

Package: pyrt (via r-universe)

October 27, 2026

Title Runtime Support for Converted 'Python' Packages

Version 0.1.0

Description Provides the 'Python' runtime semantics required by 'R' packages generated with 'pyport', including reference-semantics containers, class and method-resolution-order dispatch, exception hierarchies, iteration and slicing helpers, arbitrary-precision integer arithmetic, and bit-exact random number generation.

License MIT + file LICENSE

URL https://github.com/laueehow1986-hub/r_python_converter

BugReports https://github.com/laueehow1986-hub/r_python_converter/issues

Encoding UTF-8

Depends R (>= 4.3.0)

Imports gmp (>= 0.7-1), R6 (>= 2.5.1), stats, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.3

NeedsCompilation yes

Author Yee How Lau [aut, cre, cph]

Maintainer Yee How Lau <laueehow1986@gmail.com>

Repository <https://cran.r-universe.dev>

Date/Publication 2026-10-27 18:49:57 UTC

RemoteUrl <https://github.com/cran/pyrt>

RemoteRef HEAD

RemoteSha 1420915204336b4b4e776911bdf2a2cf342ed174

Contents

py_builtin	2
py_class	3
py_eq	5
py_exception	6
py_int	7
py_iter	8
py_len	10
py_ndarray	11
py_none	12
py_random_generator	12
py_slice	13
py_split_varargs	14
py_stdlib	15
py_str	16
Index	18

py_builtin	<i>Retrieve a Python built-in function or exception constructor by name</i>
------------	---

Description

Looks up a Python built-in function (len, range, enumerate, zip, reversed, iter, next, int, float, bool, str, repr, ascii, abs, divmod, pow, min, max, sum, all, any, sorted, list, tuple, dict, set, map, filter, isinstance, issubclass, getattr, setattr, hasattr, delattr, super, property, staticmethod, classmethod, ord, chr, hex, oct, bin, round, print, callable) or built-in exception constructor (ValueError, TypeError, KeyError, IndexError, StopIteration, RuntimeError, ZeroDivisionError, AttributeError, LookupError, AssertionError, NotImplementedError, OSError, Exception, BaseException).

Usage

```
py_builtin(name)

py_builtin_names()
```

Arguments

name	Character string naming the Python built-in.
------	--

Value

A callable R function or pyrt_class representing the built-in.

Examples

```
py_builtin("len")(py_list(list(1L, 2L, 3L)))
py_builtin("sum")(py_range(5L))
```

py_class

Python class system: C3 MRO, super(), descriptors, and with context managers

Description

py_class() defines a Python class with C3 method-resolution order (matching Python's multiple-inheritance rules), dunder hooks (`__init__`, `__call__`, `__repr__`, `__str__`, `__eq__`, `__lt__`, `__len__`, `__getitem__`, `__iter__`, `__enter__`, `__exit__`, etc.), and descriptors (`py_property()`, `py_staticmethod()`, `py_classmethod()`). Calling a `pyrt_class` (or via `$new(...)`) constructs an instance (`pyrt_instance`) with reference semantics. `py_super(cls, self)` returns a proxy that resolves attributes starting after `cls` in `self`'s MRO. `py_with(mgr, body)` implements Python's with statement (`__enter__` / `__exit__`, including exception suppression when `__exit__` returns a truthy value).

Usage

```
py_class(name, bases = list(), dict = list())

py_class(name, bases = list(), dict = list())

py_property(fget = NULL, fset = NULL, fdel = NULL)

py_staticmethod(fn)

py_classmethod(fn)

py_getattr(obj, attr, default = missing_sentinel)

py_setattr(obj, attr, value)

py_delattr(obj, attr)

py_hasattr(obj, attr)

py_super(cls, self)

py_isinstance(obj, classinfo)

py_issubclass(cls, classinfo)

py_call(obj, args = list(), kwargs = list())

py_with(mgr, body)
```

Arguments

name	Class name string.
bases	List of base pyrt_class objects (defaults to object).
dict	Named list of methods, properties, and class attributes.
fget, fset, fdel	Getter, setter, and deleter functions for py_property().
fn	Function wrapped by py_staticmethod() or py_classmethod().
obj	Object whose attribute is accessed/mutated by py_getattr(), py_setattr(), py_delattr(), or py_hasattr().
attr	Attribute name string.
default	Optional fallback returned by py_getattr() when the attribute does not exist.
value	Value assigned by py_setattr().
cls, self	Current pyrt_class and pyrt_instance passed to py_super(), or tested by py_isinstance() / py_issubclass().
classinfo	A pyrt_class, class name string, or list / py_tuple of classes for py_isinstance() and py_issubclass().
args, kwargs	Positional argument list and keyword argument list/dict passed to py_call().
mgr	Context manager object implementing __enter__ and __exit__.
body	Function function(entered_val) ... executed inside py_with().

