Processamento de Linguagens (3º ano de LEI)

Trabalho Prático 1

Relatório de Desenvolvimento

Conversor de CSV para JSON

Grupo 27

Bernardo Saraiva (a93189)

José Gonçalves (a93204)

Rui Moreira (a93232)

28 de março de 2022

Resumo

O presente trabalho prático consiste em implementar a conversão de um ficheiro em formato CSV para JSON. Tem-se como objectivo aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases dentro de textos, no contexto do módulo *'Ply'* do Python.

Conteúdo

Capítulo 1

Introdução

O presente documento encontra-se inserido no primeiro projeto prático da cadeira de Processamento de Linguagens, que se insere no 3º ano da Licenciatura em Engenharia Informática.

1.1 Estrutura do Relatório

O presente relatório encontra-se dividido em 5 secções. Nesta primeira, no capítulo 1, é introduzido o trabalho, onde se encontra a atual descrição da estrutura do relatório.

O capítulo 2, que corresponde à análise do problema, extrai-se e estrutura-se a informação que está presente no enunciado. Nesta parte vamos identificar todo o conhecimento necessário para começar a desenhar a solução.

O terceiro capítulo, 3, irá descrever a forma como foi desenvolvida a ferramenta, explicando os vários passos de leitura e processamento dos ficheiros, até à sua transformação total em ficheiros no formato JSON. No capítulo 4 demonstra-se a utilidade do programa que desenvolvemos, realizando testes para verificar se as soluções correspondem às expectativas e referindo como proceder para obter um ficheiro em JSON convertido de um ficheiro em CSV.

Por fim, no último capítulo, 5, termina-se o relatório com uma síntese e análise crítica do trabalho realizado.

1.2 Enunciado do Problema

O projeto desenvolvido tem como objetivo efetuar a conversão de um ficheiro CSV (Comma Separated Values) para ficheiros JSON (JavaScript Object Notation), recorrendo à linguagem de programação Python e às suas bibliotecas *'re'* e *'ply'*.

Para o processamento do ficheiro CSV, é de extrema importância a interpretação da primeira linha do ficheiro, que funciona como o cabeçalho da tabela. Deste modo, através da separação por vírgulas, o cabeçalho indica a que campo cada uma das informações presentes nas linhas seguintes pertence. No entanto, é de notar que o ficheiro CSV original poderá conter alguma notação no seu cabeçalho, implicando um processamento adicional antes de efetuar a sua escrita no ficheiro JSON, tal como se descreve na Secção 3.1. Com isto, a um campo poderá ser aplicada uma função de agregação, ou representar uma lista, o que implica cuidados adicionais.

Capítulo 2

Análise do Problema

2.1 Formatos de texto CSV e JSON

De modo a processar um ficheiro CSV para JSON é necessário especificar de que modo cada um dos seus elementos se irão relacionar. Assim, o cabeçalho do ficheiro CSV possuirá as diversas chaves presentes no ficheiro JSON que deverá ser criado, e todas as linhas restantes possuirão os valores a atribuir a estas mesmas chaves.

Nas secções 2.2 e 2.3, apresentam-se os 3 modos diferentes de processamento adicional que foram descritos na sub-secção 1.2 por requererem processamento adicional.

2.2 Listas

- **Listas com tamanho definido:**

  O campo descrito com um número entre chavetas representa o número de colunas que esse campo abrange.

  Exemplo:

  ```
  Número,Nome,Curso,Notas{5},,,,,
  3162,Cândido Faísca,Teatro,12,13,14,15,16
  ```

  Neste caso, o campo abrangido com chavetas conta com uma lista de 5 valores que correspondem ao campo notas.

- **Listas com um intervalo de tamanhos:**

  Para além de listas com tamanho único podemos ter listas cujo tamanho varia num determinado intervalo.

  Exemplo:

  ```
  Número,Nome,Curso,Notas{3,5},,,,,
  3162,Cândido Faísca,Teatro,12,13,14,,
  7777,Cristiano Ronaldo,Desporto,17,12,20,11,12
  ```

A notação do exemplo anterior indica que o campo Notas está representado numa lista que poderá variar entre 3 a 5 valores.

2.3 Funções de Agregação

- **Funções de agregação:**

  Para além dos exemplos referidos anteriormente, podemos ter também a presença de elementos que indicam uma função a ser aplicada a uma lista, identificado pelo extensão "::função"onde função identifica como a lista deverá ser tratada.

