

# **Compiler MATLAB to Python**

*Release 0.1.0*

**Artem Eroshev**

Apr 30, 2021



<b>1</b>	<b>cmp</b>	<b>1</b>
1.1	CMP package . . . . .	1
	<b>Python Module Index</b>	<b>33</b>
	<b>Index</b>	<b>35</b>



## 1.1 CMP package

### 1.1.1 Subpackages

#### Ast package

##### Submodules

##### Additive module

```
class cmp.ast.additive.MinusNode ( lhs, rhs )  
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode  
    Node of minus operation
```

**lhs**

**rhs**

```
class cmp.ast.additive.PlusNode ( lhs, rhs )  
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode  
    Node of plus operation
```

**lhs**

**rhs**

##### Array module

```
class cmp.ast.array.ArrayNode ( ident, content )  
    Bases: cmp.ast.node.Node  
    Create array by defined rules Expected expression like >>> zeros(1, 5) and etc
```

**content**

**ident**

```
class cmp.ast.array.ArrayVectorNode ( content )  
    Bases: cmp.ast.node.Node  
    Node of vector array Expected expression like >>> [1, 2, 3, 4 * 3] and etc
```

**content**

*Assignment module*

```
class cmp.ast.assignment.AssignmentNode ( lhs, rhs )
```

```
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
```

```
    Assignment object node: Lhs Rhs object
```

```
    lhs
```

```
    rhs
```

*Comment module*

```
class cmp.ast.comment.CommentNode ( comment )
```

```
    Bases: cmp.ast.node.Node
```

```
    comment
```

*Conditional statement module*

```
class cmp.ast.conditional_statement.ConditionalNode ( main_stmt )
```

```
    Bases: cmp.ast.node.Node
```

```
    Base node for conditional statements
```

```
    main_stmt
```

```
class cmp.ast.conditional_statement.ElseIfClauseNode ( main_stmt, stmt_list )
```

```
    Bases: cmp.ast.conditional_statement.ConditionalNode
```

```
    Node of elseif clause
```

```
    main_stmt
```

```
    stmt_list
```

```
class    cmp.ast.conditional_statement.ManyBranchConditionalNode    (    main_stmt,  
main_branch, alt_chain, alt_branch )
```

```
    Bases: cmp.ast.conditional_statement.ConditionalNode
```

```
    Node of many conditional statement
```

```
    alt_branch
```

```
    alt_chain
```

```
    main_branch
```

```
    main_stmt
```

```
class cmp.ast.conditional_statement.SimpleConditionalNode ( main_stmt, stmt_list )
```

```
    Bases: cmp.ast.conditional_statement.ConditionalNode
```

```
    Conditional node for one way statement >>> if (expression is True) >>>    do_something
```

```
    main_stmt
```

```
    stmt_list
```

```
class    cmp.ast.conditional_statement.TwoBranchConditionalNode    (    main_stmt,  
main_branch, alt_branch )
```

```
    Bases: cmp.ast.conditional_statement.ConditionalNode
```

```
    Conditional node for two way statement >>> if (expression is True) >>>    do_something >>>  
    else >>>    to_do
```

```
    alt_branch
```

**main\_branch**

**main\_stmt**

*Define clear module*

**class** cmp.ast.define\_clear.**ClearNode** ( *id\_list* )

Bases: cmp.ast.node.Node

Object of clear key word

**id\_list**

*Define global module*

**class** cmp.ast.define\_global.**GlobalNode** ( *id\_list* )

Bases: cmp.ast.node.Node

Object of global key word

**id\_list**

*Equality module*

**class** cmp.ast.equality.**NegativeEqualityNode** ( *lhs, rhs* )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Negative equality object node

**lhs**

**rhs**

**class** cmp.ast.equality.**PositiveEqualityNode** ( *lhs, rhs* )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Positive equality object node

**lhs**

**rhs**

*Error module*

**class** cmp.ast.error.**ErrorNode** ( *message* )

Bases: cmp.ast.node.Node

Node of error parsing

**message**

*Finite unit module*

**class** cmp.ast.finite\_unit.**ConstantNode** ( *const* )

Bases: cmp.ast.node.Node

Node of constant

**const**

**class** cmp.ast.finite\_unit.**IdentifierNode** ( *ident* )

Bases: cmp.ast.node.Node

Node of identifier

**ident**

**class** cmp.ast.finite\_unit.**SimpleNode** ( *content* )

Bases: cmp.ast.node.Node

Finite point in traverse tree

**content***Function module*

```
class cmp.ast.function.FunctionDeclareNode ( return_list, name )
```

Bases: `cmp.ast.node.Node`

Declaration of function

**name**

**return\_list**

```
class cmp.ast.function.FunctionNameNode ( name, input_list )
```

Bases: `cmp.ast.node.Node`

**input\_list**

**name**

```
class cmp.ast.function.FunctionNode ( declare, body )
```

Bases: `cmp.ast.node.Node`

Node of function object

**body**

**declare**

*Iterations module*

```
class cmp.ast.iterations.ForLoopNode ( iterator, express, body )
```

Bases: `cmp.ast.node.Node`

Object of FOR loop

**body**

**express**

**iter**

```
class cmp.ast.iterations.WhileLoopNode ( express, body )
```

Bases: `cmp.ast.node.Node`

**body**

**express**

*Jump statement module*

```
class cmp.ast.jump_stmt.BreakNode
```

Bases: `cmp.ast.node.Node`

Key word BREAK

```
class cmp.ast.jump_stmt.ReturnNode
```

Bases: `cmp.ast.node.Node`

Key word RETURN

*Lhs rhs node module*

```
class cmp.ast.lhs_rhs_node.LhsRhsNode ( lhs, rhs )
```

Bases: `cmp.ast.node.Node`

Base for lhs, rhs object node



**lhs****rhs***Logic module***class** cmp.ast.logic.**AndNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Object of logic AND

**lhs****rhs****class** cmp.ast.logic.**OrNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Object of logic OR

**lhs****rhs***Multiplicative module***class** cmp.ast.multiplicative.**ArrayDivNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Object of divide array

**lhs****rhs****class** cmp.ast.multiplicative.**ArrayMulNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Object of multiply array

**lhs****rhs****class** cmp.ast.multiplicative.**ArrayPowerNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Object of power array

**lhs****rhs****class** cmp.ast.multiplicative.**ArrayRDivNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Object of right divide array

**lhs****rhs****class** cmp.ast.multiplicative.**DivideNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Object of divide

**lhs**

**rhs**

**class** cmp.ast.multiplicative.**MultiplyNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Object of multiply

**lhs**

**rhs**

**class** cmp.ast.multiplicative.**PowerNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Object of power

**lhs**

**rhs**

*Node module*

**class** cmp.ast.node.**Node**

Bases: abc.ABC

Base node for AST

*Relational module*

**class** cmp.ast.relational.**GreaterEqualRelationalNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Greater or equal relational object node

**lhs**

**rhs**

**class** cmp.ast.relational.**GreaterRelationalNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Greater relational object node

**lhs**

**rhs**

**class** cmp.ast.relational.**LowerEqualRelationalNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Lower or equal object node

**lhs**

**rhs**

**class** cmp.ast.relational.**LowerRelationalNode** ( lhs, rhs )

Bases: cmp.ast.lhs\_rhs\_node.LhsRhsNode

Lower object node

**lhs**

**rhs**

*Root module*

**class** cmp.ast.root.**FileAST** ( root )

Bases: cmp.ast.node.Node

Entry point in AST

**root**

*Sparse module*

```
class cmp.ast.sparse.SparseNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Node of smudge value
```

**lhs**

**rhs**

*Transpose module*

```
class cmp.ast.transpose.TransposeNode ( expr )
    Bases: cmp.ast.node.Node
    Node of transpose operation
```

**expr**

*Unary expression module*

```
class cmp.ast.unary_expression.UnaryExpressionNode ( unary_op, expr )
    Bases: cmp.ast.node.Node
    Node of unary operator
```

**expr**

**unary\_op**

*Module contents*

```
class cmp.ast.AndNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Object of logic AND
```

**lhs**

**rhs**

```
class cmp.ast.ArrayDivNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Object of divide array
```

**lhs**

**rhs**

```
class cmp.ast.ArrayMulNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Object of multiply array
```

**lhs**

**rhs**

```
class cmp.ast.ArrayNode ( ident, content )
    Bases: cmp.ast.node.Node
    Create array by defined rules Expected expression like >>> zeros(1, 5) and etc
```

**content**

**ident**

**class** `cmp.ast.ArrayPowerNode ( lhs, rhs )`

Bases: `cmp.ast.lhs_rhs_node.LhsRhsNode`

Object of power array

**lhs**

**rhs**

**class** `cmp.ast.ArrayRDivNode ( lhs, rhs )`

Bases: `cmp.ast.lhs_rhs_node.LhsRhsNode`

Object of right divide array

**lhs**

**rhs**

**class** `cmp.ast.ArrayVectorNode ( content )`

Bases: `cmp.ast.node.Node`

Node of vector array Expected expression like >>> [1, 2, 3, 4 \* 3] and etc

**content**

**class** `cmp.ast.AssignmentNode ( lhs, rhs )`

Bases: `cmp.ast.lhs_rhs_node.LhsRhsNode`

Assignment object node: Lhs Rhs object

**lhs**

**rhs**

**class** `cmp.ast.BreakNode`

Bases: `cmp.ast.node.Node`

Key word BREAK

**class** `cmp.ast.ClearNode ( id_list )`

Bases: `cmp.ast.node.Node`

Object of clear key word

**id\_list**

**class** `cmp.ast.CommentNode ( comment )`

Bases: `cmp.ast.node.Node`

**comment**

**class** `cmp.ast.ConstantNode ( const )`

Bases: `cmp.ast.node.Node`

Node of constant

**const**

**class** `cmp.ast.DivideNode ( lhs, rhs )`

Bases: `cmp.ast.lhs_rhs_node.LhsRhsNode`

Object of divide

**lhs**

**rhs**

**class** `cmp.ast.ElseIfClauseNode` ( *main\_stmt, stmt\_list* )  
Bases: `cmp.ast.conditional_statement.ConditionalNode`  
Node of elseif clause

