syn Documentation

Release 0.0.15

Matt Bodenhamer

Apr 22, 2017

Contents

`syn` is a Python library and command-line tool that will provide metaprogramming, typing, and compilation facilities.

This project is currently in pre-alpha. Initial usage documentation and examples will be provided when the project moves to alpha in release `0.1.0`. The target date for `0.1.0` is Q3 2017.

Contents:

CHAPTER 1

syn package

Subpackages

syn.base package

Subpackages

syn.base.a package

Submodules

syn.base.a.base module

class syn.base.a.base.**Base**(**args*, ***kwargs*)
Bases: object

to_dict(*exclude=()*)
Convert the object into a dict of its declared attributes.

May exclude certain attributes by listing them in exclude.

validate()
Raise an exception if the object is missing required attributes, or if the attributes are of an invalid type.

syn.base.a.meta module

class syn.base.a.meta.**Attr**(*typ=None*, *default=None*, *doc=''*, *optional=False*, *init=None*)
Bases: object

class syn.base.a.meta.**Attrs**(**args*, ***kwargs*)
Bases: *syn.base_utils.dict.UpdateDict*

class syn.base.a.meta.**Meta**(*clsname*, *bases*, *dct*)
Bases: type

syn.base.a.meta.**preserve_attr_data**(*A*, *B*)
Preserve attr data for combining B into A.

Module contents

syn.base.b package

Submodules

syn.base.b.base module

class syn.base.b.base.**Base**(***kwargs*)
Bases: object

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

classmethod coerce(*value*, ***kwargs*)

copy(***kwargs*)

classmethod from_mapping(*value*)

classmethod from_object(*obj*)

classmethod from_sequence(*seq*)

istr(*pretty=False*, *indent=0*, *toplevel=False*)
Returns a string that, if evaluated, produces an equivalent object.

pretty(*indent=0*)
Returns a pretty-printed version if istr().

to_dict(***kwargs*)
Convert the object into a dict of its declared attributes.

May exclude certain attribute groups by listing them in exclude=[].

May include certain attribute groups (to the exclusion of all others) by listing them in include=[].

to_tuple(***kwargs*)
Convert the object into a tuple of its declared attribute values.

validate()
Raise an exception if the object is missing required attributes, or if the attributes are of an invalid type.

class syn.base.b.base.**BaseType**(*obj*)
Bases: *syn.types.a.base.Type*

attrs(***kwargs*)

type
alias of *Base*

syn.base.b.base.**init_hook**(*f*)

syn.base.b.base.**coerce_hook**(*f*)

syn.base.b.base.**setstate_hook**(*f*)

class syn.base.b.base.**Harvester**
Bases: object

syn.base.b.examine module

syn.base.b.examine.**check_idempotence**(*obj*)

syn.base.b.meta module

class syn.base.b.meta.**Attr**(**args*, ***kwargs*)
Bases: *syn.base.a.base.Base*

class syn.base.b.meta.**Attrs**(**args*, ***kwargs*)
Bases: *syn.base.a.meta.Attrs*

class syn.base.b.meta.**Meta**(*clsname*, *bases*, *dct*)
Bases: *syn.base.a.meta.Meta*

groups_enum()
Returns an enum-ish dict with the names of the groups defined for this class.

class syn.base.b.meta.**Data**
Bases: object

syn.base.b.meta.**create_hook**(*f*)

syn.base.b.meta.**pre_create_hook**(**args*, ***kwargs*)

class syn.base.b.meta.**This**
Bases: *syn.type.a.type.TypeExtension*

syn.base.b.meta.**preserve_attr_data**(*A*, *B*)
Preserve attr data for combining B into A.

syn.base.b.utils module

class syn.base.b.utils.**Counter**(***kwargs*)
Bases: *syn.base.b.base.Base*

Keyword-Only Arguments:

initial_value: ***int*** **|** ***float*** The initial value to which the counter is reset

resets: ***list*** A list of counters to reset when this counter is reset

step (*default* = 1): ***int*** **|** ***float*** Amount by which to increment the counter

threshold [Optional]: ***int*** **|** ***float*** Threshold at which to reset the counter

value (*default* = -1): ***int*** **|** ***float*** The current count

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- make_type_object: True
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: initial_value, resets, step, threshold, value

peek()

reset()

validate()

syn.base.b.wrapper module

class syn.base.b.wrapper.**ListWrapper**(***kwargs*)

Bases: *syn.base.b.base.Base*, *syn.base.b.base.Harvester*

Keyword-Only Arguments:

_list: ***list*** The wrapped list

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: _list
- copy_copy: _list
- _internal: _list
- str_exclude: _list

append(*item*)

count(*item*)

extend(*items*)

index(*item*)

insert(*index*, *item*)

pop(*index=-1*)

remove(*item*)

reverse()

sort(**args*, ***kwargs*)

validate()

Module contents

Module contents

syn.base_utils package

Submodules

syn.base_utils.alg module

Some general purpose algorithms.

syn.base_utils.alg.**defer_reduce**(*func*, *items*, *test*, *accum=None*)
Recursively reduce items by func, but only the items that do not cause test(items, accum) to return False. Returns the reduced list (accum) and the list of remaining deferred items.

syn.base_utils.context module

syn.base_utils.context.**null_context**(**args*, ***kwds*)
A context manager that does nothing.

syn.base_utils.context.**assign**(**args*, ***kwds*)
Assigns B to A.attr, yields, and then assigns A.attr back to its original value.

syn.base_utils.context.**setitem**(**args*, ***kwds*)

syn.base_utils.context.**chdir**(**args*, ***kwds*)

syn.base_utils.context.**delete**(**args*, ***kwds*)
For using then deleting objects.

syn.base_utils.context.**nested_context**(**args*, ***kwds*)

syn.base_utils.context.**capture**(**args*, ***kwds*)

syn.base_utils.context.**on_error**(**args*, ***kwds*)

syn.base_utils.debug module

class syn.base_utils.debug.**Trace**
Bases: object

c_call(*frame*, *arg*)

c_exception(*frame*, *arg*)

c_return(*frame*, *arg*)

call(*frame*, *arg*)

exception(*frame*, *arg*)

line(*frame*, *arg*)

return_(*frame*, *arg*)

class syn.base_utils.debug.**CallTrace**(*indent=0*, *tab=' '*)
Bases: *syn.base_utils.debug.Trace*

call(*frame*, *arg*)

return_(*frame*, *arg*)

syn.base_utils.debug.**call_trace**(***kwargs*)

syn.base_utils.debug.**reset_trace**(**args*, ***kwds*)

syn.base_utils.dict module

Various dict extensions.

class syn.base_utils.dict.**AttrDict**
Bases: dict

A dict whose items can be accessed as attributes.

class syn.base_utils.dict.**UpdateDict**(**args*, ***kwargs*)
Bases: dict

A dict with an extensible update() hook.

update(**args*, ***kwargs*)

class syn.base_utils.dict.**GroupDict**
Bases: *syn.base_utils.dict.AttrDict*

An AttrDict whose items are treated as sets.

complement(**args*)
Returns the difference of the union of all values and the union of the values in *args.

intersection(**args*)
Returns the intersection of the values whose keys are in *args. If *args is blank, returns the intersection of all values.

union(**args*)
Returns the union of the values whose keys are in *args. If *args is blank, returns the union of all values.

update(**args*, ***kwargs*)

class syn.base_utils.dict.**ReflexiveDict**(**args*, ***kwargs*)
Bases: *syn.base_utils.dict.AttrDict*

An AttrDict for which each key == the associated value.

class syn.base_utils.dict.**SeqDict**
Bases: *syn.base_utils.dict.AttrDict*

An AttrDict whose items are treated as sequences.

update(**args*, ***kwargs*)

class syn.base_utils.dict.**AssocDict**(**args*, ***kwargs*)
Bases: _abcoll.MutableMapping

Mapping maintained via an assoc list.

update(**args*, ***kwargs*)
Preserves order if given an assoc list.

syn.base_utils.dict.**SetDict**
alias of *GroupDict*

syn.base_utils.filters module

Various filters for processing arguments. Inteded for use in the call keyword argument to the base.Base constructor.

syn.base_utils.filters.**split**(*obj*, *sep=None*)

syn.base_utils.filters.**join**(*obj*, *sep=' '*)

syn.base_utils.filters.**dictify_strings**(*obj*, *empty=None*, *sep=None*, *typ=<type 'dict'>*)

syn.base_utils.float module

syn.base_utils.float.**feq**(*a*, *b*, *tol=1.4901161193847696e-08*, *relative=False*)

syn.base_utils.float.**cfeq**(*a*, *b*, *tol=1.4901161193847696e-08*, *relative=False*)

syn.base_utils.float.**prod**(*args*, *log=False*)

syn.base_utils.float.**sgn**(*x*)

syn.base_utils.hash module

syn.base_utils.hash.**is_hashable**(*obj*)

syn.base_utils.iter module

syn.base_utils.iter.**iterlen**(*iter*)
Returns the number of iterations remaining over iter.

syn.base_utils.iter.**is_empty**(*iter*)
Returns True if iter is empty, otherwise False.

syn.base_utils.iter.**consume**(*it*, **args*, ***kwargs*)
Consumes N items from iter. If N is None (or not given), consumes all.

syn.base_utils.iter.**first**(*it*, **args*, ***kwargs*)

syn.base_utils.iter.**last**(*it*, **args*, ***kwargs*)

syn.base_utils.iter.**iteration_length**(*N*, *start=0*, *step=1*)
Return the number of iteration steps over a list of length N, starting at index start, proceeding step elements at a time.

syn.base_utils.list module

class syn.base_utils.list.**ListView**(*lst*, *start*, *end*)
Bases: _abcoll.MutableSequence

A list view.

insert(*idx*, *obj*)

class syn.base_utils.list.**IterableList**(*values*, *position=0*, *position_mark=None*)
Bases: list

consume(*n*)

copy()

displacement()

empty()

mark()

next()

peek(*n=None*, *safe=True*)

previous()

reset()

seek(*n*, *mode=0*)

take(*n*)

class syn.base_utils.list.**DefaultList**(*default*, **args*, ***kwargs*)
Bases: list

syn.base_utils.list.**is_proper_sequence**(*seq*)

syn.base_utils.list.**is_flat**(*seq*)

syn.base_utils.list.**is_unique**(*seq*)
Returns True if every item in the seq is unique, False otherwise.

syn.base_utils.list.**indices_removed**(*lst*, *idxs*)
Returns a copy of lst with each index in idxs removed.

syn.base_utils.list.**flattened**(*seq*)

syn.base_utils.logic module

syn.base_utils.logic.**implies**(*a*, *b*)

syn.base_utils.logic.**equiv**(*a*, *b*)

syn.base_utils.logic.**xor**(*a*, *b*)

syn.base_utils.logic.**and_**(**args*)

syn.base_utils.logic.**or_**(**args*)

syn.base_utils.logic.**nand**(**args*)

syn.base_utils.logic.**nor**(**args*)

syn.base_utils.logic.**fuzzy_and**(**args*)

syn.base_utils.logic.**fuzzy_not**(*arg*)

syn.base_utils.logic.**fuzzy_nand**(**args*)

syn.base_utils.logic.**fuzzy_or**(**args*)

syn.base_utils.logic.**fuzzy_nor**(**args*)

syn.base_utils.logic.**fuzzy_implies**(*a*, *b*)

syn.base_utils.logic.**fuzzy_equiv**(*a*, *b*)

syn.base_utils.logic.**fuzzy_xor**(*a*, *b*)

syn.base_utils.logic.**collection_equivalent**(*A*, *B*)

syn.base_utils.logic.**collection_comp**(*A*, *B*, *item_func=<built-in function eq>*, *coll_func=<built-in function all>*)

syn.base_utils.order module

class syn.base_utils.order.**Precedes**(*A*, *B*)
Bases: object

syn.base_utils.order.**Succeeds**(*A*, *B*)

syn.base_utils.order.**topological_sorting**(*nodes*, *relations*)
An implementation of Kahn's algorithm.

syn.base_utils.py module

syn.base_utils.py.**mro**(*cls*)

syn.base_utils.py.**hasmethod**(*x*, *name*)

syn.base_utils.py.**import_module**(*modname*)

syn.base_utils.py.**message**(*e*)

syn.base_utils.py.**run_all_tests**(*env*, *verbose=False*, *print_errors=False*, *exclude=None*, *include=None*)

syn.base_utils.py.**index**(*seq*, *elem*)

syn.base_utils.py.**nearest_base**(*cls*, *bases*)
Returns the closest ancestor to cls in bases.

syn.base_utils.py.**get_typename**(*x*)
Returns the name of the type of x, if x is an object. Otherwise, returns the name of x.

syn.base_utils.py.**get_mod**(*cls*)
Returns the string identifying the module that cls is defined in.