  Exemplo:

```
Número,Nome,Curso,Notas{3,5}::sum,,,,,
3162,Cândido Faísca,Teatro,12,13,14,,
```

  Sendo o output representado com o somatório dos valores presentes naquele campo, assim como o identificador também é alterado, aparecendo do seguinte modo:

```
"Notas_sum": 39
```

Capítulo 3

Concepção/Desenho da Solução

Para a concepção do programa descrito, foi determinado que a melhor e mais eficaz estratégia a usar para a leitura do ficheiro seria a leitura linha a linha. O facto de este tipo de ficheiro não conter qualquer dependência entre as linhas do corpo do ficheiro (apesar de todas se relacionarem com o cabeçalho), resulta numa redução da utilização de dados e estruturas ao mínimo indispensável, tornando-se relevante guardar apenas a informação do cabeçalho para relacionar com as restantes e poder identificar a sua coluna.

Assim, para conseguir obter o resultado pretendido, recorreu-se ao módulo *'ply'*. Este tem uma sintaxe bastante específica, que trabalha essencialmente com *tokens* e *states*, identificando-os e realizando ações sobre os mesmos.

Nas secções seguintes será descrita com maior detalhe a utilização do módulo *'ply'* no funcionamento do projeto desenvolvido.

3.1 Processamento do cabeçalho

Para obter a informação acerca das colunas é necessário o processamento do cabeçalho, quer seja para obter o número de colunas ou o nome de cada uma. O cabeçalho contém também algumas notações especiais, pelo que se descrevem em seguida junto de alguns exemplos:

Deste modo, tornou-se necessário criar um estado específico para o assunto, com o nome de **header**. Este estado foi definido como "inclusive", uma vez que se pretende extender e utilizar os estados anteriores.

- **t_header_LISTSIZE** Quando detectada uma lista (quer seja de tamanho fixo ou variável), a presente função detecta o seu tamanho mínimo e máximo (caso exista). Em seguida, coloca no array *context* as palavras *"inicioLista"* no índice correspondente ao inicio da lista, *"fimLista"* nos casos em que a lista possa terminar, e apenas *"lista"* entre ambas. Em seguida será dado um exemplo, para que melhor se entenda esse método.
- **t_header_SEPARATOR** Quando detectada uma virgula (separador entre colunas), a função aumenta o indice em uma unidade, correspondendo assim à respetiva coluna. É de ressalvar que no caso das listas de tamanho variável (que também recorrem a virgulas para a separação) não chega a entrar neste token, uma vez que antes de verificar esta situação confirma se não pertence ao token LISTSIZE.

- **t_header_DATA** Aqui é detectado todo o conteúdo geral do cabeçalho, sendo utilizada a expressão *"normal"* para agregar no array *context*, exemplificando-se em seguida a sua utilização junto das expressões descritas em *t_header_LISTSIZE*.

  Para além do referido anteriormente, também conta com um array *headers*, onde se guarda o nome de cada coluna, para que seja posteriormente utilizado em cada linha do ficheiro JSON.

  Ao receber o seguinte header:

  ```
  Número,Nome,Curso,Notas{3,5},,,,,
  ```

  O array *context* resultante seria:

  ```
  ['normal', 'normal', 'normal', 'inicioLista', 'lista', 'fimLista', 'fimLista', 'fimLista'
  ```

  E o array *headers* resultante seria:

  ```
  ['Número', 'Nome', 'Curso', 'Notas', 'Notas', 'Notas', 'Notas', 'Notas']
  ```

- **t_header_NEWLINE** Nesta função são tomados os procedimentos para quando ocorre a primeira mudança de linha, ou seja, voltar ao state *INITIAL*, e começar o processamento do corpo e a sua escrita no ficheiro JSON (que será explicado na sub-secção seguinte), assim com resetar o valor do índice.

- **t_header_AGGREGATION** Este token procura encontrar a presença de funções de agregação, através da notação referida anteriormente. Quando é encontrada uma operação de agregação, é registado no array *aggregation* o nome da operação nos índices na qual se aplica, tal como acontece no array *context*. Assim, estão disponíveis as operações de soma (**sum**), média (**media**), máximo (**max**) e mínimo (**min**).