Value

py_class() returns a callable pyrt_class; instantiating it returns a pyrt_instance.

Examples

```
Counter <- py_class("Counter", dict = list(
  `__init__` = function(self, start = 0L) {
    py_setattr(self, "n", start)
  },
  inc = function(self, step = 1L) {
    py_setattr(self, "n", py_getattr(self, "n") + step)
    py_getattr(self, "n")
  }
))
c1 <- Counter(10L)
c1$inc(5L)
```

Description

py_eq(), py_ne(), py_lt(), py_le(), py_gt(), py_ge() implement Python's value equality and ordering rules (including `1 == 1.0 == True`, `gmp::bigz` comparisons, structural equality on `py_list` / `py_tuple` / `py_dict` / `py_set`, and dunder hooks `__eq__`, `__lt__`, etc.). `py_is()` tests object identity (for reference-semantics objects, whether two bindings point to the same underlying environment). `py_in()` tests membership (`x in container`), dispatching to `__contains__` when present.

Usage

```
py_eq(x, y)
```

```
py_ne(x, y)
```

```
py_lt(x, y)
```

```
py_le(x, y)
```

```
py_gt(x, y)
```

```
py_ge(x, y)
```

```
py_is(x, y)
```

```
py_in(x, y)
```

Arguments

x, y Operands.

Value

A single logical (TRUE or FALSE), or a `pyrt_ndarray` for array comparisons.

Examples

```
py_eq(1L, TRUE)
py_eq(py_list(list(1L, 2L)), py_list(list(1.0, 2.0)))
```

Description

py_exception() creates an R condition object whose S3 classes follow Python's built-in exception hierarchy (KeyError inherits from LookupError, Exception, BaseException, etc.). py_raise() signals the exception (optionally chaining __cause__ via from =). py_isinstance_exc() tests whether an exception matches a class name (or list/tuple of class names). py_try() implements Python's full try / except / else / finally semantics (where else runs only when expr finishes without raising, and finally always runs).

Usage

```
py_exception(type = "Exception", ..., value = NULL)

py_raise(exc = NULL, ..., cause = NULL)

py_isinstance_exc(exc, classes)

py_try(expr, handlers = list(), orelse = NULL, finalbody = NULL)
```

Arguments

type	Exception class name (such as "ValueError", "KeyError", "ZeroDivisionError", "StopIteration") or a pyrt_class exception subclass.
...	Positional arguments stored in exc\$args and formatted into the condition message.
value	Optional single payload (notably StopIteration(value)).
exc	An exception condition or class name passed to py_raise(), or a condition tested by py_isinstance_exc(). If NULL, py_raise() re-raises the currently active Python exception.
cause	Optional chained cause (raise exc from cause).
classes	Character vector, list, or py_tuple of exception class names or pyrt_class objects.
expr	Expression evaluated inside the try block.
handlers	Named list of handler functions (function(e) ...). A handler named "*" acts as a bare except:. Multiple exception types can be separated with " " (e.g. "KeyError IndexError").
orelse	Optional function function() ... executed if and only if expr did not raise an exception.
finalbody	Optional function function() ... always executed on exit.

Value

py_exception() returns a condition of class "pyrt_exception"; py_isinstance_exc() returns a single logical; py_try() returns the value of expr, the matched handler, or or_else().

Examples

```
e <- py_exception("KeyError", "missing")
py_isinstance_exc(e, "LookupError")
py_try(
  py_raise("IndexError", "out of range"),
  handlers = list(LookupError = function(err) err$type)
)
```

py_int

Python int() constructor and exact integer / operator semantics

Description

py_int() converts numbers, booleans, or strings (in base 0 or 2..36, with optional _ separators) to a Python integer (integer when in 32-bit signed range, gmp::bigz when larger). py_floordiv(), py_mod(), py_divmod(), py_pow(), py_bitand(), py_bitor(), py_bitxor(), py_invert(), py_lshift(), and py_rshift() implement Python's exact numeric and two's-complement bit-wise semantics (including floor division and modulo toward -Inf, and raises ZeroDivisionError on division by zero).

