions
finally:
    .... Always will be run
```

```
try:
    ...
except* type as name:
    ...
except* (type, type) as name:
    ...
else:
    ...
finally:
    ...
```

This version for handling exception groups

```
with expr:
    ...
```

```
with e as target:
    ...
```

```
with e1 as t1, e2 as t2:
    ...
```

The above is equivalent to that below

```
with e1 as t1:
    with e2 as t2:
        ...
```

```
__enter__() → target
```

If exc: `__exit__(exc_type, exc_value, tb)`

If not: `__exit__(None, None, None)`

Functions

```
@decor1
@decor2(expr)
@decor3
def name(param: type=default, ...) -> type:
    ...
```

↓

```
name = decor1(decor2(expr)(decor3(name)))
```

All parameters without defaults must occur before those with defaults

✗

```
def name(p1, p2, p3=default, p4):
```

Positional only	*Either*	*Keyword only*
`def name(p1, p2,`	`/, p3, p4, *,`	`p5, p6):`

```
def name(*spare_pos_params, **spare_kw_params):
```

Classes

```
@dec1
@dec2
class name(base1, base2):
    name1: type = expr
    name2: type

    def meth(self, ...):
        ...
```

→ `name = dec1(dec2(name))`

```
x = MyClass.method
inst = MyClass()
y = inst.method
```

`x(inst)` *Unbound*
`y()` *Bound*

→ *Variables defined here become class members, but can be accessed through* `self`

