Processamento de Linguagens e Compiladores (3º ano de LCC)

Trabalho Prático 1

Relatório de Desenvolvimento

Grupo 2

André Lucena Ribas Ferreira (A94956) Carlos Eduardo da Silva Machado (A96936)
Gonçalo Manuel Maia de Sousa (A97485)

Ano Letivo 2022/2023

Conteúdo

1 Introdução

1.1 Resumo

O presente Relatório tem como objetivo apresentar a nossa solução para o Trabalho Prático 1 da Unidade Curricular de Processamento de Linguagens e Compiladores. Especificamente, demonstramos a conceção de duas soluções para o Problema 5 "Ficheiros CSV com listas e funções de agregação", onde se procura converter um ficheiro do tipo CSV para JSON. A primeira solução utiliza reduzidamente o módulo *re*, suportando-se no Python e nas suas ferramentas, enquanto que a segunda procura maximizar o uso de expressões regulares e das funções desse mesmo módulo.

1.2 Estrutura

Este documento está organizado em diversos capítulos, de acordo com o tema retratado. No Capítulo 1 introduz-se o relatório com um breve resumo. No Capítulo 2 analisa-se detalhadamente o problema em questão, especificando quais os requistos para uma solução adequada. No Capítulo 3 apresenta-se as 2 Soluções propostas para o problema. No Capítulo 4 conclui-se o trabalho com uma síntese e considerações finais.

2 Especificação do Problema

No Enunciado escolhido, pretende-se criar um conversor de ficheiros CSV (*Comma Separated Values*) para o formato JSON, utilizando o módulo *re* da linguagem de programação *python*.

2.1 Análise

Para se resolver o problema, é necessário analisar a representação possível do ficheiro CSV e o seu significado. Notavelmente, este está contido na primeira linha, que funciona como cabeçalho e descreve cada uma das suas colunas. Para realizar a conversão corretamente, é necessário compreender cada uma das possibilidades, sempre com Título: 1. Coluna Simples (Apenas o Título); 2. Colunas com Listas (Tamanho Fixo ou Variável); 3. Colunas com Listas e Função de Agregação.

O seguinte exemplo representa cada uma das possibilidades, por ordem:

```
Número,Nome2,,,Notas3,5::sum,,,,,
```

1. Os Títulos das Colunas Simples encontram-se separadas por uma só vírgula;
2. As Colunas com Lista têm o identificador do número de colunas da lista, possivelmente com valores máximos e mínimos: $\{N, M\}$. Para além disso, é necessário colocar vírgulas para os títulos das colunas abrangidos.
3. As Funções de Agregação aparecem depois do identificador do tamanho da lista, antes das vírgulas destinadas aos campos vazios, indicadas por **::**.

Cada uma das linhas seguintes do ficheiro CSV deve corresponder à organização definida:

```
3162,Cândido,Faísca,12,13,14,,
7777,Cristiano,Ronaldo,17,12,20,11,12
264,Marcelo,Sousa,18,19,19,20,
```

O resultado deve agrupar cada uma das listas num bloco de JSON:

```
[
    {
        "Número": "3162",
        "Nome": [Cândido,Faísca],
        "Notas_sum": 39
    },
    {
        "Número": "7777",
        "Nome": [Cristiano,Ronaldo],
        "Notas_sum": 72
    },
    {
        "Número": "264",
        "Nome": [Marcelo,Sousa],
        "Notas_sum": 76
    }
]
```

1. Cada Coluna Simples deve apenas incluir o seu campo, entre aspas;
2. Cada Lista deve incluir os seus campos numa lista em JSON;
3. Cada Função de Agregação deve alterar o Título para marcar a sua identificação, colocando no JSON o resultado de aplicar a função à lista original.

2.2 Definição dos Requisitos

Como Requisitos para as nossas soluções, consideramos necessário:

1. Identificar as características de cada coluna, através da primeira linha;
2. Manter um registo dessas características;
3. Identificar a quantidade de colunas;
4. Identificar os campos das restantes linhas, de acordo com a caracterização das colunas;
5. Construir o resultado JSON a partir desses campos.

3 Resolução

Para resolver o Problema de acordo com os Requisitos enunciados, as seguintes linhas de código são comuns para ambas as soluções.

