Processamento de Linguagens

André Gonçalves Vieira
A90166

José Pedro Castro Ferreira
A89572

Simão Paulo da Gama Castel-Branco e Brito
A89482

(29 de março de 2021)

Resumo

Neste relatório, descrevemos a implementação de um conversor de ficheiros *.csv* para ficheiros *.json*.
Utilizando a linguagem de programação *Python* e o módulo *re* da mesma, a implementação do conversor baseia-se na utilização de Expressões Regulares.

Conteúdo

1 Introdução **1**

2 Especificação do Problema **2**
2.1 Enunciado 2
2.2 Formato CSV 2
2.3 Formato JSON 3

3 Implementação da Solução **4**
3.1 Leitura do Ficheiro de Input 4
3.2 Leitura dos Campos de Dados 4
3.3 Leitura e Tratamento dos Dados 6
3.4 Tratamento de Listas 6
3.5 Execução de Funções Sobre Listas 8
3.6 Produção do Ficheiro de Output 8

4 Testes e Resultados Obtidos **9**

5 Guia de Utilização **11**

6 Conclusão **11**

1 Introdução

No âmbito da Unidade Curricular de Processamento de Linguagens, foi proposto o desenvolvimento de um conversos de ficheiros do tipo *.csv* para ficheiros do tipo *.json* em *Python*, utilizando Expressões Regulares. Para tal, aplicamos os conhecimentos adquiridos nas aulas sobre Expressões Regulares.

Neste relatório vamos descrever em detalhe as decisões tomadas assim como as expressões regulares e os métodos do módulo *re* que utilizamos para atingir o produto final desejado.

O relatório está organizado da seguinte forma. Na Secção 2 abordamos o enunciado e os objetivos propostos para o problema. Depois, na Secção 3, passamos a explicar em concreto a nossa implementação para a solução do problema.

Na Secção 4 apresentamos alguns testes que fizemos e o resultado da execução da conversão de alguns ficheiros de testes. e na Secção 5 apresentamos um guia de utilização do conversor.

Por fim a Secção 6 apresenta uma síntese do artigo e as conclusões discutindo os resultados obtidos.

2 Especificação do Problema

2.1 Enunciado

O problema resolvido pelo nosso grupo foi o Enunciado 4: "Conversor genérico de CSV para JSON". Este enunciado tem como base construir um conversor capaz de transformar qualquer ficheiro com formato CSV para um ficheiro de formato JSON.

O dataset fornecido poderá também conter listas em algumas células. No entanto, o cabeçalho do ficheiro de texto em formato CSV terá um asterisco '*****' depois do nome do respetivo campo, de modo a transformar os valores da coluna numa lista ao converter no formato JSON.

Após o asterisco poderá, também, haver uma função de aglomeração, sendo estas:

- **sum**: Apresenta o somatório das notas;
- **avg**: Apresenta a média das notas;
- **max**: Apresenta a nota máxima;
- **min**: Apresenta a nota mínima;

No caso de exitir alguma destas funções, no local acima indicado, o conversor deverá aplicar a operação respetiva à lista e gerar o ficheiro em formato JSON de acordo com a indicação fornecida.

2.2 Formato CSV

Um ficheiro em formato **CSV** (*Comma Separated Values*) é um ficheiro que, inicialmente e frequentemente, é utilizado para tranformar uma Folha de Cálculo num ficheiro de texto.

```
número;nome;curso;notas*
A71823;Ana Maria;MIEI;(12,14,15,18)
A89765;João Martins;LCC;(11,16,13)
A54321;Paulo Correia;MIEFIS;(17)
```

Figura 1: Ficheiro em formato CSV.

2.3 Formato JSON

Um ficheiro em formato **JSON** (*JavaScript Object Notation*) é um ficheiro que utiliza um formato textual neutro e simples, com base no conceito de um conjunto de pares {"*campo*": "*valor*"}, sendo um concorrente do formato XML.

```
[
    {
        "número": "A71823",
        "nome": "Ana Maria",
        "curso": "MIEI",
        "notas": [12,14,15,18]
    },
    {
        "número": "A89765",
        "nome": "João Martins",
        "curso": "LCC",
        "notas": [11,16,13]
    },
    {
        "número": "A54321",
        "nome": "Paulo Correia",
        "curso": "MIEFIS",
        "notas": [17]
    }
]
```

Figura 2: Ficheiro em formato JSON.

