

ROCKSTAR

Poetic Analysis Web App

Oferece um grau de licença poética sem precedentes na composição e estrutura de programas





ROCKSTAR?

Rockstar é uma linguagem
de programação projetada
para criar programas que
também são *hair metal*
power ballads.

ROCKSTAR FEATURES

01

GRAU POÉTICO

(Tommy = 1337)

Tommy was a big bad brother.

02

cASe-InSensitive

Com exceção de variáveis
de nomes próprios

03

ECMAScript type system

Exceto undefined, usa-se
“mysterious” em vez disso

04

Comments

Isso é rock; cabe ao público
encontrar o significado.

IMPLEMENTAÇÃO, Python Lex-Yacc

A decorative graphic featuring a central sunburst with the number '01' inside. Four lightning bolts are positioned around the sunburst: one at the top-left, one at the top-right, one at the bottom-left, and one at the bottom-right.

01

LEX.py

Usado para quebrar o código em tokens especificados por regras de expressão regular

A decorative graphic featuring a central sunburst with the number '02' inside. Four lightning bolts are positioned around the sunburst: one at the top-left, one at the top-right, one at the bottom-left, and one at the bottom-right.

02

YACC.py

Reconhece a sintaxe especificada na forma de uma gramática livre de contexto

Yacc, "Yet Another Compiler Compiler"

Cada regra gramatical é definida por uma função Python em que a docstring para essa função contém a especificação gramatical livre de contexto apropriada. As declarações que compõem o corpo da função implementam as ações semânticas da regra. Cada função aceita um único argumento p que é uma sequência contendo os valores de cada símbolo gramatical na regra correspondente.

```
def p_expression_plus(p):  
    'expression : expression PLUS term'  
    #   ^           ^       ^   ^  
    # p[0]         p[1]   p[2] p[3]  
  
    p[0] = p[1] + p[3]
```


PARSING

Por exemplo, se você quiser analisar expressões aritméticas simples, você pode primeiro escrever uma especificação gramatical livre de contexto.

```
expression : expression + term
           | expression - term
           | term

term        : term * factor
           | term / factor
           | factor

factor      : NUMBER
           | ( expression )
```


PARSING

Sempre que uma regra gramatical específica é reconhecida, a ação descreve o que fazer.

Grammar

```
-----  
expression0 : expression1 + term  
             | expression1 - term  
             | term
```

```
term0       : term1 * factor  
             | term1 / factor  
             | factor
```

```
factor      : NUMBER  
             | ( expression )
```

Action

```
-----  
expression0.val = expression1.val + term.val  
expression0.val = expression1.val - term.val  
expression0.val = term.val
```

```
term0.val = term1.val * factor.val  
term0.val = term1.val / factor.val  
term0.val = factor.val
```

```
factor.val = int(NUMBER.lexval)  
factor.val = expression.val
```


BRIGADÔ!



PERGUNTAS?

github.com/Durfan/ufsj-rockstar

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**

Please keep this slide for attribution