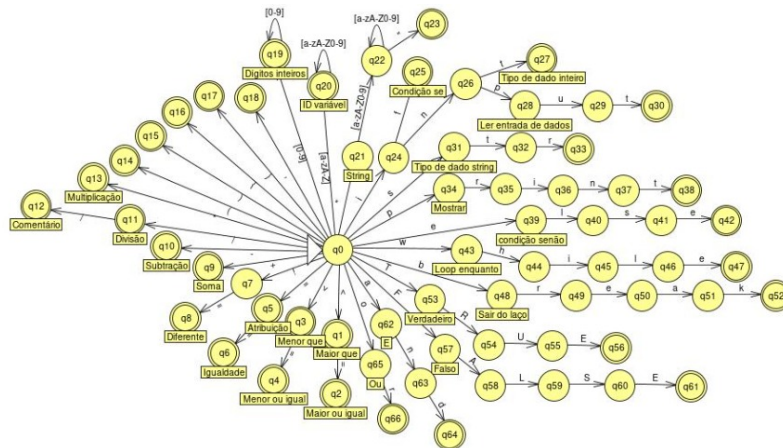


Indentificador de Tokens

Introdução

Para resolver este problema, usamos a abordagem de filtrar cada palavra inserida e indentificar os tokens, para isso usamos expressões regulares.

Autômato que marque a Linguagem



Primeiros Passos

Para esta primeira etapa vamos definir as nossas palavras reservadas, operadores relacionais, aritmeticos, atribuição e os delimitadores.

```

import re

# Palavras Reservadas
PR = [
    ('if', 'if'), ('input', 'input'), ('int', 'int'), ('str', 'str'),
    ('print', 'print'), ('else', 'else'), ('while', 'while'), ('break', 'break'),
    ('TRUE', 'True'), ('FALSE', 'False'), ('and', 'and'), ('or', 'or')
]

# Operadores Relacionais
OR = [
    ('>', '>'),
    ('>=', '>='),
    ('<', '<'),
    ('<=', '<='),
    ('==', '=='),
    ('!=', '!=')
]

OAr = [
    ('+', '+'),
    ('-', '-'),
    ('/', '/'),
    ('*', '*')
]

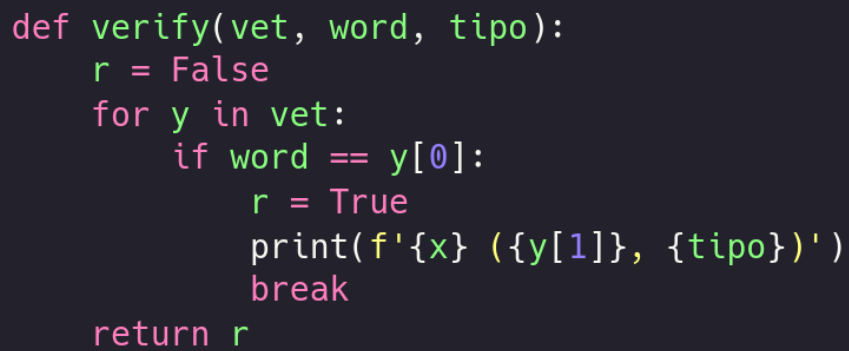
# Operador de Atribuição
OAt = [('=', '=')]

# Delimitadores
Del = [
    ('(', '('),
    (')', ')'),
    ('{', '{'),
    ('}', '}'),
    (',', ','),
    ('"', '"')
]

```

Indentificar lexemas validos e mostrar token

Nesta etapa foi criado esta função que tem objetivo de indentificar os nossos lexemas validos se o lexema é valido mostra o token caso o contrario não é exibido.



```
def verify(vet, word, tipo):  
    r = False  
    for y in vet:  
        if word == y[0]:  
            r = True  
            print(f'{x} ({y[1]}, {tipo})')  
            break  
    return r
```

Teste

Nesta ultima etapa executamos um exemplo de teste para reconhecer uma palavra que esta atribuida a variavel script que em seguida passa por um processo de split para pegar cada item da palavra, depo identificado cada lexema e exibido os tokens, posteriormente exibida os ids das variaveis e por fim o tratamento das strings.

```
script = 'if a == " baa + c " x = y 1a 10 b1 c4c 9.8'
teste = script.split()

for x in teste:

    next = False # Pular para a próxima palavra

    if verify(PR, x, 'PR'):
        continue

    if verify(OR, x, 'OR'):
        continue

    if verify(0Ar, x, '0Ar'):
        continue

    if verify(0At, x, '0At'):
        continue

    if verify(Del, x, 'Del'):
        continue

    # Ids de variáveis
    Id = re.findall(r'[a-zA-Z]+[a-zA-Z0-9]*', script)

    for y in Id:
        if x == y:
            next = True
            print(f'{x} ([a-zA-Z]+[a-zA-Z0-9]*, Id)')
            break

    if next:
        continue

    # NI - Números Inteiros
    NI = re.findall(r'[0-9]+', script)
    for y in NI:
        if x == y:
            print(f'{x} ([0-9]+, NI)')
            break
    else:
        print(f'ERRO: {x} não é um lexema válido!')
```

Conclusão

O trabalho foi utilizado uma abordagem bem simples e o automato poderia ser otimizado para trabalhar em um cenário onde realmente objetivo fosse criar um analisador lexico onde o motor disso tudo seria uma maquina de turing e também usariamos orientação objetos para ter o processo de emcapsulamento onde teriamos alguns objetos definidos no construtor como cabeça, fita, numeros de linhas e palavras reservadas e para metodos algumas coisas como atualizador de estado, estado atual, avanacar estado e outros metodos que o problema fosse exigir, metodos para cada estado a maquina de turing e claro alguns tratamentos de erros que viria deste da leitura do arquivo de texto ao fim do classe.