
syn Documentation

Release 0.0.15

Matt Bodenhamer

Apr 22, 2017

Contents

1	syn package	3
1.1	Subpackages	3
1.2	Module contents	246
2	Changelog	247
2.1	0.0.15 (2017-04-28)	247
2.2	0.0.14 (2017-03-05)	247
2.3	0.0.13 (2017-02-14)	247
2.4	0.0.12 (2017-02-12)	248
2.5	0.0.11 (2016-08-16)	248
2.6	0.0.10 (2016-08-12)	248
2.7	0.0.9 (2016-08-09)	248
2.8	0.0.8 (2016-08-09)	248
2.9	0.0.7 (2016-07-20)	248
2.10	0.0.6 (2016-07-20)	249
2.11	0.0.5 (2016-07-12)	249
2.12	0.0.4 (2016-07-11)	249
2.13	0.0.3 (2016-04-21)	250
2.14	0.0.2 (2016-04-21)	250
2.15	0.0.1 (2016-04-17)	250
3	Indices and tables	251
	Python Module Index	253

`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:

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, **kwargs)  
    A context manager that does nothing.  
  
syn.base_utils.context.assign (*args, **kwargs)  
    Assigns B to A.attr, yields, and then assigns A.attr back to its original value.  
  
syn.base_utils.context.setitem (*args, **kwargs)  
  
syn.base_utils.context.chdir (*args, **kwargs)  
  
syn.base_utils.context.delete (*args, **kwargs)  
    For using then deleting objects.  
  
syn.base_utils.context.nested_context (*args, **kwargs)  
  
syn.base_utils.context.capture (*args, **kwargs)  
  
syn.base_utils.context.on_error (*args, **kwargs)
```

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, **kws*)

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 commands'}

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

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': 1

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*; $0 \leq i \leq 0x10ffff$.

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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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.De1` (***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 subtree rooted 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 subtree rooted 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 = '2.999999999'

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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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, or_else, **kwargs*)

Bases: `syn.python.b.blocks.Block`

Positional Arguments:

test: *Expression* *body*: *list (Expression | Statement)* *or_else*: *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 subtree rooted 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', 'or_else')

- 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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

astalias 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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

astalias 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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`

astalias 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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`

astalias 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 subtree rooted 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 subtree rooted 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

astalias 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 In*

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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted by this node.**_parent** [Optional]: *Node* Parent of this nodesubclasses: *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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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.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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 subtree rooted 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 something 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*, ***kws*)

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 subtree rooted 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 subtree rooted 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

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 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/invoke 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`

S

syn, 246
syn.base, 8
syn.base.a, 4
syn.base.a.base, 3
syn.base.a.meta, 3
syn.base.b, 8
syn.base.b.base, 4
syn.base.b.examine, 5
syn.base.b.meta, 5
syn.base.b.utils, 6
syn.base.b.wrapper, 7
syn.base_utils, 15
syn.base_utils.alg, 8
syn.base_utils.context, 8
syn.base_utils.debug, 8
syn.base_utils.dict, 9
syn.base_utils.filters, 10
syn.base_utils.float, 10
syn.base_utils.hash, 10
syn.base_utils.iter, 10
syn.base_utils.list, 10
syn.base_utils.logic, 11
syn.base_utils.order, 12
syn.base_utils.py, 12
syn.base_utils.rand, 13
syn.base_utils.repl, 14
syn.base_utils.str, 14
syn.base_utils.tree, 15
syn.conf, 18
syn.conf.conf, 15
syn.conf.conf2, 15
syn.conf.vars, 17
syn.cython, 18
syn.five, 21
syn.five.num, 18
syn.five.string, 18
syn.globals, 21
syn.globals.loggers, 21
syn.globals.values, 21
syn.python, 145
syn.python.b, 145
syn.python.b.base, 21
syn.python.b.blocks, 43
syn.python.b.expressions, 54
syn.python.b.literals, 117
syn.python.b.statements, 130
syn.python.b.variables, 142
syn.schema, 155
syn.schema.b, 155
syn.schema.b.sequence, 145
syn.serialize, 155
syn.serialize.a, 155
syn.sets, 176
syn.sets.b, 176
syn.sets.b.base, 155
syn.sets.b.leaf, 157
syn.sets.b.operators, 165
syn.sets.b.range, 172
syn.tagmathon, 187
syn.tagmathon.b, 187
syn.tagmathon.b.base, 176
syn.tagmathon.b.builtin, 179
syn.tagmathon.b.compiler, 180
syn.tagmathon.b.function, 180
syn.tagmathon.b.interpreter, 186
syn.tree, 224
syn.tree.b, 224
syn.tree.b.node, 188
syn.tree.b.query, 189
syn.tree.b.tree, 223
syn.type, 228
syn.type.a, 228
syn.type.a.ext, 225
syn.type.a.type, 226
syn.types, 235
syn.types.a, 235
syn.types.a.base, 229
syn.types.a.mapping, 230

- syn.types.a.ne, 231
- syn.types.a.numeric, 233
- syn.types.a.sequence, 233
- syn.types.a.set, 234
- syn.types.a.special, 234
- syn.types.a.string, 234
- syn.util, 246
 - syn.util.constraint, 242
 - syn.util.constraint.b, 242
 - syn.util.constraint.b.base, 235
 - syn.util.constraint.b.constraints, 237
 - syn.util.constraint.b.solvers, 240
 - syn.util.log, 246
 - syn.util.log.b, 246
 - syn.util.log.b.base, 242
 - syn.util.log.b.events, 244

A

- A (syn.python.b.expressions.BinOp attribute), 68
- A (syn.schema.b.sequence.Repeat attribute), 152
- A (syn.sets.b.operators.Difference attribute), 170
- abstract() (syn.python.b.base.PythonTree method), 24
- Add (class in syn.python.b.expressions), 69
- add() (syn.util.log.b.base.Logger method), 244
- add_child() (syn.tree.b.node.Node method), 189
- add_node() (syn.tree.b.tree.Tree method), 224
- Alias (class in syn.python.b.statements), 133
- AllDifferentConstraint (class in syn.util.constraint.b.constraints), 238
- Ancestor (class in syn.tree.b.query), 189
- ancestors() (syn.tree.b.node.Node method), 189
- And (class in syn.python.b.expressions), 90
- and_() (in module syn.base_utils.logic), 11
- Any (class in syn.tree.b.query), 191
- AnyType (class in syn.type.a.type), 226
- append() (syn.base.b.wrapper.ListWrapper method), 7
- Arg (class in syn.python.b.blocks), 49
- Arguments (class in syn.python.b.blocks), 51
- arity (syn.tree.b.query.Comparison attribute), 196
- arity (syn.tree.b.query.Function attribute), 201
- arity (syn.tree.b.query.Identity attribute), 205
- as_return() (syn.python.b.base.PythonNode method), 23
- as_return() (syn.python.b.blocks.If method), 46
- as_return() (syn.python.b.statements.Pass method), 142
- as_value() (syn.python.b.base.PythonNode method), 23
- as_value() (syn.python.b.blocks.If method), 46
- as_value() (syn.python.b.statements.Assign method), 132
- assert_deepcopy_idempotent() (in module syn.base_utils.py), 12
- assert_equivalent() (in module syn.base_utils.py), 12
- assert_inequivalent() (in module syn.base_utils.py), 12
- assert_pickle_idempotent() (in module syn.base_utils.py), 12
- assert_type_equivalent() (in module syn.base_utils.py), 12
- Assign (class in syn.python.b.statements), 130
- assign() (in module syn.base_utils.context), 8
- AssocDict (class in syn.base_utils.dict), 9
- ast (syn.python.b.base.Context attribute), 25
- ast (syn.python.b.base.Del attribute), 30
- ast (syn.python.b.base.Expression attribute), 42
- ast (syn.python.b.base.Expression_ attribute), 36
- ast (syn.python.b.base.Interactive attribute), 37
- ast (syn.python.b.base.Load attribute), 27
- ast (syn.python.b.base.Module attribute), 34
- ast (syn.python.b.base.Param attribute), 31
- ast (syn.python.b.base.ProgN attribute), 40
- ast (syn.python.b.base.PythonNode attribute), 23
- ast (syn.python.b.base.RootNode attribute), 33
- ast (syn.python.b.base.Special attribute), 39
- ast (syn.python.b.base.Statement attribute), 43
- ast (syn.python.b.base.Store attribute), 28
- ast (syn.python.b.blocks.Arg attribute), 51
- ast (syn.python.b.blocks.Arguments attribute), 52
- ast (syn.python.b.blocks.Block attribute), 45
- ast (syn.python.b.blocks.For attribute), 48
- ast (syn.python.b.blocks.FunctionDef attribute), 54
- ast (syn.python.b.blocks.If attribute), 46
- ast (syn.python.b.blocks.While attribute), 49
- ast (syn.python.b.expressions.Add attribute), 70
- ast (syn.python.b.expressions.And attribute), 92
- ast (syn.python.b.expressions.Attribute attribute), 116
- ast (syn.python.b.expressions.BinaryOperator attribute), 67