Utilizaremos o módulo 're', que oferece operações de matching de expressões regulares em python.

[1]:
```
import re
from pprint import pprint
```

De seguida, obtemos o ficheiro de input do utilizador, do tipo CSV.

[2]:
```
filename = 'teste1.csv'#input("introduza o ficheiro de input:")
input = open(filename, "r", encoding="utf-8")
```

O primeiro passo de ambas as soluções é o de interpretar a primeira linha, de modo a conseguir uma caracterização das colunas que fazem parte do ficheiro. Para tal, construimos uma expressão regular que, com a utilização de grupos e da interação da função *findall* com esses mesmos grupos, nos devolve uma lista que define completamente cada uma das colunas.

Mais precisamente, cada entrada da lista é um tuplo com 4 componentes, todas do tipo *string*. A presença dos três últimos é facultativa e indica se a coluna tem elemento mínimo, se forma uma lista e qual o seu elemento máximo, e se tem uma função aplicada.

1. Título;
2. Número mínimo de elementos da lista;
3. Número máximo de elementos da lista;
4. Função aplicada à lista.

A expressão regular começa por definir uma palavra obrigatória **([^{,]+)** , capturada no primeiro grupo, que é o Título da coluna. Nesta parte, utiliza-se parênteses retos com acento circunflexo para denotar qualquer letra que não as especificadas, nomeadamente chaveta esquerda e vírgula, e depois um operador para repetir, pelo menos uma vez, essas letras.

De seguida, definimos dois blocos facultativos, utilizando o operador **(?:)** para não capturar os grupos definidos pelos parênteses necessários para o operador **?** que os torna opcionais. O primeiro representa as listas, onde pelo menos um número é garantido, definido de tal modo a pertencer ao grupo 3, capturando no grupo 2 o valor mínimo se existente. O segundo representa a função, cuja designação encontra-se capturada no grupo 4.

[3]:
```
def constroi_colunas(input):
    primeira = input.readline()[:-1]
    colunas = re.findall(r'([^{,]+)(?:{(?:(\d+),)?(\d+)})?(?:::([^,]+))?',␣
 ↪primeira)
    return colunas
```

[4]:
```
colunas = constroi_colunas(input)
pprint(colunas)
```

```
[('Aluno', '', '', ''),
 ('Número', '', '', ''),
 ('Disciplina', '', '', ''),
```

```
('Notas', '3', '5', 'len'),
('Faltas', '', '', '')]
```

Deste modo, podemos satisfazer aos requisitos 1, 2 e 3.

3.1 Solução 1

A primeira solução apenas utiliza a lista *colunas* para construir o resultado. Com ela, podemos percorrer cada linha, coluna a coluna, e construir progressivamente a nossa solução.

Para cada linha, realizamos uma separação por vírgulas para conseguir cada campo, satisfazendo o requisito 4. Depois, dependendo da coluna, construímos a solução, satisfazendo o requisito 5.

[5]:
```
def constroi_resultado_1(colunas, input):
    res = '[\n'
    #0->título, 1->min, 2->max, 3->fun
    for linha in input:
        linha = re.sub(r'\n$', r'', linha)
        split = re.split(r',', linha)
        r = 0 #ponto da lista split a ler
        res += '\t{\n'
        for i in range(len(colunas)):
            if colunas[i][2] == '':
                res += '\t\t\"%s\": \"%s\",\n' % (colunas[i][0], split[r])
                r += 1
            else:
                max = int(colunas[i][2])
                lis = []
                for l in range(max):
                    if split[r] != '':
                        lis.append((split[r]))
                    r+=1
                if colunas[i][3] == '':
                    res+='\t\t\"%s\": %s,\n' % (colunas[i][0], str(map(int,lis)))
                else:
                    fun = eval(colunas[i][3])
                    res+='\t\t\"%s_%s\": %s,\n' % (colunas[i][0], colunas[i][3],␣
↪str(fun(list(map(int,lis)))))
        res = re.sub(r',\n$', '\n\t},\n', res) #res[:-2] + '\n\t},\n'
    res = re.sub(r',\n$', '\n]', res) #res[:-2] + '\n]'
    return res
```

3.2 Solução 2

Esta solução é a implementação principal deste projeto. A inspiração para a sua conceção nasce da ideia de tentar "codificar" o significado de cada uma das colunas numa única expressão regular, de modo a evitar, no passo final da solução, enquanto se percorre cada uma das linhas, de também percorrer cada coluna isoladamente e continuamente endereçar a lista *colunas*. Esta solução é mais interessante, de um ponto de vista conceptual, por envolver uma maior complexidade de expressões regulares e por ser, subjetivamente, mais elegante.