syn.base_utils.py.**compose**(**funcs*)

syn.base_utils.py.**assert_equivalent**(*o1*, *o2*)
Asserts that o1 and o2 are distinct, yet equivalent objects

syn.base_utils.py.**assert_inequivalent**(*o1*, *o2*)
Asserts that o1 and o2 are distinct and inequivalent objects

syn.base_utils.py.**assert_type_equivalent**(*o1*, *o2*)
Asserts that o1 and o2 are distinct, yet equivalent objects of the same type

syn.base_utils.py.**assert_pickle_idempotent**(*obj*)
Assert that obj does not change (w.r.t. ==) under repeated picklings

syn.base_utils.py.**assert_deepcopy_idempotent**(*obj*)
Assert that obj does not change (w.r.t. ==) under repeated deepcopies

syn.base_utils.py.**rgetattr**(*obj*, *attr*, **args*)

syn.base_utils.py.**callables**(*obj*, *exclude_sys=True*)

syn.base_utils.py.**is_subclass**(*x*, *typ*)

syn.base_utils.py.**getitem**(*mapping*, *item*, *default=None*, *allow_none_default=False*, *delete=False*)

syn.base_utils.py.**same_lineage**(*o1*, *o2*)
Returns True iff o1 and o2 are of the same class lineage (that is, a direct line of descent, without branches).

syn.base_utils.py.**type_partition**(*lst*, **types*)

syn.base_utils.py.**subclasses**(*cls*, *lst=None*)
Recursively gather subclasses of cls.

syn.base_utils.py.**unzip**(*seq*)

syn.base_utils.py.**this_module**(*npop=1*)
Returns the module object of the module this function is called from

syn.base_utils.py.**eprint**(*out*, *flush=True*)

syn.base_utils.py.**harvest_metadata**(*fpath*, *abspath=False*, *template='__{}__'*)

syn.base_utils.py.**tuple_append**(*tup*, *x*)

syn.base_utils.py.**get_fullname**(*x*)

syn.base_utils.py.**tuple_prepend**(*x*, *tup*)

syn.base_utils.py.**elog**(*exc*, *func*, *args=None*, *kwargs=None*, *str=<type 'str'>*, *pretty=True*, *name=''*)
For logging exception-raising function invocations during randomized unit tests.

syn.base_utils.py.**ngzwarn**(*value*, *name*)

syn.base_utils.py.**full_funcname**(*func*)

syn.base_utils.py.**hangwatch**(*timeout*, *func*, **args*, ***kwargs*)

syn.base_utils.py.**safe_vars**(**args*, ***kwargs*)

syn.base_utils.py.**getfunc**(*obj*, *name=''*)
Get the function corresponding to name from obj, not the method.

class syn.base_utils.py.**Partial**(*f*, *args=None*, *indexes=None*, *kwargs=None*)
Bases: object

Partial function object that allows specification of which indices are "baked in".

syn.base_utils.py.**pyversion**()

syn.base_utils.py.**getkey**(*mapping*, *value*, *default=None*, *use_id=False*)
Returns the first key mapping to value, as encountered via iteritems(), otherwise default. Obviously, works best for injective maps.

syn.base_utils.rand module

Random value-generating utilities. Intended mainly for generating random values for testing purposes (i.e. finding edge cases).

syn.base_utils.rand.**rand_bool**(*thresh=0.5*, ***kwargs*)

syn.base_utils.rand.**rand_int**(*min_val=-9223372036854775808*, *max_val=9223372036854775807*, ***kwargs*)

syn.base_utils.rand.**rand_float**(*lb=None*, *ub=None*, ***kwargs*)

syn.base_utils.rand.**rand_complex**(*imag_only=False*, ***kwargs*)

syn.base_utils.rand.**rand_long**(*min_len=None*, *max_len=None*, ***kwargs*)

syn.base_utils.rand.**rand_str**(*min_char=0*, *max_char=255*, *min_len=0*, *max_len=10*, *func=<built-in function chr>*, ***kwargs*)
For values in the (extended) ASCII range, regardless of Python version.

syn.base_utils.rand.**rand_unicode**(*min_char=0*, *max_char=1114111*, *min_len=0*, *max_len=10*, ***kwargs*)
For values in the unicode range, regardless of Python version.

syn.base_utils.rand.**rand_bytes**(***kwargs*)

syn.base_utils.rand.**rand_list**(***kwargs*)

syn.base_utils.rand.**rand_tuple**(***kwargs*)

syn.base_utils.rand.**rand_dict**(***kwargs*)

syn.base_utils.rand.**rand_set**(***kwargs*)

syn.base_utils.rand.**rand_frozenset**(***kwargs*)

syn.base_utils.rand.**rand_none**(***kwargs*)

syn.base_utils.rand.**rand_dispatch**(*typ*, ***kwargs*)

syn.base_utils.rand.**rand_primitive**(***kwargs*)

syn.base_utils.rand.**rand_hashable**(***kwargs*)

syn.base_utils.repl module

class syn.base_utils.repl.**REPL**(*prompt=''*)
Bases: object

command_help = {'q': 'quit', 'h': 'display available commands', 'e': 'eval the argument', '?': 'display available comma

commands = {'q': <function quit>, 'h': <function print_commands>, 'e': <function eval>, '?': <function print_command

eval(*expr*)

print_commands(***kwargs*)

quit(**args*, ***kwargs*)

class syn.base_utils.repl.**repl_command**(*name*, *help=''*)
Bases: object

syn.base_utils.str module

syn.base_utils.str.**quote_string**(*obj*)

syn.base_utils.str.**outer_quotes**(*string*)

syn.base_utils.str.**break_quoted_string**(*string*, *pattern*, *repl=None*)

syn.base_utils.str.**break_around_line_breaks**(*string*)

syn.base_utils.str.**escape_line_breaks**(*string*)

syn.base_utils.str.**escape_null**(*string*)

syn.base_utils.str.**escape_for_eval**(*string*)

syn.base_utils.str.**chrs**(*lst*)

syn.base_utils.str.**safe_chr**(*x*)

syn.base_utils.str.**safe_str**(*x*, *encoding='utf-8'*)

syn.base_utils.str.**safe_unicode**(*x*)

syn.base_utils.str.**safe_print**(*x*, *encoding='utf-8'*)

syn.base_utils.str.**istr**(*obj*, *pretty=False*, *indent=0*)

syn.base_utils.tree module

syn.base_utils.tree.**seq_list_nested**(*b*, *d*, *x=0*, *top_level=True*)
Create a nested list of iteratively increasing values.

b: branching factor d: max depth x: starting value (default = 0)

Module contents

syn.conf package

Submodules

syn.conf.conf module

class syn.conf.conf.**YAMLMixin**
Bases: syn.conf.conf.DictMixin

classmethod from_file(*fil*)

syn.conf.conf2 module

class syn.conf.conf2.**ConfDict**(***kwargs*)
Bases: *syn.base.b.base.Base*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()

- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

class syn.conf.conf2.**ConfList**(***kwargs*)

Bases: *syn.base.b.wrapper.ListWrapper*

Keyword-Only Arguments:

_list: <Schema> The wrapped list

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: None
- •min_len: None
- •optional_none: False
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Groups:

- •_all: _list
- •copy_copy: _list
- •_internal: _list
- •str_exclude: _list

schema = <syn.schema.b.sequence.Repeat {'set': <syn.sets.b.operators.Union {'_id': None, '_node_count': 21, '_name':

class syn.conf.conf2.**Conf**(***kwargs*)

Bases: *syn.conf.conf2.ConfDict*

Keyword-Only Arguments:

_env: *dict* vars: *ConfDict*

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •optional_none: False
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Groups:

- •_all: _env, vars

syn.conf.vars module

class syn.conf.vars.**Vars**(***kwargs*)

Bases: *syn.base.b.base.Base*

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •env_default: False
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •optional_none: False
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()

- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

classmethod **coerce**(*value*)

Module contents

syn.cython package

Subpackages

Module contents

syn.five package

Submodules

syn.five.num module

syn.five.string module

syn.five.string.**strf**
alias of *unicode*

class syn.five.string.**unicode**(*object=''*) → unicode object
Bases: basestring

unicode(string[, encoding[, errors]]) -> unicode object

Create a new Unicode object from the given encoded string. encoding defaults to the current default string encoding. errors can be 'strict', 'replace' or 'ignore' and defaults to 'strict'.

capitalize() → unicode
Return a capitalized version of S, i.e. make the first character have upper case and the rest lower case.

center(*width*[, *fillchar*]) → unicode
Return S centered in a Unicode string of length width. Padding is done using the specified fill character (default is a space)

count(*sub*[, *start*[, *end*]]) → int
Return the number of non-overlapping occurrences of substring sub in Unicode string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

decode([*encoding*[, *errors*]]) → string or unicode
Decodes S using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeDecodeError. Other possible values are 'ignore' and 'replace' as well as any other name registered with codecs.register_error that is able to handle UnicodeDecodeErrors.

encode([*encoding*[, *errors*]]) → string or unicode
Encodes S using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a

UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → bool
Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs([*tabsize*]) → unicode
Return a copy of S where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int
Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → unicode
Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int
Like S.find() but raise ValueError when the substring is not found.

isalnum() → bool
Return True if all characters in S are alphanumeric and there is at least one character in S, False otherwise.

isalpha() → bool
Return True if all characters in S are alphabetic and there is at least one character in S, False otherwise.

isdecimal() → bool
Return True if there are only decimal characters in S, False otherwise.

isdigit() → bool
Return True if all characters in S are digits and there is at least one character in S, False otherwise.

islower() → bool
Return True if all cased characters in S are lowercase and there is at least one cased character in S, False otherwise.

isnumeric() → bool
Return True if there are only numeric characters in S, False otherwise.

isspace() → bool
Return True if all characters in S are whitespace and there is at least one character in S, False otherwise.

istitle() → bool
Return True if S is a titlecased string and there is at least one character in S, i.e. upper- and titlecase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

isupper() → bool
Return True if all cased characters in S are uppercase and there is at least one cased character in S, False otherwise.

join(*iterable*) → unicode
Return a string which is the concatenation of the strings in the iterable. The separator between elements is S.

ljust (*width*[, *fillchar*]) → int
Return S left-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).

lower () → unicode
Return a copy of the string S converted to lowercase.

lstrip ([*chars*]) → unicode
Return a copy of the string S with leading whitespace removed. If chars is given and not None, remove characters in chars instead. If chars is a str, it will be converted to unicode before stripping

partition (*sep) -> (head*, *sep*, *tail*)
Search for the separator sep in S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.

replace (*old*, *new*[, *count*]) → unicode
Return a copy of S with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.

rfind (*sub*[, *start*[, *end*]]) → int
Return the highest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

rindex (*sub*[, *start*[, *end*]]) → int
Like S.rfind() but raise ValueError when the substring is not found.

rjust (*width*[, *fillchar*]) → unicode
Return S right-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).

rpartition (*sep) -> (head*, *sep*, *tail*)
Search for the separator sep in S, starting at the end of S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return two empty strings and S.

rsplit ([*sep*[, *maxsplit*]]) → list of strings
Return a list of the words in S, using sep as the delimiter string, starting at the end of the string and working to the front. If maxsplit is given, at most maxsplit splits are done. If sep is not specified, any whitespace string is a separator.

rstrip ([*chars*]) → unicode
Return a copy of the string S with trailing whitespace removed. If chars is given and not None, remove characters in chars instead. If chars is a str, it will be converted to unicode before stripping

split ([*sep*[, *maxsplit*]]) → list of strings
Return a list of the words in S, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.

splitlines (*keepends=False*) → list of strings
Return a list of the lines in S, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true.

startswith (*prefix*[, *start*[, *end*]]) → bool
Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip ([*chars*]) → unicode
Return a copy of the string S with leading and trailing whitespace removed. If chars is given and not None,

remove characters in chars instead. If chars is a str, it will be converted to unicode before stripping

swapcase() → unicode
Return a copy of S with uppercase characters converted to lowercase and vice versa.

title() → unicode
Return a titlecased version of S, i.e. words start with title case characters, all remaining cased characters have lower case.

translate(*table*) → unicode
Return a copy of the string S, where all characters have been mapped through the given translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, Unicode strings or None. Unmapped characters are left untouched. Characters mapped to None are deleted.

upper() → unicode
Return a copy of S converted to uppercase.

zfill(*width*) → unicode
Pad a numeric string S with zeros on the left, to fill a field of the specified width. The string S is never truncated.

syn.five.string.**unichr**(*i*) → Unicode character
Return a Unicode string of one character with ordinal i; 0 <= i <= 0x10ffff.