- ast (syn.python.b.expressions.BinOp attribute), 69
- ast (syn.python.b.expressions.BitAnd attribute), 86
- ast (syn.python.b.expressions.BitOr attribute), 83
- ast (syn.python.b.expressions.BitXor attribute), 84
- ast (syn.python.b.expressions.BooleanOperator attribute), 89
- ast (syn.python.b.expressions.BoolOp attribute), 90
- ast (syn.python.b.expressions.Call attribute), 113
- ast (syn.python.b.expressions.Comparator attribute), 94
- ast (syn.python.b.expressions.Compare attribute), 96
- ast (syn.python.b.expressions.Div attribute), 74
- ast (syn.python.b.expressions.Eq attribute), 97

ast (syn.python.b.expressions.Expr attribute), 55
 ast (syn.python.b.expressions.FloorDiv attribute), 76
 ast (syn.python.b.expressions.Gt attribute), 103
 ast (syn.python.b.expressions.GtE attribute), 105
 ast (syn.python.b.expressions.IfExp attribute), 115
 ast (syn.python.b.expressions.In attribute), 109
 ast (syn.python.b.expressions.Invert attribute), 65
 ast (syn.python.b.expressions.Is attribute), 106
 ast (syn.python.b.expressions.IsNot attribute), 107
 ast (syn.python.b.expressions.Keyword attribute), 112
 ast (syn.python.b.expressions.LShift attribute), 80
 ast (syn.python.b.expressions.Lt attribute), 100
 ast (syn.python.b.expressions.LtE attribute), 102
 ast (syn.python.b.expressions.MatMult attribute), 87
 ast (syn.python.b.expressions.Mod attribute), 77
 ast (syn.python.b.expressions.Mult attribute), 73
 ast (syn.python.b.expressions.Not attribute), 64
 ast (syn.python.b.expressions.NotEq attribute), 99
 ast (syn.python.b.expressions.NotIn attribute), 110
 ast (syn.python.b.expressions.Operator attribute), 57
 ast (syn.python.b.expressions.Or attribute), 93
 ast (syn.python.b.expressions.Pow attribute), 79
 ast (syn.python.b.expressions.RShift attribute), 81
 ast (syn.python.b.expressions.Sub attribute), 71
 ast (syn.python.b.expressions.UAdd attribute), 61
 ast (syn.python.b.expressions.UnaryOp attribute), 60
 ast (syn.python.b.expressions.UnaryOperator attribute), 58
 ast (syn.python.b.expressions.USub attribute), 63
 ast (syn.python.b.literals.Bytes attribute), 122
 ast (syn.python.b.literals.List attribute), 125
 ast (syn.python.b.literals.Literal attribute), 118
 ast (syn.python.b.literals.NameConstant attribute), 130
 ast (syn.python.b.literals.Num attribute), 119
 ast (syn.python.b.literals.Sequence attribute), 124
 ast (syn.python.b.literals.Set attribute), 129
 ast (syn.python.b.literals.Str attribute), 121
 ast (syn.python.b.literals.Tuple attribute), 127
 ast (syn.python.b.statements.Alias attribute), 135
 ast (syn.python.b.statements.Assign attribute), 132
 ast (syn.python.b.statements.Break attribute), 139
 ast (syn.python.b.statements.Continue attribute), 140
 ast (syn.python.b.statements.EmptyStatement attribute), 138
 ast (syn.python.b.statements.Import attribute), 136
 ast (syn.python.b.statements.Pass attribute), 142
 ast (syn.python.b.statements.Return attribute), 133
 ast (syn.python.b.variables.Name attribute), 143
 ast (syn.python.b.variables.Starred attribute), 145
 AstUnsupported, 24
 Attr (class in syn.base.a.meta), 3
 Attr (class in syn.base.b.meta), 5
 AttrDict (class in syn.base_utils.dict), 9
 Attribute (class in syn.python.b.expressions), 115

Attribute (class in syn.tree.b.query), 192
 attributes() (syn.tree.b.node.Node method), 189
 Attrs (class in syn.base.a.meta), 3
 Attrs (class in syn.base.b.meta), 5
 attrs() (in module syn.types.a.base), 230
 attrs() (syn.base.b.base.BaseType method), 5
 attrs() (syn.types.a.base.Type method), 229
 attrs() (syn.types.a.base.TypeType method), 229
 Axis (class in syn.tree.b.query), 193

B

B (syn.python.b.expressions.BinOp attribute), 68
 B (syn.sets.b.operators.Difference attribute), 170
 Base (class in syn.base.a.base), 3
 Base (class in syn.base.b.base), 4
 Basestring (class in syn.types.a.string), 234
 BaseType (class in syn.base.b.base), 5
 BinaryOperator (class in syn.python.b.expressions), 66
 BinOp (class in syn.python.b.expressions), 67
 BitAnd (class in syn.python.b.expressions), 84
 BitOr (class in syn.python.b.expressions), 82
 BitXor (class in syn.python.b.expressions), 83
 Block (class in syn.python.b.blocks), 43
 body (syn.python.b.base.Expression_ attribute), 36
 Bool (class in syn.types.a.numeric), 233
 BooleanOperator (class in syn.python.b.expressions), 87
 BoolOp (class in syn.python.b.expressions), 89
 bounds (syn.python.b.literals.Sequence attribute), 124
 bounds (syn.python.b.literals.Set attribute), 129
 bounds (syn.python.b.literals.Tuple attribute), 127
 Break (class in syn.python.b.statements), 138
 break_around_line_breaks() (in module syn.base_utils.str), 14
 break_quoted_string() (in module syn.base_utils.str), 14
 BuiltinFunction (class in syn.tagmathon.b.builtin), 179
 Bytes (class in syn.python.b.literals), 121
 Bytes (class in syn.types.a.string), 234

C

c_call() (syn.base_utils.debug.Trace method), 8
 c_exception() (syn.base_utils.debug.Trace method), 8
 c_return() (syn.base_utils.debug.Trace method), 8
 Call (class in syn.python.b.expressions), 112
 Call (class in syn.tagmathon.b.function), 182
 call() (syn.base_utils.debug.CallTrace method), 9
 call() (syn.base_utils.debug.Trace method), 8
 call() (syn.tagmathon.b.builtin.BuiltinFunction method), 180
 call() (syn.tagmathon.b.function.Function method), 182
 call_coerce (syn.type.a.type.TypeType attribute), 227
 call_trace() (in module syn.base_utils.debug), 9
 call_validate (syn.type.a.type.TypeType attribute), 227
 Callable (class in syn.type.a.ext), 225
 callables() (in module syn.base_utils.py), 12

- CallTrace (class in syn.base_utils.debug), 9
- capitalize() (syn.five.string.unicode method), 18
- capture() (in module syn.base_utils.context), 8
- center() (syn.five.string.unicode method), 18
- cfeq() (in module syn.base_utils.float), 10
- chdir() (in module syn.base_utils.context), 8
- check() (syn.type.a.ext.Callable method), 225
- check() (syn.type.a.ext.Hashable method), 226
- check() (syn.type.a.ext.Mapping method), 225
- check() (syn.type.a.ext.Sequence method), 225
- check() (syn.type.a.ext.Tuple method), 225
- check() (syn.type.a.type.AnyType method), 226
- check() (syn.type.a.type.MultiType method), 227
- check() (syn.type.a.type.Schema method), 228
- check() (syn.type.a.type.Set method), 228
- check() (syn.type.a.type.Type method), 226
- check() (syn.type.a.type.TypeType method), 227
- check() (syn.type.a.type.ValuesType method), 227
- check() (syn.util.constraint.b.base.Constraint method), 236
- check() (syn.util.constraint.b.base.Problem method), 237
- check() (syn.util.constraint.b.constraints.AllDifferentConstraint method), 239
- check() (syn.util.constraint.b.constraints.EqualConstraint method), 239
- check() (syn.util.constraint.b.constraints.FunctionConstraint method), 238
- check_idempotence() (in module syn.base.b.examine), 5
- Child (class in syn.tree.b.query), 194
- children() (syn.tree.b.node.Node method), 189
- choose_var() (syn.util.constraint.b.solvers.RecursiveBacktrack method), 242
- chrs() (in module syn.base_utils.str), 14
- ClassWrapper (class in syn.sets.b.leaf), 161
- coerce() (syn.base.b.base.Base class method), 4
- coerce() (syn.conf.vars.Vars class method), 18
- coerce() (syn.type.a.ext.Mapping method), 225
- coerce() (syn.type.a.ext.Sequence method), 225
- coerce() (syn.type.a.ext.Tuple method), 225
- coerce() (syn.type.a.type.AnyType method), 226
- coerce() (syn.type.a.type.MultiType method), 227
- coerce() (syn.type.a.type.Schema method), 228
- coerce() (syn.type.a.type.Set method), 228
- coerce() (syn.type.a.type.Type method), 226
- coerce() (syn.type.a.type.TypeType method), 227
- coerce() (syn.type.a.type.ValuesType method), 227
- coerce_hook() (in module syn.base.b.base), 5
- collect() (in module syn.types.a.base), 230
- collect() (syn.types.a.base.Type method), 229
- collect_by_type() (syn.tree.b.node.Node method), 189
- collect_nodes() (syn.tree.b.node.Node method), 189
- collect_rootward() (syn.tree.b.node.Node method), 189
- collection_comp() (in module syn.base_utils.logic), 12
- collection_equivalent() (in module syn.base_utils.logic), 11
- command_display_current_value() (syn.types.a.ne.DiffExplorer method), 231
- command_display_current_value() (syn.types.a.ne.ValueExplorer method), 231
- command_display_value() (syn.types.a.ne.DiffExplorer method), 231
- command_display_value() (syn.types.a.ne.ValueExplorer method), 231
- command_down() (syn.types.a.ne.DiffExplorer method), 231
- command_down() (syn.types.a.ne.ValueExplorer method), 231
- command_find() (syn.types.a.ne.DiffExplorer method), 231
- command_help (syn.base_utils.repl.REPL attribute), 14
- command_help (syn.types.a.ne.DiffExplorer attribute), 231
- command_help (syn.types.a.ne.ValueExplorer attribute), 231
- command_step() (syn.types.a.ne.DiffExplorer method), 231
- command_step() (syn.types.a.ne.ValueExplorer method), 231
- command_up() (syn.types.a.ne.DiffExplorer method), 231
- command_up() (syn.types.a.ne.ValueExplorer method), 231
- commands (syn.base_utils.repl.REPL attribute), 14
- commands (syn.types.a.ne.DiffExplorer attribute), 231
- commands (syn.types.a.ne.ValueExplorer attribute), 231
- Comparator (class in syn.python.b.expressions), 93
- Compare (class in syn.python.b.expressions), 95
- Comparison (class in syn.tree.b.query), 195
- compile_to_python() (in module syn.tagmation.b.compiler), 180