```
@classmethod
def meth(cls, ...):
    ...
```

```
@staticmethod
def meth(...):
    ...
```

If assigned to via `self`*, will create a new instance variable with the same name which hides the class member.*

Simple Statements

`name1 = name2 = expr`
`name: type = expr`
`(a, b, *c) = (1, 2, 3, 4, 5)`
`x[3:5] = items`

`name += expr` → `+=` `-=` `*=` `/=` `//=` `%=` `**=` `&=` `|=` `^=` `>>=` `<<=` `@=`

`assert expr`
`assert expr, message`
`pass`
`del name, name, ...`
`return expr`
`break`
Only in `for` *or* `while`
`yield expr`
`yield from generator`
`raise exception`
`raise exception from exception`
`continue`

`import module, module, ...`
`import module as name`
`from module import (name, name, ...)`
`from module import *`
`from __future__ import feature`
Must occur before all but docstrings & comments
`global name, name, ...`
`nonlocal name, name, ...`
`type name = type`

Operators

18 `(...)` *Parenthesized exprs* | `[...]` *List literal or comp* | `{...}` *Set literal or comp* | `{key: val ...}` *Dict literal or comp*

17 `x[...]` *Index* | `x[start:end:step]` *Slicing* | `x(...)` *Function call* | `x.attr` *Attribute reference*

16 `await x` | 15 `**` *Exponentiation* | 14 `+x` *Unary +ve* | `-x` *Unary -ve* | `~x` *Bitwise NOT*

13 `*` *Multiplication* | `/` *Division* | `//` *Floor division* | `%` *Remainder* | `@` *Matrix multiplication*

Extensions only, not valid for any builtin

12 `+` *Addition* | `-` *Subtraction* | 11 `<<` *Left bit shift* | `>>` *Right bit shift*

10 `&` *Bitwise AND* | 9 `^` *Bitwise XOR* | 8 `|` *Bitwise OR*

All operators group left-to-right except `**` *and* `if ... else ...` *which group right-to-left*

7 `in` | `not in` *Membership* | `is` | `is not` *Identity* | `==` | `!=` *Equality* | `<` | `<=` | `>=` | `!=` *Comparisons*

6 `not x` *Logical NOT* | 5 `and` *Logical AND* | 4 `or` *Logical OR* | 3 `x if y else z` *Conditional expr.*

2 `lambda params: expr` *Lambda (anon. function)* | 1 `:=` *Assignment expr.*

Standard Types

Numeric: `bool` `int` `float` `complex`
Sets: `frozenset` `set`
Sequences: `str` `bytes` `tuple` `memoryview` `list` `bytearray`
Singletons: `None` `NotImplemented` `Ellipsis`
Mappings: `dict`

Types in blue are immutable

Built-in Functions

`abs(`*x*`)`	Absolute value of *x*
`aiter(`*iterable*`)`	Return *iterator* on async iterable
`all(`*iterable*`)`	`True` iff all in *iterable* are true
`anext(`*async_iter*`)`	Next value from *async_iter*
`anext(`*a_iter*`, `*default*`)`	If no value return *default*
`any(`*iterable*`)`	`True` iff any in *iterable* are true
`ascii(`*x*`)`	As `repr()` but escape non-ASCII
`bin(`*x*`)`	Convert integer *x* to binary string
`bool(`*object*`)`	Convert an *object* to `bool`
`breakpoint(…)`	Drop into debugger at this point
`bytearray(`*source*`)`	Convert `bytes`-like *source* to array
`bytearray(`*src*`, `*enc*`)`	Convert `str` to array with encoding
`bytearray(`*s*`, `*e*`, `*errs*`)`	As with `str.encode()`
`bytes(…)`	Params as per `bytearray()`
`callable(`*x*`)`	`True` iff *x* is callable
`chr(`*x*`)`	Return `str` for Unicode code point *x*
`@classmethod(`*x*`)`	Transform *x* into class method
`compile(`*source*`, `*filename*`, `*mode*`, `*flags*`=0,` *dont_inherit*`=False,` *optimize*`=-1)`	Compile *source* into code obj runnable by `exec()` or `eval()`
`complex(`*x*`)`	Convert number/`str` to `complex`
`complex(`*real*`, `*imag*`)`	Create `complex` from *real* & *imag*
`delattr(`*obj*`, `*name*`)`	Delete attr *name* from *obj*
`dict(**`*kwarg*`)`	Create `dict` from keyword args
`dict(`*map*`, **`*kw*`)`	Create from existing mapping
`dict(`*seq*`, **`*kw*`)`	Create from iterable of `(k, v)`
`dir()`	Return list of names in scope
`dir(`*x*`)`	Return list of attributes on *x*
`divmod(`*a*`, `*b*`)`	Return (int quotient, remainder)
`enumerate(`*it*`, `*start*`=1)`	Yield `(n, i)` from iterable *it*
`eval(`*expr*`, `*globals*`=None, `*locals*`=None)`	Evaluate *expr*
`exec(`*x*`, `*globals*`=None, `*locals*`=None, `*closure*`=None)`	Execute *x* as `str` or code obj, always returns `None`
`filter(`*func*`, `*seq*`)`	Yield i from *seq* where *func*(i) true
`float(`*x*`)`	Convert number/`str` to `float`
`format(`*x*`, `*spec*`)`	Format *x* according to *spec*
`frozenset(`*seq*`)`	Create `frozenset` from *seq*
`getattr(`*obj*`, `*name*`)`	Return attr *name* from *obj*
`getattr(`*o*`, `*n*`, `*default*`)`	If attr missing, return *default*
`globals()`	Return `dict` of current globals
`hasattr(`*obj*`, `*name*`)`	Return `True` iff *obj* has attr *name*
`hash(`*x*`)`	Return hash value of *x*
`help(`*request*`)`	Invoke built-in help system
`hex(`*x*`)`	Convert integer *x* to hex string
`id(`*x*`)`	Return unique ID of object
`input()`	Return line of text entered by user
`input(`*prompt*`)`	Print *prompt* before input
`int(`*number*`)`	Convert *number* to `int`
`int(`*x*`, `*base*`=10)`	Convert string *x* in specified *base*
`isinstance(`*obj*`, `*type*`)`	Return `True` iff *obj* is of *type*
`issubclass(`*cls*`, `*info*`)`	Return `True` iff *obj* is subclass of anything in *info* (can be a `tuple` or union)
`iter(`*obj*`)`	Return iterator on *obj*
`len(`*x*`)`	Return number of items in *x*
`list(`*x*`)`	Create `list` from iterable *seq*
`locals()`	Return `dict` of current locals

Built-in Functions *(continued)*

`map(`*func*`, `*seq*`, *`*`*iterables*`)` Apply *func* to each item of *seq*—if *interables* supplied, *func* gets additional args

`max(`seq`, key=None)` Return max item in *seq*—if specified *key* should be 1-arg func to return sort value

`max(`*seq*`, *, `*default*`, `*key*`=None)`
If *seq* empty, return *default* instead of `ValueError`

`max(`*arg1*`, `*arg2*`, *`*`*more*`)` Use args instead of iterable

`memoryview(`*x*`)` Create `memoryview` from *x*

`min(`…`)` As `max()` but minimum item

`next(`*iter*`, `*default*`)` Next item from *iter*—if specified, *default* is returned instead of `StopIteration`

`object()` Create instance of `object` base

`oct(`*x*`)` Convert integer *x* to octal string

`open(`…`)` See the later on `open()` later

`ord(`*x*`)` Return `int` code point of 1-char `str`

`pow(`*base*`, `*exp*`, `*mod*`=None)`
Return *base* to power *exp*, optionally modulo *mod*

`print(`*`*`*objects*`, `*sep*`=" ", `*end*`="\n",`
file`=None, `*flush*`=False)`
Print objects to *file* (or stdout), joined by *sep*, then *end*

`property(`*fget*`, `*fset*`, `*fdel*`, `*doc*`)` Return property

`range(`*stop*`)` Return range type from `0` to *stop*

`range(`*start*`, `*stop*`, `*step*`=1)` From *start* to *stop* in *step*

`repr(`*obj*`)` Return printable representation

`reversed(`*seq*`)` Yield items in *seq* in reverse order

`round(`*num*`, `*ndigits*`=None)` Round *num* to n digits, or int

`set(`*x*`)` Create `set` from iterable *x*

`setattr(`*obj*`, `*name*`, `*val*`)` Set attr *name* to *val* on *obj*

`slice(`…`)` As range, but returns `slice` object

`sorted(`*seq*`, `*key*`=None, `*reverse*`=False)`
Return sorted list from *seq*, *key* as per `max()`

`@staticmethod(`*x*`)` Transform *x* into static method

`str(`*obj*`)` Return `str` version of *obj*

`str(`*obj*`, `*encoding*`="utf-8", `*errors*`="strict")`
Convert bytes-like *obj* to string form

`sum(`*seq*`, `*start*`=0)` Sum *seq*, starting with *start*

`super(`*type*`, `*obj*`=None)` Return proxy for base class of *type*, if specified, *obj*.`__mro__` attr used for resolution order

`tuple(`*seq*`)` Create `tuple` from iterable *seq*

`type(`*obj*`)` Return type of *obj*

`type(`*name*`, `*bases*`, `*dict*`, `*`*`*kwds*`)` Return new `type`

`vars(`*obj*`)` Return the `__dict__` of obj

`zip(`*`*`*seq*`, `*strict*`=False)`
Yield n-tuple from corresponding items in *seqs*, if *strict* raise `ValueError` if any iterable is of different length

`__import__(`*name*`, `*globals*`=None, `*locals*`=None,`
fromlist`=(), `*level*`=0)`
Implementation behind the `import` statement

open()

mode	
`r`	Open for reading
`w`	Open for writing
`x`	Fail if already exists
`a`	Open for append
`r+`	Read and write
`w+`	As `r+` but truncate
`…t`	Text mode (default)
`…b`	Binary mode

buffering	
`<0`	Default policy
`0`	No buffering
`1`	Line buffering
`>1`	Set buffer size

closefd

If an open file descriptor passed to `open()`, only close it on close if `closefd` is true

newline	
`None`	Read: all endings to `\n`, write:`\n` to `os.linesep`
`""`	Recognise all endings on read, but no translations
`"\n"`	} For all these options, recgonise only this value
`"\r"`	} as a line ending when reading, and translate
