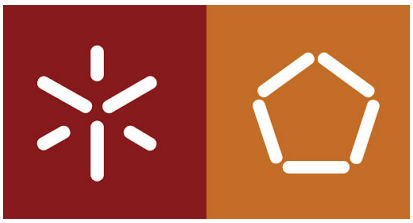

Universidade do Minho

Licenciatura em Engenharia Informática

Processamento de Linguagens

Trabalho Prático: Conversor de ficheiros CSV para JSON

Grupo 50

Rui Monteiro (A93179) Rodrigo Rodrigues (A93201)
Daniel Azevedo (A93324)

Ano Letivo 2021/2022



1 Resumo

Este documento é um relatório de desenvolvimento sobre o 1º trabalho prático realizado no âmbito da Unidade Curricular Processamento de Linguagens perante a proposta de fazer um **conversor de ficheiro CSV** *(Comma separated values)* para o formato JSON. O foco deste trabalho é escrever Expressões Regulares (ER) para descrição de padrões em *streams* de texto. Todo o código aqui apresentado está na íntegra no ficheiro em anexo.

Conteúdo

2 Introdução

As Expressões Regulares (ER) estão na base de inúmeras funcionalidades em *websites* e editores de texto, permitindo descrever padrões em *streams* de texto e transformar texto com base em regras. Uma das grandes vantagens de *regEx* é a possibilidade de definir os nossos próprios critérios de pesquisa para um padrão que atenda às nossas necessidades. Neste trabalho, através do uso de ER e do módulo *re* do Python, desenvolvemos uma solução para um **conversor de ficheiro CSV** para o formato JSON *(JavaScript Object Notation)*.

3 Ficheiros CSV com listas e funções de agregação

3.1 Análise do Problema

Um ficheiro CSV é um ficheiro de texto simples no qual as informações são separadas por vírgulas e é comum em aplicativos de folhas de cálculo. Os ficheiros de entrada (*datasets*) com que estamos a trabalhar podem conter listas e funções de agregação. As listas são formadas por conjuntos de campos, pelo que podem ser:

- Listas com tamanho definido **N**
- Listas com um intervalo de tamanhos **{N,M}**

Por sua vez, as funções de agregação podem ser aplicadas a listas, seguindo a notação Campo{N,M}::funcao, onde *funcao* é aplicada à lista de comprimento variável entre N e M de valores respetivos ao campo *Campo*.

Após análise do problema naturalmente se reconhece que este problema se baseia em interpretação, identificação de padrões e filtragem de texto, sendo que as Expressões Regulares revelam-se úteis na sua resolução.

3.2 Resolução do Problema

3.2.1 Expressões Regulares para tratamento do cabeçalho e do corpo do ficheiro CSV

Sabendo que a primeira linha do ficheiro CSV funciona como cabeçalho que define o que representa cada coluna, é importante capturar os diferentes campos/colunas. Para tal foi utilizada a seguinte expressão regular

```
cabecalho = r'([\wà-úÀ-Ú\/]+)({(\d),(\d)}|{(\d)})?((\:\:)([\wà-úÀ-Ú\/]+))?'
```

que contempla as diferentes possibilidades de um cabeçalho (com ou sem lista e/ou função de agregação, e os diferentes campos possíveis).

Para as restantes linhas (corpo do ficheiro CSV) foi utilizada a seguinte expressão regular

```
corpo = r'([^,\n]*)'
```

que captura todos os tokens separados por vírgula, excluindo vírgulas e *newline*'s.

3.2.2 Verificar o tipo de elementos

Havendo listas no *dataset*, é importante verificar o tipo de elementos que constituem a lista de forma a converter e apresentar corretamente a informação no ficheiro JSON, bem como para eventuais aplicações de funções de agregação a essas listas. De igual forma, noutros campos do CSV será feita a verificação.

Para verificar se um elemento do ficheiro CSV é um número(positivo ou negativo) ou uma String, é utilizada a função *checkNumber* que aplica ao elemento a seguinte expressão regular

```
expNum = r'((\-|\+)?\d+(.\d+)?)'
```

que verifica se o elemento é um inteiro, um *float* ou uma string caso não exista *match* com o conteúdo do ficheiro CVS. Esta verificação ocorre com o objetivo de escrever, no ficheiro JSON, o respetivo elemento com o tipo correto (caso seja uma string irá iniciar e finalizar com " (aspas), o que não acontece com valores numéricos).