- complement() (syn.base_utils.dict.GroupDict method), 9
- complement() (syn.sets.b.base.SetNode method), 156
- complement() (syn.sets.b.leaf.SetWrapper method), 159
- complement() (syn.sets.b.range.Range method), 173
- Complex (class in syn.types.a.numeric), 233
- compose() (in module syn.base_utils.py), 12
- cond (syn.tree.b.query.Where attribute), 223
- Conf (class in syn.conf.conf2), 16
- ConfDict (class in syn.conf.conf2), 15
- ConfList (class in syn.conf.conf2), 16
- Constraint (class in syn.util.constraint.b.base), 236
- consume() (in module syn.base_utils.iter), 10
- consume() (syn.base_utils.list.IterableList method), 10
- Context (class in syn.python.b.base), 24
- Continue (class in syn.python.b.statements), 139
- copy() (syn.base.b.base.Base method), 4
- copy() (syn.base_utils.list.IterableList method), 11

copy() (syn.util.constraint.b.base.Domain method), 236
count() (syn.base.b.wrapper.ListWrapper method), 7
count() (syn.five.string.unicode method), 18
Counter (class in syn.base.b.utils), 6
create_hook() (in module syn.base.b.meta), 5
current_frame (syn.tagmathon.b.interpreter.Env attribute), 187
current_value (syn.types.a.ne.DiffExplorer attribute), 231

D

Data (class in syn.base.b.meta), 5
decode() (syn.five.string.unicode method), 18
deep_comp() (in module syn.types.a.ne), 231
deep_freq() (in module syn.types.a.ne), 232
DefaultList (class in syn.base_utils.list), 11
defer_reduce() (in module syn.base_utils.alg), 8
Del (class in syn.python.b.base), 28
delete() (in module syn.base_utils.context), 8
delim (syn.python.b.literals.Sequence attribute), 124
depth_first() (syn.tree.b.node.Node method), 189
depth_first() (syn.tree.b.tree.Tree method), 224
depth_first() (syn.types.a.ne.DiffExplorer method), 231
depth_first() (syn.types.a.ne.ValueExplorer method), 231
Descendant (class in syn.tree.b.query), 196
descendants() (syn.tree.b.node.Node method), 189
deserialize() (in module syn.types.a.base), 230
deserialize() (syn.types.a.base.Type class method), 229
deserialize() (syn.types.a.mapping.Mapping class method), 230
deserialize() (syn.types.a.sequence.Sequence class method), 233
deserialize() (syn.types.a.special.NONE class method), 234
deserialize_dispatch() (syn.types.a.base.Type class method), 229
Dict (class in syn.types.a.mapping), 230
dictify_strings() (in module syn.base_utils.filters), 10
Difference (class in syn.sets.b.operators), 169
difference() (syn.sets.b.base.SetNode method), 156
difference() (syn.sets.b.leaf.SetWrapper method), 159
difference() (syn.sets.b.range.Range method), 173
DifferentLength (class in syn.types.a.ne), 232
DifferentTypes (class in syn.types.a.ne), 232
DiffersAtAttribute (class in syn.types.a.ne), 232
DiffersAtIndex (class in syn.types.a.ne), 232
DiffersAtKey (class in syn.types.a.ne), 232
DiffExplorer (class in syn.types.a.ne), 231
dispatch() (syn.type.a.type.Type class method), 226
dispatch() (syn.types.a.base.Type class method), 229
displacement() (syn.base_utils.list.IterableList method), 11
display() (syn.sets.b.base.SetNode method), 156
display() (syn.sets.b.leaf.ClassWrapper method), 162
display() (syn.sets.b.leaf.Empty method), 165

display() (syn.sets.b.leaf.SetWrapper method), 159
display() (syn.sets.b.leaf.TypeWrapper method), 161
display() (syn.sets.b.operators.SetOperator method), 166
display() (syn.sets.b.range.Range method), 173
display() (syn.sets.b.range.StrRange method), 176
display() (syn.type.a.ext.Callable method), 225
display() (syn.type.a.ext.Hashable method), 226
display() (syn.type.a.ext.Mapping method), 225
display() (syn.type.a.ext.Sequence method), 225
display() (syn.type.a.ext.Tuple method), 225
display() (syn.type.a.type.AnyType method), 226
display() (syn.type.a.type.MultiType method), 227
display() (syn.type.a.type.Schema method), 228
display() (syn.type.a.type.Set method), 228
display() (syn.type.a.type.Type method), 226
display() (syn.type.a.type.TypeType method), 227
display() (syn.type.a.type.ValuesType method), 227
display() (syn.types.a.ne.DiffExplorer method), 231
display() (syn.types.a.ne.ValueExplorer method), 231
display() (syn.util.constraint.b.base.Constraint method), 236
display() (syn.util.constraint.b.base.Domain method), 236
display() (syn.util.constraint.b.base.Problem method), 237
display() (syn.util.constraint.b.constraints.AllDifferentConstraint method), 239
display() (syn.util.constraint.b.constraints.EqualConstraint method), 239
display() (syn.util.constraint.b.constraints.FunctionConstraint method), 238
Div (class in syn.python.b.expressions), 73
do_nothing() (in module syn.tree.b.tree), 224
Domain (class in syn.util.constraint.b.base), 235
down() (syn.types.a.ne.DiffExplorer method), 231
down() (syn.types.a.ne.ValueExplorer method), 231

E

elems (syn.schema.b.sequence.SchemaNode attribute), 147
elog() (in module syn.base_utils.py), 13
emit() (syn.python.b.base.Expression_ method), 36
emit() (syn.python.b.base.PythonNode method), 23
emit() (syn.python.b.base.PythonTree method), 24
emit() (syn.python.b.base.RootNode method), 33
emit() (syn.python.b.blocks.Arg method), 51
emit() (syn.python.b.blocks.Arguments method), 52
emit() (syn.python.b.blocks.For method), 48
emit() (syn.python.b.blocks.FunctionDef method), 54
emit() (syn.python.b.blocks.If method), 46
emit() (syn.python.b.blocks.While method), 49
emit() (syn.python.b.expressions.Attribute method), 116
emit() (syn.python.b.expressions.BinOp method), 69
emit() (syn.python.b.expressions.BoolOp method), 90
emit() (syn.python.b.expressions.Call method), 113

- emit() (syn.python.b.expressions.Compare method), 96
- emit() (syn.python.b.expressions.Expr method), 55
- emit() (syn.python.b.expressions.IfExp method), 115
- emit() (syn.python.b.expressions.Keyword method), 112
- emit() (syn.python.b.expressions.Operator method), 57
- emit() (syn.python.b.expressions.UnaryOp method), 60
- emit() (syn.python.b.literals.Bytes method), 122
- emit() (syn.python.b.literals.NameConstant method), 130
- emit() (syn.python.b.literals.Num method), 119
- emit() (syn.python.b.literals.Sequence method), 124
- emit() (syn.python.b.literals.Str method), 121
- emit() (syn.python.b.statements.Alias method), 135
- emit() (syn.python.b.statements.Assign method), 132
- emit() (syn.python.b.statements.EmptyStatement method), 138
- emit() (syn.python.b.statements.Import method), 136
- emit() (syn.python.b.statements.Return method), 133
- emit() (syn.python.b.variables.Name method), 143
- emit() (syn.python.b.variables.Starred method), 145
- emit2() (syn.python.b.blocks.Arguments method), 52
- emit3() (syn.python.b.blocks.Arguments method), 52
- emit_block() (syn.python.b.blocks.Block method), 45
- emit_decorators() (syn.python.b.blocks.FunctionDef method), 54
- Empty (class in syn.sets.b.leaf), 164
- empty() (syn.base_utils.list.IterableList method), 11
- EmptyStatement (class in syn.python.b.statements), 136
- encode() (syn.five.string.unicode method), 18
- endswith() (syn.five.string.unicode method), 19
- enumerate() (in module syn.types.a.base), 230
- enumerate() (syn.schema.b.sequence.Sequence method), 153
- enumerate() (syn.sets.b.base.SetNode method), 156
- enumerate() (syn.sets.b.leaf.ClassWrapper method), 162
- enumerate() (syn.sets.b.leaf.Empty method), 165
- enumerate() (syn.sets.b.leaf.SetWrapper method), 159
- enumerate() (syn.sets.b.leaf.TypeWrapper method), 161
- enumerate() (syn.sets.b.operators.Difference method), 170
- enumerate() (syn.sets.b.operators.Intersection method), 169
- enumerate() (syn.sets.b.operators.Product method), 172
- enumerate() (syn.sets.b.operators.SetOperator method), 166
- enumerate() (syn.sets.b.operators.Union method), 168
- enumerate() (syn.sets.b.range.Range method), 173
- enumerate() (syn.sets.b.range.StrRange method), 176
- enumerate() (syn.types.a.base.Type class method), 229
- enumeration_value() (in module syn.types.a.base), 230
- enumeration_value() (syn.type.a.type.AnyType method), 226
- enumeration_value() (syn.type.a.type.MultiType method), 227
- enumeration_value() (syn.type.a.type.Type method), 226
- enumeration_value() (syn.type.a.type.TypeType method), 227
- enumeration_value() (syn.type.a.type.ValuesType method), 227
- enumeration_value() (syn.types.a.base.Type class method), 229
- Env (class in syn.tagmathon.b.interpreter), 186
- eprint() (in module syn.base_utils.py), 13
- Eq (class in syn.python.b.expressions), 96
- Eq (class in syn.tree.b.query), 198
- EqualConstraint (class in syn.util.constraint.b.constraints), 239
- equiv() (in module syn.base_utils.logic), 11
- escape_for_eval() (in module syn.base_utils.str), 14
- escape_line_breaks() (in module syn.base_utils.str), 14
- escape_null() (in module syn.base_utils.str), 14
- estr() (in module syn.types.a.base), 230
- estr() (syn.types.a.base.Type method), 229
- estr() (syn.types.a.mapping.Mapping method), 230
- estr() (syn.types.a.numeric.Long method), 233
- estr() (syn.types.a.numeric.Numeric method), 233
- estr() (syn.types.a.sequence.Sequence method), 233
- estr() (syn.types.a.set.Set method), 234
- estr() (syn.types.a.special.NONE method), 234
- estr() (syn.types.a.string.Bytes method), 234
- estr() (syn.types.a.string.Unicode method), 234
- eval() (in module syn.tagmathon.b.interpreter), 187
- eval() (syn.base_utils.repl.REPL method), 14
- eval() (syn.tagmathon.b.base.SyntagmathonNode method), 178
- eval() (syn.tagmathon.b.base.Variable method), 179