**main\_stmt**

**stmt\_list**

**class** `cmp.ast.ErrorNode` ( *message* )  
Bases: `cmp.ast.node.Node`  
Node of error parsing

**message**

**class** `cmp.ast.FileAST` ( *root* )  
Bases: `cmp.ast.node.Node`  
Entry point in AST

**root**

**class** `cmp.ast.ForLoopNode` ( *iterator, express, body* )  
Bases: `cmp.ast.node.Node`  
Object of FOR loop

**body**

**express**

**iter**

**class** `cmp.ast.FunctionDeclareNode` ( *return\_list, name* )  
Bases: `cmp.ast.node.Node`  
Declaration of function

**name**

**return\_list**

**class** `cmp.ast.FunctionNameNode` ( *name, input\_list* )  
Bases: `cmp.ast.node.Node`

**input\_list**

**name**

**class** `cmp.ast.FunctionNode` ( *declare, body* )  
Bases: `cmp.ast.node.Node`  
Node of function object

**body**

**declare**

**class** `cmp.ast.GlobalNode` ( *id\_list* )  
Bases: `cmp.ast.node.Node`  
Object of global key word

```

    id_list

class cmp.ast.GreaterEqualRelationalNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Greater or equal relational object node

    lhs

    rhs

class cmp.ast.GreaterRelationalNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Greater relational object node

    lhs

    rhs

class cmp.ast.IdentifierNode ( ident )
    Bases: cmp.ast.node.Node
    Node of identifier

    ident

class cmp.ast.LowerEqualRelationalNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Lower or equal object node

    lhs

    rhs

class cmp.ast.LowerRelationalNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Lower object node

    lhs

    rhs

class cmp.ast.ManyBranchConditionalNode ( main_stmt, main_branch, alt_chain, alt_branch )
    Bases: cmp.ast.conditional_statement.ConditionalNode
    Node of many conditional statement

    alt_branch

    alt_chain

    main_branch

    main_stmt

class cmp.ast.MinusNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Node of minus operation

    lhs

    rhs

```

```

class cmp.ast.MultiplyNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Object of multiply

    lhs

    rhs

class cmp.ast.NegativeEqualityNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Negative equality object node

    lhs

    rhs

class cmp.ast.Node
    Bases: abc.ABC
    Base node for AST

class cmp.ast.OrNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Object of logic OR

    lhs

    rhs

class cmp.ast.PlusNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Node of plus operation

    lhs

    rhs

class cmp.ast.PositiveEqualityNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Positive equality object node

    lhs

    rhs

class cmp.ast.PowerNode ( lhs, rhs )
    Bases: cmp.ast.lhs_rhs_node.LhsRhsNode
    Object of power

    lhs

    rhs

class cmp.ast.ReturnNode
    Bases: cmp.ast.node.Node
    Key word RETURN

class cmp.ast.SimpleConditionalNode ( main_stmt, stmt_list )
    Bases: cmp.ast.conditional_statement.ConditionalNode
    Conditional node for one way statement >>> if (expression is True) >>>    do_something

```

**main\_stmt**

**stmt\_list**

**class** `cmp.ast.SimpleNode ( content )`

Bases: `cmp.ast.node.Node`

Finite point in traverse tree

**content**

**class** `cmp.ast.SparseNode ( lhs, rhs )`

Bases: `cmp.ast.lhs_rhs_node.LhsRhsNode`

Node of smudge value

**lhs**

**rhs**

**class** `cmp.ast.TransposeNode ( expr )`

Bases: `cmp.ast.node.Node`

Node of transpose operation

**expr**

**class** `cmp.ast.TwoBranchConditionalNode ( main_stmt, main_branch, alt_branch )`

Bases: `cmp.ast.conditional_statement.ConditionalNode`

Conditional node for two way statement >>> if (expression is True) >>> do\_something >>>  
else >>> to\_do

**alt\_branch**

**main\_branch**

**main\_stmt**

**class** `cmp.ast.UnaryExpressionNode ( unary_op, expr )`

Bases: `cmp.ast.node.Node`

Node of unary operator

**expr**

**unary\_op**

**class** `cmp.ast.WhileLoopNode ( express, body )`

Bases: `cmp.ast.node.Node`

**body**

**express**

## CLI package

*Submodules*

*CLI module*

**class** `cmp.cli.cli.Command ( *args, **kwargs )`

Bases: `argparse.ArgumentParser`

`cmp.helpers.singleton.Singleton`

CLI interface class

`cmp.helpers.log.LogMixin`,



**execute ( )**

Entry point of program CMP for executed in command line

**Return type** None

*Handlers check module*

**class** `cmp.cli.handlers_check.AbstractHandler`

Bases: `cmp.cli.handlers_check.Handler`, `cmp.helpers.log.LogMixin`

Abstract handler for CLI interface CMP

**abstract handle ( args )**

**Return type** Optional[str]

**set\_next ( handler )**

**Return type** Handler

**class** `cmp.cli.handlers_check.CheckOutputFile`

Bases: `cmp.cli.handlers_check.AbstractHandler`

Check -of/-output-file key. If it is true, result data will write in file by directed file. In default result show in output stream

**handle ( args )**

**Return type** Optional[str]

**class** `cmp.cli.handlers_check.CheckServerKey`

Bases: `cmp.cli.handlers_check.AbstractHandler`

Check server option in input stream. If key -S/-server is true, TCP server will be started

**handle ( args )**

**Return type** Optional[str]

**static network\_execute ( message, parser )**

Special entry point of program CMP for working in TCP server

**Return type** Optional[str]

**class** `cmp.cli.handlers_check.CheckStringKey`

Bases: `cmp.cli.handlers_check.AbstractHandler`

Check -s/-string and -p/-path key in input stream. Only one of two keys expected. If -string is true, data will get from input stream. If -path is true, data will read from file by directed path

**handle ( args )**

**Return type** Optional[str]

**class** `cmp.cli.handlers_check.GetResult`

Bases: `cmp.cli.handlers_check.AbstractHandler`

After checks all keys this class generated result

**handle ( args )**

**Return type** Optional[str]

**class** `cmp.cli.handlers_check.Handler`

Bases: `abc.ABC`

Base handler without subject area

**abstract handle** ( *args* )

**Return type** Optional[str]

**abstract set\_next** ( *handler* )  
Return type `Handler`

## Module contents

## Grammar package

## Subpackages

### Submodules

### Lexer module

```

class cmp.grammar.lexer.Lexer
    Bases: cmp.helpers.log.LogMixin
    Executive lexer object. Containing primary tokens. This class give next token

    D = '[0-9]'

    E = '[DdEe][+-]?[0-9]+'

    L = '[a-zA-Z_]\'

    constant = '[0-9]+([DdEe][+-]?[0-9]+)? | [0-9]*"."[0-9]+([DdEe][+-]?[0-9]+)? | [0-9]+."'[0-9]*([D-
dEe][+-]?[0-9]+)?'

    constant_1 = '[0-9]+([DdEe][+-]?[0-9]+)?'

    constant_2 = '[0-9]*"."[0-9]+([DdEe][+-]?[0-9]+)?'

    constant_3 = '[0-9]+."'[0-9]*([DdEe][+-]?[0-9]+)?'

    identifier = '[a-zA-Z_]([a-zA-Z_] | [0-9])*'

    input ( data_ )

        Return type None

    keywords = {'break': 'BREAK', 'clear': 'CLEAR', 'else': 'ELSE', 'elseif': 'ELSEIF', 'end': 'END',
'for': 'FOR', 'function': 'FUNCTION', 'global': 'GLOBAL', 'if': 'IF', 'return': 'RETURN', 'while':
'WHILE'}

    literals = ['~', ',', '!', ':', '=', '(', ')', '[', ']', '&', '-', '+', '*', '/', '\\', '>', '<', '|']

    states = (('string', 'exclusive'),)

    t_ANY_COMMENT ( token_ )
        [%] | [n]
        Return type LexToken

    t_ARRAY_DIV = '\\\./'

    t_ARRAY_MUL = '\\\.\^*'

    t_ARRAY_POW = '\\\.\^'

    t_ARRAY_RDIV = '\\\./'

    t_CONSTANT ( token_ )
        Return type LexToken

```

```

t_EQ_OP = '=='

t_GE_OP = '\\>='

t_IDENTIFIER ( token_ )
    Return type LexToken

t_LE_OP = '<='

t_NEWLINE ( token_ )
    n
    Return type LexToken

t_NE_OP = '(~=) | (!=)'

t_STRING_LITERAL ( token_ )
    '[^\\n]*'
    Return type LexToken

t_TRANSPOSE ( token_ )
    Return type LexToken

t_error ( token_ )
    Error handler lexer
    Return type None

t_ignore_WHITESPACE = '\\s+'

t_string_TCOMMENT = '[^\\n]+'

t_string_error ( token_ )
    Error handler lexer for string state
    Return type None

t_string_ignore = ""

token ( )
    Return type LexToken

tokens = ('IDENTIFIER', 'CONSTANT', 'STRING_LITERAL', 'LE_OP', 'GE_OP', 'EQ_OP',
'NE_OP', 'ARRAY_MUL', 'ARRAY_POW', 'ARRAY_DIV', 'ARRAY_RDIV', 'TRANSPOSE',
'NEWLINE', 'COMMENT', 'TCOMMENT', 'FOR', 'WHILE', 'BREAK', 'IF', 'ELSE', 'ELSEIF',
'END', 'FUNCTION', 'RETURN', 'GLOBAL', 'CLEAR')

transpose = ""|\\\\"

transpose_1 = ""

transpose_2 = "\\\"

```

#### Parser module

```

class cmp.grammar.parser.Parser ( lexer=<class 'cmp.grammar.lexer.Lexer'>, yacc_debug=False )
    Bases: object
    Executive parser object. Containing primary reduce rules. This class build AST

```

**errors ( )**

Return type `Iterator[str]`

**handlers** = {'!=': <class 'cmp.ast.equality.NegativeEqualityNode'>, '&': <class 'cmp.ast.logic.AndNode'>, '\*': <class 'cmp.ast.multiplicative.MultiplyNode'>, '+': <class 'cmp.ast.additive.PlusNode'>, '-': <class 'cmp.ast.additive.MinusNode'>, '\*.': <class 'cmp.ast.multiplicative.ArrayMulNode'>, './': <class 'cmp.ast.multiplicative.ArrayDivNode'>, './.': <class 'cmp.ast.multiplicative.ArrayRDivNode'>, '^': <class 'cmp.ast.multiplicative.ArrayPowerNode'>, '/': <class 'cmp.ast.multiplicative.DivideNode'>, '<': <class 'cmp.ast.relational.LowerRelationalNode'>, '<=': <class 'cmp.ast.relational.LowerEqualRelationalNode'>, '==': <class 'cmp.ast.equality.PositiveEqualityNode'>, '>': <class 'cmp.ast.relational.GreaterRelationalNode'>, '>=': <class 'cmp.ast.relational.GreaterEqualRelationalNode'>, '^': <class 'cmp.ast.multiplicative.PowerNode'>, '|': <class 'cmp.ast.logic.OrNode'>}

