

---

# **SharedParser**

***Release 1.0***

**Bogdana Simionica, Stefan Stefanache**

**Dec 19, 2022**



**CONTENTS:**

<b>1</b>	<b>fcd_team</b>	<b>1</b>
1.1	grammar module . . . . .	1
1.2	output module . . . . .	2
1.3	parser module . . . . .	2
<b>2</b>	<b>Indices and tables</b>	<b>5</b>
	<b>Python Module Index</b>	<b>7</b>
	<b>Index</b>	<b>9</b>



## 1.1 grammar module

Implementation of grammar

```
class grammar.GrammarParseSeparators(value)
```

Bases: Enum

An enumeration.

```
class grammar.Grammar(starting_symbol, terminals=NOTHING, nonterminals=NOTHING,  
                      productions=NOTHING)
```

Bases: object

This class implements the basic functionality of working with a grammar, including parsing a grammar specification file written in a reduced form of EBNF.

**starting\_symbol:** str

Initial state symbol (S)

**terminals:** List[str]

List of terminals

**nonterminals:** List[str]

List of nonterminals

**productions:** Dict[str, List[str]]

Mapping of nonterminals to lists of corresponding productions

**static get\_grammar\_from\_file**(*file\_name*)

Reads grammar specification from a file and returns the corresponding Grammar.

**Raises**

**The grammar specification must be written in the specified reduced –**  
EBNF form and the file must exist (see example specification). Otherwise, an exception is thrown.

**Parameters**

**file\_name** (str) – Path to the file containing the grammar definition

**Return type**

*Grammar*

**Returns**

Corresponding Grammar if file exists and contains a valid definition

**get\_productions\_for\_nonterminal**(*nonterminal*)

Returns the productions corresponding to a given nonterminal.

**Parameters**

**nonterminal** (str) – The nonterminal to get productions for

**Return type**

List[str]

**verify\_CFG()**

Checks if the Grammar is context-free.

**Return type**

bool

## 1.2 output module

**class** output.ParserOutput(*grammar*)

Bases: object

**get\_recursive\_table**(*index, parent, right\_sibling, string\_products, products\_stack*)

**get\_string\_products**(*alpha*)

**get\_table**(*alpha*)

**print\_pretty\_table**()

**print\_pretty\_table\_to\_file**(*filename*)

**print\_table**()

**class** output.Row(*info, parent, right\_sibling*)

Bases: object

**return\_data**()

## 1.3 parser module

**class** parser.Parser(*grammar, state=ParsingStates.NORMAL\_STATE, current\_position=1, alpha=NOTHING, beta=NOTHING, w=NOTHING, index\_error=0*)

Bases: object

**advance**()

**algorithm\_descendent\_recursive**(*filename*)

**alpha:** list

**another\_try**()

**back**()

**beta:** list

```
build_string_of_prod()
current_position: int
expand()
grammar: Grammar
index_error: int
momentary_insuccess()
read_sequence_from_file(filename)
state: ParsingStates
success()
w: list

class parser.ParsingStates(value)
    Bases: Enum
    An enumeration.
    BACK_STATE = 'b'
    ERROR_STATE = 'e'
    FINAL_STATE = 'f'
    NORMAL_STATE = 'q'
```





## INDICES AND TABLES

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



## PYTHON MODULE INDEX

### g

grammar, 1

### o

output, 2

### p

parser, 2



## INDEX

### A

`advance()` (*parser.Parser* method), 2  
`algorithm_descendent_recursive()` (*parser.Parser* method), 2  
`alpha` (*parser.Parser* attribute), 2  
`another_try()` (*parser.Parser* method), 2

### B

`back()` (*parser.Parser* method), 2  
`BACK_STATE` (*parser.ParsingStates* attribute), 3  
`beta` (*parser.Parser* attribute), 2  
`build_string_of_prod()` (*parser.Parser* method), 2

### C

`current_position` (*parser.Parser* attribute), 3

### E

`ERROR_STATE` (*parser.ParsingStates* attribute), 3  
`expand()` (*parser.Parser* method), 3

### F

`FINAL_STATE` (*parser.ParsingStates* attribute), 3

### G

`get_grammar_from_file()` (*grammar.Grammar* static method), 1  
`get_productions_for_nonterminal()` (*grammar.Grammar* method), 1  
`get_recursive_table()` (*output.ParserOutput* method), 2  
`get_string_products()` (*output.ParserOutput* method), 2  
`get_table()` (*output.ParserOutput* method), 2  
`grammar`  
    module, 1  
`Grammar` (class in *grammar*), 1  
`grammar` (*parser.Parser* attribute), 3  
`GrammarParseSeparators` (class in *grammar*), 1

### I

`index_error` (*parser.Parser* attribute), 3

### M

module  
    *grammar*, 1  
    *output*, 2  
    *parser*, 2  
`momentary_insuccess()` (*parser.Parser* method), 3

### N

`nonterminals` (*grammar.Grammar* attribute), 1  
`NORMAL_STATE` (*parser.ParsingStates* attribute), 3

### O

`output`  
    module, 2

### P

`parser`  
    module, 2  
`Parser` (class in *parser*), 2  
`ParserOutput` (class in *output*), 2  
`ParsingStates` (class in *parser*), 3  
`print_pretty_table()` (*output.ParserOutput* method), 2  
`print_pretty_table_to_file()` (*output.ParserOutput* method), 2  
`print_table()` (*output.ParserOutput* method), 2  
`productions` (*grammar.Grammar* attribute), 1

### R

`read_sequence_from_file()` (*parser.Parser* method), 3  
`return_data()` (*output.Row* method), 2  
`Row` (class in *output*), 2

### S

`starting_symbol` (*grammar.Grammar* attribute), 1  
`state` (*parser.Parser* attribute), 3  
`success()` (*parser.Parser* method), 3

### T

`terminals` (*grammar.Grammar* attribute), 1

## V

`verify_CFG()` (*grammar.Grammar method*), [2](#)

## W

`w` (*parser.Parser attribute*), [3](#)