Module contents

Additional Python 2/3 compatibility facilities.

syn.five.**range**(**args*, ***kwargs*)

syn.globals package

Submodules

syn.globals.loggers module

syn.globals.values module

Module contents

syn.python package

Subpackages

syn.python.b package

Submodules

syn.python.b.base module

class syn.python.b.base.**PythonNode**(***kwargs*)
Bases: *syn.tree.b.node.Node*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list

- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

as_return(***kwargs*)

as_value(*obj*, **args*, ***kwargs*)

ast = None

emit(***kwargs*)

expressify_statements(*obj*, **args*, ***kwargs*)

classmethod from_ast(*ast*, ***kwargs*)

maxver = '100'

minver = '0'

resolve_progn(*obj*, **args*, ***kwargs*)

to_ast(***kwargs*)

validate()

variables(***kwargs*)

viewable(***kwargs*)

class syn.python.b.base.**PythonTree**(*root*, ***kwargs*)
Bases: *syn.tree.b.tree.Tree*

Positional Arguments:

root: *RootNode* The root node of the tree

Keyword-Only Arguments:

id_dict: *dict* (any => *Node*) Mapping of ids to nodes

node_counter: *Counter* Node id counter

node_types: *list* (*basestring*) List of all tree node types

nodes: *list* (*Node*) List of all tree nodes

type_dict: *dict* (any => *list* (*Node*)) Mapping of type names to node lists

Class Options:

- args: ('root',)
- autodoc: True
- coerce_args: False
- id_equality: False

- •init_validate: True
- •make_hashable: False
- •make_type_object: True
- •optional_none: False
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Groups:

- •_all: id_dict, node_counter, node_types, nodes, root, type_dict
- •generate_exclude: id_dict, node_counter, node_types, nodes, type_dict
- •eq_exclude: node_counter
- •str_exclude: id_dict, node_counter, node_types, nodes, type_dict

abstract()

emit(**kwargs)

to_ast(**kwargs)

exception syn.python.b.base.**AstUnsupported**
Bases: exceptions.Exception

exception syn.python.b.base.**PythonError**
Bases: exceptions.Exception

class syn.python.b.base.**Context**(**kwargs)
Bases: *syn.python.b.base.PythonNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

classmethod from_ast(*ast*, ***kwargs*)

maxver = '100'

minver = '0'

to_ast(***kwargs*)

class syn.python.b.base.**Load**(***kwargs*)
Bases: *syn.python.b.base.Context*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Load*

maxver = '100'

minver = '0'

class syn.python.b.base.**Store**(**kwargs*)
Bases: *syn.python.b.base.Context*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

•args: ()

•autodoc: True

•coerce_args: False

- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Store*

maxver = '100'

minver = '0'

class syn.python.b.base.**Del**(**kwargs*)

Bases: *syn.python.b.base.Context*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Del*

maxver = '100'

minver = '0'

class syn.python.b.base.**Param**(**kwargs*)
Bases: *syn.python.b.base.Context*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True

- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Param*

maxver = '2.9999999999'

minver = '0'

class syn.python.b.base.**RootNode**(***kwargs*)
Bases: *syn.python.b.base.PythonNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent

- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

emit (***kwargs*)

expressify_statements (*obj*, **args*, ***kwargs*)

classmethod from_ast (*ast*, ***kwargs*)

maxver = '100'

minver = '0'

resolve_progn (*obj*, **args*, ***kwargs*)

to_ast (***kwargs*)

class syn.python.b.base.**Module** (***kwargs*)

Bases: *syn.python.b.base.RootNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False

- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Module*

maxver = '100'

minver = '0'

class syn.python.b.base.**Expression_**(***kwargs*)
Bases: *syn.python.b.base.RootNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: 1
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent

- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Expression*

body
itemgetter(item, ...) –> itemgetter object

Return a callable object that fetches the given item(s) from its operand. After f = itemgetter(2), the call f(r) returns r[2]. After g = itemgetter(2, 5, 3), the call g(r) returns (r[2], r[5], r[3])

emit(***kwargs*)

classmethod from_ast(*ast*, ***kwargs*)

maxver = '100'

minver = '0'

to_ast(***kwargs*)

class syn.python.b.base.**Interactive**(***kwargs*)
Bases: *syn.python.b.base.RootNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False

- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Interactive*

maxver = '100'

minver = '0'

class syn.python.b.base.**Special**(***kwargs*)
Bases: *syn.python.b.base.PythonNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent

- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

maxver = '100'

minver = '0'

validate()

class syn.python.b.base.**ProgN**(**kwargs)

Bases: *syn.python.b.base.Special*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()

•init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast = None

expressify_statements(***kwargs*)

maxver = '100'

minver = '0'

resolve_progn(*obj*, **args*, ***kwargs*)

value(*obj*, **args*, ***kwargs*)

valuify(***kwargs*)

class syn.python.b.base.**NoAST**
Bases: object

Dummy class to prevent binding to a specific ast object.

class syn.python.b.base.**Expression**(***kwargs*)
Bases: *syn.python.b.base.PythonNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent

- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *NoAST*

maxver = '100'

minver = '0'

class syn.python.b.base.**Statement**(***kwargs*)
Bases: *syn.python.b.base.PythonNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()

•init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast = None

maxver = '100'

minver = '0'

syn.python.b.base.**from_ast**(*ast*, ***kwargs*)

syn.python.b.base.**from_source**(*src*, *mode='exec'*)

syn.python.b.blocks module

class syn.python.b.blocks.**Block**(***kwargs*)

Bases: *syn.python.b.base.Statement*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* body: *list* (*Expression* | *Statement*) col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, body, col_offset, indent_amount, lineno
- copy_copy: _list, body
- ast_convert_attr: body
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: body, col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent

- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

emit_block(*head*, *body*, ***kwargs*)

maxver = '100'

minver = '0'

valuify_block(*body*, *name*, ***kwargs*)

class syn.python.b.blocks.**If**(*test*, *body*, *orelse*, ***kwargs*)
Bases: *syn.python.b.blocks.Block*

Positional Arguments:

test: *Expression* body: *list* (*Expression* | *Statement*) orelse: *list* (*Expression* | *Statement*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('test', 'body', 'orelse')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()

- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, body, col_offset, indent_amount, lineno, orelse, test
- copy_copy: _list, body, orelse
- ast_convert_attr: body, orelse, test
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: body, col_offset, lineno, orelse, test
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

as_return(***kwargs*)

as_value(*obj*, **args*, ***kwargs*)

ast
alias of *If*

emit(***kwargs*)

maxver = '100'

minver = '0'

resolve_progn(*obj*, **args*, ***kwargs*)

class syn.python.b.blocks.**For**(*target*, *iter*, *body*, *orelse*, ***kwargs*)
Bases: *syn.python.b.blocks.Block*

Positional Arguments:

target: *Name* | *Tuple* | *List* iter: *Expression* body: *list* (*Expression* | *Statement*) orelse: *list* (*Expression* | *Statement*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('target', 'iter', 'body', 'orelse')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, body, col_offset, indent_amount, iter, lineno, orelse, target
- copy_copy: _list, body, orelse
- ast_convert_attr: body, iter, orelse, target
- hash_exclude: _parent
- generate_exclude: _node_count, _parent

- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: body, col_offset, iter, lineno, orelse, target
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *For*

emit (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.blocks.**While**(*test*, *body*, *orelse*, ***kwargs*)
Bases: *syn.python.b.blocks.Block*

Positional Arguments:

test: *Expression* body: *list* (*Expression* | *Statement*) orelse: *list* (*Expression* | *Statement*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('test', 'body', 'orelse')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None

- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, body, col_offset, indent_amount, lineno, orelse, test
- copy_copy: _list, body, orelse
- ast_convert_attr: body, orelse, test
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: body, col_offset, lineno, orelse, test
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *While*

emit (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.blocks.**Arg**(*arg*[, *annotation*], ***kwargs*)
Bases: *syn.python.b.base.PythonNode*

Positional Arguments:

arg: *basestring* annotation [**Optional**]: *Expression*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ['arg', 'annotation']
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, annotation, arg, col_offset, indent_amount, lineno
- copy_copy: _list
- ast_convert_attr: annotation
- hash_exclude: _parent

- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: annotation, arg, col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

emit (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.blocks.**Arguments**(*args*[, *vararg*][, *kwarg*], *defaults*, ***kwargs*)

Bases: *syn.python.b.base.PythonNode*

Positional Arguments:

args: *list* (*Name*) vararg [**Optional**]: *basestring* kwarg [**Optional**]: *basestring* defaults: *list* (*Expression*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ['args', 'vararg', 'kwarg', 'defaults']
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None

- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, args, col_offset, defaults, indent_amount, kwarg, lineno, vararg
- copy_copy: _list, args, defaults
- ast_convert_attr: args, defaults
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: args, col_offset, defaults, kwarg, lineno, vararg
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of `arguments`

emit (***kwargs*)

emit2 (***kwargs*)

emit3 (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.blocks.**FunctionDef** (*name*, *args*, *body*[, *decorator_list*], ***kwargs*)
Bases: *syn.python.b.blocks.Block*

Positional Arguments:

name: *basestring* args: *Arguments* body: *list* (*Expression* | *Statement*) decorator_list [**Optional**]: *list* (*Expression*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ['name', 'args', 'body', 'decorator_list']
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, args, body, col_offset, decorator_list, indent_amount, lineno, name

- copy_copy: _list, body, decorator_list
- ast_convert_attr: args, body, decorator_list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: args, body, col_offset, decorator_list, lineno, name
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *FunctionDef*

emit(***kwargs*)

emit_decorators(***kwargs*)

maxver = '100'

minver = '0'

syn.python.b.expressions module

class syn.python.b.expressions.**Expr**(*value*, ***kwargs*)
Bases: *syn.python.b.base.Expression*

Positional Arguments:

value: *PythonNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('value',)
- autodoc: True
- coerce_args: False

- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, value
- copy_copy: _list
- ast_convert_attr: value
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, value
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Expr*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.expressions.**Operator**(***kwargs*)

Bases: *syn.python.b.base.Expression*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

emit (***kwargs*)

maxver = '100'

minver = '0'

symbol = None

class syn.python.b.expressions.**UnaryOperator**(***kwargs*)

Bases: *syn.python.b.expressions.Operator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False

- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- •repr_exclude: _list, _parent
- •ast_attr: col_offset, lineno
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

`ast` = None

`maxver` = '100'

`minver` = '0'

class `syn.python.b.expressions.`**`UnaryOp`**(*op*, *operand*, ***kwargs*)
Bases: *syn.python.b.base.Expression*

Positional Arguments:

op: *UnaryOperator* operand: *Expression*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ['op', 'operand']
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, op, operand
- copy_copy: _list
- ast_convert_attr: op, operand
- hash_exclude: _parent

- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, op, operand
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *UnaryOp*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.expressions.**UAdd**(***kwargs*)
Bases: *syn.python.b.expressions.UnaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False

- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *UAdd*

maxver = '100'

minver = '0'

symbol = '+'

class syn.python.b.expressions.**USub**(***kwargs*)
Bases: *syn.python.b.expressions.UnaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent

- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *USub*

maxver = '100'

minver = '0'

symbol = '-'

class syn.python.b.expressions.**Not**(***kwargs*)
Bases: *syn.python.b.expressions.UnaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()

•init_hooks: ()

•init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Not*

maxver = '100'

minver = '0'

symbol = 'not'

class syn.python.b.expressions.**Invert**(***kwargs*)
Bases: *syn.python.b.expressions.UnaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- •repr_exclude: _list, _parent
- •ast_attr: col_offset, lineno
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Invert*

maxver = '100'

minver = '0'

symbol = '~'

class syn.python.b.expressions.**BinaryOperator**(***kwargs*)
Bases: *syn.python.b.expressions.Operator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

maxver = '100'

minver = '0'

class syn.python.b.expressions.**BinOp**(*left*, *op*, *right*, ***kwargs*)

Bases: *syn.python.b.base.Expression*

Positional Arguments:

left: *Expression* op: *BinaryOperator* right: *Expression*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('left', 'op', 'right')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False

- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, left, lineno, op, right
- copy_copy: _list
- ast_convert_attr: left, op, right
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, left, lineno, op, right
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

A

attrgetter(attr, ...) –> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter('name'), the call f(r) returns r.name. After g = attrgetter('name', 'date'), the call g(r) returns (r.name, r.date). After h = attrgetter('name.first', 'name.last'), the call h(r) returns (r.name.first, r.name.last).

