# apyengine

*Release 1.0*

**Mark Anacker**

**Dec 01, 2023**

# CONTENTS

CHAPTER

ONE

# APYENGINE PACKAGE

## 1.1 Submodules

### 1.1.1 apyengine.apyengine module

apyengine - An environment for running Python-subset scripts.

This module implements an interpreter for a Python3 subset, along with support functions. It may be embedded in a Python3 host program to provide versatile and extensible scripting. The syntax is Python3, with some significant limitations. To wit - no classes, no importing of Python modules, and no dangerous functions like 'exec()'. This adds a great degree of security when running arbitrary scripts.

Some pre-determined Python modules (such as numpy) may be installed into the interpreter by scripts. Additional optional functionality is provided by extensions. These are full Python scripts that may be loaded on-demand by the user scripts. There are many extensions provided in the distribution, and it's easy to create new ones.

The companion project "apyshell" demonstrates how to fully use and control this engine. <https://github.com/closecrowd/apyshell>

#### Credits

- version: 1.0
- last update: 2023-Nov-17
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023 by Mark Anacker

**Note:** This package incorporates "asteval" from https://github.com/newville/asteval

**class** apyengine.apyengine.**ApyEngine**(*basepath=None*, *builtins_readonly=True*, *global_funcs=False*, *writer=None*, *err_writer=None*)

Bases: `object`

Create an instance of the ApyEngine script runner.

This class contains the interpreter for the apy language, as well as full support structures for controlling it's operation. It is intended for this class to be instanciated in a host application that will perform the support functions, and control this engine.

**__init__**(*basepath=None*, *builtins_readonly=True*, *global_funcs=False*, *writer=None*, *err_writer=None*)

Constructs an instance of the ApyEngine class.

Main entry point for apyengine. It also installs all of the script- callable utility functions.

Args:

**basepath** [The top directory where script files will be found.] (default=./)

**builtins_readonly** [If True, protect the built-in symbols from being] overridden (default=True).

**global_funcs** [If True, all variables are global, even in]

def functions.

**: If False, vars created in a def func are local to** that func (default=False). Can also be modified by setSysFlags_()

**writer** [The output stream for normal print() output.] Defauls to stdout.

err_writer : The output stream for errors. Defaults to stderr.

Returns:

Nothing.

**abortrun**()

Stop a script as soon as possible.

**addcmds**(*cmddict*, *value=None*)

register a whole dict of new commands for the scripts to use.

**check_**(*code*)

Syntax check a Python expression.

Given a string containing a Python expression, parse it and return OK if it's valid, or an error message if not.

This function can be called from within a script ("check_()"), or from the host application.

**Args:** code : An expression to check

**Returns:** 'OK' if the expression is valid 'ERR' and a message if it isn't valid None if code is empty

**clearProcs**(*exception_list=None*)

Remove all currently-defined def functions *except* those on the persistence list *or* in the passed-in exception_list.

Used to remove all "def funcs()" created by the script. Most useful when you're loading a new script programmatically.

**Args:** exception_list : A list of proc names to NOT remove.

**Returns:** None

**delProc**(*pname*)

Remove a specified proc from the engine (and the persist list if needed).

This effectively over-rides the setProcPersist() setting.

**Args:** pname : The name of the def func() to remove

**Returns:** The return value. True for success, False otherwise.

**delcmds**(*cmddict*, *value=None*)
unregister a whole dict of existing commands.

**dumpst_**(*tag=None*)

**dumpus_**()

**eval_**(*cmd*)
Directly execute a script statement in the engine.

Executes a Python statement in the context of the current Interpreter - as if it was in a script. This can set/print variables, run user def funcs(), and so forth. It can be use to make a simple REPL program.

This function can be called from within a script ("eval_()"), or from the host application.

**Args:** cmd : An expression to execute

**Returns:** The results of the expression or None if there was an error

**exit_**(*ret=0*)

**getSysFlags_**(*flagname*)

**getSysVar_**(*name*, *default=None*)

**getvar_**(*vname*, *default=None*)
returns the value of a script variable to the host program

**install_**(*modname*)
Install a pre-authorized Python module into the engine's symbol table.

This is callable from a script with the 'install_()' command. Only modules in the MODULE_LIST list in astutils.py can be installed. Once installed, they can not be uninstalled during this run of apyshell.

