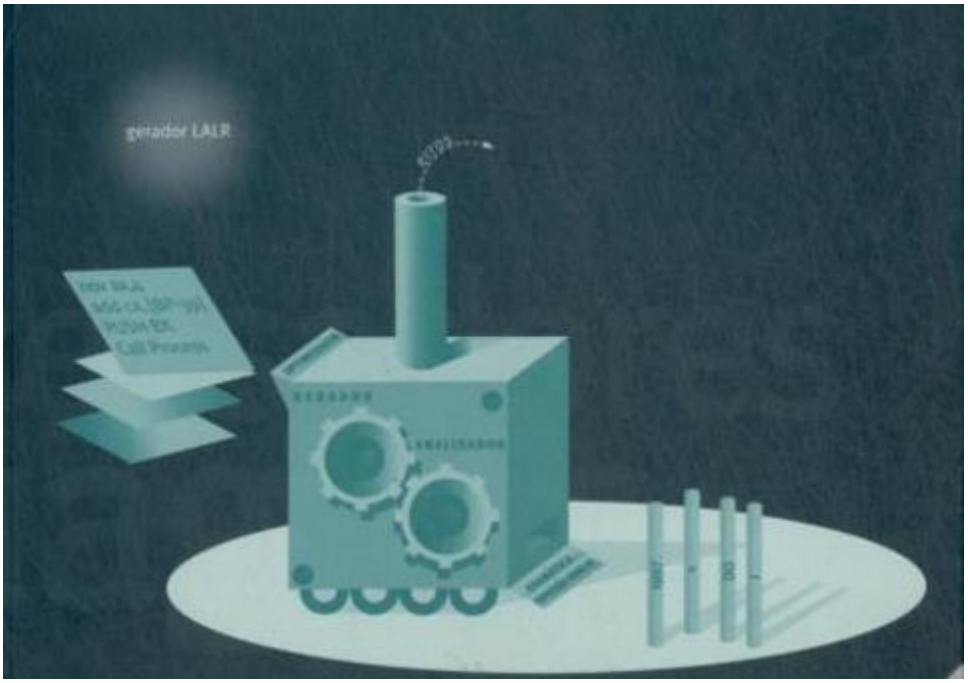
ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Processamento de Linguagens

Licenciatura em Engenharia de Sistemas Informáticos

Trabalho Prático nº 1



Luís Gonçalves 18851 – a18851@alunos.ipca.pt

Luís Esteves 16960 – a16960@alunos.ipca.pt

Exercício II

Introdução

A biblioteca PLY (Python Lex-Yacc) é uma ferramenta poderosa para análise léxica e sintática em Python. Neste exercício será explorado o uso da biblioteca PLY para implementar um analisador léxico em Python, que analisa tokens de um determinado conjunto de instruções de entrada. A análise léxica é uma etapa fundamental em qualquer processo de compilação e interpretação de código pois é responsável por identificar os elementos básicos da linguagem, como palavras-chave, identificadores, números, operadores e símbolos. Neste trabalho, será demonstrado como utilizar a biblioteca PLY para criar um analisador léxico eficiente e robusto, capaz de lidar com diferentes tipos de entradas e garantir a precisão na identificação dos tokens.

Tokens Utilizados

Para o desenvolvimento deste programa decidimos que seriam estes o tokens utilizados.

```
    #products -> Dedicado para ler frases do tipo : PRODUTO=twix=2.3=13.
    #products2 -> Dedicado para ler frases do tipo : PRODUTO=twix.
    #error -> Dedicado para dar erro para quando encontra frases do tipo
: QUANTIA c20, c70.
    #money_on_machine -> Dedicado para ler frases do tipo : MOEDEIRO=0.
    #intro_money -> Dedicado para ler frases do tipo : QUANTIA c20, c70.
    #cancel -> Dedicado para ler frases do tipo : CANCELAR
    #skip -> Dedicado para passar \n,\t e \r a frente

tokens = ["products", "products2", "money_on_machine", "intro_money",
"cancel", "error", "skip"]
```