**property has\_errors**

Return type `bool`

**p\_additive\_expression ( p )**

**additive\_expression** : *multiplicative\_expression*  
additive\_expression '+' *multiplicative\_expression*  
additive\_expression '-' *multiplicative\_expression*

Return type `None`

**p\_and\_expression ( p )**

**and\_expression** : *equality\_expression*  
and\_expression '&' *equality\_expression*

Return type `None`

**p\_array\_element ( p )**

**array\_element** : *expression*  
*expression\_statement*

Return type `None`

**p\_array\_expression ( p )**

**array\_expression** : IDENTIFIER '(' *index\_expression\_list* ')'

Return type `None`

**p\_array\_list ( p )**

**array\_list** : *array\_element*  
*array\_list* *array\_element*

Return type `None`

**p\_assignment\_expression ( p )**

**assignment\_expression** : postfix\_expression '=' *expression*

Return type `None`

**p\_assignment\_statement ( p )**

**assignment\_statement** : *assignment\_expression* *eastmt*

Return type `None`

**p\_clear\_statement ( p )**

**clear\_statement** : CLEAR *identifier\_list* *eastmt*

**Return type** None

**p\_comment\_statement** (*p*)

comment\_statement : COMMENT TCOMMENT

**Return type** None

**p\_constant\_expression** (*p*)

constant\_expression : CONSTANT

**Return type** None

**p\_elseif\_clause** (*p*)

elseif\_clause : *ELSEIF* expression statement\_list

elseif\_clause ELSEIF expression statement\_list

**Return type** None

**p\_eostmt** (*p*)

eostmt : ','

','

NEWLINE

**Return type** None

**p\_equality\_expression** (*p*)

equality\_expression : *relational\_expression*

equality\_expression EQ\_OP relational\_expression

equality\_expression NE\_OP relational\_expression

**Return type** None

**p\_error** (*p*)

**Return type** None

**p\_expression** (*p*)

expression : *or\_expression*

expression ':' or\_expression

**Return type** None

**p\_expression\_statement** (*p*)

expression\_statement : *eostmt*

expression eostmt

**Return type** None

**p\_func\_declare** (*p*)

func\_declare : *func\_declare\_lhs*

func\_return\_list '=' func\_declare\_lhs

func\_declare\_invoke\_error

**Return type** None

**p\_func\_declare\_invoke\_error** (*p*)

func\_declare\_invoke\_error : *func\_return\_list* '='

func\_return\_list

**Return type** None

```

p_func_declare_lhs ( p )
    func_declare_lhs : IDENTIFIER
    IDENTIFIER '(' ' '
    IDENTIFIER '(' func_identifier_list ' '

    Return type None

p_func_identifier_list ( p )
    func_identifier_list : IDENTIFIER
    func_identifier_list ',' IDENTIFIER

    Return type None

p_func_return_list ( p )
    func_return_list : IDENTIFIER
    '[' func_identifier_list ']'

    Return type None

p_func_statement ( p )
    func_statement : FUNCTION func_declare eostmt statement_list END
    func_statement_error

    Return type None

p_func_statement_error ( p )
    func_statement_error : FUNCTION error eostmt statement_list END

    Return type None

p_global_statement ( p )
    global_statement : GLOBAL identifier_list eostmt

    Return type None

p_identifier_expression ( p )
    identifier_expression : IDENTIFIER

    Return type None

p_identifier_list ( p )
    identifier_list : IDENTIFIER
    identifier_list IDENTIFIER

    Return type None

p_index_expression ( p )
    index_expression : ':'
    expression

    Return type None

p_index_expression_list ( p )
    index_expression_list : index_expression
    index_expression_list ',' index_expression

    Return type None

p_iteration_statement ( p )
    iteration_statement : WHILE expression statement_list END eostmt
    FOR IDENTIFIER '=' expression statement_list END eostmt

```

FOR '(' IDENTIFIER '=' expression ')' statement\_list END eostmt

**Return type** None

**p\_jump\_statement** ( *p* )

**jump\_statement** : *BREAK eostmt*

RETURN eostmt

**Return type** None

**p\_multiplicative\_expression** ( *p* )

**multiplicative\_expression** : *unary\_expression*

multiplicative\_expression '\*' unary\_expression

multiplicative\_expression '/' unary\_expression

multiplicative\_expression '^' unary\_expression

multiplicative\_expression ARRAY\_MUL unary\_expression

multiplicative\_expression ARRAY\_DIV unary\_expression

multiplicative\_expression ARRAY\_RDIV unary\_expression

multiplicative\_expression ARRAY\_POW unary\_expression

**Return type** None

**p\_or\_expression** ( *p* )

**or\_expression** : *and\_expression*

or\_expression '|' and\_expression

**Return type** None

**p\_postfix\_expression** ( *p* )

**postfix\_expression** : *primary\_expression*

array\_expression

postfix\_expression TRANSPOSE

**Return type** None

**p\_primary\_expression** ( *p* )

**primary\_expression** : *identifier\_expression*

constant\_expression

string\_literal\_expression

'(' expression ')'

'[' '']

'[' array\_list '']

**Return type** None

**p\_relational\_expression** ( *p* )

**relational\_expression** : *additive\_expression*

relational\_expression '<' additive\_expression

relational\_expression '>' additive\_expression

relational\_expression LE\_OP additive\_expression

relational\_expression GE\_OP additive\_expression

**Return type** None

**p\_selection\_statement** ( *p* )

**selection\_statement** : *IF expression statement\_list END eostmt*

IF expression statement\_list ELSE statement\_list END eostmt

IF expression statement\_list elseif\_clause END eostmt

IF expression statement\_list elseif\_clause ELSE statement\_list END eostmt

selection\_statement\_invoke\_error

selection\_statement\_error

**Return type** None

**p\_selection\_statement\_error** ( *p* )

selection\_statement\_error : IF error

**Return type** None

**p\_selection\_statement\_invoke\_error** ( *p* )

selection\_statement\_invoke\_error : IF expression statement\_list

**Return type** None

**p\_statement** ( *p* )

statement : *global\_statement*

clear\_statement

assignment\_statement

expression\_statement

selection\_statement

iteration\_statement

jump\_statement

func\_statement

comment\_statement

**Return type** None

**p\_statement\_list** ( *p* )

statement\_list : *statement*

statement\_list statement

statement\_list\_error

**Return type** None

**p\_statement\_list\_error** ( *p* )

statement\_list\_error : statement\_list error

**Return type** None

**p\_string\_literal\_expression** ( *p* )

string\_literal\_expression : STRING\_LITERAL

**Return type** None

**p\_translation\_unit** ( *p* )

translation\_unit : statement\_list

**Return type** None

**p\_unary\_expression** ( *p* )

unary\_expression : *postfix\_expression*

unary\_operator postfix\_expression

**Return type** None

**p\_unary\_operator** ( *p* )

unary\_operator : '+'

'\_'

'~'

**Return type** None



**parse** ( *text*, *debug\_level=False* )

Return type Any

**precedence** = (('right', '-'), ('right', '~'), ('right', '+'))

*Module contents*

**class** cmp.grammar.**Lexer**

Bases: cmp.helpers.log.LogMixin

Executive lexer object. Containing primary tokens. This class give next token

**D** = '[0-9]'

**E** = '[DdEe][+-]?[0-9]+'

**L** = '[a-zA-Z\_]'

**constant** = '[0-9]+([DdEe][+-]?[0-9]+)?|[0-9]\*"."[0-9]+([DdEe][+-]?[0-9]+)?|[0-9]+"."[0-9]\*([DdEe][+-]?[0-9]+)?'

**constant\_1** = '[0-9]+([DdEe][+-]?[0-9]+)?'

**constant\_2** = '[0-9]\*"."[0-9]+([DdEe][+-]?[0-9]+)?'

**constant\_3** = '[0-9]+"."[0-9]\*([DdEe][+-]?[0-9]+)?'

**identifier** = '[a-zA-Z\_]([a-zA-Z\_]|[0-9])\*'

**input** ( *data\_* )

Return type None

**keywords** = {'break': 'BREAK', 'clear': 'CLEAR', 'else': 'ELSE', 'elseif': 'ELSEIF', 'end': 'END', 'for': 'FOR', 'function': 'FUNCTION', 'global': 'GLOBAL', 'if': 'IF', 'return': 'RETURN', 'while': 'WHILE'}

**literals** = ['~', ',', '!', ':', '=', '(', ')', '[', ']', '&', '-', '+', '\*', '/', '\\', '>', '<', '|']

**states** = (('string', 'exclusive'),)

**t\_ANY\_COMMENT** ( *token\_* )

[%]||[n]

Return type LexToken

**t\_ARRAY\_DIV** = '\\\./'

**t\_ARRAY\_MUL** = '\\\.\.\*'

**t\_ARRAY\_POW** = '\\\.\.\^'

**t\_ARRAY\_RDIV** = '\\\./.'

**t\_CONSTANT** ( *token\_* )

Return type LexToken

**t\_EQ\_OP** = '=='

**t\_GE\_OP** = '\\\>='

```

t_IDENTIFIER ( token_ )
    Return type LexToken

t_LE_OP = '<='

t_NEWLINE ( token_ )
    n
    Return type LexToken

t_NE_OP = '(~=) | (!=)'

t_STRING_LITERAL ( token_ )
    '[^'\n]*'
    Return type LexToken

t_TRANSPOSE ( token_ )
    Return type LexToken

t_error ( token_ )
    Error handler lexer
    Return type None

t_ignore_WHITESPACE = '\\s+'

t_string_TCOMMENT = '[^\\n]+'

t_string_error ( token_ )
    Error handler lexer for string state
    Return type None

t_string_ignore = ""

token ( )
    Return type LexToken

tokens = ('IDENTIFIER', 'CONSTANT', 'STRING_LITERAL', 'LE_OP', 'GE_OP', 'EQ_OP',
'NE_OP', 'ARRAY_MUL', 'ARRAY_POW', 'ARRAY_DIV', 'ARRAY_RDIV', 'TRANSPOSE',
'NEWLINE', 'COMMENT', 'TCOMMENT', 'FOR', 'WHILE', 'BREAK', 'IF', 'ELSE', 'ELSEIF',
'END', 'FUNCTION', 'RETURN', 'GLOBAL', 'CLEAR')

transpose = ""|\\\\"

transpose_1 = ""

transpose_2 = "\\\\"

class cmp.grammar.Parser ( lexer=<class 'cmp.grammar.lexer.Lexer'>, yacc_debug=False )
    Bases: object
    Executive parser object. Containing primary reduce rules. This class build AST

errors ( )
    Return type Iterator[str]