**Args:** modname : The module name to install

**Returns:** The return value. True for success, False otherwise.

**isDef_**(*name*)
return True if the symbol is defined

**listDefs_**(*exception_list=None*)
returns a list of currently-defined def functions

**list_Modules_**()

**loadScript_**(*filename*, *persist=False*)
load and execute a script file

**regcmd**(*name*, *func=None*)
Register a new command for the scripts to use.

**reporterr_**(*msg*)
Print error messages on the console error writer.

Prints an error message and returns it.

**Args:** msg : The message to output, and return

**Returns:** The passed-in error message

**setProcPersist**(*pname*, *flag*)
Add or remove the proc from the persist list.

This list protects script-defined functions from the clearProcs() function. This just modifies the persist list - it doesn't affect the presence of the proc in the engine itself.

**Args:** pname : The name of the def func() to presist (or not) flag : if True, add it to the persis list

if False, remove it

**Returns:** The return value. True for success, False otherwise.

**setSysFlags_**(*flagname*, *state*)

**setSysVar_**(*name*, *val*)

**setvar_**(*vname*, *val*)
Set a variable from the outside.

pass None as val to delete the variable

**stop_**(*ret=0*)

**unregcmd**(*name*)
unregister a command.

apyengine.apyengine.**dump**(*obj*, *tag=None*)

apyengine.apyengine.**findFile**(*paths*, *filename*)

apyengine.apyengine.**sanitizePath**(*path*)

### 1.1.2 apyengine.asteval module

Safe(ish) evaluation of mathematical expression using Python's ast module.

Extensively modified by Mark Anacker <closecrowd@pm.me>

Forked from: https://github.com/newville/asteval

This module provides an Interpreter class that compiles a restricted set of Python expressions and statements to Python's AST representation, and then executes that representation using values held in a symbol table. It is meant to be instanciated by the ApyEngine class in apyengine.py.

The symbol table is a simple dictionary, giving a simple, flat namespace. This comes pre-loaded with many functions from Python's builtin and math module. If numpy is installed, many numpy functions are also included. Additional symbols can be added when an Interpreter is created, but the user of that interpreter will not be able to import additional modules. Access to the symbol table is protected by a mutex, allowing multiple threads to access the global state without interfering with each other.

Expressions, including loops, conditionals, and function definitions can be compiled into ast node and then evaluated later, using the current values in the symbol table.

The result is a restricted, simplified version of Python that is somewhat safer than 'eval' because many unsafe operations (such as 'import' and 'eval') are simply not allowed.

**Many parts of Python syntax are supported, including:**

- for loops, while loops, if-then-elif-else conditionals
- try-except (including 'finally')
- function definitions with def
- advanced slicing: a[::-1], array[-3:, :, ::2]
- if-expressions: out = one_thing if TEST else other
- list comprehension out = [sqrt(i) for i in values]

**The following Python syntax elements are not supported:** Import, Exec, Lambda, Class, Global, Generators, Yield, Decorators

In addition, while many builtin functions are supported, several builtin functions that are considered unsafe are missing ('exec', and 'getattr' for example)

### Credits

- version: 1.0
- last update: 2018-Sept-29
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023 by Mark Anacker

---

**Note:**

- Based on: asteval 0.9.13 <https://github.com/newville/asteval>
- Originally by: Matthew Newville, Center for Advanced Radiation Sources, The University of Chicago, <newville@cars.uchicago.edu>

---

**class** apyengine.asteval.**Interpreter**(*symtable=None*, *usersyms=None*, *writer=None*, *err_writer=None*, *readonly_symbols=None*, *builtins_readonly=True*, *global_funcs=False*, *max_statement_length=50000*, *no_print=False*, *raise_errors=False*)

Bases: object

Create an instance of the asteval Interpreter.

This is the main class in this file.

**__init__**(*symtable=None*, *usersyms=None*, *writer=None*, *err_writer=None*, *readonly_symbols=None*, *builtins_readonly=True*, *global_funcs=False*, *max_statement_length=50000*, *no_print=False*, *raise_errors=False*)

Create an asteval Interpreter.

