RELATÓRIO TP1

João Furtado|nº21125
Rúben Gomes|nº21118

20/11/2021

—

Processamento de Linguagens

—

Alberto Simões

INDÍCE

RELATÓRIO TP1 1

INTRODUÇÃO 3

PRIMEIROS PASSOS 4

TOKENS E STATES 4

FUNÇÕES 4

PRIMEIROS PASSOS 5

IMPLEMENTAÇÃO FINAL 6

TOKENS 6

FUNÇÕES 6

IMPLEMENTAÇÃO FINAL 7

IMPLEMENTAÇÃO FINAL 8

CONCLUSÃO 9

INTRODUÇÃO

Este projeto foi realizado a respeito da proposta de trabalho do professor Alberto Simões, docente da cadeira de Processamento de Linguagens na Licenciatura de Engenharia de Sistemas Informáticos do Instituto Politécnico do Cávado e do Ave.
O trabalho proposto permite-nos escolher entre 3 opções de formatos textuais para transformar em ficheiros do formato Html e LATEX. Por conseguinte, tomamos a decisão de escolher o enunciado B que nos pede para transformar ficheiros no formato CSV nos formatos referidos anteriormente.
O relatório aqui apresentado irá ter como objetivo explicar o passo a passo da realização deste projeto.

PRIMEIROS PASSOS

No início do projeto, começamos por criar a classe chamada “CSVHTML”.

```
class CSVHTML:
```

TOKENS E STATES

Em seguida , criamos os seguintes “tokens” e “states” :

```
tokens = ("COUNTRY", "CAPITAL", "CURRENCY", "LANGUAGE")
states = (("capital", "exclusive"),
          ("moeda", "exclusive"),
          ("lingua", "exclusive"))
```

Estes “tokens” definiam a categoria das colunas do ficheiro CSV enviado pelo professor. Os “states” servem para definir um novo estado de lexing e são do tipo “exclusive”.

FUNÇÕES

Definimos a função t_ANY_error que funciona como uma regra para lidar com os erros.

```
def t_ANY_error(self, t):
    print(f"Unexpected token {t.value[:11]}")
    exit(1)
```

PRIMEIROS PASSOS

As seguintes funções criadas definem os “tokens” que escrevemos anteriormente. Mas também, os “tokens” são guardados numa lista e iniciamos o próximo “state” da variável a ser lida.

```
def t_COUNTRY(self, t):
    r"[^,]+"
    t.type = "COUNTRY"
    self.list1.append(t)
    t.lexer.begin("capital")
    return t

def t_capital_CAPITAL(self, t):
    r"[^,]+"
    t.type = "CAPITAL"
    self.list1.append(t)
    t.lexer.begin("moeda")
    return t

def t_moeda_CURRENCY(self, t):
    r"[^,]+"
    t.type = "CURRENCY"
    self.list1.append(t)
    t.lexer.begin("lingua")
    return t

def t_lingua_LANGUAGE(self, t):
    r"[^\n]+"
    t.type = "LANGUAGE"
    self.list1.append(t)
    t.lexer.begin("INITIAL")
    return t
```

IMPLEMENTAÇÃO FINAL

Com o desenvolver do programa percebemos que a solução que estávamos a criar não suportava a possibilidade de o ficheiro CSV te rum número de colunas variado. Sendo assim, decidimos criar uma versão do programa onde seria possível processar vários tipos de ficheiros CSV com formatos diferentes.

TOKENS

Começamos por criar novos tokens de modo a satisfazer a necessidade de ler um número variado de colunas.

```
tokens = ("WORD", "NEWLINE", "QUOTES","COMMENTS")
```

FUNÇÕES

A função “t_QUOTES” tem como principal objetivo ler as variáveis do tipo ‘ ‘ “Dari Persian, Pashto” ‘

```
def t_QUOTES(self, t):
    r'"[^"]+"'
    # t.type = "QUOTES"
    t = t.value.strip()
    self.list_lines.insert(self.count_words, t)
    self.count_words += 1
    pass
```

A função “t_WORD” lê todas as variáveis separadas por virgulas ou um \n

```
def t_WORD(self, t):
    r"[^,\n]+"
    # t.type = 'WORD'
    t = t.value.strip()
    self.list_lines.insert(self.count_words, t)
    self.count_words += 1
    pass
```

A função “t_NEWLINE” identifica quando há uma troca de linha e adiciona a “list_lines”, que guarda as variáveis de uma linha à “list_col” que guarda cada uma das colunas do ficheiro.

```
def t_NEWLINE(self, t):
    r"\n"
    # t.type = 'NEWLINE'
    num = self.counts
    final_list = self.list_lines.copy()
    self.list_col.insert(num, final_list)
    self.counts = self.counts + 1
    self.list_lines.clear()
    self.count_words = 0
    pass
```

A função “t_COMMENTS” ignora as linhas do ficheiro que contenham comentários.

```
def t_COMMENTS(self, t):
    r"\#.*\n"
    pass
```

A função “t_eof” serve para identificar a última linha e caso não encontre um “\n” também a guarda na lista.

```
def t_eof(self, t):
    num = self.counts
    final_list = self.list_lines.copy()
    self.list_col.insert(num, final_list)
    self.counts = self.counts + 1
```

A função “__init__” é usada como construtor da classe “Converter”. Nesta função também foram criadas as listas para as linhas e para as colunas assim como os contadores que necessitamos.

```
def __init__(self, filename):
    self.lexer = None
    self.filename = filename
    self.list_lines = []
    self.list_col = []
    self.counts = 0
    self.count_words = 0
```

A função “toc” é utilizada para fazer a iteração dos “tokens”.Também utilizamos para criar os ficheiros “Html” e “LATEX” e apresentar o output de acordo com o que o utilizador pedir.

```
def toc(self, **kwargs):
    self.lexer = plex.lex(module=self, **kwargs)
    self.lexer.input(slurp(self.filename))
    for token in iter(self.lexer.token, None):
        pass
    f1 = open("list1.html", "w")
    f2 = open("list1.tex", "w")

    pos = 0
    new_list = []  # Temporarily holds one of the columns selected
    final_list = []  # Holds all of the columns selected
    table = []  # Copy of the last list, to convert into HTML LATEX

    print("MENU")
    print("1-Choose columns to include in output;")
    print("2-Output the entire file;")
    print("0-Exit.")

    choice = input()

    if int(choice) == 1:
        num = int(input("How many columns to show?: "))
        j = 0
        while j < num:...
        chosen = pd.DataFrame(table).T.values.tolist()  # Transposing the Matrix
        f1.write(tabulate(chosen, tablefmt='html'))
        f2.write(tabulate(chosen, tablefmt='tex'))
    if int(choice) == 2:
        f1.write(tabulate(self.list_col, tablefmt='html'))
        f2.write(tabulate(self.list_col, tablefmt='tex'))

    print("Processing finished!")
```

CONCLUSÃO

Concluindo, na realização deste trabalho obtivemos variados tipo de conhecimentos acerca de funcionamento de python mais especificamente do ply.lex. Tornou-se um desafio interessante para compreender a aplicação das várias funções do ply .lex nos programas.