Usage

```
py_int(x = 0L, base = NULL)

py_add(x, y)

py_sub(x, y)

py_mul(x, y)

py_truediv(x, y)

py_floordiv(x, y)

py_mod(x, y)

py_divmod(x, y)

py_pow(x, y, mod = NULL)

py_bitand(x, y)
```

```

py_bitor(x, y)

py_bitxor(x, y)

py_invert(x)

py_lshift(x, y)

py_rshift(x, y)

py_neg(x)

py_pos(x)

py_abs(x)

py_matmul(x, y)

```

Arguments

x, y	Operands (integers, <code>gmp::bigz</code> , doubles, booleans, containers, or <code>pyrt_instance</code> objects implementing dunder methods such as <code>__add__</code>).
base	Integer base for string conversion in <code>py_int()</code> (0L or 2..36).
mod	Optional modulus for 3-argument <code>pow(x, y, mod)</code> .

Value

An R integer, `gmp::bigz`, double, complex, or dunder result.

Examples

```

py_floordiv(-7L, 2L)
py_mod(-7L, 2L)
py_bitand(-5L, 3L)
py_rshift(-8L, 2L)

```

py_iter

Python iteration protocol (iter, next), generators, and range

Description

`py_iter()` returns a stateful `pyrt_iterator` over any Python container, string, `py_range()`, generator, R vector/list, or custom class implementing `__iter__` (or `__getitem__`). `py_next()` advances an iterator by one step, returning default if supplied or raising `StopIteration` when exhausted. `py_generator()` wraps a step function (`function(yield_val) ...`) into a stateful generator supporting `next()`, `$send()`, and `$close()`. `py_for()` executes a loop body for each item of an iterable and runs `orelse()` if the loop finished without break.

Usage

```

py_iter(x)

py_next(x, default = missing_sentinel)

py_collect(x)

py_iter_step(x)

py_bounce(thunk)

py_yield(value = NULL, cont = NULL)

py_stop_iteration(value = NULL)

py_generator(step_fn)

py_break_signal()

py_for(x, body, or_else = NULL)

py_range(start, stop = NULL, step = 1L)

py_range_seq(start, stop = NULL, step = 1L)

py_enumerate(x, start = 0L)

py_zip(...)

py_reversed(x)

```

Arguments

x	An iterable or iterator.
default	Optional fallback returned by <code>py_next()</code> when x is exhausted instead of raising <code>StopIteration</code> .
thunk	Zero-argument continuation thunk for <code>py_bounce()</code> .
value	Value yielded by <code>py_yield()</code> or returned via <code>py_stop_iteration()</code> .
cont	Optional continuation function function(sent = NULL) for CPS-lowered generators.
step_fn	A zero-argument function that returns <code>py_yield(value)</code> to yield an item or <code>py_stop_iteration(ret)</code> when the generator returns.
body	Function function(item) called on each element in <code>py_for()</code> . Returning <code>py_break_signal()</code> exits the loop early without running <code>or_else</code> .
or_else	Optional zero-argument function run when <code>py_for()</code> completes normally without breaking.

start, stop, step	Integer range bounds for <code>py_range()</code> (when only one argument is given, <code>py_range(stop)</code> starts at 0L).
...	Iterables passed to <code>py_zip()</code> .

Value

`py_iter()` and `py_generator()` return `pyrt_iterator` objects; `py_next()` returns the next element; `py_collect()` returns an R list of all remaining elements.

Examples

```
it <- py_iter(py_range(1L, 6L, 2L))
py_next(it)
py_collect(it)
```

py_len	<i>Python len() and container constructors (list, tuple, dict, set)</i>
--------	---

Description

Reference-semantics containers that mirror Python's built-in list, tuple, dict, and set types (with 0-based indexing, negative indices, slices, and Python methods such as `$append()`, `$extend()`, `$pop()`, `$sort()`, `$get()`, `$setdefault()`, `$update()`, `$items()`, `$union()`, `$intersection()`, etc.). Mutating a `py_list`, `py_dict`, or `py_set` updates it in place across all variables referencing it.