This is a restricted, simplified interpreter using Python syntax. This is meant to be called from the ApyEngine class in apyengine.py.

**Args:** symtable : dictionary to use as symbol table (if *None*, one will be created).

usersyms : dictionary of user-defined symbols to add to symbol table.

writer : callable file-like object where standard output will be sent.

err_writer : callable file-like object where standard error will be sent.

readonly_symbols : symbols that the user can not assign to

builtins_readonly : whether to blacklist all symbols that are in the initial symtable

global_funcs : whether to make procs use the global symbol table

max_statement_length : Maximum length of a script statement

no_print : disable print() output if True

**abortrun**()

Terminate execution of a script.

Sets a flag that causes the currently-running script to exit as quickly as possible.

**addSymbol**(*name*, *val*)

**delSymbol**(*name*)

**static dump**(*node*, ***kw*)
Simple ast dumper.

**eval**(*expr*, *lineno=0*, *show_errors=True*)
Evaluate a single statement.

**getSymbol**(*name*)

**install**(*modname*)
Install a pre-authorized Python module into the engine's symbol table.

This is callable from a script with the 'install_()' command. Only modules in the MODULE_LIST list in astutils.py can be installed. Once installed, they can not be uninstalled during this run of apyshell.

This is called by the install_() function in apyengine.py

**Args:** modname : The module name to install

**Returns:** The return value. True for success, False otherwise.

**isReadOnly**(*varname*)
See if a script variable name is marked read-only

Script variables may be marked as read-only. This will test that status.

**Args:** varnam : The name of the variable

**Return:** True is it's read-only False if it's read-write

**node_assign**(*node*, *val*)
Assign a value (not the node.value object) to a node.

This is used by on_assign, but also by for, list comprehension, etc.

**on_arg**(*node*)
Arg for function definitions.

**on_assert**(*node*)
Assert statement.

**on_assign**(*node*)
Simple assignment.

**on_attribute**(*node*)
Extract attribute.

**on_augassign**(*node*)
Augmented assign.

**on_binop**(*node*)
Binary operator.

**on_boolop**(*node*)
Boolean operator.

**on_break**(*node*)
Break.

**on_call**(*node*)
Function execution.

**on_compare**(*node*)
comparison operators

**on_constant**(*node*)
Return constant value.

**on_continue**(*node*)
Continue.

**on_delete**(*node*)
Delete statement.

**on_dict**(*node*)
Dictionary.

**on_ellipsis**(*node*)
Ellipses.

**on_excepthandler**(*node*)
Exception handler...

**on_expr**(*node*)
Expression.

**on_expression**(*node*)
basic expression

**on_extslice**(*node*)
Extended slice.

**on_for**(*node*)
For blocks.

**on_functiondef**(*node*)
Define procedures.

**on_if**(*node*)
Regular if-then-else statement.

**on_ifexp**(*node*)
If expressions.

**on_index**(*node*)
Index.

**on_interrupt**(*node*)
Interrupt handler.

**on_list**(*node*)
List.

**on_listcomp**(*node*)
List comprehension – only up to 4 generators!

**on_module**(*node*)
Module def.

**on_name**(*node*)
Name node.

**on_nameconstant**(*node*)
named constant True, False, None in python >= 3.4

**on_num**(*node*)
Return number.

**on_pass**(*node*)
Pass statement.

**on_raise**(*node*)
Raise an error

**on_repr**(*node*)
Repr.

**on_return**(*node*)
Return statement: look for None, return special sentinal.

**on_slice**(*node*)
Simple slice.

**on_str**(*node*)
Return string.

**on_subscript**(*node*)
Subscript handling – one of the tricky parts.

**on_try**(*node*)
Try/except/else/finally blocks.

**on_tuple**(*node*)
Tuple.

**on_unaryop**(*node*)
Unary operator.

**on_while**(*node*)
While blocks.

**parse**(*text*)
Parse statement/expression to Ast representation.

**raise_exception**(*node*, *exc=None*, *msg=''*, *expr=None*, *lineno=0*)

**remove_nodehandler**(*node*)
Remove support for a node.

Returns current node handler, so that it might be re-added with add_nodehandler()

**run**(*node*, *expr=None*, *lineno=None*, *with_raise=True*)
Execute parsed Ast representation for an expression.

**set_nodehandler**(*node*, *handler*)
set node handler

**stoprun**()
Terminate execution of a script.

Sets a flag that causes the currently-running script to exit as quickly as possible, without throwing an exception.

