# Aula Teórica 4 (guião)

Semana de 13 a 17 de Março de 2023

José Carlos Ramalho

Sinopsis:

- Analisadores Léxicos;
- A ferramenta `ply`.

## Analisadores Léxicos

Símbolos terminais:

1. Sinais: são constituídos por um caráter;
2. Palavras reservadas: strings constantes;
3. Terminais variáveis: identificadores, inteiros, etc.

### Exercício: a linguagem das listas

Exemplos:

```
[]
[2]
[ 2, 4, 5]
[ 2, 4, [ 5, 7, 9], 6]
```

Quais são os terminais:

```
T = {'[', ']', num}
```

**Implementação com o `re`**

```
import re

texto_input = """
[]
[2]
erro
[ 2, 4, 5]
???
[ 2, 4, [ 5, 7, 9], 6]
]23,-98[]
"""
```

```
for num,ap,fp,sep,skip,unk in re.findall(r'''
         ([+-]?\d+)
       |  (\[)
       |  (\])
       |  (,)
       |  (\s+|\t+) # skip
       |  (.)       # situação de erro
    ''', texto_input, re.I|re.X):

    if   num      : print('NUM = ', num)
    elif ap       : print('AP')
    elif fp       : print('FP')
    elif sep      : print('SEP')
    elif skip     : pass
    elif unk      : print('ERRO: ', unk)
```

**Implementação com o `ply`**

```
# PL2023: 14 de Março, jcr
# ------------------------------------------------------------
# listas.py
#
# tokenizer for a simple list language
# ------------------------------------------------------------

import ply.lex as lex

# List of token names.   This is always required
tokens = (
   'NUM',
   'AP',
   'FP',
   'SEP'
)

# Regular expression rules for simple tokens
t_AP    = r'\['
t_FP   = r'\]'
t_SEP   = r','

# A regular expression rule with some action code
def t_NUM(t):
    r'[+\-]\d+'
    t.value = int(t.value)
    return t

# Define a rule so we can track line numbers
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)
```

```
# A string containing ignored characters (spaces and tabs)
t_ignore  = ' \t'

# Error handling rule
def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)

# Build the lexer
lexer = lex.lex()

# Test it out
data = '''
[]
[2]
erro
[ 2, 4, 5]
???
[ 2, 4, [ 5, 7, 9], 6]
]23,-98[]
'''

# Give the lexer some input
lexer.input(data)

for tok in lexer:
    print(tok)
    # print(tok.type, tok.value, tok.lineno, tok.lexpos)
```

## TPC2: Somador on/off

```
# PL2023: 14 de Março, jcr
# -----------------------------------------------------------
# somador_on_off.py
# -----------------------------------------------------------

import ply.lex as lex

# List of token names.   This is always required
tokens = (
   'NUM',
   'AP',
   'FP',
   'SEP'
)

# Regular expression rules for simple tokens
t_AP    = r'\['
t_FP   = r'\]'
t_SEP   = r','
```

```
# A regular expression rule with some action code
def t_NUM(t):
    r'[+\-]\d+'
    t.value = int(t.value)
    return t

# Define a rule so we can track line numbers
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

# A string containing ignored characters (spaces and tabs)
t_ignore  = ' \t'

# Error handling rule
def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    t.lexer.skip(1)

# Build the lexer
lexer = lex.lex()

# Test it out
data = '''
[]
[2]
erro
[ 2, 4, 5]
???
[ 2, 4, [ 5, 7, 9], 6]
]23,-98[]
'''

# Give the lexer some input
lexer.input(data)

for tok in lexer:
    print(tok)
    print(tok.type, tok.value, tok.lineno, tok.lexpos)
```

### Com condições de contexto

```
# ------------------------------------------------------------
# somador-ply.py
#
# somador de números
# tokens: on, off, = e \d+
# ------------------------------------------------------------
import ply.lex as lex
import sys
```

```
# Declare the state
states = (
   ('off','exclusive'),
)

# List of token names.   This is always required
tokens = (
    'ON',
    'OFF',
    'PRINT',
    'NUMBER'
)

# Regular expression rules for tokens in initial state
def t_OFF(t):
    r'[oO][fF][fF]'
    t.lexer.begin('off')

def t_NUMBER(t):
    r'\d+'
    t.lexer.soma = t.lexer.soma + int(t.value)

def t_ANY_PRINT(t):
    r'='
    print("soma = ", t.lexer.soma)

# Define a rule so we can track line numbers
def t_ANY_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

# A string containing ignored characters (spaces and tabs)
t_ignore  = ' \t'

# Error handling rule: remaining chars
def t_ANY_error(t):
    t.lexer.skip(1)

# ------------------------------------------------------------
# Regular expression rules for tokens in conditional state
def t_off_ON(t):
    r'[Oo][Nn]'
    t.lexer.begin('INITIAL')

def t_off_NUMBER(t):
    r'\d+'
    pass

def t_off_OFF(t):
    r'[oO][fF][fF]'
    pass
```

```
# Build the lexer
lexer = lex.lex()

# My state
lexer.soma = 0

lexer.begin('INITIAL')

# Reading input
for linha in sys.stdin:
    lexer.input(linha)
    for tok in lexer:
        print(tok)
```

## Gramáticas

Uma gramática G é um tuplo (N, T, S, P ), onde:

- N - conjunto de símbolos não terminais;
- T - conjunto de símbolos terminais;
- S - símbolo inicial;
- P - as produções.

### Gramática Independente de Contexto

Uma gramática é uma GIC se cada produção é da forma: X→α, onde X∈N e α∈(N ∪T)∗

### Gramáticas Regulares

Uma gramática é regular

- linear à direita: – X→uY,ondeX,Y ∈V eu∈A∗,ou – X→u,ondeX∈V eu∈A∗
- linear à esquerda: – X→Yu,ondeX,Y ∈V eu∈A∗,ou – X→u,ondeX∈V eu∈A∗