3.2.3 Verificar se o input tem listas

Ao invocar o *findall* para o cabeçalho, é possível obter uma lista com todos os *matches*. Desta forma, verificando o grupo de captura correspondente ao *match* da lista, é possível averiguar de forma simples se o input tem lista. Se o grupo estiver "vazio"conclui-se que não apresenta lista e em caso contrário apresenta.

```
tokens = re.findall(cabecalho,first_line)
```

3.2.4 Verificar o intervalo de comprimento da lista

Sabendo da existência de listas, é necessário verificar se a lista apresenta tamanho fixo N ou variável $\{N,M\}$.

Em caso de **lista de tamanho fixo** N, é inserido, numa lista os valores apresentados, sendo que não se pode exceder o tamanho da lista.

```
lista = []
for index in range(i,n):
    lista.append((toks[index]))
    index += 1
```

Em caso de **lista de tamanho variável** $\{N,M\}$:

- Verifica-se o valor das fronteiras N e M
- Calcula-se variável *maxIndex* que representa o índice máximo que o último valor da lista terá caso realmente sejam especificados M valores, ou seja, todos os valores possíveis, não utilizando vírgulas para preencher espaços em falta

- Insere-se na lista os valores correspondentes, indicados no ficheiro, sendo que não se pode exceder o tamanho da lista nem inserir valores correspondentes a campos do cabeçalho em posições à frente.

Considerando o seguinte cabeçalho de exemplo

```
Número,Nome,Curso,Notas{3,6}::sum,,,,,,Professor
```

podemos obter a representação gráfica dos grupos de captura obtida pelo programa:

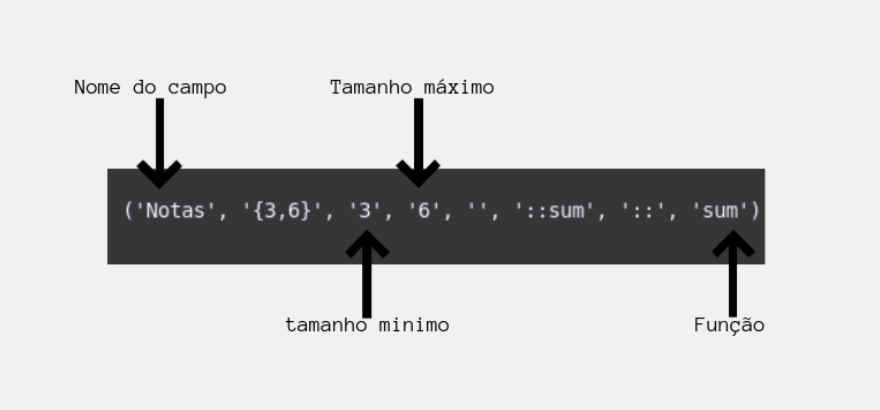

Figura 1: Grupos de captura

Considerando a seguinte linha do ficheiro de exemplo

```
3162,Cândido Faísca,Teatro,12,13,14,19,7,5.5,Ernesto
```

obtemos os seguintes *matches*

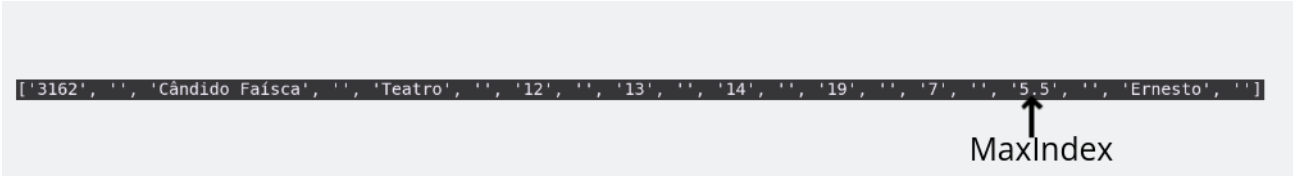

Figura 2: Matches da linha de exemplo

Por observação da Figura 2 é possível observar o último valor da lista(5.5) de tamanho variável {3,6}, encontra-se no índice 16. Ora, o *maxIndex* calculado pelo programa também é o índice 16, tal como esperado, uma vez que a lista possui o tamanho máximo possível segundo o cabeçalho (6 elementos). Desta forma, no código está garantido que não serão adicionados à lista valores de campos seguintes, pois só serão tidos em contas valores dentro dos limites estabelecidos. Caso a lista apresentasse 3/4/5 elementos, seria feito um salto de forma a avançar para o campo seguinte, não copiando/inserindo valores de campos errados na lista.

```
# Caso {a,b}
if not tok[4]:
                    a = tok[2]
                    b = tok[3]
                    n = index + (int(a)) + 2
                    maxIndex = index + (int(b)*2) - 1
                    i = index
                    lista = []

                    for index in range(i,n,2):
                        if toks[index]:
                            lista.append(toks[index])

                    i = (n+1)
                    for index in range(i,maxIndex,2):
                        if toks[index]:
                            lista.append(toks[index])
                        else:
                            break

                    if len(lista) != int(b):
                        index += int(b) - len(lista)
                    else:
                        index += 2
```

3.2.5 Verificar se existe função de agregação

Após a verificação do tamanho da lista, é necessário averiguar se esta se encontra associada a uma função. Para isso vamos à lista de *matches* da expressão regular do cabeçalho, e ao respetivo grupo de captura que terá essa informação (ver Figura 1), caso esse campo se encontre vazio não irá existir nenhuma função associada à lista correspondente.