```

```

handlers = {'!=': <class 'cmp.ast.equality.NegativeEqualityNode'>, '&': <class 'cmp.ast.logic.AndNode'>, '*': <class 'cmp.ast.multiplicative.MultiplyNode'>, '+': <class 'cmp.ast.additive.PlusNode'>, '-': <class 'cmp.ast.additive.MinusNode'>, '.*': <class 'cmp.ast.multiplicative.ArrayMulNode'>, './': <class 'cmp.ast.multiplicative.ArrayDivNode'>, './.': <class 'cmp.ast.multiplicative.ArrayRDivNode'>, '^': <class 'cmp.ast.multiplicative.ArrayPowerNode'>, '/': <class 'cmp.ast.multiplicative.DivideNode'>, '<': <class 'cmp.ast.relational.LowerRelationalNode'>, '<=': <class 'cmp.ast.relational.LowerEqualRelationalNode'>, '==': <class 'cmp.ast.equality.PositiveEqualityNode'>, '>': <class 'cmp.ast.relational.GreaterRelationalNode'>, '>=': <class 'cmp.ast.relational.GreaterEqualRelationalNode'>, '^': <class 'cmp.ast.multiplicative.PowerNode'>, '|': <class 'cmp.ast.logic.OrNode'>}

```

**property** `has_errors`

Return type `bool`

**p\_additive\_expression** (*p*)

*additive\_expression* : *multiplicative\_expression*

*additive\_expression* '+' *multiplicative\_expression*

*additive\_expression* '-' *multiplicative\_expression*

Return type `None`

**p\_and\_expression** (*p*)

*and\_expression* : *equality\_expression*

*and\_expression* '&' *equality\_expression*

Return type `None`

**p\_array\_element** (*p*)

*array\_element* : *expression*

*expression\_statement*

Return type `None`

**p\_array\_expression** (*p*)

*array\_expression* : IDENTIFIER '(' *index\_expression\_list* ')'

Return type `None`

**p\_array\_list** (*p*)

*array\_list* : *array\_element*

*array\_list* *array\_element*

Return type `None`

**p\_assignment\_expression** (*p*)

*assignment\_expression* : postfix\_expression '=' *expression*

Return type `None`

**p\_assignment\_statement** (*p*)

*assignment\_statement* : *assignment\_expression* *eastmt*

Return type `None`

**p\_clear\_statement** (*p*)

*clear\_statement* : CLEAR *identifier\_list* *eastmt*

Return type `None`

**p\_comment\_statement** (*p*)

*comment\_statement* : COMMENT TCOMMENT

**Return type** None

**p\_constant\_expression** ( *p* )  
 constant\_expression : CONSTANT

**Return type** None

**p\_elseif\_clause** ( *p* )  
 elseif\_clause : *ELSEIF expression statement\_list*  
 elseif\_clause ELSEIF expression statement\_list

**Return type** None

**p\_eostmt** ( *p* )  
 eostmt : *','*  
           *','*  
           NEWLINE

**Return type** None

**p\_equality\_expression** ( *p* )  
 equality\_expression : *relational\_expression*  
 equality\_expression EQ\_OP relational\_expression  
 equality\_expression NE\_OP relational\_expression

**Return type** None

**p\_error** ( *p* )  
**Return type** None

**p\_expression** ( *p* )  
 expression : *or\_expression*  
 expression *':'* or\_expression

**Return type** None

**p\_expression\_statement** ( *p* )  
 expression\_statement : *eostmt*  
 expression eostmt

**Return type** None

**p\_func\_declare** ( *p* )  
 func\_declare : *func\_declare\_lhs*  
 func\_return\_list '=' func\_declare\_lhs  
 func\_declare\_invoke\_error

**Return type** None

**p\_func\_declare\_invoke\_error** ( *p* )  
 func\_declare\_invoke\_error : *func\_return\_list '='*  
 func\_return\_list

**Return type** None

**p\_func\_declare\_lhs** ( *p* )  
 func\_declare\_lhs : *IDENTIFIER*  
 IDENTIFIER *'('* *'*  
 IDENTIFIER *'('* func\_identifier\_list *'*

**Return type** None

**p\_func\_identifier\_list** ( *p* )

**func\_identifier\_list** : IDENTIFIER  
 func\_identifier\_list ',' IDENTIFIER

**Return type** None

**p\_func\_return\_list** ( *p* )

**func\_return\_list** : IDENTIFIER  
 '[' func\_identifier\_list ']

**Return type** None

**p\_func\_statement** ( *p* )

**func\_statement** : FUNCTION *func\_declare eostmt statement\_list* END  
 func\_statement\_error

**Return type** None

**p\_func\_statement\_error** ( *p* )

**func\_statement\_error** : FUNCTION error eostmt statement\_list END

**Return type** None

**p\_global\_statement** ( *p* )

**global\_statement** : GLOBAL identifier\_list eostmt

**Return type** None

**p\_identifier\_expression** ( *p* )

**identifier\_expression** : IDENTIFIER

**Return type** None

**p\_identifier\_list** ( *p* )

**identifier\_list** : IDENTIFIER  
 identifier\_list IDENTIFIER

**Return type** None

**p\_index\_expression** ( *p* )

**index\_expression** : ':'  
 expression

**Return type** None

**p\_index\_expression\_list** ( *p* )

**index\_expression\_list** : *index\_expression*  
 index\_expression\_list ',' *index\_expression*

**Return type** None

**p\_iteration\_statement** ( *p* )

**iteration\_statement** : WHILE *expression statement\_list* END *eostmt*  
 FOR IDENTIFIER '=' *expression statement\_list* END *eostmt*  
 FOR '(' IDENTIFIER '=' *expression* ')' *statement\_list* END *eostmt*

**Return type** None

**p\_jump\_statement** ( *p* )

**jump\_statement** : *BREAK eostmt*  
*RETURN eostmt*

**Return type** *None*

**p\_multiplicative\_expression** ( *p* )

**multiplicative\_expression** : *unary\_expression*  
*multiplicative\_expression '\*' unary\_expression*  
*multiplicative\_expression '/' unary\_expression*  
*multiplicative\_expression '^' unary\_expression*  
*multiplicative\_expression ARRAY\_MUL unary\_expression*  
*multiplicative\_expression ARRAY\_DIV unary\_expression*  
*multiplicative\_expression ARRAY\_RDIV unary\_expression*  
*multiplicative\_expression ARRAY\_POW unary\_expression*

**Return type** *None*

**p\_or\_expression** ( *p* )

**or\_expression** : *and\_expression*  
*or\_expression '|' and\_expression*

**Return type** *None*

**p\_postfix\_expression** ( *p* )

**postfix\_expression** : *primary\_expression*  
*array\_expression*  
*postfix\_expression TRANSPOSE*

**Return type** *None*

**p\_primary\_expression** ( *p* )

**primary\_expression** : *identifier\_expression*  
*constant\_expression*  
*string\_literal\_expression*  
*'(' expression ')'*  
*'[' '']'*  
*'[' array\_list '']'*

**Return type** *None*

**p\_relational\_expression** ( *p* )

**relational\_expression** : *additive\_expression*  
*relational\_expression '<' additive\_expression*  
*relational\_expression '>' additive\_expression*  
*relational\_expression LE\_OP additive\_expression*  
*relational\_expression GE\_OP additive\_expression*

**Return type** *None*

**p\_selection\_statement** ( *p* )

**selection\_statement** : *IF expression statement\_list END eostmt*  
*IF expression statement\_list ELSE statement\_list END eostmt*  
*IF expression statement\_list elseif\_clause END eostmt*  
*IF expression statement\_list elseif\_clause ELSE statement\_list END eostmt*  
*selection\_statement\_invoke\_error*  
*selection\_statement\_error*

**Return type** *None*

**p\_selection\_statement\_error** (*p*)  
 selection\_statement\_error : IF error  
 Return type None

**p\_selection\_statement\_invoke\_error** (*p*)  
 selection\_statement\_invoke\_error : IF expression statement\_list  
 Return type None

**p\_statement** (*p*)  
 statement : *global\_statement*  
           clear\_statement  
           assignment\_statement  
           expression\_statement  
           selection\_statement  
           iteration\_statement  
           jump\_statement  
           func\_statement  
           comment\_statement  
 Return type None

**p\_statement\_list** (*p*)  
 statement\_list : *statement*  
                  statement\_list statement  
                  statement\_list\_error  
 Return type None

**p\_statement\_list\_error** (*p*)  
 statement\_list\_error : statement\_list error  
 Return type None

**p\_string\_literal\_expression** (*p*)  
 string\_literal\_expression : STRING\_LITERAL  
 Return type None

**p\_translation\_unit** (*p*)  
 translation\_unit : statement\_list  
 Return type None

**p\_unary\_expression** (*p*)  
 unary\_expression : *postfix\_expression*  
                   unary\_operator postfix\_expression  
 Return type None

**p\_unary\_operator** (*p*)  
 unary\_operator : '+'  
                  '-'  
                  '~'  
 Return type None

**parse** (*text*, *debug\_level=False*)  
 Return type Any

**precedence** = (('right', '-'), ('right', '~'), ('right', '+'))

## Helpers package

*Subpackages*

*Server package*

*Submodules*

*Server setup*

*Tcp client module*

**class** cmp.helpers.server.tcp\_client.**TCPClient** ( \*args, \*\*kwargs )

Bases: argparse.ArgumentParser

Simple TCP client for sending in MATLAB compiler

**execute** ( )

Start TCP client in CLI

**Return type** None

cmp.helpers.server.tcp\_client.**main** ( )

**Return type** None

*Tcp server module*

**class** cmp.helpers.server.tcp\_server.**TCPServer** ( consumer, host=None, port=None )

Bases: cmp.helpers.log.ServerLog

Server for service matlab compiler hostname - IP address or domen name local machine where is server will run port - the port that the server will listen to consumer - function what will give result str -> str

**async execute** ( )

Start TCP server

**Return type** None

*Module contents*

**class** cmp.helpers.server.**TCPClient** ( \*args, \*\*kwargs )

Bases: argparse.ArgumentParser

Simple TCP client for sending in MATLAB compiler

**execute** ( )

Start TCP client in CLI

**Return type** None

**class** cmp.helpers.server.**TCPServer** ( consumer, host=None, port=None )

Bases: cmp.helpers.log.ServerLog

Server for service matlab compiler hostname - IP address or domen name local machine where is server will run port - the port that the server will listen to consumer - function what will give result str -> str