- eval() (syn.tagmathon.b.function.Call method), 183
- eval() (syn.tagmathon.b.function.Function method), 182
- eval() (syn.tagmathon.b.function.SpecialCall method), 185
- eval() (syn.tree.b.query.Any method), 192
- eval() (syn.tree.b.query.Function method), 201
- eval() (syn.tree.b.query.Name method), 208
- eval() (syn.tree.b.query.Position method), 212
- eval() (syn.tree.b.query.Predicate method), 214
- Event (class in syn.util.log.b.base), 242
- exception() (syn.base_utils.debug.Trace method), 8
- expandtabs() (syn.five.string.unicode method), 19
- expected_size() (syn.sets.b.base.SetNode method), 157
- ExplorationError, 231
- explorer() (syn.types.a.ne.DiffersAtAttribute method), 232
- explorer() (syn.types.a.ne.DiffersAtIndex method), 232
- explorer() (syn.types.a.ne.DiffersAtKey method), 232
- explorer() (syn.types.a.ne.NEType method), 232
- Expr (class in syn.python.b.expressions), 54
- expressify_statements() (syn.python.b.base.ProgN method), 40

- expressify_statements() (syn.python.b.base.PythonNode method), 23
- expressify_statements() (syn.python.b.base.RootNode method), 33
- Expression (class in syn.python.b.base), 40
- Expression_ (class in syn.python.b.base), 34
- extend() (syn.base.b.wrapper.ListWrapper method), 7
- ## F
- failure() (syn.schema.b.sequence.MatchFailed method), 155
- feq() (in module syn.base_utils.float), 10
- feq_comp() (in module syn.types.a.ne), 232
- find() (syn.five.string.unicode method), 19
- find_ne() (in module syn.types.a.base), 230
- find_ne() (syn.types.a.base.Type method), 229
- find_one() (syn.tree.b.tree.Tree method), 224
- find_type() (syn.tree.b.node.Node method), 189
- first() (in module syn.base_utils.iter), 10
- flattened() (in module syn.base_utils.list), 11
- Float (class in syn.types.a.numeric), 233
- FloorDiv (class in syn.python.b.expressions), 74
- Following (class in syn.tree.b.query), 199
- following() (syn.tree.b.node.Node method), 189
- For (class in syn.python.b.blocks), 46
- format() (syn.five.string.unicode method), 19
- forward_check() (syn.util.constraint.b.solvers.RecursiveBacktrackSolver method), 242
- Frame (class in syn.tagmathon.b.interpreter), 186
- from_ast() (in module syn.python.b.base), 43
- from_ast() (syn.python.b.base.Context class method), 26
- from_ast() (syn.python.b.base.Expression_ class method), 36
- from_ast() (syn.python.b.base.PythonNode class method), 23
- from_ast() (syn.python.b.base.RootNode class method), 33
- from_file() (syn.conf.conf.YAMLMixin class method), 15
- from_mapping() (syn.base.b.base.Base class method), 4
- from_object() (syn.base.b.base.Base class method), 4
- from_sequence() (syn.base.b.base.Base class method), 4
- from_source() (in module syn.python.b.base), 43
- Frozenset (class in syn.types.a.set), 234
- full_funcname() (in module syn.base_utils.py), 13
- func (syn.tree.b.query.Function attribute), 201
- func() (syn.tree.b.query.Eq method), 199
- func() (syn.tree.b.query.Ge method), 202
- func() (syn.tree.b.query.Gt method), 203
- func() (syn.tree.b.query.Identity method), 205
- func() (syn.tree.b.query.Le method), 206
- func() (syn.tree.b.query.Lt method), 207
- func() (syn.tree.b.query.Ne method), 209
- Function (class in syn.tagmathon.b.function), 180
- Function (class in syn.tree.b.query), 200
- FunctionConstraint (class in syn.util.constraint.b.constraints), 237
- FunctionDef (class in syn.python.b.blocks), 52
- fuzzy_and() (in module syn.base_utils.logic), 11
- fuzzy_equiv() (in module syn.base_utils.logic), 11
- fuzzy_implies() (in module syn.base_utils.logic), 11
- fuzzy_nand() (in module syn.base_utils.logic), 11
- fuzzy_nor() (in module syn.base_utils.logic), 11
- fuzzy_not() (in module syn.base_utils.logic), 11
- fuzzy_or() (in module syn.base_utils.logic), 11
- fuzzy_xor() (in module syn.base_utils.logic), 11
- ## G
- Ge (class in syn.tree.b.query), 201
- gen_type (syn.types.a.base.Type attribute), 229
- gen_types (syn.types.a.base.Type attribute), 229
- generate() (in module syn.types.a.base), 230
- generate() (syn.type.a.ext.Callable method), 225
- generate() (syn.type.a.ext.Hashable method), 226
- generate() (syn.type.a.ext.Mapping method), 225
- generate() (syn.type.a.ext.Sequence method), 225
- generate() (syn.type.a.ext.Tuple method), 225
- generate() (syn.type.a.type.AnyType method), 226
- generate() (syn.type.a.type.MultiType method), 227
- generate() (syn.type.a.type.Schema method), 228
- generate() (syn.type.a.type.Set method), 228
- generate() (syn.type.a.type.Type method), 226
- generate() (syn.type.a.type.TypeType method), 227
- generate() (syn.type.a.type.ValuesType method), 227
- generate() (syn.types.a.base.Type class method), 229
- generate_set() (syn.schema.b.sequence.Or method), 150
- generate_set() (syn.schema.b.sequence.Repeat method), 152
- generate_set() (syn.schema.b.sequence.Sequence method), 153
- gensym() (syn.tagmathon.b.interpreter.Env method), 187
- gensym() (syn.tagmathon.b.interpreter.Frame method), 186
- get_fullname() (in module syn.base_utils.py), 13
- get_mod() (in module syn.base_utils.py), 12
- get_name() (syn.tagmathon.b.function.Function method), 182
- get_node_by_id() (syn.tree.b.tree.Tree method), 224
- get_one() (syn.schema.b.sequence.Sequence method), 153
- get_one() (syn.sets.b.base.SetNode method), 157
- get_one() (syn.sets.b.operators.SetOperator method), 166
- get_type_name() (in module syn.base_utils.py), 12
- getfunc() (in module syn.base_utils.py), 13
- getitem() (in module syn.base_utils.py), 12
- getkey() (in module syn.base_utils.py), 13
- globals() (syn.tagmathon.b.interpreter.Env method), 187
- GroupDict (class in syn.base_utils.dict), 9
- groups_enum() (syn.base.b.meta.Meta method), 5

Gt (class in `syn.python.b.expressions`), 102
 Gt (class in `syn.tree.b.query`), 202
 GtE (class in `syn.python.b.expressions`), 103

H

hangwatch() (in module `syn.base_utils.py`), 13
 harvest_metadata() (in module `syn.base_utils.py`), 13
 Harvester (class in `syn.base.b.base`), 5
 Hashable (class in `syn.type.a.ext`), 226
 hashable() (in module `syn.types.a.base`), 230
 hashable() (`syn.types.a.base.Type` method), 229
 hasmember() (`syn.sets.b.base.SetNode` method), 157
 hasmember() (`syn.sets.b.leaf.ClassWrapper` method), 162
 hasmember() (`syn.sets.b.leaf.Empty` method), 165
 hasmember() (`syn.sets.b.leaf.SetWrapper` method), 160
 hasmember() (`syn.sets.b.leaf.TypeWrapper` method), 161
 hasmember() (`syn.sets.b.operators.Difference` method), 170
 hasmember() (`syn.sets.b.operators.Intersection` method), 169
 hasmember() (`syn.sets.b.operators.Product` method), 172
 hasmember() (`syn.sets.b.operators.Union` method), 168
 hasmember() (`syn.sets.b.range.IntRange` method), 175
 hasmember() (`syn.sets.b.range.Range` method), 173
 hasmember() (`syn.sets.b.range.StrRange` method), 176
 hasmethod() (in module `syn.base_utils.py`), 12

I

id() (`syn.tree.b.node.Node` method), 189
 Identity (class in `syn.tree.b.query`), 204
 identity() (in module `syn.tree.b.tree`), 224
 If (class in `syn.python.b.blocks`), 45
 IfExp (class in `syn.python.b.expressions`), 113
 implies() (in module `syn.base_utils.logic`), 11
 Import (class in `syn.python.b.statements`), 135
 import_module() (in module `syn.base_utils.py`), 12
 In (class in `syn.python.b.expressions`), 108
 index() (in module `syn.base_utils.py`), 12
 index() (`syn.base.b.wrapper.ListWrapper` method), 7
 index() (`syn.five.string.unicode` method), 19
 indexed_values (`syn.type.a.type.ValuesType` attribute), 227
 indices_removed() (in module `syn.base_utils.list`), 11
 init_hook() (in module `syn.base.b.base`), 5
 insert() (`syn.base.b.wrapper.ListWrapper` method), 7
 insert() (`syn.base_utils.list.ListView` method), 10
 Int (class in `syn.types.a.numeric`), 233
 Interactive (class in `syn.python.b.base`), 36
 Intersection (class in `syn.sets.b.operators`), 168
 intersection() (`syn.base_utils.dict.GroupDict` method), 9
 intersection() (`syn.sets.b.base.SetNode` method), 157
 intersection() (`syn.sets.b.leaf.SetWrapper` method), 160
 intersection() (`syn.sets.b.range.Range` method), 173
 IntRange (class in `syn.sets.b.range`), 174

Invert (class in `syn.python.b.expressions`), 64
 Is (class in `syn.python.b.expressions`), 105
 is_empty() (in module `syn.base_utils.iter`), 10
 is_flat() (in module `syn.base_utils.list`), 11
 is_hashable() (in module `syn.base_utils.hash`), 10
 is_proper_sequence() (in module `syn.base_utils.list`), 11
 is_subclass() (in module `syn.base_utils.py`), 12
 is_typelist (`syn.type.a.type.MultiType` attribute), 227
 is_unique() (in module `syn.base_utils.list`), 11
 is_visit_primitive() (in module `syn.types.a.ne`), 232
 isalnum() (`syn.five.string.unicode` method), 19
 isalpha() (`syn.five.string.unicode` method), 19
 isdecimal() (`syn.five.string.unicode` method), 19
 isdigit() (`syn.five.string.unicode` method), 19
 islower() (`syn.five.string.unicode` method), 19
 IsNot (class in `syn.python.b.expressions`), 106
 isnumeric() (`syn.five.string.unicode` method), 19
 isspace() (`syn.five.string.unicode` method), 19
 issubset() (`syn.sets.b.base.SetNode` method), 157
 issubset() (`syn.sets.b.leaf.Empty` method), 165
 issubset() (`syn.sets.b.leaf.SetWrapper` method), 160
 issubset() (`syn.sets.b.range.Range` method), 173
 issuperset() (`syn.sets.b.base.SetNode` method), 157
 issuperset() (`syn.sets.b.leaf.Empty` method), 165
 issuperset() (`syn.sets.b.leaf.SetWrapper` method), 160