3 Implementação da Solução

O nosso conversor está preparado para receber o nome do Ficheiro de Input como argumento ao ser executado a partir do terminal e vai gerar um ficheiro de igual nome com a extensão *.json* com o resultado da conversão.

3.1 Leitura do Ficheiro de Input

Inicialmente, utilizamos uma variável *file_name* para guardar o nome do ficheiro passado como argumento (*sys.argv[1]*). De seguida, abrimos esse ficheiro com permissões de leitura.

```
file_name = sys.argv[1]
f = open(file_name,"r",encoding="utf-8")
```

Figura 3: Leitura do nome do Ficheiro de Input e Leitura do Ficheiro

De seguida, vamos utilizar o método *readlines()* aplicado ao ficheiro de Input e passar todas as linhas lidas do ficheiro para uma lista. Mais à frente, ao escrever o ficheiro *.json*, convém que conseigamos identificar qual a última linha do ficheiro de Input. Com esta estratégia de ler para uma lista, mais facilmente identificamos a última linha do ficheiro.

```
fLines = f.readlines()
```

Figura 4: Leitura das linhas do ficheiro para uma lista

3.2 Leitura dos Campos de Dados

Concluída a leitura das linhas do ficheiro, decidimos começar a sua análise por definir logo de início os campos de dados que vamos ter. Relembrando que no ficheiro *JSON*, a informação é apresentada como {*"campo":"valor"*}. Estes campos estão indicados na primeira linha do ficheiro *.csv*, ou seja, na posição de índice 0 da lista que contém todas as linhas do ficheiro que lemos.

No ficheiro de *input*, todos os campos e dados aparecem separados pelo caractér ';', exceto os últimos de cada linha que são seguidos pelo caracter '\n'. Assim sendo, vamos utilizar o método *re.split* na posição 0 da lista e a ER para fazer o *split* foi *r';|\n'*. Ao fazer este split, reparamos que no final da lista de *Strings* resultante, aparecia uma *String* vazia. Por isso fazemos um *pop()* na lista resultante que, se não lhe for passado um índice, remove o último elemento da lista (Ver Figura 5).

```
headers = re.split(r';|\n',fLines[0])
headers.pop()
```

Figura 5: Leitura das linhas do ficheiro para uma lista

Como já vimos na Secção 2, os ficheiros *.csv* podem conter listas. Sabemos que um dado campo representa uma lista se o seu nome contiver um caracter '*'. Decidimos identificar logo à partida quais os campos que vão receber listas como dados. Isto é possível criando uma lista *listsIndex* que vai guardar os índices dos campos que contêm o caracter '*' no seu nome (Ver figura 6).

```
listsIndex = []
headers_size = len(headers)
for i in range(headers_size):
    if '*' in headers[i]:
        listsIndex.append(i)
```

Figura 6: Identificação dos Campos que vão receber Listas

Depois de feito o tratamento da primeira linha do ficheiro, podemos remover essa linha da lista que contém todas as linhas do ficheiro, uma vez que a informação nela contemplada já está guardada na lista *headers*. Vamos criar uma variável *lines_checked* que contará quantas linhas já foram analisadas e uma outa variável *lines_total* que terá o número total de linhas de dados, isto é, o número total de linhas do ficheiro - 1, que é a linha dos cabeçalhos. Por último, vamos aos cabeçalhos que vão receber listas (cujos índices estão guardados na lista *listsIndex*) e vamos usar o método *re.sub* para subtituir o caracter '*' pelo caracter '_'.

```
fLines.pop(0)
lines_checked = 0
lines_total = len(fLines)

for i in listsIndex:
    headers[i] = re.sub(r'\*','_',headers[i])
```

Figura 7: Utilização do método *re.sub* e criação das variáveis auxiliares

3.3 Leitura e Tratamento dos Dados

Para tratar os dados lidos para a lista que contém todas as linhas do ficheiro, vamos iterar pelos índices da lista. Criamos em cada iteração uma lista *values* e nesta lista vamos guardar o resultado de executar um split com a mesma ER que usamos para os campos de dados (*re.split(r';|\n',fLines[i])*). Por vezes, a execução deste método deixa na última posição da lista uma *String* vazia. Se tal se verificar, damos *pop()* na lista, de forma a elimiar esse elemento.