`"\r\n"`	} any `\n` to this value on write

opener

A function which will be passed `(`*path*`, `*flags*`)`, where *flags* is as would be passed to `os.open()`, return OS file descriptor

Methods on Builtins

`int`	
`bit_length()`	Number of bits to represent
`bit_count()`	Number of bits set
`to_bytes(`*length*`=1, `*byteorder*`="big", `*signed*`=False)`	
`int.from_bytes(`*bytes*`, …)`	
`as_integer_ratio()`	Always `(`*x*`, 1)`
`is_integer()`	Always `True`

`float`	
`as_integer_ratio()`	Denominator positive
`is_integer()`	`True` iff finite integer value
`hex()`	Hex string representation
`float.fromhex(`*string*`)`	Create from hex string

Hex form: [*sign*] [`0x`] *integer* [`.` *fraction*] [`p` *exponent*]

Sequences	
`index(`*item*` [, `*start*` [, `*stop*` ]])`	Return index of first *start* ≤ *item* < *stop*, or `ValueError` if not found
`count(`*item*`)`	Return number of matches

Mutable Sequences	
`append(`*item*`)`	Appends *item*
`clear()`	Remove all items
`copy()`	Shallow copy
`extend(`*iterable*`)`	Append from *iterable*
`insert(`*index*`, `*item*`)`	Insert *item* before *index*
`pop([`*index*`])`	Remove & return at *index*, RHS if omitted
`remove(`*item*`)`	Remove first *item*, `ValueError` if none
`reverse()`	Reverse items in-place

`list`	
`sort(`*key*`=None, `*reverse*`=False)`	Sort items in-place

`memoryview`	
`tobytes(`*order*`='C')`	Return data as `bytes`
`hex([`*sep*` [, `*bytes_per_sep*` ]])`	Convert to hex digits
`tolist()`	Return as `list`
`toreadonly()`	Return read-only version
`release()`	Release underlying buffer
`cast(`*format*` [, `*shape*`])`	Cast to new format

`bytes` & `bytearray`	
`bytes.from_hex(`*s*`)`	Convert `str` of hex digits to `bytes`
`hex([`*sep*` [, `*bytes_per_sep*` ]])`	Convert to hex digits
`count(`*substr*` [, `*start*` [, `*stop*` ]])`	*substr* is `bytes` or `int`
`removeprefix(`*prefix*`)`	If `bytes` starts with *prefix*, remove it
`removesuffix(`*suffix*`)`	If `bytes` starts with *suffix*, remove it
`decode(`*encoding*`="utf-8", `*errors*`="strict")`	
`endswith(`*suffix*` [, `*start*` [, `*stop*` ]])`	Returns `bool`
`find(`*substr*` [, `*start*` [, `*stop*` ]])`	First match index, or `-1`
`index(…)`	As `find()` but raise `ValueError` if missing
`join(`*iterable*`)`	`TypeError` if *iterable* contains non-`bytes`
`str.maketrans(`*fromchars*`, `*tochars*`)`	
`partition(`*sep*`)`	Return `(`before`, `*sep*`, `after`)`
`replace(`*current*`, `*replacement*` [, `*max_count*` ])`	
`rfind(`*substring*` [, `*start*` [, `*stop*` ]])`	Final match index
`rindex(…)`	As `rfind()` but raise `ValueError` if missing
`rpartition(`*sep*`)`	`partition()` but on final match
`startswith(`*suffix*` [, `*start*` [, `*stop*` ]])`	Returns `bool`
`translate(`*table*`, `*delete*`=b'')`	Translate through *table*

These work like the `str` *version, and can be used on arbitrary* `bytes`

`center(…)` `ljust(…)` `lstrip(…)` `rjust(…)`
`rsplit(…)` `rstrip(…)` `split(…)` `strip(…)`

These assume ASCII and should not be used on arbitrary `bytes`

`capitalize()` `expandtabs(…)` `isalnum()`
`isalpha()` `isascii()` `isdigit()` `islower()`
`isspace()` `istitle()` `isupper()` `lower()`
`splitlines(…)` `swapcase()` `title()` `upper()`
`zfill(…)`

`set`	
`add(`*elem*`)`	Add *elem* to the set
`clear()`	Remove all elements
`copy()`	Return shallow copy
`discard(`*elem*`)`	As `remove()` but ignore missing
`pop()`	Remove & return arbitrary element
`remove(`*elem*`)`	Remove *elem*, `KeyError` if missing

Methods on Builtins *(continued)*

`str`	
`capitalize()`	Capitalise the first character
`casefold()`	Return casefolded string
For caseless matching—as `lower()` *for ASCII, differs for some chars*	
`center(`*width*`[, `*fillchar*`])`	Pad with *fillchar* both sides
`count(`*substring*`[, `*start*`[, `*stop*`]])`	Returns `int`
`encode(`*encoding*`="utf-8", `*errors*`="strict")`	
`endswith(`*suffix*`[, `*start*`[, `*stop*`]])`	Returns `bool`
`expandtabs(`*tabsize*`=8)`	Expand tabs to spaces
`find(`*substring*`[, `*start*`[, `*stop*`]])`	First match index, or `-1`
`format(*`*args*`, **`*kwargs*`)`	See later for formatting
`format_map(`*mapping*`)`	Use *mapping* directly
`index(…)`	As `find()` but raise `ValueError` if missing
`isalnum()`	`True` iff len > 0 & all chars alphanumeric
`isalpha()`	`True` iff len > 0 & all chars alphabetic
`isascii()`	`True` iff empty or all chars are ASCII
`isdecimal()`	`True` iff len > 0 & all chars are decimal digits
`isdigit()`	Also includes more chars (e.g. powers)
`isidentifier()`	`True` iff valid Python identifier
`islower()`	`True` iff len > 0 & all chars are lowercase
`isnumeric()`	`True` iff len > 0 & all chars are numeric
`isprintable()`	`True` iff empty or no control chars
`isspace()`	`True` iff len > 0 & all chars whitespace
`istitle()`	`True` iff len > 0 & all string is in title case
`isupper()`	`True` iff len > 0 & all chars are uppercase
`join(`*iterable*`)`	`TypeError` if *iterable* contains non-`str`
`ljust(`*width*`[, `*fillchar*`])`	Pad with *fillchar* on right
`lower()`	Return lowercased version
`lstrip(`[*chars*]`)`	Remove leading *chars*, or whitespace
`str.maketrans(`*fromchars*`[, `*tochars*`[, `*removechars*`]])`	
`partition(`*sep*`)`	Return `(`before`, `*sep*`, `after`)`
`removeprefix(`*prefix*`)`	If string starts with *prefix*, remove it
`removesuffix(`*suffix*`)`	If string ends with *suffix*, remove it
`replace(`*current*`, `*replacement*`[, `*max_count*`]])`	

`str` *(continued)*	
`rfind(`*substring*`[, `*start*`[, `*stop*`]])`	Final match index
`rindex(…)`	As `rfind()` but raise `ValueError` if missing
`rjust(`*width*`[, `*fillchar*`])`	Pad with *fillchar* on left
`rpartition(`*sep*`)`	`partition()` but on final match
`rsplit(`*sep*`=None, `*maxsplit*`=-1)`	As `split()` from right
`rstrip(`[*chars*]`)`	Remove trailing *chars*, or whitespace
`split(`*sep*`=None, `*maxsplit*`=-1)`	Split on *sep* up to *maxsplit*
`splitlines(`*keepends*`=False)`	Split on line boundaries
`startswith(`*suffix*`[, `*start*`[, `*stop*`]])`	Returns `bool`
`strip(`[*chars*]`)`	Remove leading and trailing *chars*
`swapcase()`	Swap upper/lowercase
`title()`	Return title-cased version
`translate(`*table*`)`	Translate chars through *table*
`upper()`	Return uppercased version
`zfill(`*width*`)`	Left-pad with zero digits

`dict`	
`clear()`	Remove all elements
`copy()`	Return shallow copy
`dict.fromkeys(`*iterable*`, `*value*`=None)`	New `dict` with keys from *iterable*, all with value *value*
`get(`*key*`, `*default*`=None)`	Return *default* if *key* missing
`items()`	Return view of `(`key`, `value`)` pairs
`keys()`	Return view of the keys
`pop(`*key* `[, `*default*`])`	As `get()` but remove element, if *default* is not specified then raise `KeyError` instead of `None`
`popitem()`	Remove & return in LIFO order, or `KeyError`
`setdefault(`*key*`, `*default*`=None)`	If *key* in `dict`, return its value—else insert it with value *default* and return that
`update(`[*other*`,`][`**`*kw*`])`	Update from *other* and/or *kw*
`values()`	Return view of the values

String Formatting — % operator

`%` [(*key*)] [*flags*] [*min-width*] [.*precision*] *conversion*

key
Used when `dict` *is to be passed as the right-hand parameter:*
`"%(a)s %(b)s" % {"b": "x", "a": "y"}`

flags	
`#`	Use "alternative form", detailed below
`0`	Pad with zeroes rather than spaces for numerics
`-`	If content under width, left align (otherwise right)
`' '`	Leave a single space where a sign isn't required
`+`	Force sign character (either `+` or `-`)

min-width / *precision*	
`*`	Consume width from parameter list (before value)
For `s` *conversion, value truncated to the precision width if greater*	

Alternate Forms		
Code	*Alteration*	*Example output*
`o`	Prepend `0o` to value	`0o173`
`x` / `X`	Prepend `0x` / `0X` to value	`0x7b`
`e` / `E` / `f` / `F`	Always include decimal point	`123.`
`g` / `G`	As `e`, … & preserve trailing zeroes	`1.e+02`

conversion		
Code	*Meaning*	*Example output*
`d`	Signed decimal int	`123`
`i`	Alias for `d`	`123`
`o`	Signed octal int	`173`
`u`	Alias for `d` *(obsolete)*	`123`
`x`	Signed hex int, lowercase	`7b`
`X`	Signed hex int, uppercase	`7B`
`e`	Float exponent, lowercase	`1.230e+00`
`E`	Float exponent, uppercase	`1.230E+00`
`f`	Float decimal	`1.230`
`F`	Alias for `f`	`1.230`
`g`	Chooses `e` or `f` based on exponent & precision	
`G`	Chooses `E` or `f` based on exponent & precision	
`c`	Character, accepts `str` or `int`	`A`
`r`	Format using `repr()`	`1970-01-01`
`s`	Format using `str()`	`date(1970, …)`
`a`	Format using `ascii()`	`date(1970, …)`
`%`	Not a format, use for literal `%`	`%`