B

attrgetter(attr, ...) –> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter('name'), the call f(r) returns r.name. After g = attrgetter('name', 'date'), the call g(r) returns (r.name, r.date). After h = attrgetter('name.first', 'name.last'), the call h(r) returns (r.name.first, r.name.last).

ast
alias of *BinOp*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.expressions.**Add**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()

- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Add*

maxver = '100'

minver = '0'

symbol = '+'

class syn.python.b.expressions.**Sub**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- •repr_exclude: _list, _parent
- •ast_attr: col_offset, lineno
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Sub*

maxver = '100'

minver = '0'

symbol = '-'

class syn.python.b.expressions.**Mult**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Mult*

maxver = '100'

minver = '0'

symbol = '*'

class syn.python.b.expressions.**Div**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False

- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- •repr_exclude: _list, _parent
- •ast_attr: col_offset, lineno
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Div*

maxver = '100'

minver = '0'

symbol = '/'

class syn.python.b.expressions.**FloorDiv**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list

- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *FloorDiv*

maxver = '100'

minver = '0'

symbol = '//'

class syn.python.b.expressions.**Mod**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None

•must_be_root: False

•optional_none: True

•register_subclasses: False

•repr_template:

•coerce_hooks: ()

•create_hooks: ()

•init_hooks: ()

•init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Mod*

maxver = '100'

minver = '0'

symbol = '%'

class syn.python.b.expressions.**Pow**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno

- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Pow*

maxver = '100'

minver = '0'

symbol = ''**

class syn.python.b.expressions.**LShift**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()

- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *LShift*

maxver = '100'

minver = '0'

symbol = '<<'

class syn.python.b.expressions.**RShift**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *RShift*

maxver = '100'

minver = '0'

symbol = '>>'

class syn.python.b.expressions.**BitOr**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *BitOr*

maxver = '100'

minver = '0'

symbol = '|'

class syn.python.b.expressions.**BitXor**(**kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

•args: ()

•autodoc: True

- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *BitXor*

maxver = '100'

minver = '0'

symbol = '^'

class syn.python.b.expressions.**BitAnd**(**kwargs)

Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *BitAnd*

maxver = '100'

minver = '0'

symbol = '&'

class syn.python.b.expressions.**MatMult**(***kwargs*)
Bases: *syn.python.b.expressions.BinaryOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False

- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- •repr_exclude: _list, _parent
- •ast_attr: col_offset, lineno
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

`ast` = None

`maxver` = '100'

`minver` = '3.5'

`symbol` = '@'

class `syn.python.b.expressions.`**`BooleanOperator`**`(`***kwargs*`)`
Bases: *syn.python.b.expressions.Operator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent

- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

maxver = '100'

minver = '0'

class syn.python.b.expressions.**BoolOp**(*op*, *values*, ***kwargs*)
Bases: *syn.python.b.base.Expression*

Positional Arguments:

op: *BooleanOperator* values: *list* (*Expression*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('op', 'values')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True

- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, op, values
- copy_copy: _list, values
- ast_convert_attr: op, values
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, op, values
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *BoolOp*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.expressions.**And**(***kwargs*)
Bases: *syn.python.b.expressions.BooleanOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent

- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *And*

maxver = '100'

minver = '0'

symbol = 'and'

class syn.python.b.expressions.**Or**(***kwargs*)
Bases: *syn.python.b.expressions.BooleanOperator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()

•init_hooks: ()

•init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Or*

maxver = '100'

minver = '0'

symbol = 'or'

class syn.python.b.expressions.**Comparator**(***kwargs*)
Bases: *syn.python.b.expressions.Operator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- •repr_exclude: _list, _parent
- •ast_attr: col_offset, lineno
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

`ast` = None

`maxver` = '100'

minver = '0'

class syn.python.b.expressions.**Compare**(*left*, *ops*, *comparators*, ***kwargs*)
Bases: *syn.python.b.base.Expression*

Positional Arguments:

left: *Expression* ops: *list* (*Comparator*) comparators: *list* (*Expression*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('left', 'ops', 'comparators')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, comparators, indent_amount, left, lineno, ops
- copy_copy: _list, comparators, ops
- ast_convert_attr: comparators, left, ops
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, comparators, left, lineno, ops
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Compare*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.expressions.**Eq**(***kwargs*)
Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False

- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Eq*

maxver = '100'

minver = '0'

symbol = '=='

class syn.python.b.expressions.**NotEq**(***kwargs*)

Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *NotEq*

maxver = '100'

minver = '0'

symbol = '!='

class syn.python.b.expressions.**Lt**(***kwargs*)
Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False

- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Lt*

maxver = '100'

minver = '0'

symbol = '<'

class syn.python.b.expressions.**LtE**(**kwargs*)
Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent

- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *LtE*

maxver = '100'

minver = '0'

symbol = '<='

class syn.python.b.expressions.**Gt**(***kwargs*)
Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True

- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Gt*

maxver = '100'

minver = '0'

symbol = '>'

class syn.python.b.expressions.**GtE**(***kwargs*)
Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent

- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *GtE*

maxver = '100'

minver = '0'

symbol = '>='

class syn.python.b.expressions.**Is**(***kwargs*)
Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()

•init_hooks: ()

•init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Is*

maxver = '100'

minver = '0'

symbol = 'is'

class syn.python.b.expressions.**IsNot**(***kwargs*)
Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- •repr_exclude: _list, _parent
- •ast_attr: col_offset, lineno
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *IsNot*

maxver = '100'

minver = '0'

symbol = 'is not'

class syn.python.b.expressions.**In**(***kwargs*)
Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *In*

maxver = '100'

minver = '0'

symbol = 'in'

class syn.python.b.expressions.**NotIn**(***kwargs*)
Bases: *syn.python.b.expressions.Comparator*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False

- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *NotIn*

maxver = '100'

minver = '0'

symbol = 'not in'

class syn.python.b.expressions.**Keyword**(*arg*, *value*, ***kwargs*)
Bases: *syn.python.b.base.Expression*

Positional Arguments:

arg: *basestring* value: *Expression*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('arg', 'value')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, arg, col_offset, indent_amount, lineno, value
- copy_copy: _list
- ast_convert_attr: value
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: arg, col_offset, lineno, value
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of `keyword`

emit (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.expressions.**Call** (*func*[, *args*][, *keywords*][, *starargs*][, *kwargs*], ***kwargs*)

Bases: *syn.python.b.base.Expression*

Positional Arguments:

func: *Expression* args [**Optional**]: *list* (*Expression*) keywords [**Optional**]: *list* (*Keyword*) starargs [**Optional**]: *PythonNode* kwargs [**Optional**]: *PythonNode*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ['func', 'args', 'keywords', 'starargs', 'kwargs']
- autodoc: True
- coerce_args: False

- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, args, col_offset, func, indent_amount, keywords, kwargs, lineno, starargs
- copy_copy: _list, args, keywords
- ast_convert_attr: args, func, keywords, kwargs, starargs
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: args, col_offset, func, keywords, kwargs, lineno, starargs
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Call*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.expressions.**IfExp**(*test*, *body*, *orelse*, ***kwargs*)

Bases: *syn.python.b.base.Expression*

Positional Arguments:

test: *Expression* body: *Expression* orelse: *Expression*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('test', 'body', 'orelse')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, body, col_offset, indent_amount, lineno, orelse, test
- copy_copy: _list
- ast_convert_attr: body, orelse, test
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: body, col_offset, lineno, orelse, test
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *IfExp*

emit (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.expressions.**Attribute**(*value*, *attr*, ***kwargs*)
Bases: *syn.python.b.base.Expression*

Positional Arguments:

value: *Expression* attr: *basestring*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* ctx (*default* = Load(_child_map = {}, _children_set = True, _progn_value = None, col_offset = None, indent_amount = 4, lineno = None)): *Context* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('value', 'attr')

- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, attr, col_offset, ctx, indent_amount, lineno, value
- copy_copy: _list
- ast_convert_attr: ctx, value
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: attr, col_offset, ctx, lineno, value
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Attribute*

emit(***kwargs*)

maxver = '100'

minver = '0'

syn.python.b.literals module

class syn.python.b.literals.**Literal**(***kwargs*)
Bases: *syn.python.b.base.Expression*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()

•metaclass_lookup: (‘coerce_hooks’, ‘init_hooks’, ‘create_hooks’, ‘setstate_hooks’)

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

`ast` = None

`maxver` = ‘100’

`minver` = ‘0’

class `syn.python.b.literals.`**`Num`**(*n*, ***kwargs*)
Bases: *syn.python.b.literals.Literal*

Positional Arguments:

n: *int* | *long* | *float* | *complex* The numerical value

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

•args: (‘n’,)

•autodoc: True

- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, n
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, n
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Num*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.literals.**Str**(*s*, ***kwargs*)

Bases: *syn.python.b.literals.Literal*

Positional Arguments:

s: ***basestring*** The string contents

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('s',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, s
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, s
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Str*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.literals.**Bytes**(*s*, ***kwargs*)
Bases: *syn.python.b.literals.Literal*

Positional Arguments:

s: *str*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('s',)
- autodoc: True
- coerce_args: False

- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, s
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, s
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

emit (***kwargs*)

maxver = '100'

minver = '3'

class syn.python.b.literals.**Sequence**(*elts*, ***kwargs*)

Bases: *syn.python.b.literals.Literal*

Positional Arguments:

elts: *list* (*Expression*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('elts',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, elts, indent_amount, lineno
- copy_copy: _list, elts
- ast_convert_attr: elts
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, elts, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

bounds = ('[', ']')

delim = ', '

emit (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.literals.**List**(*elts*, ***kwargs*)
Bases: *syn.python.b.literals.Sequence*

Positional Arguments:

elts: *list* (*Expression*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* ctx (*default* = Load(_child_map = {}, _children_set = True, _progn_value = None, col_offset = None, indent_amount = 4, lineno = None)): *Context* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- •args: (‘elts’,)
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: (‘coerce_hooks’, ‘init_hooks’, ‘create_hooks’, ‘setstate_hooks’)
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, ctx, elts, indent_amount, lineno
- •copy_copy: _list, elts
- •ast_convert_attr: ctx, elts
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- •repr_exclude: _list, _parent
- •ast_attr: col_offset, ctx, elts, lineno
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

ast

alias of *List*

maxver = '100'

minver = '0'

class syn.python.b.literals.**Tuple**(*elts*, ***kwargs*)
Bases: *syn.python.b.literals.List*

Positional Arguments:

elts: *list* (*Expression*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* ctx (*default* = Load(_child_map = {}, _children_set = True, _progn_value = None, col_offset = None, indent_amount = 4, lineno = None)): *Context* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('elts',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()

•init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, ctx, elts, indent_amount, lineno

•copy_copy: _list, elts

•ast_convert_attr: ctx, elts

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, ctx, elts, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Tuple*

bounds = ('(', ')')

maxver = '100'

minver = '0'

class syn.python.b.literals.**Set**(*elts*, ***kwargs*)
Bases: *syn.python.b.literals.Sequence*

Positional Arguments:

elts: *list* (*Expression*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- •args: ('elts',)
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, elts, indent_amount, lineno
- •copy_copy: _list, elts
- •ast_convert_attr: elts
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- •repr_exclude: _list, _parent
- •ast_attr: col_offset, elts, lineno
- •eq_exclude: _parent
- •getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Set*

bounds = ('{', '}')

maxver = '100'

minver = '0'

class syn.python.b.literals.**NameConstant**(*value*, ***kwargs*)
Bases: *syn.python.b.literals.Literal*

Positional Arguments:

value: [True, False, None]

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

•args: ('value',)

•autodoc: True

•coerce_args: False

•descendant_exclude: ()

•id_equality: False

•init_validate: False

•make_hashable: False

•make_type_object: True

•max_len: 0

•min_len: None

•must_be_root: False

•optional_none: True

•register_subclasses: False

•repr_template:

•coerce_hooks: ()

- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, value
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, value
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

emit (***kwargs*)

maxver = '100'

minver = '3.4'

syn.python.b.statements module

class syn.python.b.statements.**Assign** (*targets*, *value*, ***kwargs*)

Bases: *syn.python.b.base.Statement*

Positional Arguments:

targets: *list* (*Expression*) value: *Expression*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('targets', 'value')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, targets, value
- copy_copy: _list, targets
- ast_convert_attr: targets, value
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, targets, value

- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

as_value(*obj*, **args*, ***kwargs*)

ast
alias of *Assign*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.statements.**Return**([*value*], ***kwargs*)
Bases: *syn.python.b.base.Statement*

Positional Arguments:

value [**Optional**]: *Expression*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('value',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True

- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, value
- copy_copy: _list
- ast_convert_attr: value
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, value
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Return*

emit (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.statements.**Alias**(*name*[, *asname*], ***kwargs*)
Bases: *syn.python.b.base.Statement*

Positional Arguments:

name: *basestring* asname [**Optional**]: *basestring*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('name', 'asname')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, asname, col_offset, indent_amount, lineno, name
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent

- ast_attr: asname, col_offset, lineno, name
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of `alias`

emit (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.statements.**Import**(*names*, ***kwargs*)
Bases: *syn.python.b.base.Statement*

Positional Arguments:

names: *list* (*Alias*)

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('names',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True

- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno, names
- copy_copy: _list, names
- ast_convert_attr: names
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno, names
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Import*

emit(***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.statements.**EmptyStatement**(***kwargs*)
Bases: *syn.python.b.base.Statement*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent

- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

emit (***kwargs*)

maxver = '100'

minver = '0'

class syn.python.b.statements.**Break** (***kwargs*)

Bases: *syn.python.b.statements.EmptyStatement*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()