**async execute** ( )

Start TCP server

**Return type** None

*Submodules*

*Camel to snake module*

cmp.helpers.camel\_to\_snake.**camel\_to\_snake** ( name )

Convert camel case to snake case



**Return type** str

*Colors module*

**class** cmp.helpers.colors.colors

Bases: object

**BOLD** = '\x1b[1m'

**ENDC** = '\x1b[0m'

**FAIL** = '\x1b[91m'

**HEADER** = '\x1b[95m'

**OKBLUE** = '\x1b[94m'

**OKCYAN** = '\x1b[96m'

**OKGREEN** = '\x1b[92m'

**UNDERLINE** = '\x1b[4m'

**WARNING** = '\x1b[93m'

*Exceptions module*

**exception** cmp.helpers.exceptions.BadInputError ( text='' )

Bases: Exception

Exceptions of None root

*Logger module*

**class** cmp.helpers.log.LogMixin

Bases: object

Class mixin for produce logger entire class

**property** logger

**Return type** Logger

**class** cmp.helpers.log.ServerLog ( name='server' )

Bases: object

Custom logger for TCP server

*Pattern singleton module*

**class** cmp.helpers.singleton.Singleton ( \*args, \*\*kwargs )

Bases: object

Base class for release pattern Singleton

*Module contents*

**exception** cmp.helpers.BadInputError ( text='' )

Bases: Exception

Exceptions of None root

**class** cmp.helpers.LogMixin

Bases: object

Class mixin for produce logger entire class

**property** logger

**Return type** Logger

**class** `cmp.helpers.Singleton ( *args, **kwargs )`

Bases: `object`

Base class for release pattern Singleton

`cmp.helpers.camel_to_snake ( name )`

Convert camel case to snake case

**Return type** `str`

**class** `cmp.helpers.colors`

Bases: `object`

**BOLD** = `'\x1b[1m'`

**ENDC** = `'\x1b[0m'`

**FAIL** = `'\x1b[91m'`

**HEADER** = `'\x1b[95m'`

**OKBLUE** = `'\x1b[94m'`

**OKCYAN** = `'\x1b[96m'`

**OKGREEN** = `'\x1b[92m'`

**UNDERLINE** = `'\x1b[4m'`

**WARNING** = `'\x1b[93m'`

## Traverse package

### Submodules

#### Traverse AST module

**class** `cmp.traverse.traverse_ast.Visitor ( numpy_mode=False, filename=None )`

Bases: `object`

Walk through the generated AST and translating it to Python code in the specified file Recursive walking in tree invoke methods `_visit` + `NameNode` `numpy_mode` - if activated, ordinary operation will be translated

in numpy operations

`filename` - name of file where will be written python code `_depth` - inner variable for count ident

`_stack` - store for return value of function

**keywords** = `{'diag': 'np.diag', 'eye': 'np.eye', 'ones': 'np.ones', 'rand': 'np.random', 'zeros': 'np.zeros'}`

**property** `py_tab`

**Return type** `str`

**tabulate\_expr ( expr )**

Shift expression on python tab and filtering entered expression on empty symbol or symbol

**Return type** `str`

**traverse\_ast ( root )**

Main function for traverse and printing translated code. :param: `root`: FileAST - root of AST :rtype: `Optional[str]` :return: python code or None if a file was specified when creating an object

*Module contents*

### **1.1.2 Module contents**

- `genindex`
- `modindex`
- `search`



## C

`cmp`, 31

- `cmp.ast`, 7
  - `cmp.ast.additive`, 1
  - `cmp.ast.array`, 1
  - `cmp.ast.assignment`, 2
  - `cmp.ast.comment`, 2
  - `cmp.ast.conditional_statement`, 2
  - `cmp.ast.define_clear`, 3
  - `cmp.ast.define_global`, 3
  - `cmp.ast.equality`, 3
  - `cmp.ast.error`, 3
  - `cmp.ast.finite_unit`, 3
  - `cmp.ast.function`, 4
  - `cmp.ast.iterations`, 4
  - `cmp.ast.jump_stmt`, 4
  - `cmp.ast.lhs_rhs_node`, 4
  - `cmp.ast.logic`, 5
  - `cmp.ast.multiplicative`, 5
  - `cmp.ast.node`, 6
  - `cmp.ast.relational`, 6
  - `cmp.ast.root`, 6
  - `cmp.ast.sparse`, 7
  - `cmp.ast.transpose`, 7
  - `cmp.ast.unary_expression`, 7
- `cmp.cli`, 14
  - `cmp.cli.cli`, 12
  - `cmp.cli.handlers_check`, 13
- `cmp.grammar`, 21
  - `cmp.grammar.lexer`, 14
  - `cmp.grammar.parser`, 15
- `cmp.helpers`, 29
  - `cmp.helpers.camel_to_snake`, 28
  - `cmp.helpers.colors`, 29
  - `cmp.helpers.exceptions`, 29
  - `cmp.helpers.log`, 29
  - `cmp.helpers.server`, 28
    - `cmp.helpers.server.tcp_client`, 28
    - `cmp.helpers.server.tcp_server`, 28
  - `cmp.helpers.singleton`, 29
- `cmp.traverse`, 31
  - `cmp.traverse.traverse_ast`, 30



## A

AbstractHandler (class in cmp.cli.handler-  
s\_check), 13  
alt\_branch (cmp.ast.conditional\_statement.-  
ManyBranchConditionalNode  
attribute), 2  
alt\_branch (cmp.ast.conditional\_statement.Two-  
BranchConditionalNode attribute), 2  
alt\_branch (cmp.ast.ManyBranchCondition-  
alNode attribute), 10  
alt\_branch (cmp.ast.TwoBranchCondition-  
alNode attribute), 12  
alt\_chain (cmp.ast.conditional\_statement.-  
ManyBranchConditionalNode  
attribute), 2  
alt\_chain (cmp.ast.ManyBranchCondition-  
alNode attribute), 10  
AndNode (class in cmp.ast), 7  
AndNode (class in cmp.ast.logic), 5  
ArrayDivNode (class in cmp.ast), 7  
ArrayDivNode (class in cmp.ast.multiplicative),  
5  
ArrayMulNode (class in cmp.ast), 7  
ArrayMulNode (class in cmp.ast.multiplica-  
tive), 5  
ArrayNode (class in cmp.ast), 7  
ArrayNode (class in cmp.ast.array), 1  
ArrayPowerNode (class in cmp.ast), 8  
ArrayPowerNode (class in cmp.ast.multiplica-  
tive), 5  
ArrayRDivNode (class in cmp.ast), 8  
ArrayRDivNode (class in cmp.ast.multiplica-  
tive), 5  
ArrayVectorNode (class in cmp.ast), 8  
ArrayVectorNode (class in cmp.ast.array), 1  
AssignmentNode (class in cmp.ast), 8  
AssignmentNode (class in cmp.ast.assignment),  
2

## B

BadInputError, 29, 29

body (cmp.ast.ForLoopNode attribute), 9  
body (cmp.ast.function.FunctionNode  
attribute), 4  
body (cmp.ast.FunctionNode attribute), 9  
body (cmp.ast.iterations.ForLoopNode  
attribute), 4  
body (cmp.ast.iterations.WhileLoopNode  
attribute), 4  
body (cmp.ast.WhileLoopNode attribute), 12  
BOLD (cmp.helpers.colors attribute), 30  
BOLD (cmp.helpers.colors.colors attribute), 29  
BreakNode (class in cmp.ast), 8  
BreakNode (class in cmp.ast.jump\_stmt), 4

## C

camel\_to\_snake() (in module cmp.helpers), 30  
camel\_to\_snake() (in module cmp.helpers.-  
camel\_to\_snake), 28  
CheckOutputFile (class in cmp.cli.handler-  
s\_check), 13  
CheckServerKey (class in cmp.cli.handler-  
s\_check), 13  
CheckStringKey (class in cmp.cli.handler-  
s\_check), 13  
ClearNode (class in cmp.ast), 8  
ClearNode (class in cmp.ast.define\_clear), 3  
cmp  
    module, 31  
cmp.ast  
    module, 7  
cmp.ast.additive  
    module, 1  
cmp.ast.array  
    module, 1  
cmp.ast.assignment  
    module, 2  
cmp.ast.comment  
    module, 2  
cmp.ast.conditional\_statement  
    module, 2  
cmp.ast.define\_clear  
    module, 3

cmp.ast.define\_global  
     module, 3  
 cmp.ast.equality  
     module, 3  
 cmp.ast.error  
     module, 3  
 cmp.ast.finite\_unit  
     module, 3  
 cmp.ast.function  
     module, 4  
 cmp.ast.iterations  
     module, 4  
 cmp.ast.jump\_stmt  
     module, 4  
 cmp.ast.lhs\_rhs\_node  
     module, 4  
 cmp.ast.logic  
     module, 5  
 cmp.ast.multiplicative  
     module, 5  
 cmp.ast.node  
     module, 6  
 cmp.ast.relational  
     module, 6  
 cmp.ast.root  
     module, 6  
 cmp.ast.sparse  
     module, 7  
 cmp.ast.transpose  
     module, 7  
 cmp.ast.unary\_expression  
     module, 7  
 cmp.cli  
     module, 14  
 cmp.cli.cli  
     module, 12  
 cmp.cli.handlers\_check  
     module, 13  
 cmp.grammar  
     module, 21  
 cmp.grammar.lexer  
     module, 14  
 cmp.grammar.parser  
     module, 15  
 cmp.helpers  
     module, 29  
 cmp.helpers.camel\_to\_snake  
     module, 28  
 cmp.helpers.colors  
     module, 29  
 cmp.helpers.exceptions  
     module, 29  
 cmp.helpers.log  
     module, 29  
 cmp.helpers.server  
     module, 28  
 cmp.helpers.server.tcp\_client  
     module, 28

cmp.helpers.server.tcp\_server  
     module, 28  
 cmp.helpers.singleton  
     module, 29  
 cmp.traverse  
     module, 31  
 cmp.traverse.traverse\_ast  
     module, 30  
 colors (class in cmp.helpers), 30  
 colors (class in cmp.helpers.colors), 29  
 Command (class in cmp.cli.cli), 12  
 comment (cmp.ast.comment.CommentNode  
     attribute), 2  
 comment (cmp.ast.CommentNode attribute), 8  
 CommentNode (class in cmp.ast), 8  
 CommentNode (class in cmp.ast.comment), 2  
 ConditionalNode (class in cmp.ast.condition-  
     al\_statement), 2  
 const (cmp.ast.ConstantNode attribute), 8  
 const (cmp.ast.finite\_unit.ConstantNode  
     attribute), 3  
 constant (cmp.grammar.Lexer attribute), 21  
 constant (cmp.grammar.lexer.Lexer attribute),  
     14  
 constant\_1 (cmp.grammar.Lexer attribute), 21  
 constant\_1 (cmp.grammar.lexer.Lexer  
     attribute), 14  
 constant\_2 (cmp.grammar.Lexer attribute), 21  
 constant\_2 (cmp.grammar.lexer.Lexer  
     attribute), 14  
 constant\_3 (cmp.grammar.Lexer attribute), 21  
 constant\_3 (cmp.grammar.lexer.Lexer  
     attribute), 14  
 ConstantNode (class in cmp.ast), 8  
 ConstantNode (class in cmp.ast.finite\_unit), 3  
 content (cmp.ast.array.ArrayNode attribute), 1  
 content (cmp.ast.array.ArrayVectorNode  
     attribute), 1  
 content (cmp.ast.ArrayNode attribute), 7  
 content (cmp.ast.ArrayVectorNode attribute), 8  
 content (cmp.ast.finite\_unit.SimpleNode  
     attribute), 4  
 content (cmp.ast.SimpleNode attribute), 12