 issuperset() (`syn.sets.b.range.Range` method), 173
 istitle() (`syn.five.string.unicode` method), 19
 istr() (in module `syn.base_utils.str`), 15
 istr() (`syn.base.b.base.Base` method), 4
 isupper() (`syn.five.string.unicode` method), 19
 item_type (`syn.type.a.ext.Sequence` attribute), 225
 items() (`syn.tagmation.b.interpreter.Env` method), 187
 items() (`syn.tagmation.b.interpreter.Frame` method), 186
 IterableList (class in `syn.base_utils.list`), 10
 iterate() (`syn.tree.b.query.Ancestor` method), 191
 iterate() (`syn.tree.b.query.Attribute` method), 193
 iterate() (`syn.tree.b.query.Child` method), 195
 iterate() (`syn.tree.b.query.Descendant` method), 198
 iterate() (`syn.tree.b.query.Following` method), 200
 iterate() (`syn.tree.b.query.Parent` method), 211
 iterate() (`syn.tree.b.query.Preceding` method), 213
 iterate() (`syn.tree.b.query.Query` method), 215
 iterate() (`syn.tree.b.query.Root` method), 217
 iterate() (`syn.tree.b.query.Self` method), 218
 iterate() (`syn.tree.b.query.Sibling` method), 219
 iterate() (`syn.tree.b.query.Type` method), 220
 iteration_length() (in module `syn.base_utils.iter`), 10
 iterlen() (in module `syn.base_utils.iter`), 10

J

join() (in module `syn.base_utils.filters`), 10
 join() (`syn.five.string.unicode` method), 19

K

KeyDifferences (class in syn.types.a.ne), 232
 Keyword (class in syn.python.b.expressions), 110

L

last() (in module syn.base_utils.iter), 10
 lazy_enumerate() (syn.sets.b.base.SetNode method), 157
 lazy_sample() (syn.sets.b.base.SetNode method), 157
 Le (class in syn.tree.b.query), 205
 length (syn.type.a.ext.Tuple attribute), 225
 line() (syn.base_utils.debug.Trace method), 8
 List (class in syn.python.b.literals), 124
 List (class in syn.types.a.sequence), 233
 ListView (class in syn.base_utils.list), 10
 ListWrapper (class in syn.base.b.wrapper), 7
 Literal (class in syn.python.b.literals), 117
 ljust() (syn.five.string.unicode method), 19
 Load (class in syn.python.b.base), 26
 locals() (syn.tagmation.b.interpreter.Env method), 187
 Logger (class in syn.util.log.b.base), 243
 Long (class in syn.types.a.numeric), 233
 lower() (syn.five.string.unicode method), 20
 LShift (class in syn.python.b.expressions), 79
 lstrip() (syn.five.string.unicode method), 20
 Lt (class in syn.python.b.expressions), 99
 Lt (class in syn.tree.b.query), 206
 LtE (class in syn.python.b.expressions), 100

M

map_type (syn.type.a.ext.Mapping attribute), 225
 Mapping (class in syn.type.a.ext), 225
 Mapping (class in syn.types.a.mapping), 230
 mark() (syn.base_utils.list.IterableList method), 11
 Match (class in syn.schema.b.sequence), 153
 match() (syn.schema.b.sequence.Or method), 150
 match() (syn.schema.b.sequence.Repeat method), 152
 match() (syn.schema.b.sequence.Sequence method), 153
 match() (syn.schema.b.sequence.Set method), 148
 MatchFailed, 155
 MatchFailure (class in syn.schema.b.sequence), 154
 MatMult (class in syn.python.b.expressions), 86
 maxver (syn.python.b.base.Context attribute), 26
 maxver (syn.python.b.base.Del attribute), 30
 maxver (syn.python.b.base.Expression attribute), 42
 maxver (syn.python.b.base.Expression_ attribute), 36
 maxver (syn.python.b.base.Interactive attribute), 37
 maxver (syn.python.b.base.Load attribute), 27
 maxver (syn.python.b.base.Module attribute), 34
 maxver (syn.python.b.base.Param attribute), 31
 maxver (syn.python.b.base.ProgN attribute), 40
 maxver (syn.python.b.base.PythonNode attribute), 23
 maxver (syn.python.b.base.RootNode attribute), 33
 maxver (syn.python.b.base.Special attribute), 39

maxver (syn.python.b.base.Statement attribute), 43
 maxver (syn.python.b.base.Store attribute), 28
 maxver (syn.python.b.blocks.Arg attribute), 51
 maxver (syn.python.b.blocks.Arguments attribute), 52
 maxver (syn.python.b.blocks.Block attribute), 45
 maxver (syn.python.b.blocks.For attribute), 48
 maxver (syn.python.b.blocks.FunctionDef attribute), 54
 maxver (syn.python.b.blocks.If attribute), 46
 maxver (syn.python.b.blocks.While attribute), 49
 maxver (syn.python.b.expressions.Add attribute), 70
 maxver (syn.python.b.expressions.And attribute), 92
 maxver (syn.python.b.expressions.Attribute attribute), 116
 maxver (syn.python.b.expressions.BinaryOperator attribute), 67
 maxver (syn.python.b.expressions.BinOp attribute), 69
 maxver (syn.python.b.expressions.BitAnd attribute), 86
 maxver (syn.python.b.expressions.BitOr attribute), 83
 maxver (syn.python.b.expressions.BitXor attribute), 84
 maxver (syn.python.b.expressions.BooleanOperator attribute), 89
 maxver (syn.python.b.expressions.BoolOp attribute), 90
 maxver (syn.python.b.expressions.Call attribute), 113
 maxver (syn.python.b.expressions.Comparator attribute), 94
 maxver (syn.python.b.expressions.Compare attribute), 96
 maxver (syn.python.b.expressions.Div attribute), 74
 maxver (syn.python.b.expressions.Eq attribute), 97
 maxver (syn.python.b.expressions.Expr attribute), 55
 maxver (syn.python.b.expressions.FloorDiv attribute), 76
 maxver (syn.python.b.expressions.Gt attribute), 103
 maxver (syn.python.b.expressions.GtE attribute), 105
 maxver (syn.python.b.expressions.IfExp attribute), 115
 maxver (syn.python.b.expressions.In attribute), 109
 maxver (syn.python.b.expressions.Invert attribute), 65
 maxver (syn.python.b.expressions.Is attribute), 106
 maxver (syn.python.b.expressions.IsNot attribute), 107
 maxver (syn.python.b.expressions.Keyword attribute), 112
 maxver (syn.python.b.expressions.LShift attribute), 80
 maxver (syn.python.b.expressions.Lt attribute), 100
 maxver (syn.python.b.expressions.LtE attribute), 102
 maxver (syn.python.b.expressions.MatMult attribute), 87
 maxver (syn.python.b.expressions.Mod attribute), 77
 maxver (syn.python.b.expressions.Mult attribute), 73
 maxver (syn.python.b.expressions.Not attribute), 64
 maxver (syn.python.b.expressions.NotEq attribute), 99
 maxver (syn.python.b.expressions.NotIn attribute), 110
 maxver (syn.python.b.expressions.Operator attribute), 57
 maxver (syn.python.b.expressions.Or attribute), 93
 maxver (syn.python.b.expressions.Pow attribute), 79
 maxver (syn.python.b.expressions.RShift attribute), 82
 maxver (syn.python.b.expressions.Sub attribute), 71
 maxver (syn.python.b.expressions.UAdd attribute), 61

- maxver (syn.python.b.expressions.UnaryOp attribute), 60
- maxver (syn.python.b.expressions.UnaryOperator attribute), 58
- maxver (syn.python.b.expressions.USub attribute), 63
- maxver (syn.python.b.literals.Bytes attribute), 122
- maxver (syn.python.b.literals.List attribute), 125
- maxver (syn.python.b.literals.Literal attribute), 118
- maxver (syn.python.b.literals.NameConstant attribute), 130
- maxver (syn.python.b.literals.Num attribute), 119
- maxver (syn.python.b.literals.Sequence attribute), 124
- maxver (syn.python.b.literals.Set attribute), 129
- maxver (syn.python.b.literals.Str attribute), 121
- maxver (syn.python.b.literals.Tuple attribute), 127
- maxver (syn.python.b.statements.Alias attribute), 135
- maxver (syn.python.b.statements.Assign attribute), 132
- maxver (syn.python.b.statements.Break attribute), 139
- maxver (syn.python.b.statements.Continue attribute), 140
- maxver (syn.python.b.statements.EmptyStatement attribute), 138
- maxver (syn.python.b.statements.Import attribute), 136
- maxver (syn.python.b.statements.Pass attribute), 142
- maxver (syn.python.b.statements.Return attribute), 133
- maxver (syn.python.b.variables.Name attribute), 143
- maxver (syn.python.b.variables.Starred attribute), 145
- message() (in module syn.base_utils.py), 12
- message() (syn.types.a.ne.DifferentLength method), 232
- message() (syn.types.a.ne.DifferentTypes method), 232
- message() (syn.types.a.ne.DiffersAtAttribute method), 232
- message() (syn.types.a.ne.DiffersAtIndex method), 232
- message() (syn.types.a.ne.DiffersAtKey method), 232
- message() (syn.types.a.ne.KeyDifferences method), 232
- message() (syn.types.a.ne.NEType method), 232
- message() (syn.types.a.ne.NotEqual method), 232
- message() (syn.types.a.ne.SetDifferences method), 232
- Meta (class in syn.base.a.meta), 3
- Meta (class in syn.base.b.meta), 5
- minver (syn.python.b.base.Context attribute), 26
- minver (syn.python.b.base.Del attribute), 30
- minver (syn.python.b.base.Expression attribute), 42
- minver (syn.python.b.base.Expression_ attribute), 36
- minver (syn.python.b.base.Interactive attribute), 37
- minver (syn.python.b.base.Load attribute), 27
- minver (syn.python.b.base.Module attribute), 34
- minver (syn.python.b.base.Param attribute), 31
- minver (syn.python.b.base.ProgN attribute), 40
- minver (syn.python.b.base.PythonNode attribute), 23
- minver (syn.python.b.base.RootNode attribute), 33
- minver (syn.python.b.base.Special attribute), 39
- minver (syn.python.b.base.Statement attribute), 43
- minver (syn.python.b.base.Store attribute), 28
- minver (syn.python.b.blocks.Arg attribute), 51
- minver (syn.python.b.blocks.Arguments attribute), 52
- minver (syn.python.b.blocks.Block attribute), 45
- minver (syn.python.b.blocks.For attribute), 48
- minver (syn.python.b.blocks.FunctionDef attribute), 54
- minver (syn.python.b.blocks.If attribute), 46
- minver (syn.python.b.blocks.While attribute), 49
- minver (syn.python.b.expressions.Add attribute), 70
- minver (syn.python.b.expressions.And attribute), 92
- minver (syn.python.b.expressions.Attribute attribute), 117