Esta codificação apresenta-se a partir de três expressões regulares, nomeadamente:

1. **patern**, que representa o padrão de identificação de cada uma das linhas. Este define a estrutura que cada linha deve seguir, servindo como validação, enquanto também captura grupos com os elementos necessários para a produção do resultado. Esta expressão regular serve como substituta à lista obtida pelo *findall* e satisfaz os requisitos 2 e 3. Para além disso, a sua utilização como padrão serve para satisfazer o requisito 4.
2. **replace**, que traduz a estrutura de saída do ficheiro JSON, para cada linha. Esta expressão regular utiliza os grupos que serão capturados pelo anterior, colocando os Títulos prontos para o ficheiro de saída. No entanto, esta expressão regular não efetua o cálculo das funções, por ainda não se as conhecer. Ao invés disso, coloca na solução uma string com a função pronta a ser invocada com a função *eval* do *python*.
3. **fun_str**, que guarda, numa alternativa, todas as funções encontradas nas colunas do ficheiro CSV. No final, depois de se invocar o *sub* que troca *patern* por *replace*, é necessário substituir todas estas funções pela sua avaliação, tomando quaisquer parâmetros necessários, devidamente colocados por *replace*. Assim, estas duas expressões servem para satisfazer o requisito 5.

Estas expressões regulares são construídas iterativamente a partir de fragmentos, estes escolhidos dependendo da caracterização da coluna correspondente. Os fragmentos de *patern* são dois, nomeadamente:

1. **([^\n,]+),**: Fragmento de uma Coluna Simples. Nele, capturam-se palavras sem vírgulas nem *newline*.
2. **([^\n,]+(?:,[^\n,]+){%d,%d}),{0,%d},**: Fragmento de Lista

Os fragmentos da expressão regular *replace* são os da sintaxe do ficheiro JSON, dependendo do formato da coluna. Notavelmente, se a lista tiver número mínimo de elementos igual a 0, é necessário ser capaz de não encontrar elementos.

Segundo a documentação do módulo *re*, o número máximo de grupos que cada objeto do tipo *Match* consegue guardar é 99. Isto limita a solução a ter, no máximo, 99 colunas, devido à utilização de grupos consecutivos na expressão *patern*.

O seguinte ciclo percorre cada uma das colunas e constrói, iterativamente, cada uma das expressões regulares explicadas anteriormente.

Notavelmente, o caso de não haver valor mínimo de elementos da lista, considera-se que o valor mínimo é igual ao máximo, de modo a ter um fragmento de *patern* único para ambos os casos.

[6]:
```
def constroi_re_2(colunas):
    patern = fun_str = r''
    replace = '\t{\n'
    #colunas: 0 -> título; 1->min; 2->max/único; 3->função
    for i,coluna in enumerate(colunas, start=1):
        if coluna[2] == '': #coluna simples
            patern += r'([^\n,]+),'
            replace += '\t\t"%s": "\%d",\n' % (coluna[0],i)
        else: #listas
            max = int(coluna[2])
            min = max if coluna[1] == '' else int(coluna[1])
            #caso em que o min == 0 precisa de funcionar como ?
            patern += r'((?:[^\n,]+){%d,1}(?:,[^\n,]+){%d,%d}),{0,%d},' %␣
↪(int(min!=0), min-1 if min!=0 else 0, max-1, max - min)
            if coluna[3] != '': #Função
                fun_str += r'%s|' % (coluna[3])
                replace += '\t\t"%s_%s": %s([\%d]),\n' %␣
↪(coluna[0],coluna[3],coluna[3],i)
            else:
                replace += '\t\t"%s": [\%d],\n' % (coluna[0],i)

    patern = re.sub(r',$', r'\\s*', patern)
    replace = re.sub(r',\n$', '\n\t},\n', replace) #replace = replace[:-2] +␣
↪'\n\t},\n'
    fun_str = fun_str[:-1]
    return (patern,replace,fun_str)
```

[7]:
```
patern, replace, fun_str = constroi_re_2(colunas)

print(patern)
```

```
([^\n,]+),([^\n,]+),([^\n,]+),((?:[^\n,]+){1,1}(?:,[^\n,]+){2,4}),{0,2},([^\n,]+
)\s*
```

A expressão regular *replace* não coloca os parênteses retos necessários para o ficheiro JSON, por ser uma caracterização de cada uma das linhas e não do ficheiro inteiro.