## D

D (cmp.grammar.Lexer attribute), 21  
 D (cmp.grammar.lexer.Lexer attribute), 14  
 declare (cmp.ast.function.FunctionNode  
     attribute), 4  
 declare (cmp.ast.FunctionNode attribute), 9  
 DivideNode (class in cmp.ast), 8  
 DivideNode (class in cmp.ast.multiplicative), 5

## E

E (cmp.grammar.Lexer attribute), 21  
 E (cmp.grammar.lexer.Lexer attribute), 14  
 ElseIfClauseNode (class in cmp.ast), 9



ElseIfClauseNode (class in cmp.ast.condition-  
al\_statement), 2  
 ENDC (cmp.helpers.colors attribute), 30  
 ENDC (cmp.helpers.colors.colors attribute), 29  
 ErrorNode (class in cmp.ast), 9  
 ErrorNode (class in cmp.ast.error), 3  
 errors() (cmp.grammar.Parser method), 22  
 errors() (cmp.grammar.parser.Parser method),  
16  
 execute() (cmp.cli.cli.Command method), 13  
 execute() (cmp.helpers.server.tcp\_client.TCP-  
Client method), 28  
 execute() (cmp.helpers.server.tcp\_serv-  
er.TCPServer method), 28  
 execute() (cmp.helpers.server.TCPClient  
method), 28  
 execute() (cmp.helpers.server.TCPServer  
method), 28  
 expr (cmp.ast.transpose.TransposeNode  
attribute), 7  
 expr (cmp.ast.TransposeNode attribute), 12  
 expr (cmp.ast.unary\_expression.UnaryExpres-  
sionNode attribute), 7  
 expr (cmp.ast.UnaryExpressionNode attribute),  
12  
 express (cmp.ast.ForLoopNode attribute), 9  
 express (cmp.ast.iterations.ForLoopNode  
attribute), 4  
 express (cmp.ast.iterations.WhileLoopNode  
attribute), 4  
 express (cmp.ast.WhileLoopNode attribute), 12

## F

FAIL (cmp.helpers.colors attribute), 30  
 FAIL (cmp.helpers.colors.colors attribute), 29  
 FileAST (class in cmp.ast), 9  
 FileAST (class in cmp.ast.root), 6  
 ForLoopNode (class in cmp.ast), 9  
 ForLoopNode (class in cmp.ast.iterations), 4  
 FunctionDeclareNode (class in cmp.ast), 9  
 FunctionDeclareNode (class in cmp.ast.func-  
tion), 4  
 FunctionNameNode (class in cmp.ast), 9  
 FunctionNameNode (class in cmp.ast.function),  
4  
 FunctionNode (class in cmp.ast), 9  
 FunctionNode (class in cmp.ast.function), 4

## G

GetResult (class in cmp.cli.handlers\_check), 13  
 GlobalNode (class in cmp.ast), 9  
 GlobalNode (class in cmp.ast.define\_global), 3  
 GreaterEqualRelationalNode (class in cmp.ast),  
10  
 GreaterEqualRelationalNode (class in  
cmp.ast.relational), 6  
 GreaterRelationalNode (class in cmp.ast), 10

GreaterRelationalNode (class in cmp.ast.rela-  
tional), 6

## H

handle() (cmp.cli.handlers\_check.Abstrac-  
tHandler method), 13  
 handle() (cmp.cli.handlers\_check.CheckOut-  
putFile method), 13  
 handle() (cmp.cli.handlers\_check.Check-  
ServerKey method), 13  
 handle() (cmp.cli.handlers\_check.Check-  
StringKey method), 13  
 handle() (cmp.cli.handlers\_check.GetResult  
method), 13  
 handle() (cmp.cli.handlers\_check.Handler  
method), 14  
 Handler (class in cmp.cli.handlers\_check), 13  
 handlers (cmp.grammar.Parser attribute), 23  
 handlers (cmp.grammar.parser.Parser  
attribute), 16  
 has\_errors() (cmp.grammar.Parser property), 23  
 has\_errors() (cmp.grammar.parser.Parser prop-  
erty), 16  
 HEADER (cmp.helpers.colors attribute), 30  
 HEADER (cmp.helpers.colors.colors attribute),  
29

## I

id\_list (cmp.ast.ClearNode attribute), 8  
 id\_list (cmp.ast.define\_clear.ClearNode  
attribute), 3  
 id\_list (cmp.ast.define\_global.GlobalNode  
attribute), 3  
 id\_list (cmp.ast.GlobalNode attribute), 10  
 ident (cmp.ast.array.ArrayNode attribute), 1  
 ident (cmp.ast.ArrayNode attribute), 8  
 ident (cmp.ast.finite\_unit.IdentifierNode  
attribute), 3  
 ident (cmp.ast.IdentifierNode attribute), 10  
 identifier (cmp.grammar.Lexer attribute), 21  
 identifier (cmp.grammar.lexer.Lexer attribute),  
14  
 IdentifierNode (class in cmp.ast), 10  
 IdentifierNode (class in cmp.ast.finite\_unit), 3  
 input() (cmp.grammar.Lexer method), 21  
 input() (cmp.grammar.lexer.Lexer method), 14  
 input\_list (cmp.ast.function.FunctionNa-  
meNode attribute), 4  
 input\_list (cmp.ast.FunctionNameNode  
attribute), 9  
 iter (cmp.ast.ForLoopNode attribute), 9  
 iter (cmp.ast.iterations.ForLoopNode attribute),  
4

## K

keywords (cmp.grammar.Lexer attribute), 21  
 keywords (cmp.grammar.lexer.Lexer attribute),

14  
 keywords (cmp.traverse.traverse\_ast.Visitor attribute), 30

## L

L (cmp.grammar.Lexer attribute), 21  
 L (cmp.grammar.lexer.Lexer attribute), 14  
 Lexer (class in cmp.grammar), 21  
 Lexer (class in cmp.grammar.lexer), 14  
 lhs (cmp.ast.additive.MinusNode attribute), 1  
 lhs (cmp.ast.additive.PlusNode attribute), 1  
 lhs (cmp.ast.AndNode attribute), 7  
 lhs (cmp.ast.ArrayDivNode attribute), 7  
 lhs (cmp.ast.ArrayMulNode attribute), 7  
 lhs (cmp.ast.ArrayPowerNode attribute), 8  
 lhs (cmp.ast.ArrayRDivNode attribute), 8  
 lhs (cmp.ast.assignment.AssignmentNode attribute), 2  
 lhs (cmp.ast.AssignmentNode attribute), 8  
 lhs (cmp.ast.DivideNode attribute), 8  
 lhs (cmp.ast.equality.NegativeEqualityNode attribute), 3  
 lhs (cmp.ast.equality.PositiveEqualityNode attribute), 3  
 lhs (cmp.ast.GreaterEqualRelationalNode attribute), 10  
 lhs (cmp.ast.GreaterRelationalNode attribute), 10  
 lhs (cmp.ast.lhs\_rhs\_node.LhsRhsNode attribute), 5  
 lhs (cmp.ast.logic.AndNode attribute), 5  
 lhs (cmp.ast.logic.OrNode attribute), 5  
 lhs (cmp.ast.LowerEqualRelationalNode attribute), 10  
 lhs (cmp.ast.LowerRelationalNode attribute), 10  
 lhs (cmp.ast.MinusNode attribute), 10  
 lhs (cmp.ast.multiplicative.ArrayDivNode attribute), 5  
 lhs (cmp.ast.multiplicative.ArrayMulNode attribute), 5  
 lhs (cmp.ast.multiplicative.ArrayPowerNode attribute), 5  
 lhs (cmp.ast.multiplicative.ArrayRDivNode attribute), 5  
 lhs (cmp.ast.multiplicative.DivideNode attribute), 5  
 lhs (cmp.ast.multiplicative.MultiplyNode attribute), 6  
 lhs (cmp.ast.multiplicative.PowerNode attribute), 6  
 lhs (cmp.ast.MultiplyNode attribute), 11  
 lhs (cmp.ast.NegativeEqualityNode attribute), 11  
 lhs (cmp.ast.OrNode attribute), 11  
 lhs (cmp.ast.PlusNode attribute), 11  
 lhs (cmp.ast.PositiveEqualityNode attribute), 11  
 lhs (cmp.ast.PowerNode attribute), 11

lhs (cmp.ast.relational.GreaterEqualRelationalNode attribute), 6  
 lhs (cmp.ast.relational.GreaterRelationalNode attribute), 6  
 lhs (cmp.ast.relational.LowerEqualRelationalNode attribute), 6  
 lhs (cmp.ast.relational.LowerRelationalNode attribute), 6  
 lhs (cmp.ast.sparse.SparseNode attribute), 7  
 lhs (cmp.ast.SparseNode attribute), 12  
 LhsRhsNode (class in cmp.ast.lhs\_rhs\_node), 4  
 literals (cmp.grammar.Lexer attribute), 21  
 literals (cmp.grammar.lexer.Lexer attribute), 14  
 logger() (cmp.helpers.log.LogMixin property), 29  
 logger() (cmp.helpers.LogMixin property), 29  
 LogMixin (class in cmp.helpers), 29  
 LogMixin (class in cmp.helpers.log), 29  
 LowerEqualRelationalNode (class in cmp.ast), 10  
 LowerEqualRelationalNode (class in cmp.ast.relational), 6  
 LowerRelationalNode (class in cmp.ast), 10  
 LowerRelationalNode (class in cmp.ast.relational), 6