No final de cada iteração vamos executar o método *printJson(headers,values)* para escrever no ficheiro *.json* o bloco de informação correspondente à linha de dados que acabamos de ler.

```
for i in range(lines_total):
    lines_checked = lines_checked + 1
    values = []
    values = re.split(r';|\n',fLines[i])
    if(values[len(values)-1] == ''):
        values.pop()
```

Figura 8: Utilização do método *split* em cada linha do ficheiro

3.4 Tratamento de Listas

Vamos percorrer os elementos da lista *headers* cujos índices pertencem à lista *listsIndex*. Começamos por passar a informação correspondente a esses campos para uma lista, separando os elementos usando o método *re.split()*. Este split coloca uma *String* vazia na primeira e na última posição da lista, pelo que vamos fazer *pop()* e *pop(0)*.

```
for n in listsIndex:
    aux = re.split(r'\(|\)|\,',values[n])
    aux.pop()
    aux.pop(0)
```

Figura 9: *Split* da informação que representa uma lista

Vamos separar os campos de cada elemento pelo caracter '_' usando o método *re.split()* e passar os campos para a lista *funcs*. Se o elemento de índice 1 de *funcs* for uma *String* vazia, sabemos que apenas temos de apresentar a lista conforme a recebemos, corrigindo apenas a sintaxe (o que é feito pelo split apresentado na Figura 9). Se for este o caso, podemos substituir o caracter '_' por uma *String* vazia no cabeçalho. Isto é feito executando o método *re.sub(r'_','',headers[n])*. Vamos ainda verificar os tipos dos dados da lista através dos métodos **Repre-**

sentsInt e **RepresentsFloat**, que verificam se os dados podem representar Inteiros ou Floats, respetivamente. Se puderem, passamos os elementos de String para o tipo que puderem representar. No final, a lista resultante será adicionada aos campos da lista *values* no índice n.

```
if(funcs[1] == ''):
    headers[n] = re.sub(r'_','',headers[n])
    if RepresentsInt(aux[0]):
        listInts = []
        for index in range(len(aux)):
            aux[index] = int(aux[index])
    elif RepresentFloat(aux[0]):
        listFloats = []
        for index in range(len(aux)):
            aux[index] = float(aux[index])
    values[n] = aux
```

Figura 10: Transformação dos Tipos da Lista

Se *funcs[i]* não for uma *String* vazia, quer dizer que esse elemento é a função que devemos aplicar sobre a lista. Vamos proceder da mesma forma que fizemos para o caso anterior no que toca à transformação dos tipos dos dados da lista com a diferença de apenas aceitarmos inteiros e floats. Se o tipo de dados não for um desses, o resultado vai ser a *String* "INVALID_INPUT".
No final, vamos extrair a função que temos de executar para a variável *func* e vamos pôr em *values[n]* o resultado da execução do método *funcDef(func,aux)* em que *func* é a função a executar e *aux* é a lista sobre a qual vamos executar a função.

```
else:
    if RepresentsInt(aux[0]):
        listInts = []
        for index in range(len(aux)):
            aux[index] = int(aux[index])
    elif RepresentFloat(aux[0]):
        listFloats = []
        for index in range(len(aux)):
            aux[index] = float(aux[index])
    else:
        aux = "INVALID_INPUT"
    func = funcs[1]
    values[n] = funcDef(func, aux)
```

Figura 11: Transformação dos Tipos da Lista

3.5 Execução de Funções Sobre Listas

Nesta parte vamos explicar o método *funcDef(func,listArg)*. Este método recebe uma *String* **func** que é a função que vamos executar e uma lista **listArg**, sobre a qual vamos executar a função.
Se o argumento **listArg** recebido for uma *String*, vamos simplesmente retornar essa *String*.
Abaixo apresentamos a definição deste método que, dada a sua simplicidade, é passível de uma explicação muito extensa.

```
def funcDef(func,listArg):
    if listArg is str:
        return listArg
    n = len(listArg)
    total = 0
    if func == "sum":
        for i in range(n):
            total = total + listArg[i]
    elif func == "avg":
        total = funcDef("sum",listArg)/len(listArg)
    elif func == "max":
        total = max(listArg)
    elif func == "min":
        total = min(listArg)
    else:
        total = -1
    return total
```