String Literals

Escape Sequences			
`\`↳	Ignore following newline	`\r`	ASCII carriage return (CR)
`\\`	Backslash	`\t`	ASCII tab
`\'`	Single quote	`\v`	ASCII vertical tab
`\"`	Double quote	`\`*ooo*	Octal char *ooo*
`\a`	ASCII bell (BEL)	`\x`*hh*	Hex char *hh*
`\b`	ASCII backspace (BS)	`\N{`*n*`}`	Unicode char named *n*
`\f`	ASCII formfeed (FF)	`\u`*xxxx*	Unicode char hex *xxxx*
`\n`	ASCII linefeed (LF)	`\U`*xxxxxxxx*	As `\u` but 32-bit

Literal Concatenation
`"one" "two"` becomes `"onetwo"`

Multiline strings
`"""Preserves newlines and allows " and ' chars"""`

Prefixes	
`b`	`bytes` instead of `str`
`r`	Raw, backslashes treated as literal
`f`	F-strings (see later)

F-Strings

`{` *expr* [`=`] [`!` *conversion*] [`:` *format-spec*] `}`

format-spec is as `str.format()`

flags	
`=`	Display as: *expr=result*

conversion	
`!s`	Transform with `str()`
`!r`	Transform with `repr()`
`!a`	Transform with `ascii()`

String Formatting — `str.format()` and `string.Formatter`

`{` [*field-name*] [`!`*conversion*] [`:`*format-spec*] `}` → [[*fill*] *align*] [*sign*] [`z`] [`#`] [`0`] [*width*] [*grouping*] [`.`*precision*] [*type*]

field-name	
`0`	First positional argument, et.c
`foo`	Keyword argument `foo`

If omitted, refer to position arguments in sequence—so these two are equivalent: `"{} {foo} {}"` *and* `"{0} {foo} {1}"`

You cannot mix automatic (`{}`) and manual (`{0}`) numbering

conversion	
`!s`	Transform with `str()`
`!r`	Transform with `repr()`
`!a`	Transform with `ascii()`

fill for padding, space is default

align	
`<`	Left align (default)
`>`	Right align
`=`	Pad between sign & digits
`^`	Centre align

sign	
`+`	Use sign for +ve & -ve
`-`	Use sign just for -ve
`' '`	Use space for +ve

String types	
`s`	String *(default)*
Integer types	
`b`	Binary
`c`	Character
`d`	Decimal *(default)*
`o`	Octal
`x`	Hex, lowercase
`X`	Hex, uppercase
`n`	As `d`, but locale aware

Float types			
`e`	Scientific, lowercase	`g`	General *(default)*
`E`	Scientific, uppercase	`G`	General uppercase
`f`	Fixed-point	`n`	`g` but locale aware
`F`	As `f`, with `NAN` & `INF`	`%`	Float as %age

flags					
`z`	Coerce `-0` to `+0`	`#`	Alternative form	`0`	Zero-pad

grouping	
`,`	Thousands separator (see type `n` for locale-aware)
`_`	Underscore as separator for floats, `d`, `b`, `o`, `x` and `X`

String Formatting — `string.Template`

`string.Template(`*template*`)` `$$` for literal $ `$`*ident* `or ${`*ident*`}` for placeholders

Class Attributes			
`delimeter`	Ident signifier (default $)	`braceidpattern`	Regex for braced ident names
`idpattern`	Regex for ident names	`flags`	Regex flags (default `IGNORECASE`)

Methods	
`substitute(`*mapping*`={}, **`*kwargs*`)`	Instantiate template, taking arguments from *mapping* and/or *kwargs*
`safe_substitute(`*mapping*`={}, **`*kwargs*`)`	Don't raise `KeyError` or `ValueError`, leave placeholders unchanged
`is_valid()`	Return `False` iff `substitute()` would return `ValueError`
`get_identifiers()`	Return `list` of valid indentifiers in template

Unicode and Other Encodings

`bytes.decode(`*encoding*`="utf-8",` *errors*`="strict") -> str` | `str.encode(…) -> bytes`

errors	
strict	Raise `UnicodeError` (default)
ignore	Silently ignore malformed data
replace	Replace malformed with ASCII `?` on encode, `U+FFFD` on decode (official REPLACEMENT CHAR)
backslashreplace	Replace malformed with escapes: `\x`*hh*, `\u`*xxxx* or `\U`*xxxxxxxx* on encode, `\x`*hh* on decode
surrogateescape	On decode replace byte with surrogate code point, on encode revert to original byte
	The following only applicable for encoding…
xmlcharrefreplace	Replace with XML/HTML character reference (&#*num*)
namereplace	Replace with `\N{…}` escape, where the content is the name from the Unicode Character Database
	The following only applicable for utf-8/16/32…
surrogatepass	Allow surrogate code points when encoding/decoding, otherwise treated as error in a `str`

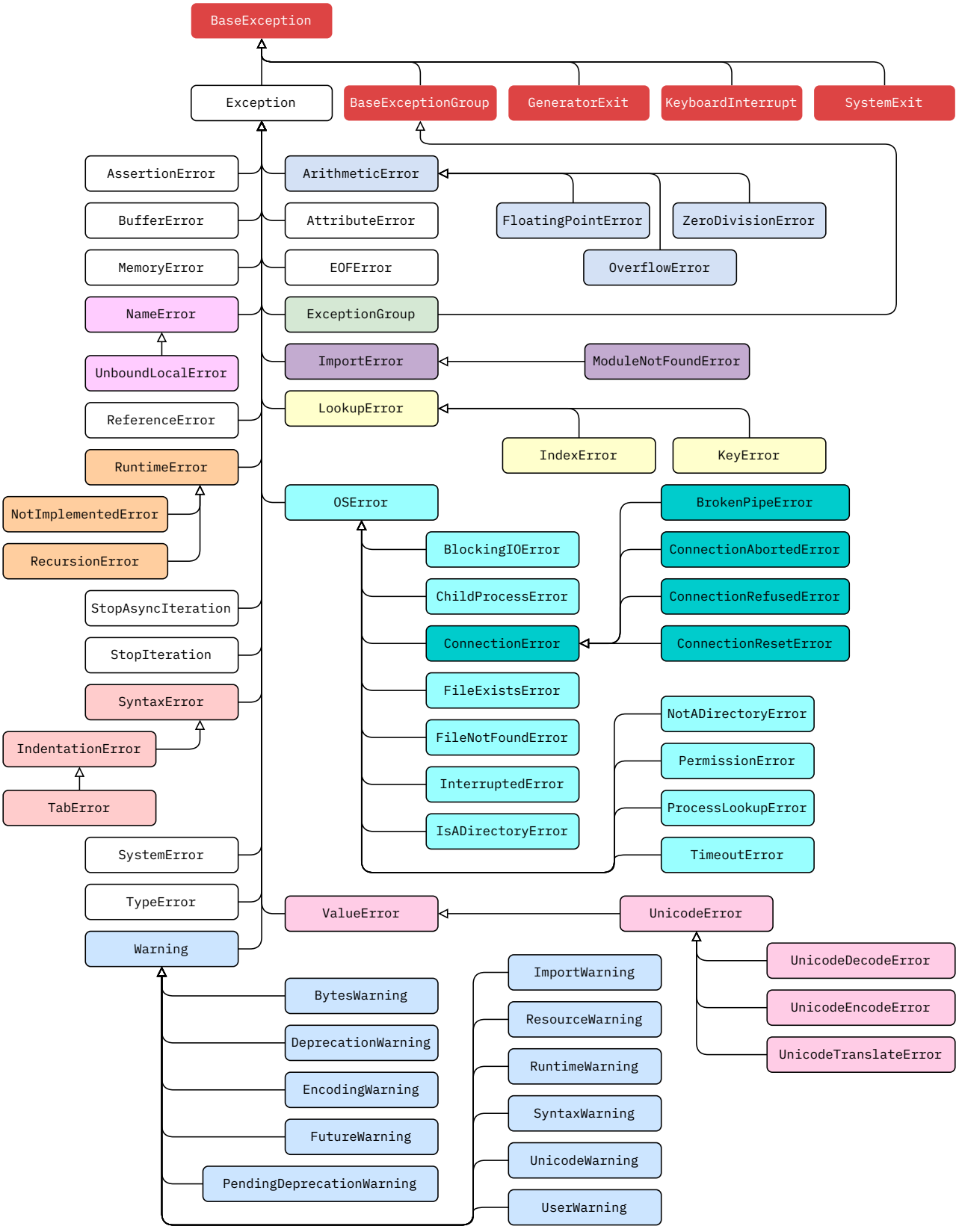
Standard Unicode Encodings

ascii	cp860	cp1250	iso2022_jp	iso8859_11	shift_jis
big5	cp861	cp1251	iso2022_jp_1	iso8859_13	shift_jis_2004
big5hkscs	cp862	cp1252	iso2022_jp_2	iso8859_14	shift_jisx0213
cp037	cp863	cp1253	iso2022_jp_2004	iso8859_15	utf_32
cp273	cp864	cp1254	iso2022_jp_3	iso8859_16	utf_32_be
cp424	cp865	cp1255	iso2022_jp_ext	johab	utf_32_le
cp437	cp866	cp1256	iso2022_kr	koi8_r	utf_16
cp500	cp869	cp1257	latin_1	koi8_t	utf_16_be
cp720	cp874	cp1258	iso8859_2	koi8_u	utf_16_le
cp737	cp875	euc_jp	iso8859_3	kz1048	utf_7
cp775	cp932	euc_jis_2004	iso8859_4	mac_cyrillic	utf_8
cp850	cp949	euc_jisx0213	iso8859_5	mac_greek	utf_8_sig
cp852	cp950	euc_kr	iso8859_6	mac_iceland	
cp855	cp1006	gb2312	iso8859_7	mac_latin2	– *Aliases not listed*
cp856	cp1026	gbk	iso8859_8	mac_roman	– *Case insensitive*
cp857	cp1125	gb18030	iso8859_9	mac_turkish	– *Can use* `-` *for* `_`
cp858	cp1140	hz	iso8859_10	ptcp154	

Other Text Encodings	
idna	Unicode in domain names (RFC 3490)
mcbs	Multibyte char set, Windows only
oem	OEM code page, Windows only
palmos	Encoding of PalmOS 3.5
punycode	IDNA in ASCII encoding (RFC 3492)
raw_unicode_escape	Latin-1 with `\U` escapes
undefined	Raise an exception for *all* conversions
unicode_escape	Used to embed Unicode in ASCII source

`bytes` *to* `bytes`, *use* `codecs.encode()`	
base64	Convert to multiline MIME base64
bz2	Compress using BZ2
hex	Convert to hex, two chars per byte
quopri	Convert to MIME quoted printable
uu	Convert to uuencoded *(to be removed in 3.13)*
zlib	Compress using gzip
`str` *to* `str`, *use* `codecs.encode()`	
rot13	Caesar-cipher of letters with offset of 13

Built-in Exceptions



Dunder Methods & Attributes

Object Attributes

`__dict__`	A mapping storing an object's attributes	`__type_params__`	Type param list of generics
`__class__`	The class to which an instance belongs	`__mro__`	`tuple` of bases for method resolution order
`__bases__`	`tuple` of base classes for a class	`__subclasses__()`	
`__name__`	The name this object was given	Returns list of live weakrefs to immediate subclasses	
`__qualname__`	Fully qualified version of name		

Basic Customisation

`__new__`(*cls*, ...)	Static method, called to create a new instance of a class	
`__init__`(*self*, ...)	Called on instance created by `__new__()` to initialise it—must also call on base classes	
`__del__`(*self*)	Finaliser (*not* destructor), called when ref count reaches zero—must also call on base classes	
`__repr__`(*self*)	Invoked for `repr(obj)`, must return `str`—ideally returns a valid Python expression	
`__str__`(*self*)	Invoked to convert to string form, must return `str`	
`__bytes__`(*self*)	Invoked to convert to binary form, must return `bytes`	
`__format__`(*self*, *spec*)	Invoked on `format(obj)` etc., must return `str`	
`__lt__`(*self*, *other*)	Rich comparison method, invoked for *obj* <= *other*	
`__le__`(*self*, *other*)	Rich comparison method, invoked for *obj* < *other*	*To generate all of these from just*
`__eq__`(*self*, *other*)	Rich comparison method, invoked for *obj* == *other*	*defining* `__eq__()` *and one*
`__ne__`(*self*, *other*)	Rich comparison method, invoked for *obj* `!=` *other*	*other, use* `functools.`
`__gt__`(*self*, *other*)	Rich comparison method, invoked for *obj* >= *other*	`total_ordering()`
`__ge__`(*self*, *other*)	Rich comparison method, invoked for *obj* > *other*	
`__hash__`(*self*)	Invoked on to get key for `dict`, `set`, ..., must return `int`, will be truncated to `Py_ssize_t`	
`__bool__`(*self*)	Invoked to cast to `bool`, must return `True` or `False`—if missing, `__len__()` is used	

Attribute Access

`__getattr__`(*self*, *name*)	Called if default access raises `AttributeError`, return value or raise `AttributeError`
`__getattribute__`(*self*, *name*)	Called first on attribute access if defined, return value or raise `AttributeError`
`__setattr__`(*self*, *name*, *value*)	Called on attribute assignment
`__delattr__`(*self*, *name*, *value*)	Called on attribute deletion, should only be defined if meaningful for this object
`__dir__`(*self*)	Invoked on `dir(obj)`, return iterable of `str` which is converted to `list` and sorted

Descriptors (see PEP 252)

`__get__`(*self*, *instance*, *owner*=`None`)	Called to get attribute of *instance*, or class (*instance* is `None`)
`__set__`(*self*, *instance*, *value*)	Called to set an attribute on *instance* to *value*
`__delete__`(*self*, *instance*)	Called to delete an attribute on *instance*
`__objclass__`	Specifies the class where this object was defined (optional, used by `inspect` module)