[8]:
```
print(replace)
```

```
        {
                "Aluno": "\1",
                "Número": "\2",
                "Disciplina": "\3",
                "Notas_len": len([\4]),
                "Faltas": "\5"
        },
```

[9]:
```
print(fun_str)
```

```
len
```

Para terminar, o resultado é construído linha a linha através de um simples *sub*, suportado pelas expressões regulares montadas anteriormente. Para terminar, um último *sub* substitui todas as instâncias de uma função pela sua avaliação e subsequente resultado.

[10]:
```
def constroi_resultado_2(input, patern, replace, fun_str):
    res = '[\n'
    for linha in input:
        #linha = linha[:-1]
        #print(linha)
        res += re.sub(patern,replace,linha)
    res = re.sub(r'_(%s)": ((?:\1)\(.*\))'%(fun_str), lambda x: '_%s": %s' % (x.
↪group(1), str(eval(x.group(2)))), res)
    res = re.sub(r',\n$', '\n]', res)#res = res[:-2] + "\n]"
    return res
```

3.3 Testes

Nesta subsecção apresentamos os testes concebidos para validar a implementação. Estes são focados em redor da Solução 2, por ser a principal do projeto.

3.3.1 Teste 1

```
"teste1.csv":
Aluno,Número,Disciplina,Notas3,5::len,Faltas
André,94956,SO,20,10,15,,,10
Bruno,94957,POO,9,9,9,9,,100
Carlos,96936,CP,18,18,10,18,10,0
Dantas,00000,ALF,0,0,0,0,,99999
Eduardo,11111,LA,7,7,7,,,5
Filipa,22222,AUC,16,10,6,,,2
Gonçalo,97485,PLC,17,10,18,,,0
```

[11]:
```python
input1 = open('teste1.csv', "r", encoding="utf-8")
colunas = constroi_colunas(input1)
print(colunas)
res1 = constroi_resultado_1(colunas, input1)
input1.close()
#print(res1)

input1 = open('teste1.csv', "r", encoding="utf-8")
colunas = constroi_colunas(input1)
(patern1,replace1,fun_str1) = constroi_re_2(colunas)
res2 = constroi_resultado_2(input1,patern1,replace1,fun_str1)
print(res2)
input1.close()

print(f"As duas soluções são iguais? Resposta: %s." % (res1 == res2))
```

```
[('Aluno', '', '', ''), ('Número', '', '', ''), ('Disciplina', '', '', ''),
('Notas', '3', '5', 'len'), ('Faltas', '', '', '')]
[
        {
                "Aluno": "André",
                "Número": "94956",
                "Disciplina": "SO",
                "Notas_len": 3,
                "Faltas": "10"
        },
        {
                "Aluno": "Bruno",
                "Número": "94957",
                "Disciplina": "POO",
                "Notas_len": 4,
                "Faltas": "100"
```

```
        },
        {
                "Aluno": "Carlos",
                "Número": "96936",
                "Disciplina": "CP",
                "Notas_len": 5,
                "Faltas": "0"
        },
        {
                "Aluno": "Dantas",
                "Número": "00000",
                "Disciplina": "ALF",
                "Notas_len": 4,
                "Faltas": "99999"
        },
        {
                "Aluno": "Eduardo",
                "Número": "11111",
                "Disciplina": "LA",
                "Notas_len": 3,
                "Faltas": "5"
        },
        {
                "Aluno": "Filipa",
                "Número": "22222",
                "Disciplina": "AUC",
                "Notas_len": 3,
                "Faltas": "2"
        },
        {
                "Aluno": "Gonçalo",
                "Número": "97485",
                "Disciplina": "PLC",
                "Notas_len": 3,
                "Faltas": "0"
        }
]
As duas soluções são iguais? Resposta: True.
```

Apenas se demonstram os seguintes testes realizados sobre a Solução 2 já que a 1ª não está desenhada para Listas com Strings.