•init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value

•repr_exclude: _list, _parent

•ast_attr: col_offset, lineno

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Break*

maxver = '100'

minver = '0'

class syn.python.b.statements.**Continue**(***kwargs*)
Bases: *syn.python.b.statements.EmptyStatement*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

•args: ()

•autodoc: True

- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Continue*

maxver = '100'

minver = '0'

class syn.python.b.statements.**Pass**(**kwargs)

Bases: *syn.python.b.statements.EmptyStatement*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, indent_amount, lineno
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

as_return(***kwargs*)

ast
alias of *Pass*

maxver = '100'

minver = '0'

syn.python.b.variables module

class syn.python.b.variables.**Name**(*id*, ***kwargs*)
Bases: *syn.python.b.base.Expression*

Positional Arguments:

id: *basestring*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* ctx (*default* = Load(_child_map = {}, _children_set = True, _progn_value = None, col_offset = None, indent_amount = 4, lineno = None)): *Context* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('id',)
- autodoc: True

- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, ctx, id, indent_amount, lineno
- copy_copy: _list
- ast_convert_attr: ctx
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, ctx, id, lineno
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast
alias of *Name*

emit(***kwargs*)

maxver = '100'

minver = '0'

variables(***kwargs*)

class syn.python.b.variables.**Starred**(*value*, ***kwargs*)
Bases: *syn.python.b.base.Expression*

Positional Arguments:

value: *Name*

Keyword-Only Arguments:

_child_map: *dict* _children_set (*default* = False): *bool* _id [**Optional**]: *int*

Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

_progn_value [**Optional**]: *object* col_offset [**Optional**]: *int* ctx (*default* = Load(_child_map = {}, _children_set = True, _progn_value = None, col_offset = None, indent_amount = 4, lineno = None)): *Context* indent_amount [**Optional**] (*default* = 4): *int*

The number of spaces to indent per indent level

lineno [**Optional**]: *int*

Class Options:

- args: ('value',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()

- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value, col_offset, ctx, indent_amount, lineno, value
- copy_copy: _list
- ast_convert_attr: ctx, value
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _child_map, _children_set, _id, _list, _name, _node_count, _parent, _progn_value
- repr_exclude: _list, _parent
- ast_attr: col_offset, ctx, lineno, value
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

ast = None

emit (***kwargs*)

maxver = '100'

minver = '3'

Module contents

Module contents

syn.schema package

Subpackages

syn.schema.b package

Submodules

syn.schema.b.sequence module

Tools for representing sets of sequences via sequence operators and sets of sequence items. The main idea is that a set of sequences is the result of a (flattened) Cartesian product over a sequence of sets.

class syn.schema.b.sequence.**SchemaNode**(***kwargs*)

Bases: *syn.tree.b.node.Node*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

set [Optional]: ***SetNode*** Internal set representation

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children, elems

Groups:

- _all: _id, _list, _name, _node_count, _parent, set
- copy_copy: _list
- hash_exclude: _parent

- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent, set
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

elems

class syn.schema.b.sequence.**Set**(*[set]*, ***kwargs*)

Bases: *syn.schema.b.sequence.SchemaNode*

Positional Arguments:

set [Optional]: *SetNode* Internal set representation

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

Class Options:

- args: ('set',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()

- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children, elems

Groups:

- _all: _id, _list, _name, _node_count, _parent, set
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent, set
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

match(*seq*, ***kwargs*)

class syn.schema.b.sequence.**Type**([*set*], ***kwargs*)

Bases: *syn.schema.b.sequence.Set*

Positional Arguments:

set [Optional]: ***SetNode*** Internal set representation

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ('set',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None

- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children, elems

Groups:

- _all: _id, _list, _name, _node_count, _parent, set
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent, set
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

class syn.schema.b.sequence.**Or**(**kwargs*)

Bases: *syn.schema.b.sequence.SchemaNode*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

set [Optional]: ***SetNode*** Internal set representation

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()

- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: 2
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children, elems

Groups:

- _all: _id, _list, _name, _node_count, _parent, set
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent, set
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

generate_set(***kwargs*)

match(*seq*, ***kwargs*)

class syn.schema.b.sequence.**Repeat**(***kwargs*)

Bases: *syn.schema.b.sequence.SchemaNode*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

greedy (*default* = True): ***bool*** Match as much as we can if True

lb (*default* = 0): ***int*** Minimum number of times to repeat

set [Optional]: ***SetNode*** Internal set representation

ub [Optional]: ***int*** Maximum number of times to repeat

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: 1
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children, elems

Groups:

- _all: _id, _list, _name, _node_count, _parent, greedy, lb, set, ub
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent, set
- repr_exclude: _list, _parent

- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

A

itemgetter(item, ...) –> itemgetter object

Return a callable object that fetches the given item(s) from its operand. After f = itemgetter(2), the call f(r) returns r[2]. After g = itemgetter(2, 5, 3), the call g(r) returns (r[2], r[5], r[3])

generate_set(***kwargs*)

match(*seq*, ***kwargs*)

validate()

class syn.schema.b.sequence.**Sequence**(***kwargs*)

Bases: *syn.schema.b.sequence.SchemaNode*

Denotes a sequence. The only SchemaNode that can denote a sequence.

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

set [Optional]: ***SetNode*** Internal set representation

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: True
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()

- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children, elems

Groups:

- _all: _id, _list, _name, _node_count, _parent, set
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent, set
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

enumerate(***kwargs*)
Iterate through all possible sequences (lists). By default, will stop after 50 items have been yielded. This value can be change by supplying a different value via the max_enumerate kwarg.

generate_set(***kwargs*)

get_one(***kwargs*)
Returns one possible sequence (list). May return the same value on multiple invocations.

match(*seq*, ***kwargs*)
If the schema matches seq, returns a list of the matched objects. Otherwise, returns MatchFailure instance.

sample(***kwargs*)
Returns one possible sequence (list). The selection is randomized.

validate()

class syn.schema.b.sequence.**Match**(***kwargs*)
Bases: *syn.base.b.wrapper.ListWrapper*

Keyword-Only Arguments:

_list: ***list*** The wrapped list

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: False
- make_hashable: False

- make_type_object: True
- max_len: None
- min_len: None
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: _list
- copy_copy: _list
- _internal: _list
- str_exclude: _list

class syn.schema.b.sequence.**MatchFailure**(***kwargs*)

Bases: *syn.base.b.base.Base*

Keyword-Only Arguments:

fails [Optional]: ***list*** List of sub-failures

message: ***basestring*** Reason for failure

seq: ***IterableList*** The sequence that failed to match

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()

- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: fails, message, seq

exception `syn.schema.b.sequence.`**`MatchFailed`**(*msg*, *seq*, *fails=None*)

Bases: `exceptions.Exception`

`failure`()

Module contents

Module contents

syn.serialize package

Subpackages

syn.serialize.a package

Module contents

Module contents

syn.sets package

Subpackages

syn.sets.b package

Submodules

syn.sets.b.base module

class `syn.sets.b.base.`**`SetNode`**(***kwargs*)

Bases: *syn.tree.b.node.Node*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: None
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

complement(*universe*)

difference(*other*)

display(***kwargs*)

Returns a pretty string representation of the set.

enumerate(***kwargs*)

expected_size()

get_one(***kwargs*)
Return one element from the set, regardless of sampling bias, without evaluating any sets.

hasmember(*item*)

intersection(**args*)

issubset(*other*)

issuperset(*other*)

lazy_enumerate(***kwargs*)
Enumerate without evaluating any sets.

lazy_sample(***kwargs*)
Sample without evaluating any sets.

sample(***kwargs*)
Return a random element from the set. Method should try to avoid introducing a sampling bias.

simplify()

size()
Returns the cardinality of the set.

size_limits()
Returns the lower and upper bounds of set size.

to_set(***kwargs*)

union(**args*)

syn.sets.b.leaf module

class syn.sets.b.leaf.**SetLeaf**(***kwargs*)
Bases: *syn.sets.b.base.SetNode*

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: True
- descendant_exclude: ()
- id_equality: False
- init_validate: False

- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

class syn.sets.b.leaf.**SetWrapper**(*set*, ***kwargs*)

Bases: *syn.sets.b.leaf.SetLeaf*

Positional Arguments:

set: *set*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- •args: ('set',)
- •autodoc: True
- •coerce_args: True
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent, set
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

complement(**args*, ***kwargs*)

difference(**args*, ***kwargs*)

display(***kwargs*)

enumerate(***kwargs*)

hasmember(*item*)

intersection(**args*, ***kwargs*)

issubset(**args*, ***kwargs*)

issuperset(**args*, ***kwargs*)

sample(***kwargs*)

size()

to_set(***kwargs*)

union(**args*, ***kwargs*)

class syn.sets.b.leaf.**TypeWrapper**(*type*, ***kwargs*)
Bases: *syn.sets.b.leaf.SetLeaf*

The idea is that a type implicitly represents the set of all of its valid instances.

Positional Arguments:

type: *Type*

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

Class Options:

- args: ('type',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()

- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, type
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

display(***kwargs*)

enumerate(***kwargs*)

hasmember(*item*)

sample(***kwargs*)

size()

to_set(***kwargs*)

class syn.sets.b.leaf.**ClassWrapper**(*type*, ***kwargs*)

Bases: *syn.sets.b.leaf.SetLeaf*

The idea is that a type implicitly represents the set of all of its subclasses, including itself.

Positional Arguments:

type: *type*

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

subclasses: *list* (*type*)

Class Options:

- •args: ('type',)
- •autodoc: True
- •coerce_args: True
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent, subclasses, type
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

display(***kwargs*)

enumerate(***kwargs*)

hasmember(*item*)

sample(***kwargs*)

size()

to_set(**kwargs)

class syn.sets.b.leaf.**Special**(**kwargs)
Bases: *syn.sets.b.leaf.SetLeaf*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: True
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list

- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

class syn.sets.b.leaf.**Empty**(**kwargs*)

Bases: *syn.sets.b.leaf.Special*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: True
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

display(***kwargs*)

enumerate(***kwargs*)

hasmember(*other*)

issubset(*other*)

issuperset(*other*)

overlaps(*other*)

size()

to_set(***kwargs*)

syn.sets.b.operators module

class syn.sets.b.operators.**SetOperator**(***kwargs*)

Bases: *syn.sets.b.base.SetNode*

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False

- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

display(***kwargs*)

enumerate(***kwargs*)

get_one(***kwargs*)

sample(***kwargs*)

size()

symbol = None

class syn.sets.b.operators.**Union**(***kwargs*)

Bases: *syn.sets.b.operators.SetOperator*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent

- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

enumerate(***kwargs*)

hasmember(*other*)

sample(***kwargs*)

size_limits()

symbol = 'I'

to_set(***kwargs*)

class syn.sets.b.operators.**Intersection**(***kwargs*)
Bases: *syn.sets.b.operators.SetOperator*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

enumerate(***kwargs*)

hasmember(*other*)

sample(***kwargs*)

size_limits()

symbol = '&'

to_set(***kwargs*)

class syn.sets.b.operators.**Difference**(***kwargs*)

Bases: *syn.sets.b.operators.SetOperator*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True

- max_len: 2
- min_len: 2
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

A

itemgetter(item, ...) –> itemgetter object

Return a callable object that fetches the given item(s) from its operand. After f = itemgetter(2), the call f(r) returns r[2]. After g = itemgetter(2, 5, 3), the call g(r) returns (r[2], r[5], r[3])

B

itemgetter(item, ...) –> itemgetter object

Return a callable object that fetches the given item(s) from its operand. After f = itemgetter(2), the call f(r) returns r[2]. After g = itemgetter(2, 5, 3), the call g(r) returns (r[2], r[5], r[3])

enumerate(***kwargs*)

hasmember(*other*)

sample(***kwargs*)

size_limits()

symbol = '-'

to_set(***kwargs*)

class syn.sets.b.operators.**Product**(***kwargs*)

Bases: *syn.sets.b.operators.SetOperator*

Cartesian Product

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list

- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

enumerate(**kwargs)

hasmember(*other*)

sample(**kwargs)

size_limits()

symbol = 'x'

to_set(**kwargs)

syn.sets.b.range module

class syn.sets.b.range.**Range**(*lb*, *ub*, ***kwargs*)

Bases: *syn.sets.b.leaf.SetLeaf*

Positional Arguments:

lb: ***int*** The lower bound

ub: ***int*** The upper bound

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ('lb', 'ub')
- autodoc: True
- coerce_args: True
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0

- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, lb, ub
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

complement(*universe*)

difference(*other*)

display(***kwargs*)

enumerate(***kwargs*)

hasmember(*other*)

intersection(**args*)

issubset(*other*)

issuperset(*other*)

overlaps(*other*)

sample(***kwargs*)

size()

to_set(***kwargs*)

union(**args*)

validate()

class syn.sets.b.range.**IntRange**(*lb*, *ub*, ***kwargs*)