Usage

```
py_len(x)

py_tuple(x = NULL)

py_list(x = NULL)

py_dict(x = NULL, ...)

py_set(x = NULL)
```

Arguments

x	An object passed to <code>py_len()</code> , or initial elements / iterable for <code>py_list()</code> , <code>py_tuple()</code> , <code>py_set()</code> , or <code>py_dict()</code> .
...	Additional named or unnamed arguments (for <code>py_dict(...)</code> , keyword pairs to insert).

Value

A container object (pyrt_list, pyrt_tuple, pyrt_dict, or pyrt_set), or an integer length from py_len().

Examples

```
xs <- py_list(list(10L, 20L, 30L))
alias <- xs
xs$append(40L)
py_len(alias)
py_getitem(alias, -1L)
```

py_ndarray	<i>Create a Python-compatible ndarray backed by an R vector/matrix/array</i>
------------	--

Description

Wraps an R atomic vector, matrix, or array in a reference-semantics pyrt_ndarray object with 0-based indexing, .shape tuple, .ndim, .size, .dtype, .T transpose, .reshape(), .flatten(), .tolist(), .sum(), .mean(), .std(), .var(), .min(), .max(), .argmin(), .argmax(), .cumsum(), .dot(), and elementwise arithmetic/comparison operators with NumPy broadcasting semantics.

Usage

```
py_ndarray(data, dtype = NULL)
```

Arguments

data	An R vector, matrix, array, pyrt_list, pyrt_tuple, or pyrt_ndarray.
dtype	Optional dtype string ("float64", "int32", "bool").

Value

A pyrt_ndarray environment.

Examples

```
np <- py_stdlib("numpy")
a <- np$array(list(list(1, 2, 3), list(4, 5, 6)))
a$shape
a$sum(axis = 0L)
```

py_none	<i>Python None, Ellipsis, and truthiness helpers</i>
---------	--

Description

Foundational singletons and truthiness rules used by R code generated with **pyport**. Phase 1 expands **pyrt** with reference-semantics containers (py_list, py_dict, py_set, py_tuple), the Python class/MRO system, exceptions, iteration, slicing, and bit-exact random.

Usage

```
py_none()
is_py_none(x)
py_ellipsis()
is_py_ellipsis(x)
py_bool(x)
```

Arguments

x Any R object.

Value

py_none() and py_ellipsis() return singleton objects; is_py_none(), is_py_ellipsis(), and py_bool() return a single logical (TRUE or FALSE).

Examples

```
is_py_none(py_none())
py_bool(0L)
py_bool("non-empty")
```

py_random_generator	<i>Bit-exact CPython random.Random generator and module-level py_random</i>
---------------------	---

Description

`py_random_generator(seed)` creates an independent `random.Random` instance whose Mersenne Twister (MT19937) state, version = 2 seeding (for `int`, `gmp::bigz`, `bool`, `str`, and `bytes`), `random()`, `getrandbits()`, `_randbelow()`, `randrange()`, `randint()`, `choice()`, `shuffle()`, `sample()`, `choices()`, `uniform()`, `triangular()`, `gauss()`, `normalvariate()`, `expovariate()`, `lognormvariate()`, `getstate()`, and `setstate()` match CPython's `random` module bit-for-bit. R's own RNG is never touched. `py_random` is a pre-instantiated module-like object exposing the same functions at top level (`py_random$seed(42L)`, `py_random$random()`, etc.).

Usage

```
py_random_generator(a = NULL)
```

```
py_random
```

Arguments

`a` Optional seed (`NULL`, integer, `gmp::bigz`, logical, character string, or raw bytes).

Format

An object of class `pyrt_Random` (inherits from `pyrt_container`) of length 20.

Value

`py_random_generator()` returns a `pyrt_Random` environment; `py_random` is the default module-level generator.

Examples

```
rng <- py_random_generator(12345L)
rng$random()
rng$randint(1L, 100L)
rng$getrandbits(100L)
```

py_slice

Python slice() and 0-based subscript helpers

Description

`py_slice(start, stop, step)` constructs a Python slice object (including its `$indices(length)` method). `py_getitem()`, `py_setitem()`, and `py_delitem()` implement 0-based indexing, negative indices, slices with arbitrary positive or negative steps (including extended slice assignment and deletion on `py_list`), dictionary key lookup, string slicing, and `__getitem__` / `__setitem__` / `__delitem__` dunder dispatch on custom classes. `py_unpack()` implements Python's sequence and starred unpacking (`a, *b, c = seq`), raising `ValueError` when the item count does not match.