- minver (syn.python.b.expressions.BinaryOperator attribute), 67
- minver (syn.python.b.expressions.BinOp attribute), 69
- minver (syn.python.b.expressions.BitAnd attribute), 86
- minver (syn.python.b.expressions.BitOr attribute), 83
- minver (syn.python.b.expressions.BitXor attribute), 84
- minver (syn.python.b.expressions.BooleanOperator attribute), 89
- minver (syn.python.b.expressions.BoolOp attribute), 90
- minver (syn.python.b.expressions.Call attribute), 113
- minver (syn.python.b.expressions.Comparator attribute), 94
- minver (syn.python.b.expressions.Compare attribute), 96
- minver (syn.python.b.expressions.Div attribute), 74
- minver (syn.python.b.expressions.Eq attribute), 97
- minver (syn.python.b.expressions.Expr attribute), 55
- minver (syn.python.b.expressions.FloorDiv attribute), 76
- minver (syn.python.b.expressions.Gt attribute), 103
- minver (syn.python.b.expressions.GtE attribute), 105
- minver (syn.python.b.expressions.IfExp attribute), 115
- minver (syn.python.b.expressions.In attribute), 109
- minver (syn.python.b.expressions.Invert attribute), 65
- minver (syn.python.b.expressions.Is attribute), 106
- minver (syn.python.b.expressions.IsNot attribute), 107
- minver (syn.python.b.expressions.Keyword attribute), 112
- minver (syn.python.b.expressions.LShift attribute), 80
- minver (syn.python.b.expressions.Lt attribute), 100
- minver (syn.python.b.expressions.LtE attribute), 102
- minver (syn.python.b.expressions.MatMult attribute), 87
- minver (syn.python.b.expressions.Mod attribute), 77
- minver (syn.python.b.expressions.Mult attribute), 73
- minver (syn.python.b.expressions.Not attribute), 64
- minver (syn.python.b.expressions.NotEq attribute), 99
- minver (syn.python.b.expressions.NotIn attribute), 110
- minver (syn.python.b.expressions.Operator attribute), 57
- minver (syn.python.b.expressions.Or attribute), 93
- minver (syn.python.b.expressions.Pow attribute), 79
- minver (syn.python.b.expressions.RShift attribute), 82
- minver (syn.python.b.expressions.Sub attribute), 71
- minver (syn.python.b.expressions.UAdd attribute), 61
- minver (syn.python.b.expressions.UnaryOp attribute), 60
- minver (syn.python.b.expressions.UnaryOperator attribute), 58
- minver (syn.python.b.expressions.USub attribute), 63
- minver (syn.python.b.literals.Bytes attribute), 122

minver (syn.python.b.literals.List attribute), 126
minver (syn.python.b.literals.Literal attribute), 118
minver (syn.python.b.literals.NameConstant attribute), 130
minver (syn.python.b.literals.Num attribute), 119
minver (syn.python.b.literals.Sequence attribute), 124
minver (syn.python.b.literals.Set attribute), 129
minver (syn.python.b.literals.Str attribute), 121
minver (syn.python.b.literals.Tuple attribute), 127
minver (syn.python.b.statements.Alias attribute), 135
minver (syn.python.b.statements.Assign attribute), 132
minver (syn.python.b.statements.Break attribute), 139
minver (syn.python.b.statements.Continue attribute), 140
minver (syn.python.b.statements.EmptyStatement attribute), 138
minver (syn.python.b.statements.Import attribute), 136
minver (syn.python.b.statements.Pass attribute), 142
minver (syn.python.b.statements.Return attribute), 133
minver (syn.python.b.variables.Name attribute), 143
minver (syn.python.b.variables.Starred attribute), 145
Mod (class in syn.python.b.expressions), 76
Module (class in syn.python.b.base), 33
mro() (in module syn.base_utils.py), 12
mro() (syn.types.a.base.TypeType.type method), 230
Mult (class in syn.python.b.expressions), 72
MultiType (class in syn.type.a.type), 227

N

Name (class in syn.python.b.variables), 142
Name (class in syn.tree.b.query), 207
name() (syn.tree.b.node.Node method), 189
NameConstant (class in syn.python.b.literals), 129
nand() (in module syn.base_utils.logic), 11
Ne (class in syn.tree.b.query), 208
nearest_base() (in module syn.base_utils.py), 12
nested_context() (in module syn.base_utils.context), 8
NEType (class in syn.types.a.ne), 232
next() (syn.base_utils.list.IterableList method), 11
ngzwarn() (in module syn.base_utils.py), 13
NoAST (class in syn.python.b.base), 40
Node (class in syn.tree.b.node), 188
node (syn.tree.b.query.Where attribute), 223
node_count() (syn.tree.b.node.Node method), 189
NONE (class in syn.types.a.special), 234
nor() (in module syn.base_utils.logic), 11
Not (class in syn.python.b.expressions), 63
NotEq (class in syn.python.b.expressions), 97
NotEqual (class in syn.types.a.ne), 232
NotIn (class in syn.python.b.expressions), 109
null_context() (in module syn.base_utils.context), 8
Num (class in syn.python.b.literals), 118
Numeric (class in syn.types.a.numeric), 233

O

on_error() (in module syn.base_utils.context), 8
Operator (class in syn.python.b.expressions), 55
Or (class in syn.python.b.expressions), 92
Or (class in syn.schema.b.sequence), 149
or_() (in module syn.base_utils.logic), 11
outer_quotes() (in module syn.base_utils.str), 14
overlaps() (syn.sets.b.leaf.Empty method), 165
overlaps() (syn.sets.b.range.Range method), 173

P

pairs() (in module syn.types.a.base), 230
pairs() (syn.types.a.base.Type method), 229
Param (class in syn.python.b.base), 30
Parent (class in syn.tree.b.query), 209
parent() (syn.tree.b.node.Node method), 189
Partial (class in syn.base_utils.py), 13
partition() (syn.five.string.unicode method), 20
Pass (class in syn.python.b.statements), 140
peek() (syn.base.b.utils.Counter method), 6
peek() (syn.base_utils.list.IterableList method), 11
plaintext() (syn.util.log.b.base.Event method), 243
plaintext() (syn.util.log.b.base.Logger method), 244
plaintext() (syn.util.log.b.events.StringEvent method), 245
pop() (syn.base.b.wrapper.ListWrapper method), 7
pop() (syn.tagmathon.b.interpreter.Env method), 187
pop() (syn.util.log.b.base.Logger method), 244
Position (class in syn.tree.b.query), 211
Pow (class in syn.python.b.expressions), 77
pre_create_hook() (in module syn.base.b.meta), 5
Precedes (class in syn.base_utils.order), 12
Preceding (class in syn.tree.b.query), 212
preceding() (syn.tree.b.node.Node method), 189
Predicate (class in syn.tree.b.query), 213
preprocess() (syn.util.constraint.b.base.Constraint method), 236
preprocess() (syn.util.constraint.b.constraints.EqualConstraint method), 239
preserve_attr_data() (in module syn.base.a.meta), 4
preserve_attr_data() (in module syn.base.b.meta), 5
pretty() (syn.base.b.base.Base method), 4
previous() (syn.base_utils.list.IterableList method), 11
primitive_form() (in module syn.types.a.base), 230
primitive_form() (syn.types.a.base.Type method), 229
print_commands() (syn.base_utils.repl.REPL method), 14
Problem (class in syn.util.constraint.b.base), 236
prod() (in module syn.base_utils.float), 10
Product (class in syn.sets.b.operators), 171
ProgN (class in syn.python.b.base), 39
push() (syn.tagmathon.b.interpreter.Env method), 187
push() (syn.util.log.b.base.Logger method), 244
PythonError, 24

PythonNode (class in syn.python.b.base), 21
 PythonTree (class in syn.python.b.base), 23
 pyversion() (in module syn.base_utils.py), 13

Q

Query (class in syn.tree.b.query), 214
 query() (syn.tree.b.tree.Tree method), 224
 query() (syn.type.a.type.Type method), 226
 query_exception() (syn.type.a.type.Type method), 226
 quit() (syn.base_utils.repl.REPL method), 14
 quote_string() (in module syn.base_utils.str), 14

R

rand_bool() (in module syn.base_utils.rand), 13
 rand_bytes() (in module syn.base_utils.rand), 14
 rand_complex() (in module syn.base_utils.rand), 13
 rand_dict() (in module syn.base_utils.rand), 14
 rand_dispatch() (in module syn.base_utils.rand), 14
 rand_float() (in module syn.base_utils.rand), 13
 rand_frozenset() (in module syn.base_utils.rand), 14
 rand_hashable() (in module syn.base_utils.rand), 14
 rand_int() (in module syn.base_utils.rand), 13
 rand_list() (in module syn.base_utils.rand), 14
 rand_long() (in module syn.base_utils.rand), 13
 rand_none() (in module syn.base_utils.rand), 14
 rand_primitive() (in module syn.base_utils.rand), 14
 rand_set() (in module syn.base_utils.rand), 14
 rand_str() (in module syn.base_utils.rand), 13
 rand_tuple() (in module syn.base_utils.rand), 14
 rand_unicode() (in module syn.base_utils.rand), 14
 Range (class in syn.sets.b.range), 172
 range() (in module syn.five), 21
 rebuild() (syn.tree.b.tree.Tree method), 224
 RecursiveBacktrackSolver (class in syn.util.constraint.b.solvers), 241
 ReflexiveDict (class in syn.base_utils.dict), 9
 register_generable (syn.type.a.ext.Mapping attribute), 226
 register_generable (syn.type.a.ext.Sequence attribute), 225
 register_generable (syn.type.a.ext.Tuple attribute), 225
 register_generable (syn.type.a.type.MultiType attribute), 227
 register_generable (syn.type.a.type.Schema attribute), 228
 register_generable (syn.type.a.type.Set attribute), 228
 register_generable (syn.type.a.type.Type attribute), 226
 register_generable (syn.type.a.type.TypeType attribute), 227
 register_generable (syn.type.a.type.ValuesType attribute), 227
 remove() (syn.base.b.wrapper.ListWrapper method), 7
 remove_child() (syn.tree.b.node.Node method), 189
 remove_node() (syn.tree.b.tree.Tree method), 224

Repeat (class in syn.schema.b.sequence), 150
 REPL (class in syn.base_utils.repl), 14
 repl_command (class in syn.base_utils.repl), 14
 replace() (syn.five.string.unicode method), 20
 replace_node() (syn.tree.b.tree.Tree method), 224
 reset() (syn.base.b.utils.Counter method), 6