Bases: *syn.sets.b.range.Range*

Positional Arguments:

lb: ***int*** The lower bound

ub: ***int*** The upper bound

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ('lb', 'ub')
- autodoc: True
- coerce_args: True
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, lb, ub
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

hasmember(*other*)

class syn.sets.b.range.**StrRange**(*lb=32*, *ub=126*, ***kwargs*)

Bases: *syn.sets.b.range.Range*

Positional Arguments:

lb (*default* = 32): *int* The lower bound

ub (*default* = 126): *int* The upper bound

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

Class Options:

- args: ('lb', 'ub')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:

- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, lb, ub
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

display(***kwargs*)

enumerate(***kwargs*)

hasmember(*other*)

sample(***kwargs*)

to_set(***kwargs*)

Module contents

Module contents

syn.tagmathon package

Subpackages

syn.tagmathon.b package

Submodules

syn.tagmathon.b.base module

class syn.tagmathon.b.base.**SyntagmathonNode**(***kwargs*)
Bases: *syn.tree.b.node.Node*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent

- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

eval(*env*, ***kwargs*)

to_python(***kwargs*)

class syn.tagmathon.b.base.**Variable**(*name*, ***kwargs*)
Bases: *syn.tagmathon.b.base.SyntagmathonNode*

Positional Arguments:

name: *basestring*

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

Class Options:

- args: ('name',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, name
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

eval(*env*, ***kwargs*)

to_python(***kwargs*)

syn.tagmathon.b.base.**vars**(**args*)

syn.tagmathon.b.builtin module

class syn.tagmathon.b.builtin.**BuiltinFunction**(*name*, *signature*, *body*, ***kwargs*)

Bases: *syn.tagmathon.b.function.Function*

Positional Arguments:

name: *Variable* | *basestring* signature: *list* (*Variable*) body: <callable>

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

pass_kwargs (*default* = False): *bool* placeholder (*default* = False): *bool* python: <callable>

Class Options:

- args: ('name', 'signature', 'body')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False

- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, body, name, pass_kwargs, placeholder, python, signature
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

call(*env*, ***kwargs*)

syn.tagmathon.b.compiler module

syn.tagmathon.b.compiler.**to_python**(*obj*, ***kwargs*)

syn.tagmathon.b.compiler.**compile_to_python**(*obj*, ***kwargs*)

syn.tagmathon.b.function module

class syn.tagmathon.b.function.**Function**(*name*, *signature*, *body*, ***kwargs*)

Bases: *syn.tagmathon.b.base.SyntagmathonNode*

Positional Arguments:

name: *Variable* | *basestring* signature: *list* (*Variable*) body: *list* (*SyntagmathonNode*) | *tuple*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

placeholder (*default* = False): *bool*

Class Options:

- •args: ('name', 'signature', 'body')
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent, body, name, placeholder, signature
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent

- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

call(*env*, ***kwargs*)

eval(*env*, ***kwargs*)

get_name()

to_python(***kwargs*)

class syn.tagmathon.b.function.**Call**(*func*, *args*, ***kwargs*)
Bases: *syn.tagmathon.b.base.SyntagmathonNode*

Positional Arguments:

func: *Function* args: *dict*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ('func', 'args')
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()

•init_order: ()

•metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

•setstate_hooks: ()

Aliases:

•_list: _children

Groups:

•_all: _id, _list, _name, _node_count, _parent, args, func

•copy_copy: _list

•hash_exclude: _parent

•generate_exclude: _node_count, _parent

•_internal: _id, _list, _name, _node_count, _parent

•repr_exclude: _list, _parent

•eq_exclude: _parent

•getstate_exclude: _parent

•str_exclude: _id, _list, _name, _node_count, _parent

eval(*env*, ***kwargs*)

to_python(***kwargs*)

class syn.tagmathon.b.function.**Special**(*name*, *signature*, *body*, ***kwargs*)
Bases: *syn.tagmathon.b.function.Function*

Positional Arguments:

name: *Variable* | *basestring* signature: *list* (*Variable*) body: *list* (*SyntagmathonNode*) | *tuple*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

placeholder (*default* = False): *bool*

Class Options:

•args: ('name', 'signature', 'body')

•autodoc: True

•coerce_args: False

•descendant_exclude: ()

•id_equality: False

•init_validate: False

•make_hashable: False

- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, body, name, placeholder, signature
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

class syn.tagmathon.b.function.**SpecialCall**(*func*, *args*, ***kwargs*)

Bases: *syn.tagmathon.b.function.Call*

Positional Arguments:

func: *Function* args: *dict*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- •args: (‘func’, ‘args’)
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: (‘coerce_hooks’, ‘init_hooks’, ‘create_hooks’, ‘setstate_hooks’)
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent, args, func
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

eval(*env*, ***kwargs*)

to_python(***kwargs*)

syn.tagmathon.b.interpreter module

class syn.tagmathon.b.interpreter.**Frame**(***kwargs*)
Bases: *syn.base.b.base.Base*

Keyword-Only Arguments:

globals: *dict* locals: *dict*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: globals, locals

gensym()

items()

set_global(*key*, *value*)

update(*dct*)

class syn.tagmathon.b.interpreter.**Env**(***kwargs*)
Bases: *syn.base.b.base.Base*

Keyword-Only Arguments:

frames: *list* (*Frame*)

Class Options:

- args: ()
- autodoc: True
- coerce_args: False

- id_equality: False
- init_validate: True
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: frames

current_frame

gensym()

globals()

items()

locals()

pop()

push(*dct*)

set_global(*key*, *value*)

update(*dct*)

syn.tagmathon.b.interpreter.**eval**(*obj*, *env=None*, ***kwargs*)

Module contents

Module contents

syn.tree package

Subpackages

syn.tree.b package

Submodules

syn.tree.b.node module

class syn.tree.b.node.**Node**(***kwargs*)

Bases: *syn.base.b.wrapper.ListWrapper*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent

- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

add_child(*node*, *index=None*)

ancestors(*include_self=False*)

attributes()

children(*reverse=False*)

collect_by_type(*typ*)
A more efficient way to collect nodes of a specified type than collect_nodes.

collect_nodes(*attr=None*, *val=None*, *key=None*)

collect_rootward(*nodes=None*)

depth_first(*func=<function <lambda>>*, *filt=<function <lambda>>*, *reverse=False*, *include_toplevel=True*, *top_level=True*, *depth=0*, *yield_depth=False*)

descendants(*include_self=False*)

find_type(*typ*, *children_only=False*)

following()

id()

name()

node_count()

parent()

preceding()

remove_child(*node*)

root()

rootward(*func=<function <lambda>>*, *filt=<function <lambda>>*, *include_toplevel=True*, *top_level=True*)

set_child_parents(*parent=None*, *recurse=False*, *override=False*)

siblings(*preceding=False*, *following=False*, *axis=False*)

validate()

exception syn.tree.b.node.**TreeError**
Bases: exceptions.Exception

syn.tree.b.query module

class syn.tree.b.query.**Ancestor**(***kwargs*)
Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

include_self (*default* = False): *bool*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, include_self
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent

- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Any**(***kwargs*)

Bases: *syn.tree.b.query.Predicate*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

eval(*node*, ***kwargs*)

class syn.tree.b.query.**Attribute**(***kwargs*)

Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()

- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Axis**(***kwargs*)

Bases: *syn.tree.b.query.Query*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False

- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

class syn.tree.b.query.**Child**(***kwargs*)

Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False

- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Comparison**(***kwargs*)

Bases: *syn.tree.b.query.Function*

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 1
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

arity = 2

class syn.tree.b.query.**Descendant**(***kwargs*)

Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

include_self (*default* = False): *bool*

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 1
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent, include_self
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent

- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Eq**(***kwargs*)

Bases: *syn.tree.b.query.Comparison*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 2
- min_len: 2
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

func()
eq(a, b) – Same as a==b.

class syn.tree.b.query.**Following**(***kwargs*)
Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

include_self (*default* = False): *bool*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()

- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, include_self
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Function**(***kwargs*)

Bases: *syn.tree.b.query.Query*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None

- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

arity = None

eval(*values*, ***kwargs*)

func = None

class syn.tree.b.query.**Ge**(***kwargs*)

Bases: *syn.tree.b.query.Comparison*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True

- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 2
- min_len: 2
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

func()
ge(a, b) – Same as a>=b.

class syn.tree.b.query.**Gt**(**kwargs*)
Bases: *syn.tree.b.query.Comparison*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 2
- •min_len: 2
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

func()
gt(a, b) – Same as a>b.

class syn.tree.b.query.**Identity**(**kwargs)
Bases: *syn.tree.b.query.Function*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent

- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

arity = 1

func(*x*)

class syn.tree.b.query.**Le**(***kwargs*)

Bases: *syn.tree.b.query.Comparison*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 2
- min_len: 2
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

func()
le(a, b) – Same as a<=b.

class syn.tree.b.query.**Lt**(**kwargs)
Bases: *syn.tree.b.query.Comparison*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 2
- min_len: 2
- must_be_root: False
- optional_none: True
- register_subclasses: False

- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

func()
lt(a, b) – Same as a<b.

class syn.tree.b.query.**Name**(*name*, ***kwargs*)
Bases: *syn.tree.b.query.Predicate*

Positional Arguments:

name: *basestring*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

name_attr (*default* = _name): *basestring*

Class Options:

- args: ('name',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()

- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, name, name_attr
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

eval(*node*, ***kwargs*)

class syn.tree.b.query.**Ne**(***kwargs*)

Bases: *syn.tree.b.query.Comparison*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 2
- •min_len: 2
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

func()

ne(a, b) – Same as a!=b.

class syn.tree.b.query.**Parent**(**kwargs*)

Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent

- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Position**(*pos*, ***kwargs*)

Bases: *syn.tree.b.query.Predicate*

Positional Arguments:

pos: *int*

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

pos_attr (*default* = _nodeset_position): *basestring* start_offset (*default* = 0): *int*

Class Options:

- args: ('pos',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()

- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, pos, pos_attr, start_offset
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

eval(*node*, ***kwargs*)

class syn.tree.b.query.**Preceding**(***kwargs*)

Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

include_self (*default* = False): *bool*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False

- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, include_self
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Predicate**(***kwargs*)

Bases: *syn.tree.b.query.Query*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False

- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: 0
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

eval(*node*, ***kwargs*)

class syn.tree.b.query.**Query**(***kwargs*)

Bases: *syn.tree.b.node.Node*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Root**(***kwargs*)

Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent

- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Self**(***kwargs*)

Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Sibling**(***kwargs*)

Bases: *syn.tree.b.query.Axis*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

following (*default* = False): *bool* preceding (*default* = False): *bool*

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- one_true: [('following', 'preceding')]
- optional_none: True
- register_subclasses: False

- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, following, preceding
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Type**(*type=<syn.type.a.type.AnyType object at 0x7fa0c114f910>*, ***kwargs*)

Bases: *syn.tree.b.query.Query*

Positional Arguments:

type (*default* = <syn.type.a.type.AnyType object at 0x7fa0c114f910>): *Type*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ('type',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False

- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 1
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, type
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

iterate(*node*, ***kwargs*)

class syn.tree.b.query.**Value**(*value*, ***kwargs*)

Bases: *syn.tree.b.query.Query*

Positional Arguments:

value: any

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ('value',)
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 0
- min_len: None
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent, value
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

class syn.tree.b.query.**Where**(***kwargs*)

Bases: *syn.tree.b.query.Query*

Keyword-Only Arguments:

_id [Optional]: *int* Integer id of the node

_list: *list* Child nodes

_name [Optional]: *basestring* Name of the node (for display purposes)

_node_count: *int* The number of nodes in the subtreerooted by this node.