Dunder Methods & Attributes *(continued)*

Attribute Storage	
`__slots__`	Specifies `tuple` of attribute names instead of using `__dict__`, a little faster & smaller
Context Managers	
`__enter__(self)`	Called at the start of a with block—return value is bound to the target name specified
`__exit__(self, exc_type, exc_value, traceback)`	If exit by exception, called with exception details— otherwise all `None`
Handling Positions Arguments in Class Pattern Matching	
`__match_args__`	Tuple of attribute names to which to map positional args to type-based case in a `match`
Callables	
`__call__(self, ...)`	Called when the object is called as a function, with all arguments pass after *self*
Containers	
`__len__(self)`	Called on `len(obj)`, should return an `int >=0` indicating number of items
`__length_hint__(self)`	Return estimated length—optional, only for performance, may return `NotImplemented`
`__getitem__(self, key)`	Implement lookup *self*`[`*key*`]`—if missing, raise `IndexError` or `KeyError`
`__setitem__(self, key, value)`	Implement assignment *self*`[`*key*`]=`*value*—improper keys, raise as for `__getitem__()`
`__delitem__(self, key)`	Implement deletion `del` *self*`[`*key*`]`—improper keys, raise as for `__getitem__()`
`__missing__(self, key)`	Called by `dict.__getitem__()` for missing keys (e.g. overridden by `defaultdict`)
`__iter__(self)`	Called to obtain an iterator over to the container—for mappings, iterate over keys
`__reversed__(self)`	Implement if you can offer a better reverse iterator than `__len__()` & `__getitem__()`
`__contains__(self, key)`	Implement if you can offer membership test better than iteration—return `True` iff *key* in *self*

Numerics: binary operations			
`__add__(self, other)`	Implements `+`	`__divmod__(self, other)`	Implements `divmod()`
`__sub__(self, other)`	Implements `-`	`__pow__(self, other[, modulo])`	`pow(x, ...)` & `**`
`__mul__(self, other)`	Implements `*`	`__lshift__(self, other)`	Implements `<<`
`__matmul__(self, other)`	Implements `@`	`__rshift__(self, other)`	Implements `>>`
`__truediv__(self, other)`	Implements `/`	`__and__(self, other)`	Implements `&`
`__floordiv__(self, other)`	Implements `//`	`__xor__(self, other)`	Implements `^`
`__mod__(self, other)`	Implements `%`	`__or__(self, other)`	Implements `|`

Numerics: reversed binary operations—called on right operand with args swapped iff left doesn't implement above, and right is different type			
`__radd__(self, other)`	Implements `+`	`__rdivmod__(self, other)`	Implements `divmod()`
`__rsub__(self, other)`	Implements `-`	`__rpow__(self, other)`	Impl. `pow(x, ...)` & `**`
`__rmul__(self, other)`	Implements `*`	`__rlshift__(self, other)`	Implements `<<`
`__rmatmul__(self, other)`	Implements `@`	`__rrshift__(self, other)`	Implements `>>`
`__rtruediv__(self, other)`	Implements `/`	`__rand__(self, other)`	Implements `&`
`__rfloordiv__(self, other)`	Implements `//`	`__rxor__(self, other)`	Implements `^`
`__rmod__(self, other)`	Implements `%`	`__ror__(self, other)`	Implements `|`

Dunder Methods & Attributes *(continued)*

Numerics: augmented arithmetic assignments			
`__iadd__(self, other)`	Implements `+=`	`__ipow__(self, other[, modulo])`	Implements `**=`
`__isub__(self, other)`	Implements `-=`	`__ilshift__(self, other)`	Implements `<<=`
`__imul__(self, other)`	Implements `*=`	`__irshift__(self, other)`	Implements `>>=`
`__imatmul__(self, other)`	Implements `@=`	`__iand__(self, other)`	Implements `&=`
`__itruediv__(self, other)`	Implements `/=`	`__ixor__(self, other)`	Implements `^=`
`__ifloordiv__(self, other)`	Implements `//=`	`__ior__(self, other)`	Implements `|=`
`__imod__(self, other)`	Implements `%=`		
Numerics: unary operations			
`__neg__(self)`	Implements unary `+`	`__abs__(self)`	Implements `abs(x)`
`__pos__(self)`	Implements unary `-`	`__invert__(self)`	Implements `~`
Numerics: conversions			
`__complex__(self)`	Implements `complex(x)`	`__abs__(self)`	Implements `abs()`
`__int__(self)`	Implements `int(x)`	`__index__(self)`	Used for slices, `hex()`, …
Numerics: rounding operations			
`__round__(self[, ndigits])`	Implements `round(x)`	`__floor__(self)`	Implements `math.floor(x)`
`__trunc__(self)`	Implements `math.trunc(x)`	`__ceil__(self)`	Implements `math.ceil(x)`

Buffer Types	
`__buffer__(self, flags)`	Called when buffer requested from *self*, see `inspect.BufferFlags` for meaning of *flags*
`__release_buffer__(self, buffer)`	Called when buffer no longer needed, *buffer* is what `__buffer__()` returned
Class Creation & Metaclasses	
`__init_subclass__(cls)`	Class method, called when this class is subclassed, where *cls* is the new subclass
`__set_name__(self, owner, name)`	Called when class *owner* defined if instance *self* is class member *name* in *owner*
`__mro_entries__(self, bases)`	Called on object which isn't a `type` used as a base class to return `tuple` of classes to use
`__prepare__(name, bases, **kw)`	Class method on metaclass, returns `dict`-like object to use for attribute storage
`__instancecheck__(self, instance)`	Method on metaclass, return `True` iff *instance* is an instance of this class
`__subclasscheck__(self, subclass)`	Method on metaclass, return `True` iff *subclass* is a subclass of this class
`__class_getitem__(cls, key)`	Return the specialization of a generic class by type arguments found in *key*
Asynchronous Constructs	
`__await__(self)`	Makes object awaitable: called to obtain iterator when used with `await`
`__aiter__(self)`	Called with used with `async for`, must return an async iterator
`__anext__(self)`	Called to obtain an awaitable which yields next item, or raises `StopAsyncIteration`
`__aenter__(self)`	The async analogue of `__enter__()`, must return an awaitable
`__aexit__(self, exc_type, exc_value, traceback)`	Async analogue of `__exit__()`, must return an awaitable

Dunder Methods & Attributes *(continued)*

Function Attributes	
`__globals__`	Read-only reference to the `dict` holding the function's globals (the module in which it's defined)
`__closure__`	Either `None` or a `tuple` of cells with bindings for free variables (cells have `cell_contents` attr)
`__doc__`	Either `None` of the function's docstring
`__name__`	Function's name (for lambdas, will be `"<lambda>"`)
`__qualname__`	Function's fully qualified name
`__module__`	Either `None`, or the module in which the function was defined
`__defaults__`	Either `None`, or a `tuple` containing defaults for positional parameters
`__code__`	The code object representing the compiled function body
`__dict__`	Namespace for arbitrary function attributes
`__annotations__`	A `dict` of annotations for function parameters, with key `"return"` for return type annotation
`__kwdefaults__`	Either `None`, or a `dict` containing defaults for keyword parameters
`__type_params__`	A `tuple` of type parameters for generic functions

Bound Instance Method Attributes			
`__self__`	Instance to which method is bound	`__name__`	Same as `__func__.__name__`
`__func__`	The original function objecty	`__module__`	Same as `__func__.__module__`
`__doc__`	Same as `__func__.__doc__`		

Development Mode

To enable development mode: `-X dev` | `PYTHONDEVMODE="1"`

Effects

I. Behave as `-W default`—shows following warnings which are normally filtered:
   - `DeprecationWarning`, `ImportWarning`, `PendingDeprecationWarning`, `ResourceWarning`

II. Behave as `PYTHONMALLOC=debug`—adds debug hooks to check for buffer over/underflow, API & GIL violations

III. Call `faulthandler.enable()` at startup to install handlers for `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS`, `SIGKILL`
   - Handler dumps Python stack traceback on a crash, acts like `-X faulthandler` or `PYTHONFAULTHANDLER="1"`

IV. Enable `asyncio` debug mode, acts like `PYTHONASYNCIODEBUG="1"`

V. Check *encoding* and *errors* arguments on every call for encoding/decoding—by default they're sometimes ignored

VI. `io.IOBase` destructor logs `close()` exceptions

VII. Sets `dev_mode` attribute of `sys.flags` to `True`, so user code can enable its own checks

Coroutines

To enable debug mode: `PYTHONASYNCIODEBUG="1"` | `asyncio.run(…, debug=True)` | `loop.set_debug()`

Effects of enabling asyncio debug mode

I. Check for coroutines that were not awaited and logs them—can detect cases where `await` was missed
II. Raise exceptions in non-threadsafe APIs if called from the incorrect thread
III. Execution time of I/O selectors are logged if it takes too long
IV. Callbacks taking longer than 100ms are logged (use `loop.slow_callback_duration` to change the threshold)