Funções

1. t_products;

a. Esta função atribui a uma variável 'r' uma expressão regular que procura uma sequência de caracteres que começa com letra maiúscula ou minúscula, seguida por zero ou mais espaços em branco, seguida por um sinal de igual (=), seguida por um ou mais dígitos (0-9), seguido por um ponto opcional (.), seguido por zero ou mais dígitos (0-9), seguido por um sinal de igual (=) e terminando com um ponto (.). Essa expressão regular é usada para extrair informações sobre um produto de uma linha de texto.
b. A função usa a variável global “products_dictionary” para criar um dicionário de produtos que é armazenado numa variável chamada “products_dictionary” através de uma função auxiliar chamada “splitF”.
c. A função adiciona o nome do produto, a quantidade e o preço do produto às respetivas listas 'products', 'amounts' e 'prices'.

```
def t_products(t):
    r"PRODUTO=([a-zA-Z]+[ ]*[a-zA-Z]*)+=[0-9]+(\.)?[0-9]*=[0-
9]+\."
    global products_dictionary
    line = t.value
    type = 0
    products_dictionary = funcAux.splitF(line, type)
    products.append(products_dictionary["Produto"])
    amounts.append(products_dictionary["Quantidade"])
    prices.append(products_dictionary["Preco"])
```

2. t_money_on_machine;

A função armazena a quantidade de dinheiro na máquina na variável “money_on_machine”.

```
def t_money_on_machine(t):
    r"MOEDEIRO[ ]*=[0-9]+\."
    global money_on_machine
    line = t.value
    type = 1
    money_on_machine = funcAux.splitF(line, type)
    pass
```

3. Slurp

Esta função analisa o conteúdo de um ficheiro texto.

```
def slurp(filename):
    fh = codecs.open(filename, 'rb', encoding='utf-8')
    contents = fh.readlines()
    fh.close()
    documento = ""
    print("Ficheiro que vai ser anlisado:\n")
    for linha in contents:
        documento = documento + linha
    print(documento)
    return documento
```

4. SplitF

A função divide uma palavra em partes com base em um caractere específico e, dependendo do tipo, devolve as partes como um dicionário ou uma string.

5. t_intro_money;

A função lê e interpreta uma entrada de texto que representa uma quantia em dinheiro usando expressões regulares para identificar padrões que correspondem à supracitada quantia em dinheiro.

Quando um padrão é identificado, a função armazena a quantia na variável “money_inserted", atualiza um contador de identificação chamado "id_count" e adiciona a quantia inserida na variável "total_insertion".

```
def t_intro_money(t):
    r"QUANTIA([ ]*(c(5|10|20|50))?(e(1|2))?[ ]*(\,|\.))*"
    global money_inserted, id_count, total_insertion
    line = t.value
    line = funcAux.clear_points(line)
    type = 2
    money_inserted = funcAux.splitF(str(line),
money_inserted, id_count, type)
    id_count += 1
    total_insertion.append(money_inserted["total"])
```

6. t_products2.

Esta função usa expressões regulares para identificar padrões que correspondem a um produto.

Quando o padrão é identificado, a função guarda o nome do produto numa variável e limpa pontos e outros caracteres especiais do nome do produto.

Por fim, atualiza um contador de identificação chamado "id_count".

```
def t_products2(t):
    r"PRODUTO=([a-zA-Z]+[ ]*[a-zA-Z]*)+(\.)"
    global desired_product
    line = t.value
    line = funcAux.clear_points(line)
    type = 3
    desired_product = funcAux.splitF(str(line),
money_inserted, id_count, type)
```

7. t_error;

8. Outras funções;

- Clear_points para tirar os pontos encontrados no final das linhas
- Clear_letter para limpar letras do input
- Money_counter para contar o tipo e quantidade de moedas
- Count_money_id para somar o total da quantia inserida

```
def t_cancel(t):
    r"CANCELAR|cancelar|Cancelar"
    global cancel
    cancel = t.value

def t_skip(t):
    r"\n|\t|\r"
    pass
```

E estas funções para erros e cancelar compra.

Bibliografia

Pedro R. Santos, Thibault Langlois (2014) Compiladores – Da Teoria à Prática, FCA - Editora Informática

https://devguide.python.org/

https://johnidm.gitbooks.io/compiladores-para-humanos/content/part1/