_parent [Optional]: *Node* Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: 2
- min_len: 2
- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent

- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

cond

itemgetter(item, ...) –> itemgetter object

Return a callable object that fetches the given item(s) from its operand. After f = itemgetter(2), the call f(r) returns r[2]. After g = itemgetter(2, 5, 3), the call g(r) returns (r[2], r[5], r[3])

node

itemgetter(item, ...) –> itemgetter object

Return a callable object that fetches the given item(s) from its operand. After f = itemgetter(2), the call f(r) returns r[2]. After g = itemgetter(2, 5, 3), the call g(r) returns (r[2], r[5], r[3])

syn.tree.b.tree module

class syn.tree.b.tree.**Tree**(*root*, ***kwargs*)

Bases: *syn.base.b.base.Base*

Positional Arguments:

root: ***Node*** The root node of the tree

Keyword-Only Arguments:

id_dict: ***dict*** **(any =>** ***Node*****)** Mapping of ids to nodes

node_counter: ***Counter*** Node id counter

node_types: ***list*** **(*basestring*)** List of all tree node types

nodes: ***list*** **(*Node*)** List of all tree nodes

type_dict: ***dict*** **(any =>** ***list*** **(*Node*))** Mapping of type names to node lists

Class Options:

- args: ('root',)
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()

- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: id_dict, node_counter, node_types, nodes, root, type_dict
- generate_exclude: id_dict, node_counter, node_types, nodes, type_dict
- eq_exclude: node_counter
- str_exclude: id_dict, node_counter, node_types, nodes, type_dict

add_node(*node*, ***kwargs*)

depth_first(*node=<function do_nothing>*, *stop_test=<function do_nothing>*, *_return=<function identity>*, *current_node='root'*, ***kwargs*)

find_one(**args*, ***kwargs*)

get_node_by_id(*node_id*)

query(*q*, *context=None*)

rebuild(***kwargs*)
Repopulate the node-tracking data structures. Shouldn't really ever be needed.

remove_node(*node*, ***kwargs*)

replace_node(*source*, *dest*, ***kwargs*)

search_rootward(*node=<function do_nothing>*, *stop_test=<function do_nothing>*, *_return=<function identity>*, *current_node='root'*, ***kwargs*)

validate()

syn.tree.b.tree.**do_nothing**(**args*, ***kwargs*)

syn.tree.b.tree.**identity**(*x*)

Module contents

Module contents

syn.type package

Subpackages

syn.type.a package

Submodules

syn.type.a.ext module

class syn.type.a.ext.**Callable**
Bases: *syn.type.a.type.TypeExtension*

The value must be callable.

check(*value*)

display()

generate(***kwargs*)

class syn.type.a.ext.**Sequence**(*item_type*, *seq_type=<class '_abcoll.Sequence'>*)
Bases: *syn.type.a.type.TypeExtension*

The value must be a sequence whose values are the provided type.

check(*values*)

coerce(*values*, ***kwargs*)

display()

generate(***kwargs*)

item_type

register_generable = True

rst()

seq_type

class syn.type.a.ext.**Tuple**(*types*, *length=None*, *uniform=False*)
Bases: *syn.type.a.type.TypeExtension*

For defining tuple types.

check(*values*)

coerce(*values*, ***kwargs*)

display()

generate(***kwargs*)

length

register_generable = True

rst()

types

uniform

class syn.type.a.ext.**Mapping**(*value_type*, *map_type=<class '_abcoll.Mapping'>*)
Bases: *syn.type.a.type.TypeExtension*

The value must be a mapping whose values are the provided type.

check(*dct*)

coerce(*dct*, ***kwargs*)

display()

generate(***kwargs*)

map_type

register_generable = True

rst()

value_type

class syn.type.a.ext.**Hashable**
Bases: *syn.type.a.type.TypeExtension*

The value must be hashable.

check(*value*)

display()

generate(***kwargs*)

class syn.type.a.ext.**This**
Bases: *syn.type.a.type.TypeExtension*

syn.type.a.type module

class syn.type.a.type.**Type**
Bases: object

A representation for various possible types syn supports.

check(*value*)

coerce(*value*, ***kwargs*)

classmethod dispatch(*obj*)

display()
Returns a quasi-intuitive string representation of the type.

enumeration_value(*x*, ***kwargs*)
Return the enumeration value for *x* for this type.

generate(***kwargs*)
Returns a value for this type.

query(*value*)

query_exception(*value*)

register_generable = False

rst()
Returns a string representation of the type for RST documentation.

validate(*value*)

class syn.type.a.type.**AnyType**
Bases: *syn.type.a.type.Type*

check(*value*)

coerce(*value*, ***kwargs*)

display()

enumeration_value(*x*, ***kwargs*)

generate(***kwargs*)

validate(*value*)

class syn.type.a.type.**TypeType**(*typ*)
Bases: *syn.type.a.type.Type*

call_coerce

call_validate

check(*value*)

coerce(*value*, ***kwargs*)

display()

enumeration_value(*x*, ***kwargs*)

generate(***kwargs*)

register_generable = **True**

rst()

type

validate(*value*)

class syn.type.a.type.**ValuesType**(*values*)
Bases: *syn.type.a.type.Type*

A set (or list) of values, any of which is valid.

Think of this is a denotational definition of the type.

check(*value*)

coerce(*value*, ***kwargs*)

display()

enumeration_value(*x*, ***kwargs*)

generate(***kwargs*)

indexed_values

register_generable = **True**

validate(*value*)

values

class syn.type.a.type.**MultiType**(*types*)
Bases: *syn.type.a.type.Type*

A tuple of type specifiers, any of which may be valid.

check(*value*)

coerce(*value*, ***kwargs*)

display()

enumeration_value(*x*, ***kwargs*)

generate(***kwargs*)

is_typelist

register_generable = True

rst()

typelist

typemap

types

typestr

validate(*value*)

class syn.type.a.type.**Set**(*set*)
Bases: *syn.type.a.type.Type*

For explicitly wrapping a SetNode as a type (since automatic dispatching cannot be implemented at this level).

check(*value*)

coerce(*value*, ***kwargs*)

display()

generate(***kwargs*)

register_generable = True

validate(*value*)

class syn.type.a.type.**Schema**(*schema*)
Bases: *syn.type.a.type.Type*

For explicitly wrapping a Schema as a type (since automatic dispatching cannot be implemented at this level).

check(*value*)

coerce(*value*, ***kwargs*)

display()

generate(***kwargs*)

register_generable = True

validate(*value*)

class syn.type.a.type.**TypeExtension**
Bases: *syn.type.a.type.Type*

For extending the type system.

validate(*value*)

Module contents

Module contents

syn.types package

Subpackages

syn.types.a package

Submodules

syn.types.a.base module

class syn.types.a.base.**Type**(*obj*)
Bases: object

attrs(**kwargs*)

collect(*func*, **kwargs*)

classmethod deserialize(*dct*, **kwargs_*)

classmethod deserialize_dispatch(*obj*)

classmethod dispatch(*obj*)

classmethod enumerate(**kwargs*)

classmethod enumeration_value(*x*, **kwargs*)

estr(**kwargs*)
Should return a string that can eval into an equivalent object

find_ne(*other*, *func=<built-in function eq>*, **kwargs*)

gen_type = None

gen_types = None

classmethod generate(**kwargs*)

hashable(*self_*)

pairs(**kwargs*)

primitive_form(**kwargs*)

rstr(**kwargs*)
The idea is somethinig like a recursive str().

ser_args = ()

ser_attrs = None

ser_kwargmap = {}

ser_kwargs = ()

serialize(**kwargs*)

classmethod serialize_type(*typ*, **kwargs*)

type
alias of object

classmethod type_dispatch(*typ*)

visit(*k*, **kwargs*)

visit_len(**kwargs*)

class syn.types.a.base.**TypeType**(*obj*)
Bases: *syn.types.a.base.Type*

attrs(**kwargs*)

class type(*object*) → the object's type
Bases: object

type(name, bases, dict) -> a new type

mro() → list
return a type's method resolution order

syn.types.a.base.**deserialize**(*obj*, ***kwargs*)

syn.types.a.base.**enumerate**(*typ*, ***kwargs*)

syn.types.a.base.**estr**(*obj*, ***kwargs*)
Return a string that can evaluate into an equivalent object.

NOTE: this function is experimental and not fully supported.

syn.types.a.base.**find_ne**(*a*, *b*, *func=<built-in function eq>*, ***kwargs*)

syn.types.a.base.**generate**(*typ*, ***kwargs*)

syn.types.a.base.**attrs**(*obj*, ***kwargs*)

syn.types.a.base.**hashable**(*obj*, ***kwargs*)

syn.types.a.base.**rstr**(*obj*, ***kwargs*)

syn.types.a.base.**serialize**(*obj*, ***kwargs*)

syn.types.a.base.**visit**(*obj*, *k=0*, ***kwargs*)

syn.types.a.base.**safe_sorted**(*obj*, ***kwargs*)

syn.types.a.base.**pairs**(*obj*, ***kwargs*)

syn.types.a.base.**enumeration_value**(*typ*, *x*, ***kwargs*)

syn.types.a.base.**primitive_form**(*obj*, ***kwargs*)
Return obj, if possible, in a form composed of primitive or builtin objects.

syn.types.a.base.**collect**(*obj*, *func=<function <lambda>>*, ***kwargs*)

syn.types.a.mapping module

class syn.types.a.mapping.**Mapping**(**args*, ***kwargs*)
Bases: *syn.types.a.base.Type*

classmethod deserialize(*dct*, ***kwargs*)

estr(***kwargs*)

type
alias of *Mapping*

class syn.types.a.mapping.**Dict**(**args*, ***kwargs*)
Bases: *syn.types.a.mapping.Mapping*

type
alias of dict

syn.types.a.ne module

class syn.types.a.ne.**ValueExplorer**(*value*, *index=None*, *key=None*, *attr=None*, *prompt='(ValEx) '*, *step=1*)

Bases: *syn.base_utils.repl.REPL*

command_display_current_value()

command_display_value()

command_down(*num='1'*)

command_help = {'c': 'display current_value', 'e': 'eval the argument', 'd': 'go down the stack', 'h': 'display available

command_step(*step='1'*)

command_up(*num='1'*)

commands = {'c': <function command_display_current_value>, 'e': <function eval>, 'd': <function command_down>, 'h

depth_first(*leaves_only=False*)

display()

down()

reset()

step(*step=None*)

up()

class syn.types.a.ne.**DiffExplorer**(*A*, *B*, *prompt='(DiffEx) '*)

Bases: *syn.base_utils.repl.REPL*

command_display_current_value()

command_display_value()

command_down(*num='1'*)

command_find()

command_help = {'c': 'display current_value', 'e': 'eval the argument', 'd': 'go down the stack', 'f': 'find the inequality

command_step(*step='1'*)

command_up(*num='1'*)

commands = {'c': <function command_display_current_value>, 'e': <function eval>, 'd': <function command_down>, 'f

current_value

depth_first(***kwargs*)

display()

down()

reset()

step(**args*, ***kwargs*)

up()

value

exception syn.types.a.ne.**ExplorationError**

Bases: exceptions.Exception

syn.types.a.ne.**deep_comp**(*A*, *B*, *func=<built-in function eq>*, ***kwargs*)

syn.types.a.ne.**feq_comp**(*a*, *b*, *tol=1.4901161193847696e-08*, *relative=True*)

syn.types.a.ne.**deep_feq**(*A*, *B*, *tol=1.4901161193847696e-08*, *relative=True*)

syn.types.a.ne.**is_visit_primitive**(*obj*)
Returns true if properly visiting the object returns only the object itself.

class syn.types.a.ne.**NEType**(*A*, *B*)
Bases: object

explorer()

message()

class syn.types.a.ne.**NotEqual**(*A*, *B*)
Bases: *syn.types.a.ne.NEType*

message()

class syn.types.a.ne.**DiffersAtIndex**(*A*, *B*, *index*)
Bases: *syn.types.a.ne.NEType*

explorer()

message()

class syn.types.a.ne.**DiffersAtKey**(*A*, *B*, *key*)
Bases: *syn.types.a.ne.NEType*

explorer()

message()

class syn.types.a.ne.**DiffersAtAttribute**(*A*, *B*, *attr*)
Bases: *syn.types.a.ne.NEType*

explorer()

message()

class syn.types.a.ne.**DifferentLength**(*A*, *B*)
Bases: *syn.types.a.ne.NEType*

message()

class syn.types.a.ne.**DifferentTypes**(*A*, *B*)
Bases: *syn.types.a.ne.NEType*

message()

class syn.types.a.ne.**SetDifferences**(*A*, *B*)
Bases: *syn.types.a.ne.NEType*

message()

class syn.types.a.ne.**KeyDifferences**(*A*, *B*)
Bases: *syn.types.a.ne.NEType*

message()

syn.types.a.numeric module

class syn.types.a.numeric.**Numeric**(*obj*)
Bases: *syn.types.a.base.Type*

estr(**kwargs*)

type = None

class syn.types.a.numeric.**Bool**(*obj*)
Bases: *syn.types.a.numeric.Numeric*

type
alias of bool

class syn.types.a.numeric.**Int**(*obj*)
Bases: *syn.types.a.numeric.Numeric*

type
alias of int

class syn.types.a.numeric.**Long**(*obj*)
Bases: *syn.types.a.numeric.Numeric*

estr(**kwargs*)

type
alias of long

class syn.types.a.numeric.**Float**(*obj*)
Bases: *syn.types.a.numeric.Numeric*

type
alias of float

class syn.types.a.numeric.**Complex**(*obj*)
Bases: *syn.types.a.numeric.Numeric*

ser_args = ('real', 'imag')

type
alias of complex

syn.types.a.sequence module

class syn.types.a.sequence.**Sequence**(*obj*)
Bases: *syn.types.a.base.Type*

classmethod deserialize(*seq*, **kwargs*)

estr(**kwargs*)

type
alias of *Sequence*

class syn.types.a.sequence.**List**(*obj*)
Bases: *syn.types.a.sequence.Sequence*

type
alias of list

class syn.types.a.sequence.**Tuple**(*obj*)
Bases: *syn.types.a.sequence.Sequence*

type
alias of `tuple`

syn.types.a.set module

class `syn.types.a.set.`**Set**(**args*, ***kwargs*)
Bases: *syn.types.a.base.Type*

estr(***kwargs*)

type
alias of `set`

class `syn.types.a.set.`**FrozenSet**(**args*, ***kwargs*)
Bases: *syn.types.a.set.Set*

type
alias of `frozenset`

syn.types.a.special module

class `syn.types.a.special.`**NONE**(*obj*)
Bases: *syn.types.a.base.Type*

classmethod deserialize(*dct*, ***kwargs*)

estr(***kwargs*)

classmethod serialize_type(*typ*, ***kwargs*)

type
alias of `NoneType`

syn.types.a.string module

class `syn.types.a.string.`**String**(*obj*)
Bases: *syn.types.a.base.Type*

type
alias of `str`

class `syn.types.a.string.`**Unicode**(*obj*)
Bases: *syn.types.a.string.String*

estr(***kwargs*)

rstr(***kwargs*)

type
alias of `unicode`

class `syn.types.a.string.`**Bytes**(*obj*)
Bases: *syn.types.a.string.String*

estr(***kwargs*)

type = None

class syn.types.a.string.**Basestring**(*obj*)
Bases: *syn.types.a.string.String*

type
alias of basestring

Module contents

Module contents

Encapsulates certain functionality that ought to be available for all Python objects.