 reset() (syn.base_utils.list.IterableList method), 11
 reset() (syn.types.a.ne.DiffExplorer method), 231
 reset() (syn.types.a.ne.ValueExplorer method), 231
 reset() (syn.util.log.b.base.Logger method), 244
 reset_trace() (in module syn.base_utils.debug), 9
 resolve_progn() (syn.python.b.base.ProgN method), 40
 resolve_progn() (syn.python.b.base.PythonNode method), 23
 resolve_progn() (syn.python.b.base.RootNode method), 33
 resolve_progn() (syn.python.b.blocks.If method), 46
 Return (class in syn.python.b.statements), 132
 return_() (syn.base_utils.debug.CallTrace method), 9
 return_() (syn.base_utils.debug.Trace method), 8
 reverse() (syn.base.b.wrapper.ListWrapper method), 7
 rfind() (syn.five.string.unicode method), 20
 rgetattr() (in module syn.base_utils.py), 12
 rindex() (syn.five.string.unicode method), 20
 rjust() (syn.five.string.unicode method), 20
 Root (class in syn.tree.b.query), 215
 root() (syn.tree.b.node.Node method), 189
 RootNode (class in syn.python.b.base), 31
 rootward() (syn.tree.b.node.Node method), 189
 rpartition() (syn.five.string.unicode method), 20
 RShift (class in syn.python.b.expressions), 80
 rsplit() (syn.five.string.unicode method), 20
 rst() (syn.type.a.ext.Mapping method), 226
 rst() (syn.type.a.ext.Sequence method), 225
 rst() (syn.type.a.ext.Tuple method), 225
 rst() (syn.type.a.type.MultiType method), 228
 rst() (syn.type.a.type.Type method), 226
 rst() (syn.type.a.type.TypeType method), 227
 rstr() (in module syn.types.a.base), 230
 rstr() (syn.types.a.base.Type method), 229
 rstr() (syn.types.a.string.Unicode method), 234
 rstrip() (syn.five.string.unicode method), 20
 run_all_tests() (in module syn.base_utils.py), 12

S

safe_chr() (in module syn.base_utils.str), 14
 safe_print() (in module syn.base_utils.str), 15
 safe_sorted() (in module syn.types.a.base), 230
 safe_str() (in module syn.base_utils.str), 14
 safe_unicode() (in module syn.base_utils.str), 15
 safe_vars() (in module syn.base_utils.py), 13
 same_lineage() (in module syn.base_utils.py), 13
 sample() (syn.schema.b.sequence.Sequence method), 153
 sample() (syn.sets.b.base.SetNode method), 157

- sample() (syn.sets.b.leaf.ClassWrapper method), 162
- sample() (syn.sets.b.leaf.SetWrapper method), 160
- sample() (syn.sets.b.leaf.TypeWrapper method), 161
- sample() (syn.sets.b.operators.Difference method), 170
- sample() (syn.sets.b.operators.Intersection method), 169
- sample() (syn.sets.b.operators.Product method), 172
- sample() (syn.sets.b.operators.SetOperator method), 166
- sample() (syn.sets.b.operators.Union method), 168
- sample() (syn.sets.b.range.Range method), 173
- sample() (syn.sets.b.range.StrRange method), 176
- Schema (class in syn.type.a.type), 228
- schema (syn.conf.conf2.ConfList attribute), 16
- SchemaNode (class in syn.schema.b.sequence), 145
- search_rootward() (syn.tree.b.tree.Tree method), 224
- seek() (syn.base_utils.list.IterableList method), 11
- Self (class in syn.tree.b.query), 217
- seq_list_nested() (in module syn.base_utils.tree), 15
- seq_type (syn.type.a.ext.Sequence attribute), 225
- SeqDict (class in syn.base_utils.dict), 9
- Sequence (class in syn.python.b.literals), 122
- Sequence (class in syn.schema.b.sequence), 152
- Sequence (class in syn.type.a.ext), 225
- Sequence (class in syn.types.a.sequence), 233
- ser_args (syn.types.a.base.Type attribute), 229
- ser_args (syn.types.a.numeric.Complex attribute), 233
- ser_attrs (syn.types.a.base.Type attribute), 229
- ser_kwargmap (syn.types.a.base.Type attribute), 229
- ser_kwargs (syn.types.a.base.Type attribute), 229
- serialize() (in module syn.types.a.base), 230
- serialize() (syn.types.a.base.Type method), 229
- serialize_type() (syn.types.a.base.Type class method), 229
- serialize_type() (syn.types.a.special.NONE class method), 234
- Set (class in syn.python.b.literals), 127
- Set (class in syn.schema.b.sequence), 147
- Set (class in syn.type.a.type), 228
- Set (class in syn.types.a.set), 234
- set_child_parents() (syn.tree.b.node.Node method), 189
- set_global() (syn.tagmathon.b.interpreter.Env method), 187
- set_global() (syn.tagmathon.b.interpreter.Frame method), 186
- SetDict (in module syn.base_utils.dict), 9
- SetDifferences (class in syn.types.a.ne), 232
- setitem() (in module syn.base_utils.context), 8
- SetLeaf (class in syn.sets.b.leaf), 157
- SetNode (class in syn.sets.b.base), 155
- SetOperator (class in syn.sets.b.operators), 165
- setstate_hook() (in module syn.base.b.base), 5
- SetWrapper (class in syn.sets.b.leaf), 158
- sgn() (in module syn.base_utils.float), 10
- Sibling (class in syn.tree.b.query), 218
- siblings() (syn.tree.b.node.Node method), 189
- SimpleSolver (class in syn.util.constraint.b.solvers), 240
- simplify() (syn.sets.b.base.SetNode method), 157
- size() (syn.sets.b.base.SetNode method), 157
- size() (syn.sets.b.leaf.ClassWrapper method), 162
- size() (syn.sets.b.leaf.Empty method), 165
- size() (syn.sets.b.leaf.SetWrapper method), 160
- size() (syn.sets.b.leaf.TypeWrapper method), 161
- size() (syn.sets.b.operators.SetOperator method), 166
- size() (syn.sets.b.range.Range method), 173
- size_limits() (syn.sets.b.base.SetNode method), 157
- size_limits() (syn.sets.b.operators.Difference method), 170
- size_limits() (syn.sets.b.operators.Intersection method), 169
- size_limits() (syn.sets.b.operators.Product method), 172
- size_limits() (syn.sets.b.operators.Union method), 168
- solutions() (syn.util.constraint.b.solvers.RecursiveBacktrackSolver method), 242
- solutions() (syn.util.constraint.b.solvers.SimpleSolver method), 241
- solutions() (syn.util.constraint.b.solvers.Solver method), 240
- Solver (class in syn.util.constraint.b.solvers), 240
- sort() (syn.base.b.wrapper.ListWrapper method), 8
- Special (class in syn.python.b.base), 37
- Special (class in syn.sets.b.leaf), 163
- Special (class in syn.tagmathon.b.function), 183
- SpecialCall (class in syn.tagmathon.b.function), 184
- split() (in module syn.base_utils.filters), 10
- split() (syn.five.string.unicode method), 20
- splitlines() (syn.five.string.unicode method), 20
- Starred (class in syn.python.b.variables), 144
- startswith() (syn.five.string.unicode method), 20
- Statement (class in syn.python.b.base), 42
- step() (syn.types.a.ne.DiffExplorer method), 231
- step() (syn.types.a.ne.ValueExplorer method), 231
- Store (class in syn.python.b.base), 27
- Str (class in syn.python.b.literals), 119
- strf (in module syn.five.string), 18
- String (class in syn.types.a.string), 234
- StringEvent (class in syn.util.log.b.events), 244
- strip() (syn.five.string.unicode method), 20
- StrRange (class in syn.sets.b.range), 175
- Sub (class in syn.python.b.expressions), 70
- subclasses() (in module syn.base_utils.py), 13
- Succeeds() (in module syn.base_utils.order), 12
- swapcase() (syn.five.string.unicode method), 21
- symbol (syn.python.b.expressions.Add attribute), 70
- symbol (syn.python.b.expressions.And attribute), 92
- symbol (syn.python.b.expressions.BitAnd attribute), 86
- symbol (syn.python.b.expressions.BitOr attribute), 83
- symbol (syn.python.b.expressions.BitXor attribute), 84
- symbol (syn.python.b.expressions.Div attribute), 74
- symbol (syn.python.b.expressions.Eq attribute), 97

symbol (syn.python.b.expressions.FloorDiv attribute), 76
 symbol (syn.python.b.expressions.Gt attribute), 103
 symbol (syn.python.b.expressions.GtE attribute), 105
 symbol (syn.python.b.expressions.In attribute), 109
 symbol (syn.python.b.expressions.Invert attribute), 66
 symbol (syn.python.b.expressions.Is attribute), 106
 symbol (syn.python.b.expressions.IsNot attribute), 108
 symbol (syn.python.b.expressions.LShift attribute), 80
 symbol (syn.python.b.expressions.Lt attribute), 100
 symbol (syn.python.b.expressions.LtE attribute), 102
 symbol (syn.python.b.expressions.MatMult attribute), 87
 symbol (syn.python.b.expressions.Mod attribute), 77
 symbol (syn.python.b.expressions.Mult attribute), 73
 symbol (syn.python.b.expressions.Not attribute), 64
 symbol (syn.python.b.expressions.NotEq attribute), 99
 symbol (syn.python.b.expressions.NotIn attribute), 110
 symbol (syn.python.b.expressions.Operator attribute), 57
 symbol (syn.python.b.expressions.Or attribute), 93
 symbol (syn.python.b.expressions.Pow attribute), 79
 symbol (syn.python.b.expressions.RShift attribute), 82
 symbol (syn.python.b.expressions.Sub attribute), 72
 symbol (syn.python.b.expressions.UAdd attribute), 61
 symbol (syn.python.b.expressions.USub attribute), 63
 symbol (syn.sets.b.operators.Difference attribute), 170
 symbol (syn.sets.b.operators.Intersection attribute), 169
 symbol (syn.sets.b.operators.Product attribute), 172
 symbol (syn.sets.b.operators.SetOperator attribute), 166
 symbol (syn.sets.b.operators.Union attribute), 168
 syn (module), 246
 syn.base (module), 8
 syn.base.a (module), 4
 syn.base.a.base (module), 3
 syn.base.a.meta (module), 3
 syn.base.b (module), 8
 syn.base.b.base (module), 4
 syn.base.b.examine (module), 5
 syn.base.b.meta (module), 5
 syn.base.b.utils (module), 6
 syn.base.b.wrapper (module), 7
 syn.base_utils (module), 15
 syn.base_utils.alg (module), 8
 syn.base_utils.context (module), 8
 syn.base_utils.debug (module), 8
 syn.base_utils.dict (module), 9
 syn.base_utils.filters (module), 10
 syn.base_utils.float (module), 10
 syn.base_utils.hash (module), 10
 syn.base_utils.iter (module), 10