**unimplemented**(*node*)
Unimplemented nodes.

**user_defined_symbols**()
Return a set of symbols that have been added to symtable after construction.

I.e., the symbols from self.symtable that are not in self.no_deepcopy.

**Args:** None

**Returns:** A set of symbols in symtable that are not in self.no_deepcopy

**class** apyengine.asteval.**Procedure**(*name*, *interp*, *doc=None*, *lineno=0*, *body=None*, *args=None*, *kwargs=None*, *vararg=None*, *varkws=None*)

Bases: object

Procedure - user-defined function for asteval.

This stores the parsed ast nodes as from the 'functiondef' ast node for later evaluation.

**__init__**(*name*, *interp*, *doc=None*, *lineno=0*, *body=None*, *args=None*, *kwargs=None*, *vararg=None*, *varkws=None*)

TODO: init params.

apyengine.asteval.**dump**(*obj*, *tag=None*)

apyengine.asteval.**dumpnode**(*obj*, *tag=None*)

### 1.1.3 apyengine.astutils module

astutils - utility functions for asteval

#### Credits

- version: 1.0
- last update: 2023-Nov-13
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023 by Mark Anacker

**Note:** Originally by: Matthew Newville, The University of Chicago, <newville@cars.uchicago.edu>

**class** apyengine.astutils.**Empty**

Bases: object

Empty class.

This class is used as a return value in the __call__() and on_return() methods in asteval.Interpreter. If differentiates between an empty return and one with an expression.

**__init__**()

TODO: docstring in public method.

**class** apyengine.astutils.**ExceptionHolder**(*node*, *exc=None*, *msg=''*, *expr=None*, *lineno=0*)

Bases: object

Exception handler support.

This class carries the info needed to properly route and handle exceptions. It's generally called from on_raise() in asteval.py

**__init__**(*node*, *exc=None*, *msg=''*, *expr=None*, *lineno=0*)

Create a new Exception report object

Holds some exception metadata.

**Args:** node : Node that had an exception exc : The exception msg : Error message expr : Expression that caused the exception lineno : Source file line numner

**get_error**()
Retrieve error data.

apyengine.astutils.**NAME_MATCH**(*string=None*, *pos=0*, *endpos=9223372036854775807*, ***, *pattern=None*)
Matches zero or more characters at the beginning of the string.

**class** apyengine.astutils.**NameFinder**
Bases: ast.NodeVisitor

Find all symbol names used by a parsed node.

**__init__**()
TODO: docstring in public method.

**generic_visit**(*node*)
TODO: docstring in public method.

apyengine.astutils.**get_ast_names**(*astnode*)
Return symbol Names from an AST node.

apyengine.astutils.**install_python_module**(*symtable*, *modname*, *modlist*, *rename=True*)
Install a pre-defined Python module.

This function will install one of the Python modules (listed in MODULE_LIST) directly into the symbol table. Some of the functions in the modules are renamed to prevent conflicts with other modules. Once installed, they can not be uninstalled during this run of apyshell.

This is called by the install() function in asteval.py

**Args:** symtable : The symbol table to install into. modname : The module name to install. modlist : A list of currently-installed modules. rename : If True, add an '_' to each function name.

**Returns:** The return value. True for success, False otherwise.

apyengine.astutils.**make_symbol_table**(*modlist*, ***kwargs*)
Create a default symbol table

This function creates the default symbol table, and installs some pre-defined symbols.

