

Lark Options	
parser = "earley"	Earley - default
parser = "lalr"	LALR(1)
debug=True	Enable debug prints
lexer="standard"	Revert to simple lexer
ambiguity='explicit'	Return all derivations for Earley
start= " foo "	Set starting rule
cache=True	Enable grammar caching
transformer=...	Apply transformer to tree (for LALR)
propagate_positions	Fill tree instances with line number information
maybe_placeholders	[] returns None when not matched
keep_all_tokens = True	Don't remove unnamed terminals
postlex	Provide a wrapper for the lexer
tree_class	Provide an alternative for Tree
regex=True	Use the regex module

Tree Reference	
tree.data	Rule name
tree.children	Rule matches
tree.meta	Positional information, if enabled
print(tree.pretty())	
tree.iter_subtrees()	Iterate all subtrees
tree.find_data("foo")	Find by rule
tree.find_pred(...)	Find by predicate
tree1 == tree2	

Token Reference	
token.type	Terminal name
token.value	Matched text
token.pos_in_stream	Index in source text
token.line	
token.column	
token.end_line	
token.end_column	
token.end_pos	
len(token)	
Tokens inherit from str, so all string operations are valid (such as token.upper()).	

Grammar Definitions	
rule: ...	Define a rule
TERM: ...	Define a terminal
rule.n: ...	Rule with priority n
TERM.n: ...	Terminal with priority n
// text	Comment
%ignore ...	Ignore terminal in input
%import ...	Import terminal from file
%declare TERM	Declare a terminal without a pattern (used for postlex)
t{p1, p2}: ...	Define template
rule: t{foo, bar}	Use template

Rules consist of values, other rules and terminals.

Terminals only consist of values and other terminals.



By erezsh
cheatography.com/erezsh/

Published 26th May, 2018.
 Last updated 29th September, 2020.
 Page 1 of 2.

Sponsored by **CrosswordCheats.com**
 Learn to solve cryptic crosswords!
<http://crosswordcheats.com>

Grammar Patterns

<code>foo bar</code>	Match sequence
<code>(foo bar)</code>	Group together (for operations)
<code>foo bar</code>	Match one or the other
<code>foo?</code>	Match 0 or 1 instances
<code>[foo bar]</code>	Match 0 or 1 instances
<code>foo*</code>	Match 0 or more instances
<code>foo+</code>	Match 1 or more instances
<code>foo~3</code>	Match exactly 3 instances
<code>foo~3..5</code>	Match between 3 to 5 instances

Terminal Atoms

<code>" str ing "</code>	String to match
<code>" str ing "i</code>	Case-insensitive string
<code>/regexp/</code>	Regular Expression
<code>/re/imslux</code>	Regular Expression with flags
<code>" a".." z "</code>	Literal range

Tree Shaping

<code>rule: " foo " BAR</code>	"foo" will be filtered out
<code>!rule: " foo " BAR</code>	"foo" will be kept
<code>rule: /foo/ BAR</code>	/foo/ will be kept
<code>_TERM</code>	Filter out this terminal
<code>_rule</code>	Always inline this rule
<code>?rule: ...</code>	Inline if matched 1 child
<code>foo bar -> new_name</code>	Rename this derivation

Rules are a branch (node) in the resulting tree, and its children are its matches, in the order of matching.

Terminals (tokens) are always values in the tree, never branches.

Inlining rules means removing their branch and replacing it with their children.

Examples

```
// Define template for comma-separated list
cs_list{item}: item ("," item)*
// Use template to make a list of numbers
number_list: cs_list{ number }
// Example of a terminal for a Python comment
PY_COMMENT: /#[^\n]*/
// Example of a terminal for C comment
C_COMMENT: "/" /.?/s " */"
```