Usage

```

py_slice(start = NULL, stop = NULL, step = NULL)

py_getitem(x, key)

py_setitem(x, key, value)

py_delitem(x, key)

py_unpack(x, n_before, n_after = 0L, has_star = FALSE)

```

Arguments

start, stop, step	Slice bounds (NULL for omitted bounds; step cannot be 0L).
x	Target object (container, string, vector, or pyrt_instance).
key	0-based integer index, pyrt_slice, dictionary key, or py_tuple of indices.
value	Replacement value for py_setitem().
n_before	Number of mandatory targets before a starred target (or the exact target count when has_star = FALSE).
n_after	Number of mandatory targets after the starred target.
has_star	Logical; TRUE when the target list contains *rest.

Value

py_slice() returns a pyrt_slice object; py_getitem() returns the indexed element or sliced sub-container; py_setitem() and py_delitem() mutate reference containers in place and return the modified object invisibly; py_unpack() returns a flat R list of unpacked values.

Examples

```

xs <- py_list(list("a", "b", "c", "d", "e"))
py_getitem(xs, -1L)
py_getitem(xs, py_slice(1L, 4L, 2L))
py_setitem(xs, py_slice(1L, 3L), list("B", "C", "CC"))
xs

```

py_split_varargs	<i>Runtime helpers for transpiled *args/**kwargs, assert, and signals</i>
------------------	---

Description

py_split_varargs() partitions an R list(...) into a py_tuple of extra positional arguments (\$args) and a py_dict of extra keyword arguments (\$kwargs), raising TypeError when unexpected positional or keyword arguments are passed. py_return_signal(), py_continue_signal(), is_py_return(), is_py_continue(), and is_py_break() propagate return/break/continue out of py_try() and py_with() closures. py_assert() raises AssertionError when test is falsy under py_bool().

Usage

```
py_split_varargs(dots, vararg = TRUE, kwarg = TRUE)
```

```
py_return_signal(value = py_none())
```

```
is_py_return(x)
```

```
py_continue_signal()
```

```
is_py_continue(x)
```

```
is_py_break(x)
```

```
py_assert(test, msg = NULL)
```

Arguments

dots	R <code>list(...)</code> collected from a function's ... parameter.
vararg	Logical; whether the Python function accepts <code>*args</code> .
kwarg	Logical; whether the Python function accepts <code>**kwargs</code> .
value	Return value wrapped in <code>py_return_signal()</code> .
x	Object tested by <code>is_py_return()</code> , <code>is_py_continue()</code> , or <code>is_py_break()</code> .
test	Condition tested by <code>py_assert()</code> .
msg	Optional message evaluated and passed to <code>AssertionError</code> if test is falsy.

Value

`py_split_varargs()` returns `list(args = py_tuple, kwargs = py_dict)`.

py_stdlib

Retrieve a Python standard-library module implemented in pyrt

Description

Returns an environment implementing the requested Python standard-library module (math, itertools, functools, operator, collections, re, json, copy, bisect, heapq, string, datetime, dataclasses, enum, random, sys).

Usage

```
py_stdlib(name)
```

```
py_stdlib_names()
```

Arguments

name Module name string (e.g. "math", "itertools", "collections").

Value

A pyrt_module environment.

Examples

```
m <- py_stdlib("math")
m$gcd(48L, 18L)
it <- py_stdlib("itertools")
py_collect(it$islice(it$count(10L, 2L), 4L))
```

py_str	<i>Python str(), repr(), ascii(), format(), and str methods</i>
--------	---

Description

py_str(), py_repr(), and py_ascii() format values using Python's string and representation rules (dispatching to __str__ / __repr__ on custom instances). py_format(value, spec) implements Python's Format Specification Mini-Language ([[fill]align][sign][#][0][width][grouping_option][.precision]). py_fstring(...) evaluates an f-string's parts. py_str_method(s, method) dispatches Python str methods (split, rsplit, splitlines, join, strip, lstrip,rstrip, replace, find, rfind, index, rindex, count, startswith, endswith, upper, lower, capitalize, title, swapcase, zfill, center, ljust, rjust, partition, rpartition, removeprefix, removesuffix, isdigit, isalpha, isalnum, isspace, isupper, islower, format).