3.3.2 Teste 2

```
"teste2p.csv":
Personagens3,Cenários3,5,Adereços0,4,Atos3
'Anjo','Diabo','Joane','Barca do Infero','A Outra␣
 ↪Barca','Fila','Mar','Praia','Espadas','Sapatos','Cabra','Capacete',1,2,3
'Carlos da Maia','Maria Eduarda','Pedro da Maia','O Ramalhete','A␣
 ↪Toca','Consultório do Carlos','Vila Balsac',,'Vénus␣
 ↪Citereia','Cascatazinha','Cipreste e Cedro',,10,15,20
'Blimunda','Baltasar Mateus','D.João V','Convento de␣
 ↪Mafra','Passarola','Oficina','Terreiro do Paço',,,,,,,,50,99,3
'Vasco da Gama','Marte','Vénus','Olimpo','Torre de␣
 ↪Belem','India',,,'Bússola',,,,,44,55,66
```

[12]:
```
input1 = open('teste2p.csv', "r", encoding="utf-8")
colunas = constroi_colunas(input1)
print(colunas)
(patern1,replace1,fun_str1) = constroi_re_2(colunas)
res2 = constroi_resultado_2(input1,patern1,replace1,fun_str1)
print(res2)
input1.close()
```

```
[('Personagens', '', '3', ''), ('Cenários', '3', '5', ''), ('Adereços', '0',
'4', ''), ('Atos', '', '3', '')]
[
        {
                "Personagens": ['Anjo','Diabo','Joane'],
                "Cenários": ['Barca do Infero','A Outra
Barca','Fila','Mar','Praia'],
                "Adereços": ['Espadas','Sapatos','Cabra','Capacete'],
                "Atos": [1,2,3]
        },
        {
                "Personagens": ['Carlos da Maia','Maria Eduarda','Pedro da
Maia'],
                "Cenários": ['O Ramalhete','A Toca','Consultório do
Carlos','Vila Balsac'],
                "Adereços": ['Vénus Citereia','Cascatazinha','Cipreste e
Cedro'],
                "Atos": [10,15,20]
        },
        {
                "Personagens": ['Blimunda','Baltasar Mateus','D.João V'],
                "Cenários": ['Convento de Mafra','Passarola','Oficina','Terreiro
do Paço'],
                "Adereços": [],
                "Atos": [50,99,3]
        },
```

```
        {
                "Personagens": ['Vasco da Gama','Marte','Vénus'],
                "Cenários": ['Olimpo','Torre de Belem','India'],
                "Adereços": ['Bússola'],
                "Atos": [44,55,66]
        }
]
```

3.3.3 Teste 3

```
"teste3p.csv":
Kilometros5::sum,Motos1,4::sorted,Carros0,3::len,Biciclos1,2
10,20,30,40,50,"Morini X-Cape","Voge 300 Rally",,,"Mercedes C220␣
↪Classic",,,"BICICLETA 500 UNICÓRNIO",
64,104,47,121,84,"Casal K186 Phantom 5","Famel E-XF","BMW R 18",,"Fiat␣
↪Punto","Ford Fiesta","Ford Model T","BMX WIPE 500","BTT"

0,1,9,20,100,"Vespa GTS 300,BMW C 400 X","Kymco AK 550","Benelli Leoncino␣
↪500",,,,"BTT ALL Mountain",
```

[13]:
```
input1 = open('teste3p.csv', "r", encoding="utf-8")
colunas = constroi_colunas(input1)
print(colunas)
(patern1,replace1,fun_str1) = constroi_re_2(colunas)
res2 = constroi_resultado_2(input1,patern1,replace1,fun_str1)
print(res2)
input1.close()
```

```
[('Kilometros', '', '5', 'sum'), ('Motos', '1', '4', 'sorted'), ('Carros', '0',
'3', 'len'), ('Biciclos', '1', '2', '')]
[
        {
                "Kilometros_sum": 150,
                "Motos_sorted": ['Morini X-Cape', 'Voge 300 Rally'],
                "Carros_len": 1,
                "Biciclos": ["BICICLETA 500 UNICÓRNIO"]
        },
        {
                "Kilometros_sum": 420,
                "Motos_sorted": ['BMW R 18', 'Casal K186 Phantom 5', 'Famel
E-XF'],
                "Carros_len": 3,
                "Biciclos": ["BMX WIPE 500","BTT"]
        },

        {
                "Kilometros_sum": 130,
                "Motos_sorted": ['Benelli Leoncino 500', 'Kymco AK 550', 'Vespa
GTS 300,BMW C 400 X'],
                "Carros_len": 0,
                "Biciclos": ["BTT ALL Mountain"]
        }
]
```