**Args:** modlist : list names of currently-installed modules **kwargs : optional additional symbol name, value pairs to include in symbol table

**Returns:** symbol_table : dict a symbol table that can be used in *asteval.Interpereter*

apyengine.astutils.**op2func**(*op*)
Return function for operator nodes.

apyengine.astutils.**safe_add**(*a*, *b*)
safe version of add

apyengine.astutils.**safe_lshift**(*a*, *b*)
safe version of lshift

apyengine.astutils.**safe_mult**(*a*, *b*)
safe version of multiply

apyengine.astutils.**safe_pow**(*base*, *exp*)
safe version of pow

apyengine.astutils.**split_**(*s*, *str=''*, *num=0*)
replacement for string split()

apyengine.astutils.**strcasecmp_**(*s1*, *s2*)
case-insensitive string compare

apyengine.astutils.**type_**(*obj*, **varargs*, ***varkws*)
type that prevents varargs and varkws

apyengine.astutils.**valid_symbol_name**(*name*)
Determine whether the input symbol name is a valid name.

This checks for Python reserved words, and that the name matches the regular expression `[a-zA-Z_][a-zA-Z0-9_]`

**Args:**

**name** [str] name to check for validity.

**Returns:**

**valid** [bool] whether name is a a valid symbol name

## 1.2 Module contents

ApyEngine - An interpreter for running Python-subset scripts.

This package contains an interpreter for a safe subset of the Python3 language. It does NOT run stand-alone, but must be imported into a host application.

The companion project "apyshell" demonstrates how to fully use and control this engine. <https://github.com/closecrowd/apyshell>

### Credits

- version: 1.0
- last update: 2023-Nov-17
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023 by Mark Anacker

---

**Note:**

- This package incorporates "asteval" from https://github.com/newville/asteval

---

**class** apyengine.**ApyEngine**(*basepath=None*, *builtins_readonly=True*, *global_funcs=False*, *writer=None*, *err_writer=None*)
Bases: `object`

Create an instance of the ApyEngine script runner.

This class contains the interpreter for the apy language, as well as full support structures for controlling it's operation. It is intended for this class to be instanciated in a host application that will perform the support functions, and control this engine.

**__init__**(*basepath=None*, *builtins_readonly=True*, *global_funcs=False*, *writer=None*, *err_writer=None*)
Constructs an instance of the ApyEngine class.

Main entry point for apyengine. It also installs all of the script- callable utility functions.

Args:

**basepath** [The top directory where script files will be found.] (default=./)

**builtins_readonly** [If True, protect the built-in symbols from being] overridden (default=True).

**global_funcs** [If True, all variables are global, even in]

def functions.

**: If False, vars created in a def func are local to** that func (default=False). Can also be modified by setSysFlags_()

**writer** [The output stream for normal print() output.] Defauls to stdout.

err_writer : The output stream for errors. Defaults to stderr.

Returns:

Nothing.

**abortrun**()
Stop a script as soon as possible.

**addcmds**(*cmddict*, *value=None*)
register a whole dict of new commands for the scripts to use.

**check_**(*code*)
Syntax check a Python expression.

Given a string containing a Python expression, parse it and return OK if it's valid, or an error message if not.

This function can be called from within a script ("check_()"), or from the host application.

**Args:** code : An expression to check

**Returns:** 'OK' if the expression is valid 'ERR' and a message if it isn't valid None if code is empty

**clearProcs**(*exception_list=None*)
Remove all currently-defined def functions *except* those on the persistence list *or* in the passed-in exception_list.

Used to remove all "def funcs()" created by the script. Most useful when you're loading a new script programmatically.

**Args:** exception_list : A list of proc names to NOT remove.

**Returns:** None

**delProc**(*pname*)
Remove a specified proc from the engine (and the persist list if needed).

This effectively over-rides the setProcPersist() setting.

**Args:** pname : The name of the def func() to remove

**Returns:** The return value. True for success, False otherwise.

**delcmds**(*cmddict*, *value=None*)
unregister a whole dict of existing commands.

**dumpst_**(*tag=None*)

**dumpus_**()

**eval_**(*cmd*)
Directly execute a script statement in the engine.

Executes a Python statement in the context of the current Interpreter - as if it was in a script. This can set/print variables, run user def funcs(), and so forth. It can be use to make a simple REPL program.

This function can be called from within a script ("eval_()"), or from the host application.