  Exemplo:

  ```
  Número,Nome,Curso,Notas{3,5}::sum,,,,,
  ```

  E o array *aggregation* ficaria do seguinte modo:

  ```
  ['no_Agregation', 'no_Agregation', 'no_Agregation', 'sum', 'sum', 'sum', 'sum', 'sum']
  ```

3.2 Processamento do corpo

O corpo do CSV será processado com um conjunto de linhas que serão iteradas uma a uma. Para cada linha, e para cada coluna, apanha-se o valor com o token *t_DATA* e, utilizando o seu índice, verificamos qual o contexto através do array *context* construído no processamento do cabeçalho, segue-se os seguintes passos dependendo se o contexto for:

- **normal** - Neste caso, escreve-se diretamente no ficheiro JSON, utilizando os padrões deste formato. Segue um exemplo, onde $nome\_coluna$ é extraído do array *headers* com o respetivo índice e o $valor$ é capturado pelo *token t_DATA*:

```
nomeColuna : valor
```

- **inicioLista** - Aqui, começa-se o estado *"listReader"*. Neste estado, lêem-se todos os valores do ficheiro CSV, colocando-os num array com o nome *opList* (lista à qual poderá ser aplicada uma operação) até chegarmos ao índice cujo *context* seja "fimLista", ou caso especial de ser só uma lista com um elemento, ou seja, se o índice seguinte do array *context* for "normal", voltando ao estado "INITIAL".
- **lista** - neste caso encontra-se ativo o estado *"listReader"*, adicionando só o valor ao array *opList,* que guarda os valores da lista.
- **fimLista** - neste caso, encontra-se ativo o estado *"listReader"*, colocando o valor no array *opList*. Em seguida, retorna-se ao estado *INITIAL*. Antes de voltar ao estado "INITIAL", se no índice atual do array *aggregation* for "no_Aggregation", escrevemos no ficheiro JSON a lista:

```
nomeColuna : [todos valores que temos no array opList]
```

, caso contrario aplica-se a função de agregação guardada no *array aggregation* e escreve-se no ficheiro o resultado obtido pela função de agregação.

```
nomeColuna_nomeFunçãoAgregação : (resultado obtido função agregação)
```

Capítulo 4

Testes e Resultados

4.1 Procedimento para obter ficheiro JSON a partir de CSV

Para obter o resultado pretendido, é necessário passar como argumento o nome do ficheiro CSV, como se exemplifica no seguinte comando:

```
python3.9 csv2json.py filename.csv
```

4.2 Resultados

Para demonstrar a eficácia do programa, segue um exemplo que encapsula todas as funcionalidades do conversor implementado:

4.2.1 Ficheiro CSV de Input

```
Numero,Nome,Cursos{3,5},,,,,Notas{3,5},,,,,
3212132,Faisca Mcqueen,Teatro,Politica,Arquitertura,,,12,13,,,
7734234324,Cristina Ferreira,Musica,Engenharia,Matematica,,,17,12,20,11,12
34234223,Afonso Henriques,Enfermagem,RI,Medicina,Comunicacao,,18,19,19,20,
```

4.2.2 Ficheiro JSON de Output

```
[
        {
                "Notas" : [12,14],
                "Numero" : "3162",
                "Nome" : "Candido Faisca",
                "Curso" : "Teatro",
                "Idade" : "99"
        },
        {
                "Notas" : [17,12,20,11,12],
                "Numero" : "7777",
                "Nome" : "Cristiano Ronaldo",
                "Curso" : "Desporto",
                "Idade" : "99"
        },
        {
                "Notas" : [18,19,19,20,18],
                "Numero" : "264",
                "Nome" : "Marcelo Sousa",
                "Curso" : "Ciencia Politica",
                "Idade" : "99"
        }
]
```

Nota: É de salientar que neste exemplo, existe um erro nos limites da primeiro lista, como tal, no terminal após a execução aparecerá a seguinte mensagem de erro:

```
O ficheiro CSV possui um erro nos limites da lista na linha 2.
```

Capítulo 5

Conclusão

Dado os resultados dos testes, pode-se concluir que o programa consegue converter correctamente ficheiros CSV de vários tipos para JSON. Considera-se que o programa desenvolvido não só atinge os objetivos propostos, suportando um conjunto de funcionalidades adicionais, sem comprometer a velocidade e desempenho do mesmo.

Com o presente trabalho considera-se que se obteve uma boa implementação do enunciado proposto, conseguindo aplicar e consolidar em todo o momento os conhecimentos lecionados tanto na linguagem Python, bem como os seus módulos *'re'* e *'ply'*.