```
async def name(…) -> type:
    …
```

Defines a coroutine—it is a `SyntaxError` *to use* `yield from` *in a coroutine, or to use* `async for` *or* `async with` *outside of the body of a coroutine*

```
asyncio.run(name(), debug=None, loop_factory=None)
```

Default event loop: `asyncio.new_event_loop()`

Runs the pass coroutine, managing event loop and executor. Cannot be called when another event loop is running in same thread.

```
with asyncio.Runner() as runner:
    runner.run(first())
    runner.run(second())
```

Use `asyncio.Runner` *to run multiple coroutines in parallel without having to group them under a single coroutine as you would with* `asyncio.run()`

```
await coroutine(…)
```

Block current coroutine until done

```
task = asyncio.create_task(coroutine())
…
await task
```

Can discard `task` *for "fire and forget" operation, or interact with methods and* `await` *it*

Three types of awaitable object

`Task` | `Future` | Coroutine

`Task` *methods*

`done()`	Return `True` iff coroutine either returned, raised an exception or was cancelled
`result()`	Return result of task, or raise exception that terminated it, or `CancelledError` or `InvalidStateError`
`exception()`	Return exception that terminated task, or `None`, or raise `CancelledError` or `InvalidStateError`
`cancel(msg=None)`	Throw `CancelledError` within coroutine at next event loop cycle, optionally containing *msg*
`cancelled()` Return `True` iff coroutine cancelled	`get_name()` / `set_name(value)` Name shown in `repr()`
`get_context()` Get associated context	`get_coro()` Get wrapped coroutine
`get_stack(limit=None)` / `print_stack(limit=None, file=None)`	Get or print traceback info
`add_done_callback(callback, context=None)`	`remove_done_callback(callback)`

```
async with asyncio.TaskGroup() as tg:
    t1 = tg.create_task(some_coro(…))
    t2 = ts.create_task(another_coro(…))
Only exit the block once all tasks are done
```

If any task in the group fails other than cancelled, the remaining tasks in the group are immediately cancelled. Exceptions raised are wrapped in an `ExceptionGroup` *(or* `BaseExceptionGroup`*), except* `KeyboardInterrupt` *and* `SystemExit`*, which are re-raised.*

Coroutines *(continued)*

```
async for target in iterable:
    body
else:
    else-block
```

Semantically equivalent to code on right

```
iter = (iterable)
iter = type(iter).__aiter__(iter)

while running:
    try:
        target = await type(iter).__anext__(iter)
    except StopAsyncIteration:
        running = False
    else:
        body
else:
    else-block
```

```
manager = (expr)
aenter = type(manager).__aenter__
aexit = type(manager).__aexit__
value = await aenter(manager)
hit_except = False

try:
    target = value
    body
except:
    hit_except = True
    if not await aexit(manager, *sys.exc_info()):
        raise
finally:
    if not hit_except:
        await aexit(manager, None, None, None)
```

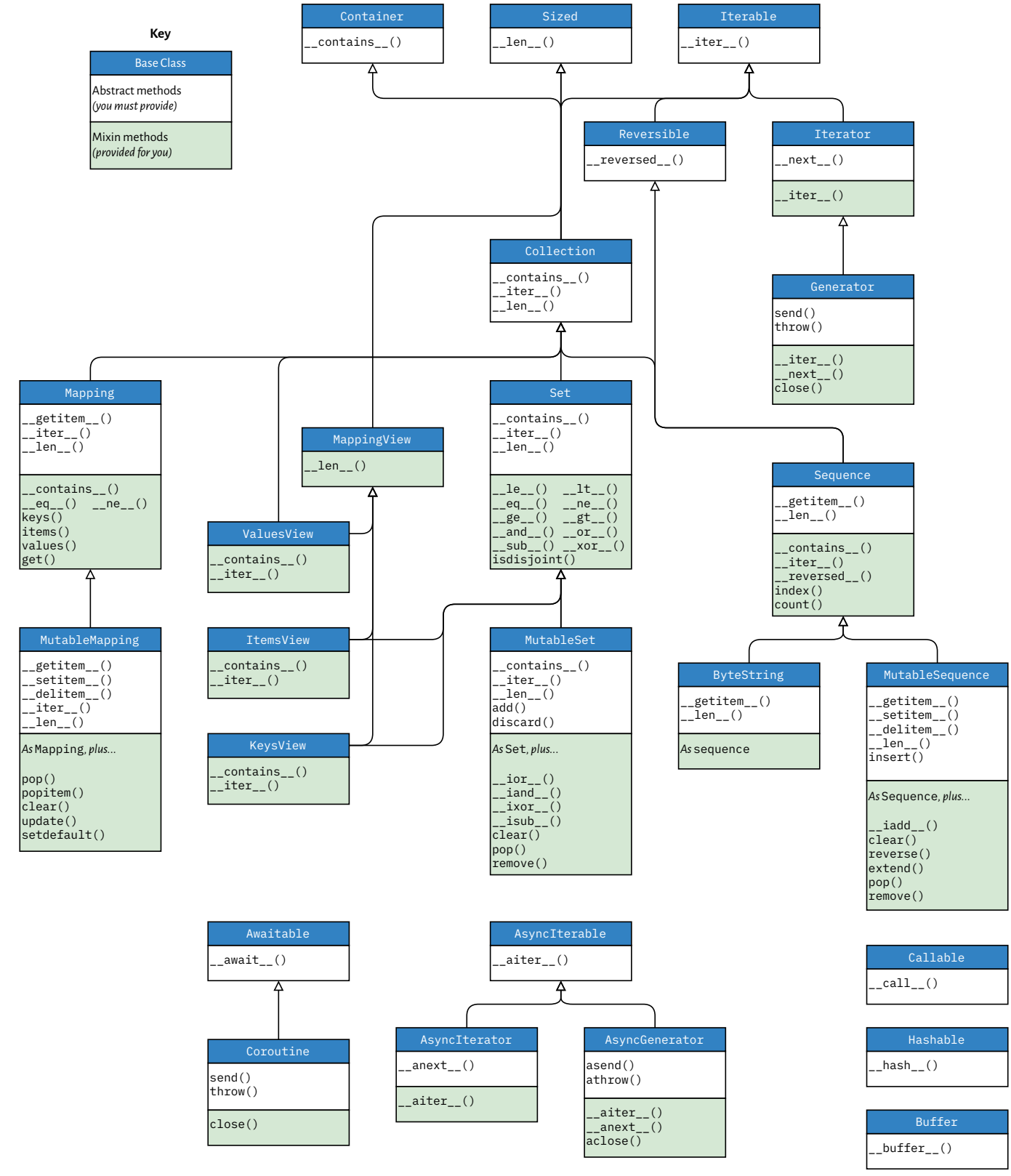
Semantically equivalent to code on left