3.3 Análise de Resultados

Tendo como input o dataset *emd.csv*, obtemos o seguinte ficheiro *JSON* tal como esperado:

Por observação dos resultados obtidos, é imediato verificar os seguintes aspetos:

- As strings estão envoltas de aspas, e os valores numéricos não.
- Cada conjunto de chavetas representa uma linha do ficheiro original CSV e contempla os respetivos campos (dentro das chavetas).
- Os conjuntos de chavetas estão separados por vírgulas, sendo que não existe vírgula após o último conjunto de chavetas (tal como seria de esperar).
- Todos os conjuntos de chavetas estão envoltos de parêntesis retos que abrem e fecham na primeira e última linha do ficheiro, respetivamente.

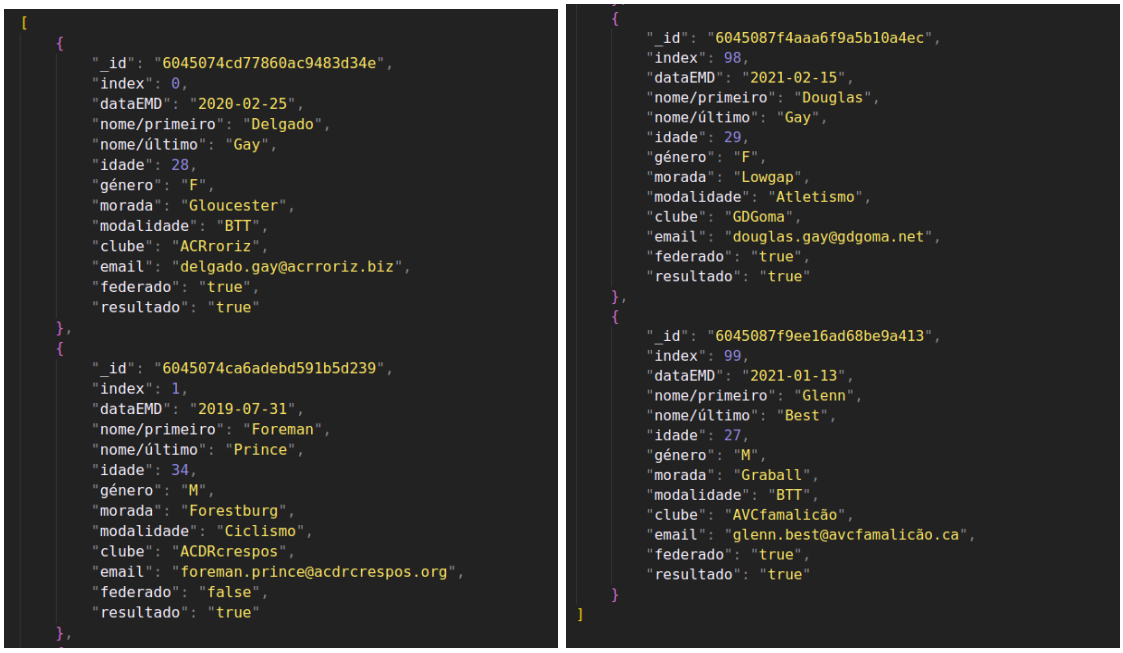

Figura 3: Excerto inicial(esquerda) e final(direita) do ficheiro .json gerado após conversão do dataset *emd.csv*

Após análise de resultados obtidos, e com o auxílio de ferramentas de verificação de ficheiros .json como *JSON Formatter Validator - debug JSON data with advanced formatting and validation algorithms* considera-se ter um elevado grau de correção e ter alcançado os objetivos propostos.

4 Comentários Finais e Conclusão

Através da realização deste trabalho prático, foi possível consolidar a matéria lecionada nas aulas práticas, nomeadamente expressões regulares e também obter uma maior experiência em Python, permitindo a melhorar a capacidade de de escrita de expressões de regulares e o seu modo de manipular em Python, com o auxílio do módulo *re*.