**Args:** cmd : An expression to execute

**Returns:** The results of the expression or None if there was an error

**exit_**(*ret=0*)

**getSysFlags_**(*flagname*)

**getSysVar_**(*name*, *default=None*)

**getvar_**(*vname*, *default=None*)
returns the value of a script variable to the host program

**install_**(*modname*)
Install a pre-authorized Python module into the engine's symbol table.

This is callable from a script with the 'install_()' command. Only modules in the MODULE_LIST list in astutils.py can be installed. Once installed, they can not be uninstalled during this run of apyshell.

**Args:** modname : The module name to install

**Returns:** The return value. True for success, False otherwise.

**isDef_**(*name*)
return True if the symbol is defined

**listDefs_**(*exception_list=None*)
returns a list of currently-defined def functions

**list_Modules_**()

**loadScript_**(*filename*, *persist=False*)
load and execute a script file

**regcmd**(*name*, *func=None*)
Register a new command for the scripts to use.

**reporterr_**(*msg*)
Print error messages on the console error writer.

Prints an error message and returns it.

**Args:** msg : The message to output, and return

**Returns:** The passed-in error message

**setProcPersist**(*pname*, *flag*)
Add or remove the proc from the persist list.

This list protects script-defined functions from the clearProcs() function. This just modifies the persist list - it doesn't affect the presence of the proc in the engine itself.

**Args:** pname : The name of the def func() to presist (or not) flag : if True, add it to the persis list

if False, remove it

**Returns:** The return value. True for success, False otherwise.

**setSysFlags_**(*flagname*, *state*)

**setSysVar_**(*name*, *val*)

**setvar_**(*vname*, *val*)
Set a variable from the outside.

pass None as val to delete the variable

**stop_**(*ret=0*)

**unregcmd**(*name*)
unregister a command.

**class** apyengine.**Interpreter**(*symtable=None*, *usersyms=None*, *writer=None*, *err_writer=None*, *readonly_symbols=None*, *builtins_readonly=True*, *global_funcs=False*, *max_statement_length=50000*, *no_print=False*, *raise_errors=False*)
Bases: object

Create an instance of the asteval Interpreter.

This is the main class in this file.

**__init__**(*symtable=None*, *usersyms=None*, *writer=None*, *err_writer=None*, *readonly_symbols=None*, *builtins_readonly=True*, *global_funcs=False*, *max_statement_length=50000*, *no_print=False*, *raise_errors=False*)
Create an asteval Interpreter.

This is a restricted, simplified interpreter using Python syntax. This is meant to be called from the ApyEngine class in apyengine.py.

**Args:** symtable : dictionary to use as symbol table (if *None*, one will be created).

usersyms : dictionary of user-defined symbols to add to symbol table.

writer : callable file-like object where standard output will be sent.

err_writer : callable file-like object where standard error will be sent.

readonly_symbols : symbols that the user can not assign to

builtins_readonly : whether to blacklist all symbols that are in the initial symtable

global_funcs : whether to make procs use the global symbol table

max_statement_length : Maximum length of a script statement

no_print : disable print() output if True

**abortrun**()
Terminate execution of a script.

Sets a flag that causes the currently-running script to exit as quickly as possible.

**addSymbol**(*name*, *val*)

**delSymbol**(*name*)

**static dump**(*node*, ***kw*)
Simple ast dumper.

**eval**(*expr*, *lineno=0*, *show_errors=True*)
Evaluate a single statement.

**getSymbol**(*name*)

**install**(*modname*)
Install a pre-authorized Python module into the engine's symbol table.

This is callable from a script with the 'install_()' command. Only modules in the MODULE_LIST list in astutils.py can be installed. Once installed, they can not be uninstalled during this run of apyshell.

This is called by the install_() function in apyengine.py

**Args:** modname : The module name to install

**Returns:** The return value. True for success, False otherwise.

**isReadOnly**(*varname*)
See if a script variable name is marked read-only

Script variables may be marked as read-only. This will test that status.

**Args:** varnam : The name of the variable

**Return:** True is it's read-only False if it's read-write

**node_assign**(*node*, *val*)
Assign a value (not the node.value object) to a node.

This is used by on_assign, but also by for, list comprehension, etc.