syn.util package

Subpackages

syn.util.constraint package

Subpackages

syn.util.constraint.b package

Submodules

syn.util.constraint.b.base module

class syn.util.constraint.b.base.**Domain**(*vars*, ***kwargs*)
Bases: *syn.base.b.base.Base*

Positional Arguments:

vars: *Mapping* (any => *SetNode*)

Class Options:

- args: ('vars',)
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()

- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Groups:

- •_all: vars

copy(**args*, ***kwargs*)

display(***kwargs*)

class syn.util.constraint.b.base.**Constraint**(*args*, ***kwargs*)
Bases: *syn.base.b.base.Base*

Positional Arguments:

args: *Sequence*

Class Options:

- •args: ('args',)
- •autodoc: True
- •coerce_args: False
- •id_equality: False
- •init_validate: True
- •make_hashable: True
- •make_type_object: True
- •optional_none: False
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Groups:

- •_all: args

check(***kwargs*)

display(***kwargs*)

preprocess(*domain*, ***kwargs*)

class syn.util.constraint.b.base.**Problem**(*domain*, *constraints*, ***kwargs*)
Bases: *syn.base.b.base.Base*

Positional Arguments:

domain: *Domain* constraints: *list* (*Constraint*)

Class Options:

- args: ('domain', 'constraints')
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: constraints, domain

check(*theory*)

display(***kwargs*)

validate()

syn.util.constraint.b.constraints module

class syn.util.constraint.b.constraints.**FunctionConstraint**(*func*, *args*, ***kwargs*)
Bases: *syn.util.constraint.b.base.Constraint*

Positional Arguments:

func: <callable> args: *Sequence*

Class Options:

- args: ('func', 'args')
- autodoc: True
- coerce_args: False

- id_equality: False
- init_validate: True
- make_hashable: True
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: args, func

check(***kwargs*)

display(***kwargs*)

class syn.util.constraint.b.constraints.**AllDifferentConstraint**(*args*, ***kwargs*)

Bases: *syn.util.constraint.b.base.Constraint*

Positional Arguments:

args: *Sequence*

Class Options:

- args: ('args',)
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: True
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()

- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: args

check(**kwargs)

display(**kwargs)

class syn.util.constraint.b.constraints.**EqualConstraint**(*arg*, *value*, **kwargs)
Bases: *syn.util.constraint.b.base.Constraint*

Positional Arguments:

arg: *basestring* value: any

Keyword-Only Arguments:

args: *Sequence*

Class Options:

- args: ('arg', 'value')
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: True
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: arg, args, value

check(**kwargs)

display(**kwargs)

preprocess(*domain*, **kwargs)

syn.util.constraint.b.solvers module

class syn.util.constraint.b.solvers.**Solver**(*problem*, ***kwargs*)
Bases: *syn.base.b.base.Base*

Positional Arguments:

problem: *Problem*

Class Options:

- args: ('problem',)
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: problem

solutions(***kwargs*)

class syn.util.constraint.b.solvers.**SimpleSolver**(*problem*, ***kwargs*)
Bases: *syn.util.constraint.b.solvers.Solver*

Enumerates through all possible solutions. Do not use for any substantially-sized domains!!!

Positional Arguments:

problem: *Problem*

Class Options:

- args: ('problem',)
- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True

- •make_hashable: False
- •make_type_object: True
- •optional_none: False
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Groups:

- •_all: problem

solutions(***kwargs*)

class syn.util.constraint.b.solvers.**RecursiveBacktrackSolver**(*problem*, ***kwargs*)
Bases: *syn.util.constraint.b.solvers.Solver*

Positional Arguments:

problem: *Problem*

Keyword-Only Arguments:

forward_checking (*default* = True): *bool* selection_method (*default* = mrv): ['mrv', 'random']

Class Options:

- •args: ('problem',)
- •autodoc: True
- •coerce_args: False
- •id_equality: False
- •init_validate: True
- •make_hashable: False
- •make_type_object: True
- •optional_none: False
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')

- setstate_hooks: ()

Groups:

- _all: forward_checking, problem, selection_method

choose_var(*uvars*, ***kwargs*)

forward_check(**args*, ***kwds*)

solutions(***kwargs*)

Module contents

Module contents

syn.util.log package

Subpackages

syn.util.log.b package

Submodules

syn.util.log.b.base module

class syn.util.log.b.base.**Event**(***kwargs*)

Bases: *syn.tree.b.node.Node*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

Class Options:

- args: ()
- autodoc: True
- coerce_args: False
- descendant_exclude: ()
- id_equality: False
- init_validate: False
- make_hashable: False
- make_type_object: True
- max_len: None
- min_len: None

- must_be_root: False
- optional_none: True
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Aliases:

- _list: _children

Groups:

- _all: _id, _list, _name, _node_count, _parent
- copy_copy: _list
- hash_exclude: _parent
- generate_exclude: _node_count, _parent
- _internal: _id, _list, _name, _node_count, _parent
- repr_exclude: _list, _parent
- eq_exclude: _parent
- getstate_exclude: _parent
- str_exclude: _id, _list, _name, _node_count, _parent

plaintext(***kwargs*)

class syn.util.log.b.base.**Logger**(*root*, ***kwargs*)

Bases: *syn.tree.b.tree.Tree*

Positional Arguments:

root: *Event* The root node of the tree

Keyword-Only Arguments:

current_parent: *Event* id_dict: *dict* (any => *Node*)

Mapping of ids to nodes

node_counter: *Counter* Node id counter

node_types: *list* (*basestring*) List of all tree node types

nodes: *list* (*Node*) List of all tree nodes

type_dict: *dict* (any => *list* (*Node*)) Mapping of type names to node lists

Class Options:

- args: ('root',)

- autodoc: True
- coerce_args: False
- id_equality: False
- init_validate: True
- make_hashable: False
- make_type_object: True
- optional_none: False
- register_subclasses: False
- repr_template:
- coerce_hooks: ()
- create_hooks: ()
- init_hooks: ()
- init_order: ()
- metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- setstate_hooks: ()

Groups:

- _all: current_parent, id_dict, node_counter, node_types, nodes, root, type_dict
- generate_exclude: id_dict, node_counter, node_types, nodes, type_dict
- eq_exclude: node_counter
- str_exclude: id_dict, node_counter, node_types, nodes, type_dict

add(*event*)

plaintext(***kwargs*)

pop()

push(*event*)

reset()

syn.util.log.b.events module

class syn.util.log.b.events.**StringEvent**(***kwargs*)

Bases: *syn.util.log.b.base.Event*

Keyword-Only Arguments:

_id [Optional]: ***int*** Integer id of the node

_list: ***list*** Child nodes

_name [Optional]: ***basestring*** Name of the node (for display purposes)

_node_count: ***int*** The number of nodes in the subtreerooted by this node.

_parent [Optional]: ***Node*** Parent of this node

s (*default* =): *basestring*

Class Options:

- •args: ()
- •autodoc: True
- •coerce_args: False
- •descendant_exclude: ()
- •id_equality: False
- •init_validate: False
- •make_hashable: False
- •make_type_object: True
- •max_len: None
- •min_len: None
- •must_be_root: False
- •optional_none: True
- •register_subclasses: False
- •repr_template:
- •coerce_hooks: ()
- •create_hooks: ()
- •init_hooks: ()
- •init_order: ()
- •metaclass_lookup: ('coerce_hooks', 'init_hooks', 'create_hooks', 'setstate_hooks')
- •setstate_hooks: ()

Aliases:

- •_list: _children

Groups:

- •_all: _id, _list, _name, _node_count, _parent, s
- •copy_copy: _list
- •hash_exclude: _parent
- •generate_exclude: _node_count, _parent
- •_internal: _id, _list, _name, _node_count, _parent
- •repr_exclude: _list, _parent
- •eq_exclude: _parent
- •getstate_exclude: _parent
- •str_exclude: _id, _list, _name, _node_count, _parent

plaintext(***kwargs*)

Module contents

Module contents

Module contents

Module contents

CHAPTER 2

Changelog

0.0.15 (2017-04-28)

- Added syn.util.constraint
- Added syn.util.log
- Added syn.python.b
- Added syn.tagmathon.b

0.0.14 (2017-03-05)

- Further integrated syn.schema and syn.sets into syn.type
- Added generation capabilities to syn.b.tree
- Added enhanced validation capabilities to syn.b.tree
- Added attribute attribute preservation for sub-class definitions
- Added pre-create hooks
- Added support for alternate means of attribute specification (syn.base.b.Harvester)
- Added ordering utilities (syn.base_utils.order)

0.0.13 (2017-02-14)

- Fixed issue preventing definition of custom __hash__ methods
- Integrated most syn.types functionality into syn.base.b.Base

0.0.12 (2017-02-12)

- Added iteration methods to syn.tree.b.Node
- Added syn.tree.query
- Added syn.types

0.0.11 (2016-08-16)

- Added syn.schema.sequence.Type for explicit type specifications
- Added repr template functionality to syn.base.b.Base
- Added Type random generation
- Added automatic metadata harvesting for sub-packages

0.0.10 (2016-08-12)

- Added lazy sampling and enumeration to syn.sets
- Removed syn.sets.Complement
- Added syn.sets.Product
- Added syn.schema.sequence (syn.schema.b.sequence)

0.0.9 (2016-08-09)

- Fixed setup.py for wheel

0.0.8 (2016-08-09)

- Added display() and rst() methods to Type classes (syn.type.a)
- Added class member/invocation auto-documentation
- Added make_hashable functionality to Base
- Added syn.sets (syn.sets.b)

0.0.7 (2016-07-20)

- Moved check_idempotence to syn.base.b.examine

0.0.6 (2016-07-20)

- Added context management utilities to base_utils
- Moved metaclass data population code to base.b.meta
- Added rudimentary init functionality to base.a.Attr and base.a.Base
- Added register_subclass functionality
- Refactored (improved) internal hook processing
- Added setstate_hook functionality
- Added _aliases functionality
- Added base.b.Base.istr()
- Added syn.tree functionality (syn.tree.b)
- Added syn.type.This type for recursive type definitions

0.0.5 (2016-07-12)

- **Added conversion classmethods to Base:**
  - from_object()
  - from_mapping()
  - from_sequence()
- Added _data member to Base for metaclass-populated values
- Fixed bug in _seq_opts propagation
- Added _seq_opts.metaclass_lookup functionality
- Changed init_hooks and coerce_hooks over to metaclass_lookup (allows subclasses to override hooks)
- Added create_hook functionality
- **Added hook decorators:**
  - init_hook
  - coerce_hook
  - create_hook
- Removed 3.3 as a supported version

0.0.4 (2016-07-11)

- Added init_hooks to base.Base
- Refactored sequence-based options to be defined in Base._seq_opts
- **Added Type extensions:**
  - Hashable
  - Tuple

- Added conf.vars
- Added coerce_hooks to base.Base

0.0.3 (2016-04-21)

- Added syn.conf module
- Added syn.five module
- Added coerce() classmethod to base.Base
- Added Mapping Type extension

0.0.2 (2016-04-21)

- Fixed type.MultiType typemap references for subclasses
- **Added Type extensions:**
  - Callable
  - Sequence
- Added attribute groups to base.Base
- **Added base.Base class options:**
  - id_equality
  - init_order
- **Added base.Attr attributes:**
  - group
  - groups
  - call
  - init
  - internal
- Added group-based excludes and includes to base.Base.to_dict()

0.0.1 (2016-04-17)

Initial release.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z