3.3.4 Teste 4

```
"teste4p.csv":
Pessoas1,3,Fruta,Animais2,Lanche4::len,Probabilidade,Trofeus0,2::sorted
André,Carlos,Gonçalo,Maça,Cão,Gato,"Croissant","Lanche","Bolo de Arroz","Nata",0.
 ↪5,,,
Amor Martim,Eugénio Isidora,,Banana,Aye-aye,tardígrado,"Pão de␣
 ↪ló","Anjo","Guardanapo","Suspiro",0.1,"Darwin Award","Nobel da Paz"
Horácio Cipriano,,,Pitaia,Lova-a-deus,Bicho-da-seda,"Donut","Torrada","Pão com␣
 ↪chouriço","Bola de Berlim",0,"Bola de Ouro"
Urban Niclas,Patrik Barbro,Utônio,Pepino,Coelacanth,T-Rex,"Barra de␣
 ↪Chocolate","Batatas fritas","Sandes","Pão",0.99,"Turing Award","IgNobel"
```

[14]:
```
input1 = open('teste4p.csv', "r", encoding="utf-8")
colunas = constroi_colunas(input1)
print(colunas)
(patern1,replace1,fun_str1) = constroi_re_2(colunas)
res2 = constroi_resultado_2(input1,patern1,replace1,fun_str1)
print(res2)
input1.close()
```

```
[('Pessoas', '1', '3', ''), ('Fruta', '', '', ''), ('Animais', '', '2', ''),
('Lanche', '', '4', 'len'), ('Probabilidade', '', '', ''), ('Trofeus', '0', '2',
'sorted')]
[
        {
                "Pessoas": [André,Carlos,Gonçalo],
                "Fruta": "Maça",
                "Animais": [Cão,Gato],
                "Lanche_len": 4,
                "Probabilidade": "0.5",
                "Trofeus_sorted": []
        },
        {
                "Pessoas": [Amor Martim,Eugénio Isidora],
                "Fruta": "Banana",
                "Animais": [Aye-aye,tardígrado],
                "Lanche_len": 4,
                "Probabilidade": "0.1",
                "Trofeus_sorted": ['Darwin Award', 'Nobel da Paz']
        },
        {
                "Pessoas": [Horácio Cipriano],
                "Fruta": "Pitaia",
                "Animais": [Lova-a-deus,Bicho-da-seda],
                "Lanche_len": 4,
                "Probabilidade": "0",
                "Trofeus_sorted": ['Bola de Ouro']
```

```
        },
        {
                "Pessoas": [Urban Niclas,Patrik Barbro,Utônio],
                "Fruta": "Pepino",
                "Animais": [Coelacanth,T-Rex],
                "Lanche_len": 4,
                "Probabilidade": "0.99",
                "Trofeus_sorted": ['IgNobel', 'Turing Award']
        }
]
```

4 Conclusão

Resumindo, este projeto consistiu na implementação de um conversor de ficheiros CSV para JSON utilizando o modulo *re* de python. Desta forma, exploramos as capacidades do módulo e os limites das expressões regulares para realizar filtros de texto.

Consideramos que a implementação apresentada cumpre os requisitos do problema, para além de possibilitar o processamento de ficheiros que tenham um número arbitrário de caracteres brancos entre as linhas do input e de mostrar o resultado das funções que são aplicadas, independentemente do seu tipo.

Tudo isto foi realizado explorando o máximo o modulo proposto, através de expressões regulares construídas durante a execução. Propomos que no futuro a solução apresentada possa ser expandida para que seja capaz de processar ficheiros de maior complexidade. Também sugerimos ser possivel utilizar esta estrutura de implementação para outros tipos de conversores, possivelmente simplificando os mesmos.

Referências

[1] https://docs.python.org/3/library/re.html