## M

main() (in module cmp.helpers.server.tcp\_client), 28  
 main\_branch (cmp.ast.conditional\_statement.ManyBranchConditionalNode attribute), 2  
 main\_branch (cmp.ast.conditional\_statement.TwoBranchConditionalNode attribute), 3  
 main\_branch (cmp.ast.ManyBranchConditionalNode attribute), 10  
 main\_branch (cmp.ast.TwoBranchConditionalNode attribute), 12  
 main\_stmt (cmp.ast.conditional\_statement.ConditionalNode attribute), 2  
 main\_stmt (cmp.ast.conditional\_statement.ElseIfClauseNode attribute), 2  
 main\_stmt (cmp.ast.conditional\_statement.ManyBranchConditionalNode attribute), 2  
 main\_stmt (cmp.ast.conditional\_statement.SimpleConditionalNode attribute), 2  
 main\_stmt (cmp.ast.conditional\_statement.TwoBranchConditionalNode attribute), 3  
 main\_stmt (cmp.ast.ElseIfClauseNode attribute), 9  
 main\_stmt (cmp.ast.ManyBranchConditionalNode attribute), 10  
 main\_stmt (cmp.ast.SimpleConditionalNode attribute), 12

main\_stmt (cmp.ast.TwoBranchConditionalNode attribute), 12  
 ManyBranchConditionalNode (class in cmp.ast), 10  
 ManyBranchConditionalNode (class in cmp.ast.conditional\_statement), 2  
 message (cmp.ast.error.ErrorNode attribute), 3  
 message (cmp.ast.ErrorNode attribute), 9  
 MinusNode (class in cmp.ast), 10  
 MinusNode (class in cmp.ast.additive), 1  
 module  
   cmp, 31  
   cmp.ast, 7  
   cmp.ast.additive, 1  
   cmp.ast.array, 1  
   cmp.ast.assignment, 2  
   cmp.ast.comment, 2  
   cmp.ast.conditional\_statement, 2  
   cmp.ast.define\_clear, 3  
   cmp.ast.define\_global, 3  
   cmp.ast.equality, 3  
   cmp.ast.error, 3  
   cmp.ast.finite\_unit, 3  
   cmp.ast.function, 4  
   cmp.ast.iterations, 4  
   cmp.ast.jump\_stmt, 4  
   cmp.ast.lhs\_rhs\_node, 4  
   cmp.ast.logic, 5  
   cmp.ast.multiplicative, 5  
   cmp.ast.node, 6  
   cmp.ast.relational, 6  
   cmp.ast.root, 6  
   cmp.ast.sparse, 7  
   cmp.ast.transpose, 7  
   cmp.ast.unary\_expression, 7  
   cmp.cli, 14  
   cmp.cli.cli, 12  
   cmp.cli.handlers\_check, 13  
   cmp.grammar, 21  
   cmp.grammar.lexer, 14  
   cmp.grammar.parser, 15  
   cmp.helpers, 29  
   cmp.helpers.camel\_to\_snake, 28  
   cmp.helpers.colors, 29  
   cmp.helpers.exceptions, 29  
   cmp.helpers.log, 29  
   cmp.helpers.server, 28  
   cmp.helpers.server.tcp\_client, 28  
   cmp.helpers.server.tcp\_server, 28  
   cmp.helpers.singleton, 29  
   cmp.traverse, 31  
   cmp.traverse.traverse\_ast, 30  
 MultiplyNode (class in cmp.ast), 11  
 MultiplyNode (class in cmp.ast.multiplicative), 6

## N

name (cmp.ast.function.FunctionDeclareNode attribute), 4  
 name (cmp.ast.function.FunctionNameNode attribute), 4  
 name (cmp.ast.FunctionDeclareNode attribute), 9  
 name (cmp.ast.FunctionNameNode attribute), 9  
 NegativeEqualityNode (class in cmp.ast), 11  
 NegativeEqualityNode (class in cmp.ast.equality), 3  
 network\_execute() (cmp.cli.handlers\_check.CheckServerKey static method), 13  
 Node (class in cmp.ast), 11  
 Node (class in cmp.ast.node), 6

## O

OKBLUE (cmp.helpers.colors attribute), 30  
 OKBLUE (cmp.helpers.colors.colors attribute), 29  
 OKCYAN (cmp.helpers.colors attribute), 30  
 OKCYAN (cmp.helpers.colors.colors attribute), 29  
 OKGREEN (cmp.helpers.colors attribute), 30  
 OKGREEN (cmp.helpers.colors.colors attribute), 29  
 OrNode (class in cmp.ast), 11  
 OrNode (class in cmp.ast.logic), 5

## P

p\_additive\_expression() (cmp.grammar.Parser method), 23  
 p\_additive\_expression() (cmp.grammar.parser.Parser method), 16  
 p\_and\_expression() (cmp.grammar.Parser method), 23  
 p\_and\_expression() (cmp.grammar.parser.Parser method), 16  
 p\_array\_element() (cmp.grammar.Parser method), 23  
 p\_array\_element() (cmp.grammar.parser.Parser method), 16  
 p\_array\_expression() (cmp.grammar.Parser method), 23  
 p\_array\_expression() (cmp.grammar.parser.Parser method), 16  
 p\_array\_list() (cmp.grammar.Parser method), 23  
 p\_array\_list() (cmp.grammar.parser.Parser method), 16  
 p\_assignment\_expression() (cmp.grammar.Parser method), 23  
 p\_assignment\_expression() (cmp.grammar.parser.Parser method), 16  
 p\_assignment\_statement() (cmp.grammar.Parser method), 23

[p\\_assignment\\_statement\(\)](#) (cmp.grammar.parser.Parser method), [16](#)  
[p\\_clear\\_statement\(\)](#) (cmp.grammar.Parser method), [23](#)  
[p\\_clear\\_statement\(\)](#) (cmp.grammar.parser.Parser method), [16](#)  
[p\\_comment\\_statement\(\)](#) (cmp.grammar.Parser method), [23](#)  
[p\\_comment\\_statement\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_constant\\_expression\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_constant\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_elseif\\_clause\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_elseif\\_clause\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_eostmt\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_eostmt\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_equality\\_expression\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_equality\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_error\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_error\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_expression\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_expression\\_statement\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_expression\\_statement\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_func\\_declare\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_func\\_declare\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_func\\_declare\\_invoke\\_error\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_func\\_declare\\_invoke\\_error\(\)](#) (cmp.grammar.parser.Parser method), [17](#)  
[p\\_func\\_declare\\_lhs\(\)](#) (cmp.grammar.Parser method), [24](#)  
[p\\_func\\_declare\\_lhs\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_func\\_identifier\\_list\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_func\\_identifier\\_list\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_func\\_return\\_list\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_func\\_return\\_list\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_func\\_statement\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_func\\_statement\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_func\\_statement\\_error\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_func\\_statement\\_error\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_global\\_statement\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_global\\_statement\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_identifier\\_expression\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_identifier\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_identifier\\_list\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_identifier\\_list\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_index\\_expression\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_index\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_index\\_expression\\_list\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_index\\_expression\\_list\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_iteration\\_statement\(\)](#) (cmp.grammar.Parser method), [25](#)  
[p\\_iteration\\_statement\(\)](#) (cmp.grammar.parser.Parser method), [18](#)  
[p\\_jump\\_statement\(\)](#) (cmp.grammar.Parser method), [26](#)  
[p\\_jump\\_statement\(\)](#) (cmp.grammar.parser.Parser method), [19](#)  
[p\\_multiplicative\\_expression\(\)](#) (cmp.grammar.Parser method), [26](#)  
[p\\_multiplicative\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [19](#)  
[p\\_or\\_expression\(\)](#) (cmp.grammar.Parser method), [26](#)  
[p\\_or\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [19](#)  
[p\\_postfix\\_expression\(\)](#) (cmp.grammar.Parser method), [26](#)  
[p\\_postfix\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [19](#)  
[p\\_primary\\_expression\(\)](#) (cmp.grammar.Parser method), [26](#)  
[p\\_primary\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [19](#)  
[p\\_relational\\_expression\(\)](#) (cmp.grammar.Parser method), [26](#)  
[p\\_relational\\_expression\(\)](#) (cmp.grammar.parser.Parser method), [19](#)  
[p\\_selection\\_statement\(\)](#) (cmp.grammar.Parser method), [26](#)  
[p\\_selection\\_statement\(\)](#) (cmp.grammar.parser.Parser method), [19](#)

p\_selection\_statement\_error() (cmp.grammar.  
 Parser method), 27  
 p\_selection\_statement\_error() (cmp.grammar.  
 parser.Parser method), 20  
 p\_selection\_statement\_invoke\_error() (cmp.-  
 grammar.Parser method), 27  
 p\_selection\_statement\_invoke\_error() (cmp.-  
 grammar.parser.Parser method), 20  
 p\_statement() (cmp.grammar.Parser method),  
 27  
 p\_statement() (cmp.grammar.parser.Parser  
 method), 20  
 p\_statement\_list() (cmp.grammar.Parser  
 method), 27  
 p\_statement\_list() (cmp.grammar.parser.Parser  
 method), 20  
 p\_statement\_list\_error() (cmp.grammar.Parser  
 method), 27  
 p\_statement\_list\_error() (cmp.grammar.parser.-  
 Parser method), 20  
 p\_string\_literal\_expression() (cmp.grammar.-  
 Parser method), 27  
 p\_string\_literal\_expression() (cmp.grammar.-  
 parser.Parser method), 20  
 p\_translation\_unit() (cmp.grammar.Parser  
 method), 27  
 p\_translation\_unit() (cmp.grammar.parser.-  
 Parser method), 20  
 p\_unary\_expression() (cmp.grammar.Parser  
 method), 27  
 p\_unary\_expression() (cmp.grammar.parser.-  
 Parser method), 20  
 p\_unary\_operator() (cmp.grammar.Parser  
 method), 27  
 p\_unary\_operator() (cmp.grammar.parser.-  
 Parser method), 20  
 parse() (cmp.grammar.Parser method), 27  
 parse() (cmp.grammar.parser.Parser method),  
 21  
 Parser (class in cmp.grammar), 22  
 Parser (class in cmp.grammar.parser), 15  
 PlusNode (class in cmp.ast), 11  
 PlusNode (class in cmp.ast.additive), 1  
 PositiveEqualityNode (class in cmp.ast), 11  
 PositiveEqualityNode (class in cmp.ast.equali-  
 ty), 3  
 PowerNode (class in cmp.ast), 11  
 PowerNode (class in cmp.ast.multiplicative), 6  
 precedence (cmp.grammar.Parser attribute), 28  
 precedence (cmp.grammar.parser.Parser  
 attribute), 21  
 py\_tab() (cmp.traverse.traverse\_ast.Visitor  
 property), 30