**on_arg**(*node*)
Arg for function definitions.

**on_assert**(*node*)
Assert statement.

**on_assign**(*node*)
Simple assignment.

**on_attribute**(*node*)
Extract attribute.

**on_augassign**(*node*)
Augmented assign.

**on_binop**(*node*)
Binary operator.

**on_boolop**(*node*)
Boolean operator.

**on_break**(*node*)
Break.

**on_call**(*node*)
Function execution.

**on_compare**(*node*)
comparison operators

**on_constant**(*node*)
Return constant value.

**on_continue**(*node*)
Continue.

**on_delete**(*node*)
Delete statement.

**on_dict**(*node*)
Dictionary.

**on_ellipsis**(*node*)
Ellipses.

**on_excepthandler**(*node*)
Exception handler...

**on_expr**(*node*)
Expression.

**on_expression**(*node*)
basic expression

**on_extslice**(*node*)
Extended slice.

**on_for**(*node*)
For blocks.

**on_functiondef**(*node*)
Define procedures.

**on_if**(*node*)
Regular if-then-else statement.

**on_ifexp**(*node*)
If expressions.

**on_index**(*node*)
Index.

**on_interrupt**(*node*)
Interrupt handler.

**on_list**(*node*)
List.

**on_listcomp**(*node*)
List comprehension – only up to 4 generators!

**on_module**(*node*)
Module def.

**on_name**(*node*)
Name node.

**on_nameconstant**(*node*)
named constant True, False, None in python >= 3.4

**on_num**(*node*)
Return number.

**on_pass**(*node*)
Pass statement.

**on_raise**(*node*)
Raise an error

**on_repr**(*node*)
Repr.

**on_return**(*node*)
Return statement: look for None, return special sentinal.

**on_slice**(*node*)
Simple slice.

**on_str**(*node*)
Return string.

**on_subscript**(*node*)
Subscript handling – one of the tricky parts.

**on_try**(*node*)
Try/except/else/finally blocks.

**on_tuple**(*node*)
Tuple.

**on_unaryop**(*node*)
Unary operator.

**on_while**(*node*)
While blocks.

**parse**(*text*)
Parse statement/expression to Ast representation.

**raise_exception**(*node*, *exc=None*, *msg=''*, *expr=None*, *lineno=0*)

**remove_nodehandler**(*node*)
Remove support for a node.

Returns current node handler, so that it might be re-added with add_nodehandler()

**run**(*node*, *expr=None*, *lineno=None*, *with_raise=True*)
Execute parsed Ast representation for an expression.

**set_nodehandler**(*node*, *handler*)
set node handler

**stoprun**()
Terminate execution of a script.

Sets a flag that causes the currently-running script to exit as quickly as possible, without throwing an exception.

**unimplemented**(*node*)
Unimplemented nodes.

**user_defined_symbols**()
Return a set of symbols that have been added to symtable after construction.

I.e., the symbols from self.symtable that are not in self.no_deepcopy.

**Args:** None

**Returns:** A set of symbols in symtable that are not in self.no_deepcopy

**class** apyengine.**NameFinder**
Bases: ast.NodeVisitor

Find all symbol names used by a parsed node.

**__init__**()
TODO: docstring in public method.

**generic_visit**(*node*)
TODO: docstring in public method.

apyengine.**get_ast_names**(*astnode*)
Return symbol Names from an AST node.

apyengine.**make_symbol_table**(*modlist*, ***kwargs*)
Create a default symbol table

This function creates the default symbol table, and installs some pre-defined symbols.

**Args:** modlist : list names of currently-installed modules **kwargs : optional additional symbol name, value pairs to include in symbol table

**Returns:** symbol_table : dict a symbol table that can be used in *asteval.Interpereter*

apyengine.**valid_symbol_name**(*name*)
Determine whether the input symbol name is a valid name.

This checks for Python reserved words, and that the name matches the regular expression `[a-zA-Z_][a-zA-Z0-9_]`

**Args:**

**name** [str] name to check for validity.

**Returns:**

**valid** [bool] whether name is a a valid symbol name

# PYTHON MODULE INDEX

## a

# INDEX

## Symbols



## A



## C



## D



## E



## F



## G

## I



## L



## M



## N



## O

## P



## R



## S

## T



## U



## V