Usage

```
py_str(x = "")

py_repr(x)

py_ascii(x)

py_format(value, spec = "", conversion = -1L)

py_fstring(...)

py_str_method(s, method)
```

Arguments

x, value Value to convert or format.

spec Format specification string (e.g. ">10.2f", "08x", " ,d", "+.3e", "^12s").

conversion	Integer conversion code for f-string FormattedValue (-1L = none, 115L = !s, 114L = !r, 97L = !a).
...	Parts passed to py_fstring().
s	A single character string for py_str_method().
method	Name of the Python str method.

Value

A character string (or container / logical / integer for str methods that return lists, tuples, booleans, or indices).

Examples

```
py_format(3.14159, ">+10.2f")
py_format(255L, "#06x")
py_format(1234567L, ",d")
py_str_method(" hello, world ", "strip")()
```

Index

* datasets

- py_random_generator, 12
- is_py_break (py_split_varargs), 14
- is_py_continue (py_split_varargs), 14
- is_py_ellipsis (py_none), 12
- is_py_none (py_none), 12
- is_py_return (py_split_varargs), 14
- py_abs (py_int), 7
- py_add (py_int), 7
- py_ascii (py_str), 16
- py_assert (py_split_varargs), 14
- py_bitand (py_int), 7
- py_bitor (py_int), 7
- py_bitxor (py_int), 7
- py_bool (py_none), 12
- py_bounce (py_iter), 8
- py_break_signal (py_iter), 8
- py_builtin, 2
- py_builtin_names (py_builtin), 2
- py_call (py_class), 3
- py_class, 3
- py_classmethod (py_class), 3
- py_collect (py_iter), 8
- py_continue_signal (py_split_varargs), 14
- py_delattr (py_class), 3
- py_delitem (py_slice), 13
- py_dict (py_len), 10
- py_divmod (py_int), 7
- py_ellipsis (py_none), 12
- py_enumerate (py_iter), 8
- py_eq, 5
- py_exception, 6
- py_floordiv (py_int), 7
- py_for (py_iter), 8
- py_format (py_str), 16
- py_fstring (py_str), 16
- py_ge (py_eq), 5
- py_generator (py_iter), 8
- py_getattr (py_class), 3
- py_getitem (py_slice), 13
- py_gt (py_eq), 5
- py_hasattr (py_class), 3
- py_in (py_eq), 5
- py_int, 7
- py_invert (py_int), 7
- py_is (py_eq), 5
- py_isinstance (py_class), 3
- py_isinstance_exc (py_exception), 6
- py_issubclass (py_class), 3
- py_iter, 8
- py_iter_step (py_iter), 8
- py_le (py_eq), 5
- py_len, 10
- py_list (py_len), 10
- py_lshift (py_int), 7
- py_lt (py_eq), 5
- py_matmul (py_int), 7
- py_mod (py_int), 7
- py_mul (py_int), 7
- py_ndarray, 11
- py_ne (py_eq), 5
- py_neg (py_int), 7
- py_next (py_iter), 8
- py_none, 12
- py_pos (py_int), 7
- py_pow (py_int), 7
- py_property (py_class), 3
- py_raise (py_exception), 6
- py_random (py_random_generator), 12
- py_random_generator, 12
- py_range (py_iter), 8
- py_range_seq (py_iter), 8
- py_repr (py_str), 16
- py_return_signal (py_split_varargs), 14
- py_reversed (py_iter), 8
- py_rshift (py_int), 7

`py_set` (`py_len`), 10
`py_setattr` (`py_class`), 3
`py_setitem` (`py_slice`), 13
`py_slice`, 13
`py_split_varargs`, 14
`py_staticmethod` (`py_class`), 3
`py_stdlib`, 15
`py_stdlib_names` (`py_stdlib`), 15
`py_stop_iteration` (`py_iter`), 8
`py_str`, 16
`py_str_method` (`py_str`), 16
`py_sub` (`py_int`), 7
`py_super` (`py_class`), 3
`py_truediv` (`py_int`), 7
`py_try` (`py_exception`), 6
`py_tuple` (`py_len`), 10
`py_unpack` (`py_slice`), 13
`py_with` (`py_class`), 3
`py_yield` (`py_iter`), 8
`py_zip` (`py_iter`), 8