```
async with expr as target:
    body
```

`asyncio` *functions*	
`sleep(`*delay*`, `*result*`=None)`	Block for *delay* seconds, optionally returning *result* to caller
`gather(*`*awaitables*`, `*return_exceptions*`=False)`	Older equivalent to `TaskGroup` with weaker guarantees
`eager_task_factory(`…`)`	Coroutines start executing immediately on construction
`create_eager_task_factory(`*constructor*`)`	Create factory as above but use *constructor* to create new tasks
`shield(`*awaitable*`)`	Return *awaitable* but ignoring cancellations (`CancelledError` still raised externally)
`timeout(`*delay*`)`	Async context manager which cancels running task after *delay* secs & raises `TimeoutError`
`timeout_at(`*when*`)`	As `timeout()` but specified as absolute time rather than delay
`wait_for(`*awaitable*`, `*timeout*`)`	If *timeout* not `None` & *awaitable* not done in *timeout* secs, cancel it & raise `TimeoutError`
`wait(*`*awaitables*`, `*timeout*`=None, `*return_when*`=ALL_COMPLETED)`	Returns `(done, pending)`—no cancel on timeout
`as_completed(`*awaitables*`, `*timeout*`=None)`	Run *awaitables* concurrently—return iter of awaitables returning results
`run_coroutine_threadsafe(`*coro*`, `*loop*`)`	Submit coroutine *coro* to specified *loop*, safe to call from different OS thread
`current_task(`*loop*`=None)`	Return currently running `Task`, or None—uses current loop by default
`all_tasks(`*loop*`=None)`	Return `set` of all not-yet-done `Task` objects—uses current loop by default
`iscoroutine(`*obj*`)`	Return `True` iff *obj* is a bare coroutine (not a `Task` or `Future` or anything else)

Standard Library: Abstract Base Classes

collections.abc



Standard Library: Collections & Dates and Times

collections

namedtuple(*typename*, *fields*, *rename*=**False**, *defaults*=**None**, *module*=**None**)

_make(*iterable*) New instance from *iterable* **_asdict**() Convert to **dict**

_make(**kwargs**) Return new instances with values of fields updated as per *kwargs*

deque([*iterable*[, *maxlen*]]) If *maxlen* specified, on insertion when full, items dropped from opposite end

append(*item*) **appendleft**(*item*) **clear**() **copy**() **count**(*item*) **extend**(*iterable*)

extendleft(*iterable*) **index**(*item*[, *start*[, *stop*]]) Index of first match within range, or **ValueError**

insert(*index*, *item*) Insert *item* before item at *index*—**insert**(0, *x*) equivalent to **appendleft**(*x*)

pop(*item*) **popleft**(*item*) If missing: **IndexError** **remove**(*item*) If missing: **ValueError**

reverse() Reverse items in-place **rotate**(*n*=1) Rotate items *n* places to right, or left if *n* is negative

ChainMap(***maps*) Single, updateable view of multiple maps—reads check all maps, updates only the first

new_child(*m*=**None**, ****kw*) Return new **ChainMap** with *m* (or empty **dict**) inserted as first map

Counter(*iterable*) Subclass of **dict** for counting hashable objects—can be used as a multimap

elements() Yield each item as many times as its count **total**() Sum of all counts

most_common([*n*]) Return list of (elem, count) for *n* (or all) most common elements in descending count order

subtract(*iterable*) As **update**() but subtracts **update**(*iterable*) Add counts to existing totals

OrderedDict(*iterable*) Less useful since 3.6 **dict** iterates in insert order, but has a couple of methods **dict** lacks

popitem(*last*=**True**) FIFO order if *last* is false **move_to_end**(*key*, *last*=**True**) Move to left end if *last* is false

defaultdict(*default_factory*, ...) Use *default_factory* to construct missing items—all other args passed to **dict**()

datetime

date(*year*, *month*, *day*) **date.today**() **date.fromtimestamp**(*timestamp*)

date.fromordinal(*ord*) **date.fromisoformat**(*str*) **date.fromisocalendar**(*yr*, *week*, *day*)

replace(*year*, *month*, *day*) Omit any to leave value unchanged **to_ordinal**() **isoformat**()

weekday() Monday is **0** **isoweekday**() Monday is **1** **ctime**() **strftime**(*format*)

isocalendar() Return namedtuple with *year*, *week*, *weekday* *Attributes:* **year** **month** **day**

time(*hour*=0, *minute*=0, *second*=0, *microsecond*=0, *tzinfo*=**None**, *fold*=0) *fold of* **1** *means later of two DST duplicates*

time.fromisoformat(*str*) **strftime**(*format*) **utcoffset**() **dst**() **tzname**()

replace(*hour*, *minute*, *second*, *microsecond*, *tzinfo*, *fold*) **isoformat**(*timespec*="auto")

timespec values: "auto" "hours" "minutes" "seconds" "milliseconds" "microseconds"

Attributes: **hour** **minute** **second** **microsecond** **tzinfo** **fold**

datetime(*year*, *month*, *day*, *hour*=0, *minute*=0, *second*=0, *microsecond*=0, *tzinfo*=**None**, *fold*=0)

datetime.today() **datetime.now**(*tz*=**None**) **datetime.utcnow**() *Deprecated, use* **now(utc)**

datetime.fromtimestamp(*timestamp*, *tz*=**None**) **datetime.utcfromtimestamp**(*timestamp*)

datetime.fromordinal(*ord*) **datetime.combine**(*date*, *time*, *tzinfo*=*time*.**tzinfo**)

datetime.fromisoformat(*str*) **datetime.fromisocalendar**(*year*, *week*, *day*)

Standard Library: Dates and Times *(continued)*

`datetime.strptime(`*str*`,` *format*`)` `date()` `time()` `timetz()` `astimezone(`*tz*`=None)`

`replace(`*year*`,` *month*`,` *day*`,` *hour*`,` *minute*`,` *second*`,` *microsecond*`,` *tzinfo*`,` *fold*`)` `utcoffset()`

`dst()` `tzname()` `timetiple()` `utctimetuple()` `toordinal()` `timestamp()`

`weekday()` `isoweekday()` `isocalendar()` `isoformat(`*sep*`="T",` *timespec*`="auto")`

`ctime()` `strftime(`*format*`)`

Attributes: `year` `month` `day` `hour` `minute` `second` `microsecond` `tzinfo` `fold`

`timedelta(`*days*`=0,` *seconds*`=0,` *microseconds*`=0,` *milliseconds*`=0,` *minutes*`=0,` *hours*`=0,` *weeks*`=0)`

`total_seconds()` For units other than seconds use division—e.g. `x / timedelta(hours=1)`

`tzinfo` *Abstract base class, not to be instantiated directly*

`utcoffset(`*datetime*`)` Timezone of *datetime* ignored, used to calculate DST adjustment if applicable

`dst(`*datetime*`)` DST offset at *datetime*, `None` if not known `tzname(`*datetime*`)` Return name at *datetime*

`fromutc(`*datetime*`)` Zone of *datetime* must be `self`, even though time expressed is always treated as UTC

`timezone(`*offset*`,` *name*`=None)` Simple `tzinfo` subclass for fixed offsets from UTC (i.e. no DST)

Class attributes: `utc` Equivalent to `timezone(timedelta(0))`

`zoneinfo`

`ZoneInfo(`*key*`)` `tzinfo` subclass using system zoneinfo, or `tzdata` pkg—example *key*: `America/New_York`

`ZoneInfo.from_file(`*fileobj*`,` *key*`=None)` `ZoneInfo.no_cache(`*key*`)` Bypass constructor's cache

`ZoneInfo.clear_cache(`*only_keys*`=None)` Clear all cached zones, or just those in *only_keys*

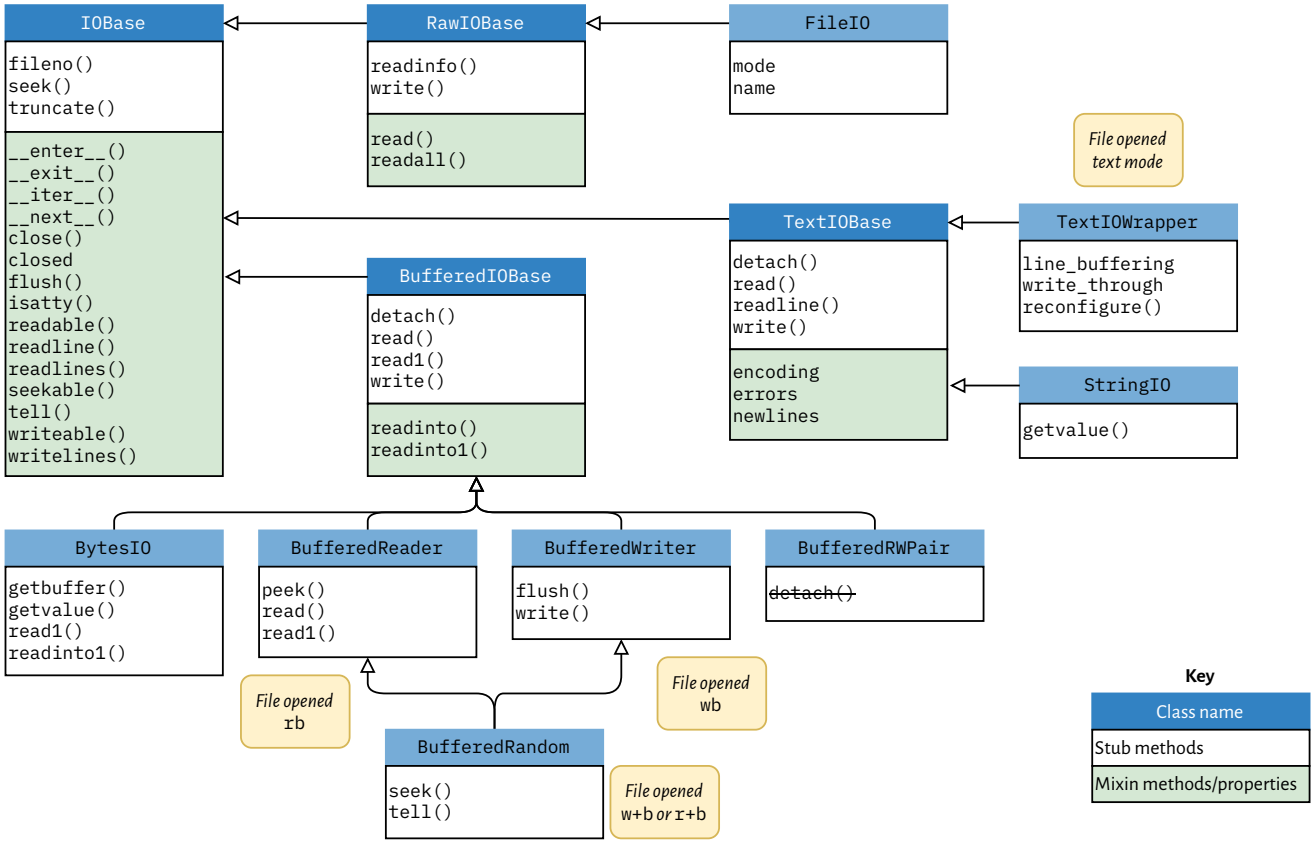
Attributes: `key`

`available_timezones()` Return `set` of zone keys `reset_tzpath(`*to*`=None)` Use `None` to restore default

`strptime()` *and* `strftime()`

Code	*Meaning*	*Example output*	*Code*	*Meaning*	*Example output*
`%a`	Locale abbrev. weekday name	`Tue`	`%p`	Locale equivalent of AM/PM	`PM`
`%A`	Locale full weekday name	`Tuesday`	`%S`	Seconds as a 2-digit decimal	`17`
`%b`	Locale abbrev. month name	`Jan`	`%U`	Week number, Sunday first	`00`
`%B`	Locale full month name	`January`	`%w`	Weekday as 1-digit decimal	`2`
`%c`	Locale format	`Tue Jan 2 15:16:17 2024`	`%W`	Week number, Monday first	`01`
`%d`	Day of month as 2-digit decimal	`02`	`%x`	Locale date representation	`01/02/24`
`%f`	µsec as 6-digits, `strptime()` only	`012345`	`%X`	Locale time representation	`15:16:17`
`%H`	Hour as 24-hour 2-digit decimal	`15`	`%y`	Year as 2-digit decimal	`24`
`%I`	Hour as 12-hour 2-digit decimal	`03`	`%Y`	Year as 4-digit decimal	`2024`
`%j`	Day of the year as 3-digit decimal	`002`	`%z`	Time zone offset	`+0100`
`%m`	Month as a 2-digit decimal	`01`	`%Z`	TIme zone name	`BST`
`%M`	Minute as 2-digit decimal	`16`	`%%`	Literal percent sign	`%`

Standard Library: I/O



IOBase	
close()	Flush and close, further calls ignored
closed	**True** iff stream is closed
fileno()	Return underlying FD, **OSError** if none
flush()	Write buffered data, no-op for unbuffered
isatty()	**True** iff stream is interactive
readable()	If **False**, **read()** raises **OSError**
readline(*size***=-1)**	If +ve, at most *size* bytes read
readlines(*hint***=-1)**	List of lines, stop after *hint* chars
seek(*offset***, *whence*=os.SEEK_SET)**	

seek() *whence* values		
os.SEEK_SET	**0**	*offset* from start of stream
os.SEEK_CUR	**1**	*offset* from current stream position
os.SEEK_END	**2**	*offset* from end of stream

offset should be non-negative for **SEEK_SET**, *is typically negative for* **SEEK_END**, *and may be positive or negative for* **SEEK_CUR**

IOBase	
tell()	Return current stream position
truncate(*size***=None)**	**None**: truncate to current position
Note: truncating doesn't change current stream position	
writeable()	**False**: **read()** & **truncate()** **OSError**
writelines(*lines***)**	Line separators are not added

RawIOBase	
read(*size***=-1)**	Read up to *size* bytes, or to EOF if unspecified May return **None** if non-blocking and no data available
readall()	Read & return all bytes to EOF
readinto(*b***)**	Read bytes into **bytearray** (or similar) *b*
write(*data***)**	Write **bytes** data, return bytes written

BufferedIOBase	
detach()	Separate & return underlying raw stream, or raise **UnsupportedOperation** if not applicable
read(*size***=-1)**	As **RawIOBase**, raise **BlockingIOError** if no data on a non-blocking stream

Standard Library: I/O *(continued)* & OS

BufferedIOBase *(continued)*

`read1(`*size*`=-1)`	As `read()`, at most one system read
`readinto(`*b*`)`	As `RawIOBase`, but `BlockingIOError`
`readinto1(`*b*`)`	As `readinto()`, at most one system read
`write(`*b*`)`	As `RawIOBase`, but `BlockingIOError`
Attribute: `raw`	Underlying raw stream (if defined)

BufferedReader

`BufferedReader(`*raw*`, `*buffer_size*`=DEFAULT_BUFFER_SIZE)`	
`peek(`*size*`=0)`	Return bytes stream without moving pos.
`read(`*size*`=-1)`	`read1(`*size*`=-1)`

BufferedRandom

`BufferedRandom(`*raw*`, `*buffer_size*`=DEFAULT_BUFFER_SIZE)`	

BufferedRWPair

`BufferedRWPair(`*reader*`, `*writer*`, `*buffer_size*`=…)`	

FileIO

`FileIO(`*name*`, `*mode*`='r', `*closefd*`=True, `*opener*`=None)`	
Attribute: `mode`	Mode as given in constructor
Attribute: `name`	File name, or descriptor if no name given

BytesIO

`BytesIO(`*initial_bytes*`=b'')`	
`getbuffer()`	Get readable/writeable view of buffer
`getvalue()`	Return copy of buffer as `bytes`

BufferedWriter

`BufferedWriter(`*raw*`, `*buffer_size*`=DEFAULT_BUFFER_SIZE)`	
`flush()`	May raise `BlockingIOError`
`write(`*b*`)`	

os *(files and dirs)*

`access(`*path*`, `*mode*`, `*dir_fd*`=None, `*effective_ids*`=False, `*follow_symlinks*`=True)`			`True` iff access permitted
mode: `F_OK` Exists	`R_OK` Read	`W_OK` Write	`X_OK` Execute
`chdir(`*path*`)` Change CWD to path	`fchdir(`*path*`)` *fd* must be a directory descriptor		`getcwd()` `getcwdb()`
`chflags(`*path*`, `*flags*`, `*follow_symlinks*`=True)`	Set flags to bitwise-OR of *flags* from `stat` mod.		