## R

return\_list (cmp.ast.function.FunctionDe-  
 declareNode attribute), 4

return\_list (cmp.ast.FunctionDeclareNode  
 attribute), 9  
 ReturnNode (class in cmp.ast), 11  
 ReturnNode (class in cmp.ast.jump\_stmt), 4  
 rhs (cmp.ast.additive.MinusNode attribute), 1  
 rhs (cmp.ast.additive.PlusNode attribute), 1  
 rhs (cmp.ast.AndNode attribute), 7  
 rhs (cmp.ast.ArrayDivNode attribute), 7  
 rhs (cmp.ast.ArrayMulNode attribute), 7  
 rhs (cmp.ast.ArrayPowerNode attribute), 8  
 rhs (cmp.ast.ArrayRDivNode attribute), 8  
 rhs (cmp.ast.assignment.AssignmentNode  
 attribute), 2  
 rhs (cmp.ast.AssignmentNode attribute), 8  
 rhs (cmp.ast.DivideNode attribute), 9  
 rhs (cmp.ast.equality.NegativeEqualityNode  
 attribute), 3  
 rhs (cmp.ast.equality.PositiveEqualityNode  
 attribute), 3  
 rhs (cmp.ast.GreaterEqualRelationalNode  
 attribute), 10  
 rhs (cmp.ast.GreaterRelationalNode attribute),  
 10  
 rhs (cmp.ast.lhs\_rhs\_node.LhsRhsNode  
 attribute), 5  
 rhs (cmp.ast.logic.AndNode attribute), 5  
 rhs (cmp.ast.logic.OrNode attribute), 5  
 rhs (cmp.ast.LowerEqualRelationalNode  
 attribute), 10  
 rhs (cmp.ast.LowerRelationalNode attribute),  
 10  
 rhs (cmp.ast.MinusNode attribute), 10  
 rhs (cmp.ast.multiplicative.ArrayDivNode  
 attribute), 5  
 rhs (cmp.ast.multiplicative.ArrayMulNode  
 attribute), 5  
 rhs (cmp.ast.multiplicative.ArrayPowerNode  
 attribute), 5  
 rhs (cmp.ast.multiplicative.ArrayRDivNode  
 attribute), 5  
 rhs (cmp.ast.multiplicative.DivideNode  
 attribute), 6  
 rhs (cmp.ast.multiplicative.MultiplyNode  
 attribute), 6  
 rhs (cmp.ast.multiplicative.PowerNode  
 attribute), 6  
 rhs (cmp.ast.MultiplyNode attribute), 11  
 rhs (cmp.ast.NegativeEqualityNode attribute),  
 11  
 rhs (cmp.ast.OrNode attribute), 11  
 rhs (cmp.ast.PlusNode attribute), 11  
 rhs (cmp.ast.PositiveEqualityNode attribute), 11  
 rhs (cmp.ast.PowerNode attribute), 11  
 rhs (cmp.ast.relational.GreaterEqualRelation-  
 alNode attribute), 6  
 rhs (cmp.ast.relational.GreaterRelationalNode  
 attribute), 6  
 rhs (cmp.ast.relational.LowerEqualRelation-



alNode attribute), 6  
 rhs (cmp.ast.relational.LowerRelationalNode attribute), 6  
 rhs (cmp.ast.sparse.SparseNode attribute), 7  
 rhs (cmp.ast.SparseNode attribute), 12  
 root (cmp.ast.FileAST attribute), 9  
 root (cmp.ast.root.FileAST attribute), 7

## S

ServerLog (class in cmp.helpers.log), 29  
 set\_next() (cmp.cli.handlers\_check.AbstractHandler method), 13  
 set\_next() (cmp.cli.handlers\_check.Handler method), 14  
 SimpleConditionalNode (class in cmp.ast), 11  
 SimpleConditionalNode (class in cmp.ast.conditional\_statement), 2  
 SimpleNode (class in cmp.ast), 12  
 SimpleNode (class in cmp.ast.finite\_unit), 3  
 Singleton (class in cmp.helpers), 30  
 Singleton (class in cmp.helpers.singleton), 29  
 SparseNode (class in cmp.ast), 12  
 SparseNode (class in cmp.ast.sparse), 7  
 states (cmp.grammar.Lexer attribute), 21  
 states (cmp.grammar.lexer.Lexer attribute), 14  
 stmt\_list (cmp.ast.conditional\_statement.ElseIfClauseNode attribute), 2  
 stmt\_list (cmp.ast.conditional\_statement.SimpleConditionalNode attribute), 2  
 stmt\_list (cmp.ast.ElseIfClauseNode attribute), 9  
 stmt\_list (cmp.ast.SimpleConditionalNode attribute), 12

## T

t\_ANY\_COMMENT() (cmp.grammar.Lexer method), 21  
 t\_ANY\_COMMENT() (cmp.grammar.lexer.Lexer method), 14  
 t\_ARRAY\_DIV (cmp.grammar.Lexer attribute), 21  
 t\_ARRAY\_DIV (cmp.grammar.lexer.Lexer attribute), 14  
 t\_ARRAY\_MUL (cmp.grammar.Lexer attribute), 21  
 t\_ARRAY\_MUL (cmp.grammar.lexer.Lexer attribute), 14  
 t\_ARRAY\_POW (cmp.grammar.Lexer attribute), 21  
 t\_ARRAY\_POW (cmp.grammar.lexer.Lexer attribute), 14  
 t\_ARRAY\_RDIV (cmp.grammar.Lexer attribute), 21  
 t\_ARRAY\_RDIV (cmp.grammar.lexer.Lexer attribute), 14  
 t\_CONSTANT() (cmp.grammar.Lexer method), 21

t\_CONSTANT() (cmp.grammar.lexer.Lexer method), 14  
 t\_EQ\_OP (cmp.grammar.Lexer attribute), 21  
 t\_EQ\_OP (cmp.grammar.lexer.Lexer attribute), 15  
 t\_error() (cmp.grammar.Lexer method), 22  
 t\_error() (cmp.grammar.lexer.Lexer method), 15  
 t\_GE\_OP (cmp.grammar.Lexer attribute), 21  
 t\_GE\_OP (cmp.grammar.lexer.Lexer attribute), 15  
 t\_IDENTIFIER() (cmp.grammar.Lexer method), 22  
 t\_IDENTIFIER() (cmp.grammar.lexer.Lexer method), 15  
 t\_ignore\_WHITESPACE (cmp.grammar.Lexer attribute), 22  
 t\_ignore\_WHITESPACE (cmp.grammar.lexer.Lexer attribute), 15  
 t\_LE\_OP (cmp.grammar.Lexer attribute), 22  
 t\_LE\_OP (cmp.grammar.lexer.Lexer attribute), 15  
 t\_NE\_OP (cmp.grammar.Lexer attribute), 22  
 t\_NE\_OP (cmp.grammar.lexer.Lexer attribute), 15  
 t\_NEWLINE() (cmp.grammar.Lexer method), 22  
 t\_NEWLINE() (cmp.grammar.lexer.Lexer method), 15  
 t\_string\_error() (cmp.grammar.Lexer method), 22  
 t\_string\_error() (cmp.grammar.lexer.Lexer method), 15  
 t\_string\_ignore (cmp.grammar.Lexer attribute), 22  
 t\_string\_ignore (cmp.grammar.lexer.Lexer attribute), 15  
 t\_STRING\_LITERAL() (cmp.grammar.Lexer method), 22  
 t\_STRING\_LITERAL() (cmp.grammar.lexer.Lexer method), 15  
 t\_string\_TCOMMENT (cmp.grammar.Lexer attribute), 22  
 t\_string\_TCOMMENT (cmp.grammar.lexer.Lexer attribute), 15  
 t\_TRANSPOSE() (cmp.grammar.Lexer method), 22  
 t\_TRANSPOSE() (cmp.grammar.lexer.Lexer method), 15  
 tabulate\_expr() (cmp.traverse.traverse\_ast.Visitor method), 30  
 TCPClient (class in cmp.helpers.server), 28  
 TCPClient (class in cmp.helpers.server.tcp\_client), 28  
 TCPServer (class in cmp.helpers.server), 28  
 TCPServer (class in cmp.helpers.server.tcp\_server), 28  
 token() (cmp.grammar.Lexer method), 22  
 token() (cmp.grammar.lexer.Lexer method), 15

tokens (cmp.grammar.Lexer attribute), 22  
 tokens (cmp.grammar.lexer.Lexer attribute), 15  
 transpose (cmp.grammar.Lexer attribute), 22  
 transpose (cmp.grammar.lexer.Lexer attribute), 15  
 transpose\_1 (cmp.grammar.Lexer attribute), 22  
 transpose\_1 (cmp.grammar.lexer.Lexer attribute), 15  
 transpose\_2 (cmp.grammar.Lexer attribute), 22  
 transpose\_2 (cmp.grammar.lexer.Lexer attribute), 15  
 TransposeNode (class in cmp.ast), 12  
 TransposeNode (class in cmp.ast.transpose), 7  
 traverse\_ast() (cmp.traverse.traverse\_ast.Visitor method), 30  
 TwoBranchConditionalNode (class in cmp.ast), 12  
 TwoBranchConditionalNode (class in cmp.ast.-conditional\_statement), 2

## U

unary\_op (cmp.ast.unary\_expression.UnaryExpressionNode attribute), 7  
 unary\_op (cmp.ast.UnaryExpressionNode attribute), 12  
 UnaryExpressionNode (class in cmp.ast), 12  
 UnaryExpressionNode (class in cmp.ast.unary\_expression), 7  
 UNDERLINE (cmp.helpers.colors attribute), 30  
 UNDERLINE (cmp.helpers.colors.colors attribute), 29

## V

Visitor (class in cmp.traverse.traverse\_ast), 30

## W

WARNING (cmp.helpers.colors attribute), 30  
 WARNING (cmp.helpers.colors.colors attribute), 29  
 WhileLoopNode (class in cmp.ast), 12  
 WhileLoopNode (class in cmp.ast.iterations), 4