 syn.base_utils.list (module), 10
 syn.base_utils.logic (module), 11
 syn.base_utils.order (module), 12
 syn.base_utils.py (module), 12
 syn.base_utils.rand (module), 13
 syn.base_utils.repl (module), 14
 syn.base_utils.str (module), 14
 syn.base_utils.tree (module), 15
 syn.conf (module), 18
 syn.conf.conf (module), 15
 syn.conf.conf2 (module), 15
 syn.conf.vars (module), 17
 syn.cython (module), 18
 syn.five (module), 21
 syn.five.num (module), 18
 syn.five.string (module), 18
 syn.globals (module), 21
 syn.globals.loggers (module), 21
 syn.globals.values (module), 21
 syn.python (module), 145
 syn.python.b (module), 145
 syn.python.b.base (module), 21
 syn.python.b.blocks (module), 43
 syn.python.b.expressions (module), 54
 syn.python.b.literals (module), 117
 syn.python.b.statements (module), 130
 syn.python.b.variables (module), 142
 syn.schema (module), 155
 syn.schema.b (module), 155
 syn.schema.b.sequence (module), 145
 syn.serialize (module), 155
 syn.serialize.a (module), 155
 syn.sets (module), 176
 syn.sets.b (module), 176
 syn.sets.b.base (module), 155
 syn.sets.b.leaf (module), 157
 syn.sets.b.operators (module), 165
 syn.sets.b.range (module), 172
 syn.tagmathon (module), 187
 syn.tagmathon.b (module), 187
 syn.tagmathon.b.base (module), 176
 syn.tagmathon.b.builtin (module), 179
 syn.tagmathon.b.compiler (module), 180
 syn.tagmathon.b.function (module), 180
 syn.tagmathon.b.interpreter (module), 186
 syn.tree (module), 224
 syn.tree.b (module), 224
 syn.tree.b.node (module), 188
 syn.tree.b.query (module), 189
 syn.tree.b.tree (module), 223
 syn.type (module), 228
 syn.type.a (module), 228
 syn.type.a.ext (module), 225
 syn.type.a.type (module), 226
 syn.types (module), 235
 syn.types.a (module), 235
 syn.types.a.base (module), 229
 syn.types.a.mapping (module), 230
 syn.types.a.ne (module), 231
 syn.types.a.numeric (module), 233

syn.types.a.sequence (module), 233
syn.types.a.set (module), 234
syn.types.a.special (module), 234
syn.types.a.string (module), 234
syn.util (module), 246
syn.util.constraint (module), 242
syn.util.constraint.b (module), 242
syn.util.constraint.b.base (module), 235
syn.util.constraint.b.constraints (module), 237
syn.util.constraint.b.solvers (module), 240
syn.util.log (module), 246
syn.util.log.b (module), 246
syn.util.log.b.base (module), 242
syn.util.log.b.events (module), 244
SyntagmathonNode (class in syn.tagmathon.b.base), 176

T

take() (syn.base_utils.list.IterableList method), 11
This (class in syn.base.b.meta), 5
This (class in syn.type.a.ext), 226
this_module() (in module syn.base_utils.py), 13
title() (syn.five.string.unicode method), 21
to_ast() (syn.python.b.base.Context method), 26
to_ast() (syn.python.b.base.Expression_ method), 36
to_ast() (syn.python.b.base.PythonNode method), 23
to_ast() (syn.python.b.base.PythonTree method), 24
to_ast() (syn.python.b.base.RootNode method), 33
to_dict() (syn.base.a.base.Base method), 3
to_dict() (syn.base.b.base.Base method), 5
to_python() (in module syn.tagmathon.b.compiler), 180
to_python() (syn.tagmathon.b.base.SyntagmathonNode method), 178
to_python() (syn.tagmathon.b.base.Variable method), 179
to_python() (syn.tagmathon.b.function.Call method), 183
to_python() (syn.tagmathon.b.function.Function method), 182
to_python() (syn.tagmathon.b.function.SpecialCall method), 185
to_set() (syn.sets.b.base.SetNode method), 157
to_set() (syn.sets.b.leaf.ClassWrapper method), 163
to_set() (syn.sets.b.leaf.Empty method), 165
to_set() (syn.sets.b.leaf.SetWrapper method), 160
to_set() (syn.sets.b.leaf.TypeWrapper method), 161
to_set() (syn.sets.b.operators.Difference method), 170
to_set() (syn.sets.b.operators.Intersection method), 169
to_set() (syn.sets.b.operators.Product method), 172
to_set() (syn.sets.b.operators.Union method), 168
to_set() (syn.sets.b.range.Range method), 173
to_set() (syn.sets.b.range.StrRange method), 176
to_tuple() (syn.base.b.base.Base method), 5
topological_sorting() (in module syn.base_utils.order), 12
Trace (class in syn.base_utils.debug), 8
translate() (syn.five.string.unicode method), 21
Tree (class in syn.tree.b.tree), 223

TreeError, 189
Tuple (class in syn.python.b.literals), 126
Tuple (class in syn.type.a.ext), 225
Tuple (class in syn.types.a.sequence), 233
tuple_append() (in module syn.base_utils.py), 13
tuple_prepend() (in module syn.base_utils.py), 13
Type (class in syn.schema.b.sequence), 148
Type (class in syn.tree.b.query), 219
Type (class in syn.type.a.type), 226
Type (class in syn.types.a.base), 229
type (syn.base.b.base.BaseType attribute), 5
type (syn.type.a.type.TypeType attribute), 227
type (syn.types.a.base.Type attribute), 229
type (syn.types.a.mapping.Dict attribute), 230
type (syn.types.a.mapping.Mapping attribute), 230
type (syn.types.a.numeric.Bool attribute), 233
type (syn.types.a.numeric.Complex attribute), 233
type (syn.types.a.numeric.Float attribute), 233
type (syn.types.a.numeric.Int attribute), 233
type (syn.types.a.numeric.Long attribute), 233
type (syn.types.a.numeric.Numeric attribute), 233
type (syn.types.a.sequence.List attribute), 233
type (syn.types.a.sequence.Sequence attribute), 233
type (syn.types.a.sequence.Tuple attribute), 233
type (syn.types.a.set.FrozenSet attribute), 234
type (syn.types.a.set.Set attribute), 234
type (syn.types.a.special.NONE attribute), 234
type (syn.types.a.string.Basestring attribute), 235
type (syn.types.a.string.Bytes attribute), 234
type (syn.types.a.string.String attribute), 234
type (syn.types.a.string.Unicode attribute), 234
type_dispatch() (syn.types.a.base.Type class method), 229
type_partition() (in module syn.base_utils.py), 13
TypeExtension (class in syn.type.a.type), 228
typelist (syn.type.a.type.MultiType attribute), 228
typemap (syn.type.a.type.MultiType attribute), 228
types (syn.type.a.ext.Tuple attribute), 225
types (syn.type.a.type.MultiType attribute), 228
typestr (syn.type.a.type.MultiType attribute), 228
TypeType (class in syn.type.a.type), 227
TypeType (class in syn.types.a.base), 229
TypeType.type (class in syn.types.a.base), 229
TypeWrapper (class in syn.sets.b.leaf), 160

U

UAdd (class in syn.python.b.expressions), 60
UnaryOp (class in syn.python.b.expressions), 58
UnaryOperator (class in syn.python.b.expressions), 57
unichr() (in module syn.five.string), 21
unicode (class in syn.five.string), 18
Unicode (class in syn.types.a.string), 234
uniform (syn.type.a.ext.Tuple attribute), 225
Union (class in syn.sets.b.operators), 166

union() (syn.base_utils.dict.GroupDict method), 9
 union() (syn.sets.b.base.SetNode method), 157
 union() (syn.sets.b.leaf.SetWrapper method), 160
 union() (syn.sets.b.range.Range method), 173
 unzip() (in module syn.base_utils.py), 13
 up() (syn.types.a.ne.DiffExplorer method), 231
 up() (syn.types.a.ne.ValueExplorer method), 231
 update() (syn.base_utils.dict.AssocDict method), 9
 update() (syn.base_utils.dict.GroupDict method), 9
 update() (syn.base_utils.dict.SeqDict method), 9
 update() (syn.base_utils.dict.UpdateDict method), 9
 update() (syn.tagmathon.b.interpreter.Env method), 187
 update() (syn.tagmathon.b.interpreter.Frame method),
 186
 UpdateDict (class in syn.base_utils.dict), 9
 upper() (syn.five.string.unicode method), 21
 USub (class in syn.python.b.expressions), 61

V

validate() (syn.base.a.base.Base method), 3
 validate() (syn.base.b.base.Base method), 5
 validate() (syn.base.b.utils.Counter method), 6
 validate() (syn.base.b.wrapper.ListWrapper method), 8
 validate() (syn.python.b.base.PythonNode method), 23
 validate() (syn.python.b.base.Special method), 39
 validate() (syn.schema.b.sequence.Repeat method), 152
 validate() (syn.schema.b.sequence.Sequence method),
 153
 validate() (syn.sets.b.range.Range method), 173
 validate() (syn.tree.b.node.Node method), 189
 validate() (syn.tree.b.tree.Tree method), 224
 validate() (syn.type.a.type.AnyType method), 227
 validate() (syn.type.a.type.MultiType method), 228
 validate() (syn.type.a.type.Schema method), 228
 validate() (syn.type.a.type.Set method), 228
 validate() (syn.type.a.type.Type method), 226
 validate() (syn.type.a.type.TypeExtension method), 228
 validate() (syn.type.a.type.TypeType method), 227
 validate() (syn.type.a.type.ValuesType method), 227
 validate() (syn.util.constraint.b.base.Problem method),
 237
 Value (class in syn.tree.b.query), 220
 value (syn.types.a.ne.DiffExplorer attribute), 231
 value() (syn.python.b.base.ProgN method), 40
 value_type (syn.type.a.ext.Mapping attribute), 226
 ValueExplorer (class in syn.types.a.ne), 231
 values (syn.type.a.type.ValuesType attribute), 227
 ValuesType (class in syn.type.a.type), 227
 valuify() (syn.python.b.base.ProgN method), 40
 valuify_block() (syn.python.b.blocks.Block method), 45
 Variable (class in syn.tagmathon.b.base), 178
 variables() (syn.python.b.base.PythonNode method), 23
 variables() (syn.python.b.variables.Name method), 144
 Vars (class in syn.conf.vars), 17

vars() (in module syn.tagmathon.b.base), 179
 viewable() (syn.python.b.base.PythonNode method), 23
 visit() (in module syn.types.a.base), 230
 visit() (syn.types.a.base.Type method), 229
 visit_len() (syn.types.a.base.Type method), 229

W

Where (class in syn.tree.b.query), 221
 While (class in syn.python.b.blocks), 48

X

xor() (in module syn.base_utils.logic), 11

Y

YAMLMixin (class in syn.conf.conf), 15

Z

zfill() (syn.five.string.unicode method), 21