`chmod(`*path*`, `*mode*`, `*dir_fd*`=None, `*follow_symlinks*`=True)`	Set mode to bitwise-OR of *mode* from `stat` mod.		
`chown(`*path*`, `*uid*`, `*guid*`, `*dir_fd*`=None, `*follow_symlinks*`=True)`	Set *uid* or *guid* to `-1` to leave unchanged		
`lchflags(`*path*`, `*flags*`)`	`lchmod(`*path*`, `*mode*`)`	`lchown(`*path*`, `*uid*`, `*guid*`)`	Same as `follow_symlinks=False`
`chroot(`*path*`)`	Change root dirtectory to *path*		
`link(`*src*`, `*dst*`, `*src_dir_fd*`=None, `*dst_dir_fd*`=None, `*follow_symlinks*`=True)`	Create *dst* as hard link to *src*		
`symlink(`*src*`, `*dst*`, `*target_is_directory*`=False, `*dir_fd*`=None)`	Create *dst* as soft link to *src*		
`listdir(`*path*`='.')`	`listdrives()`	`listmounts(`*volume*`)`	`listvolumes()`
`mkdir(`*path*`, `*mode*`=0o777, `*dir_fd*`=None)`	If already exists raises `FileExistsError`		
`makedirs(`*path*`, `*mode*`=0o777, `*exist_ok*`=False)`	*mode* used for leaf, `umask` for any other parents created		
`mkfifo(`*path*`, `*mode*`=0o666, `*dir_fd*`=None)`	`mknod(`*path*`, `*mode*`, `*device*`=0, `*dir_fd*`=None)`		
`major(`*device*`)` `minor(`*device*`)`	`makedev(`*major*`, `*minor*`)`	Extract & recreate device maj/min ver from device num	
`pathconf(`*path*`, `*name*`)`	Return value of path config *name* for *path*—for names on platform, see `pathconf_names`		
`readlink(`*path*`, `*dir_fd*`=None)`	Return destination of symlink *path*—if not symlink, raise `OSError`		
`remove(`*path*`, `*dir_fd*`=None)`	If *path* is not a file: `OSError`—use `rmdir()` for directories		
`removedirs(`*path*`)`	Also remove now-empty parents—only `OSError` if fail to remove *path*		

Standard Library: OS *(continued)*

`os` *(files and dirs, continued)*

`rename``(`*src*`, `*dst*`, `*src_dir_fd*`=None, `*dst_dir_fd*`=None)` Whether to error on already extant *dst* is platform-specific

`renames``(`*src*`, `*dst*`)` Creates missing *dst* components like `makedirs()`, then `removedirs()` on *src*

`replace``(`*src*`, `*dst*`, `*src_dir_fd*`=None, `*dst_dir_fd*`=None)` Like `rename()` but silently replaces existing *dst* if possible

`rmdir``(`*path*`, `*dir_fd*`=None)` If *path* is not a directory or isn't empty: `OSError`—use `remove()` for files

`scandir``(`*path*`='.')` Returns iterator of `DirEntry`—skips `.` and `..` and faster than `listdir()` plus `stat()`

Use `with` *on the returned object, or make sure you either exhaust the iterator fully or call* `close()` *on it, to ensure resources are freed*

`DirEntry` *Attributes:* **`name`** Entry's basename **`path`** Full path, including `name`

`inode()` **`is_dir`**`(`*follow_symlinks*`=True)` **`is_file`**`(`*follow_symlinks*`=True)`

`is_symlink()` **`is_junction()`** **`stat`**`(`*follow_symlinks*`=True)`

`stat``(`*path*`, `*dir_fd*`=None, `*follow_symlinks*`=True)` Returns `stat_result` object for specified file

`lstat``(`*path*`, `*dir_fd*`=None)` Equivalent to `stat(…, follow_symlinks=False)`

`stat_result` **`st_mode`** Mode bits **`st_ino`** Inode **`st_dev`** Containing device

`st_nlink` Num. hard links **`st_uid`** Owner UID **`st_gid`** Owner GID **`st_size`** In bytes

`st_atime` Access **`st_mtime`** Content modified **`st_ctime`** Metadata modified

`st_atime_ns` In nanoseconds **`st_mtime_ns`** In nanoseconds **`st_ctime_ns`** In nanoseconds

`st_birthtime` File creation **`st_birthtime_ns`** In nanoseconds *(these two may* `AttributeError`*)*

The following attributes are platform-specific and may not be available—consult the Python docs for more details

`st_blocks` **`st_blksize`** **`st_rdev`** **`st_flags`** **`st_gen`** **`st_fstype`** **`st_rsize`**

`st_creator` **`st_type`** **`st_file_attributes`** **`st_reparse_tag`**

`statvfs(path, dir_fd=None, follow_symlinks=True)` Returns `statvfs_result` object for filesystem

`statvfs_result` **`f_bsize`** Block size **`f_frsize`** Fragment size

`f_blocks` FS size in blocks **`f_bfree`** Num. free blocks **`f_bavail`** Free blocks for unprivileged users

`f_files` Num. inodes **`f_ffree`** Num. free inodes **`f_favail`** Free inodes for unprivileged users

`f_fsid` FS ID (`int`) **`f_flag`** Mount flags **`f_namemax`** Max. filename length, in bytes

`supports_dir_fd` `set` of functions in `os` supporting *dir_fd* parameter on current platform

`supports_effective_ids` `set` of functions in `os` supporting `effective_ids=True` on current platform

`supports_fd` `set` of functions in `os` supporting specifying *path* as an open FD on current platform

`supports_follow_symlinks` `set` of functions in `os` supporting `follow_symlinks=False` on current platform

`sync()` Flush write cache to physical storage

`truncate``(`*path*`, `*length*`)` Truncate *path* to at most *length*

`unlink``(`*path*`, `*dir_fd*`=None)` Semantically equivalent to `remove()`

`utime``(`*path*`, `*times*`=None, [`*ns*`,] `*dir_fd*`=None, `*follow_symlinks*`=True)`

Update access and modified times for *path*—it is an error to specify both *times* and *ns*, and if specified either should be a `tuple` of `(atime, mtime)`, in `int` or `float` seconds for *times* or `int` nanoseconds for *ns*. If neither, use current time as *ns*.

Standard Library: OS *(continued)*

os *(files and dirs, continued)*

walk(*top*, *topdown*=**True**, *onerror*=**None**, *followlinks*=**False**) Generate filenames in directory tree

Yields (*dirpath*, *dirnames*, *filenames*): *dirpath* is **str** of current path, *dirnames* is **list** of subdirs, *filenames* is **list** of files.

When *topdown* is **True**, can modify *dirnames* in-place to prune recursion, *onerror* is function passed **OSError** instance as arg

fwalk(*top*='.', *topdown*=**True**, *onerror*=**None**, *follow_symlinks*=**False**, *dir_fd*=**None**)

As **walk()** but yields (*dirpath*, *dirnames*, *filenames*, *dir_fd*) and supports *dir_fd* itself—*dir_fd* is closed after each iteration

memfd_create(*name*, *flags*=**MFD_CLOEXEC**) Create memory-mapped file in anonymous namespace

flags	**MFD_CLOEXEC**	**MFD_ALLOW_SEALING**	**MFD_HUGETLB**	**MFD_HUGE_SHIFT**	
	MFD_HUGE_MASK	**MFD_HUGE_64KB**	**MFD_HUGE_512KB**	**MFD_HUGE_1MB**	
	MFD_HUGE_2MB	**MFD_HUGE_8MB**	**MFD_HUGE_16MB**	**MFD_HUGE_32MB**	
	MFD_HUGE_256MB	**MFD_HUGE_512MB**	**MFD_HUGE_1GB**	**MFD_HUGE_2GB**	**MFD_HUGE_16GB**

eventfd(*initval*, *flags*=**EFD_CLOEXEC**) Return new file event descriptor—*initval* is initial counter, must be 32-bit

flags	**EFD_CLOEXEC**	**EFD_NONBLOCK**	**EFD_SEMAPHORE**

eventfd_read(*event_fd*) If **EFD_SEMAPHORE** set then return **1** & decrement value, else return value & set to zero

eventfd_write(*event_fd*, *value*) Increment counter by *value*

getxattr(*path*, *attr*, *follow_symlinks*=**True**) Return value of extended filesystem *attr* on *path*

listxattr(*path*=**None**, *follow_symlinks*=**True**) Return **list** of extended filesystem attrs—default to CWD

removexattr(*path*, *attr*, *follow_symlinks*=**True**) Remove extended filesystem *attr* from *path*

setxattr(*path*, *attr*, *value*, *flags*=**0**, *follow_symlinks*=**True**) Set value of extended filesystem *attr* on *path*

flags	**XATTR_REPLACE**	Error if attr not already set	**XATTR_CREATE**	Error if attr already set

stat

Flags that can be used with **chmod()**: **S_ISUID** Set UID **S_ISGID** Set GID **S_ISVTX** Sticky bit

User: **S_IRWXU** Mask — **S_IRUSR** / **S_IWUSR** / **S_IXUSR** Group: **S_IRWXG** — **S_IRGRP** / **S_IWGRP** / **S_IXGRP**

Other: **S_IRWXO** — **S_IROTH** / **S_IWOTH** / **S_IXOTH** **S_ENFMT** V7 aliases: **S_IREAD** / **S_IWRITE** / **S_IEXEC**

lags that can be used with **chflags()**: **UF_NODUMP** / **UF_IMMUTABLE** / **UF_APPEND** / **UF_OPAQUE** / **UF_NOUNLINK**

UF_COMPRESSED / **UF_HIDDEN** / **SF_ARCHIVED** / **SF_IMMUTABLE** / **SF_APPEND** / **SF_NOUNLINK** / **SF_SNAPSHOT**

Test mode for specific file types: ***S_ISDIR***(*mode*) ***S_ISCHR***(*mode*) ***S_ISBLK***(*mode*) ***S_ISREG***(*mode*)

S_ISFIFO(*mode*) ***S_ISLNK***(*mode*) ***S_ISSOCK***(*mode*) ***S_ISDOOR***(*mode*) ***S_ISPORT***(*mode*) ***S_ISWHT***(*mode*)

S_IMODE(*mode*) Part of mode used by **chmod()** **S_IFMT**(*mode*) Part of mode checked by functions above

Indexes into the 10-tuple returned by **stat()**, **fstat()** *and* **lstat()**:

ST_MODE / **ST_INO** / **ST_DEV** / **ST_NLINK** / **ST_UID** / **ST_GID** / **ST_SIZE** / **ST_ATIME** / **ST_MTIME** / **ST_CTIME**

Flags used in **ST_MODE** *field: (generally more readable to use the functions above instead, however):*

S_IFSOCK / **S_IFLNK** / **S_IFREG** / **S_IFBLK** / **S_IFDIR** / **S_IFCHR** / **S_IFIFO** / **S_IFDOOR** / **S_IFPOR** / **S_IFWHT**

Other functions:

filemode(*mode*) Convert to string of form **"-rwxrwxrwx"**

Standard Library: Iterators for Efficient Looping

`itertools` *(terminating on the shortest input sequence)*	
`accumulate(`*iterable*`, `*function*`=operator.add, `*initial*`=None))`	Using (e.g.) `add` yields P_0, P_0+P_1, P_0+P_1+P_2, …
`batched(`*iterable*`, `*n*`)`	Yield *n*-tuples with items from *iterable* in batches (final one may be less than *n*)
`chain(*`*iterables*`)`	Yields from each iterator until fully exhausted before moving on to the next
`chain.from_iterable(`*iterable*`)`	As `chain()` but *iterable* is lazily evaluated to yield iterables to consume
`compress(`*data*`, `*selectors*`)`	Yields items from *data* only when corresponding item from *selectors* is true
`dropwhile(`*predicate*`, `*iterable*`)`	Drops items from *iterable* while 1-argument *predicate* returns true, then yield the rest
`filterfalse(`*predicate*`, `*iterable*`)`	Yield items from *iterable* where 1-argument *predicate* returns false
`groupby(`*iterable*`, `*key*`=None)`	Pass items to 1-argument *key* and yield contiguous groups with the same key
`islice(`*iterable*`, `*stop*`)` `islice(`*iterable*`, `*start*`, `*stop*`[, `*step*`]])`	As sequence slicing, but for iterators
`pairwise(`*iterable*`)`	Yields overlapping consecutive pairs: (P_0, P_1), (P_1, P_2), (P_2, P_3), …
`starmap(`*function*`, `*iterable*`)`	Each item from *iterable* is a sequence of arguments passed to *function*
`takewhile(`*predicate*`, `*iterable*`)`	Yields items from *iterable* while 1-argument *predicate* returns true, then stop
`tee(`*iterable*`, `*n*`=2)`	Return *n* independent iterators on the underlying *iterable*
`zip_longest(*`*iterables*`, `*fillvalue*`=None)`	As builtin `zip()` but missing values from shorter *iterables* filled with *fillvalue*
`itertools` *(infinite iterators)*	
`count(`*start*`=0, `*step*`=1)`	As `range()` but iterates indefinitely—yields *start* first, then increments by *step* each time
`cycle(`*iterable*`)`	Yields items from *iterable*, and also takes a copy so it can then repeat them indefinitely
`repeat(`*object*`, `[*times*]`)`	Yields *object* indefiitely, or a maximum of *times* times if specified
`itertools` *(combinatorics)*	
`product(*`*iterables*`, `*repeat*`=1)`	Yields `tuple`s which cover the Cartesian join of supplied iterables
`permutations(`*iterable*`, `*r*`=None)`	Yields `tuple`s which cover all permutations of choosing *r* items, defaults to all
`combinations(`*iterable*`, `*r*`)`	Yields `tuple`s which cover all combinations of choosing *r* items
`combinations_with_replacement(`*iterable*`, `*r*`)`	As above but same item may be chosen more than once