Figura 12: Definição do método *funcDef(func,listArg)*

3.6 Produção do Ficheiro de Output

Logo após a abertura do ficheiro de *input*, vamos abrir também um ficheiro com nome igual ao de *input* com exntensão *.json* com permissões de escrita.
Antes do tratamento das linhas de dados, vamos escrever no ficheiro de *output* o caracter '['. No final do programa vamos escrever também o caracter ']'. Para cada linha de informação vamos executar o método printJson(headers,values). Temos de seguir as regras de sintaxe dos ficheiros *.json* como a escrita de vírgulas no final de todas as linhas de dados exceto a última, a escrita de vírgulas a seguir a todos os blocos de informação exceto o último, entre outras. É também importante contemplar que um valor do tipo **float** seja limitado a duas casas decimais. Abaixo apresentamos a definição deste método.

```
def printJson(headers,values):
    json.write("\n\t{")

    for i in range(headers_size):
        if i + 1 == len(headers):
            if(isinstance(values[i],float)):
                fNota = "{:.2f}".format(values[i])
                json.write(f"\n\t\t\"{headers[i]}\": {fNota}")
            elif(isinstance(values[i], int) or isinstance(values[i], list)):
                json.write(f"\n\t\t\"{headers[i]}\": {values[i]}")
            else:
                json.write(f"\n\t\t\"{headers[i]}\": \"{values[i]}\"")
        else:
            if(isinstance(values[i],float)):
                fNota = "{:.2f}".format(values[i])
                json.write(f"\n\t\t\"{headers[i]}\": {fNota},")
            elif(isinstance(values[i], int) or isinstance(values[i], list)):
                json.write(f"\n\t\t\"{headers[i]}\": {values[i]},")
            else:
                json.write(f"\n\t\t\"{headers[i]}\": \"{values[i]}\",")
```

Figura 13: Definição do método *printJson(headers,values)*

4 Testes e Resultados Obtidos

Após o desenvolvimento da solução descrita na Secção 3, desenvolvemos alguns ficheiros de teste:

- **test3L.csv** - 4 campos e 3 linhas de informação
- **test568L.csv** - 4 campos e 568 linhas de informação
- **test2107L.csv** - 4 campos e 2107 linhas de informação
- **test10C.csv** - 10 campos e 106 linhas de informação

Em cada um dos ficheiros existe pelo menos um campo que representa uma lista ou a execução de uma função sobre uma lista. Assim garantimos o funcionamento do conversor para qualquer ficheiro com extensão *.csv*.
Após a execução dos testes, verificamos que os ficheiros de output correspondiam ao pretendido:

- **test3L.json** - 3 blocos com 4 campos de informação
- **test568L.json** - 568 blocos com 4 campos de informação
- **test2107L.json** - 2107 blocos com 4 campos de informação
- **test10C.json** - 106 blocos com 10 campos de informação

Assim, conseideramos que os testes mostram que o conversor funciona para qualquer ficheiro de extensão *.csv*.

```
numero;nome;curso;notas*max
A71823;Ana Maria;MIEI;(12,14,15,18)
A89765;João Martins;LCC;(11,16,13)
A54321;Paulo Correia;MIEFIS;(17)
```

Figura 14: Ficheiro **test3L.csv**

Figura 15: Ficheiro **test3L.json**

5 Guia de Utilização

Para utilizar o conversor que desenvolvemos basta executar o comando seguinte:

```
python3 csv2json.py fileName.csv
```

Figura 16: Comando de execução do conversor

Isto resultará na criação de um ficheiro *.json* com o nome do ficheiro passado como argumento, na diretoria do ficheiro passado como argumento.

6 Conclusão

Através da realização deste trabalho houve uma consolidação dos conhecimentos adquiridos nas aulas teóricas e teorico-práticas já lecionadas relativos ao uso de Expressões Regulares para reconhecimento de padrões em *Strings* lidas de ficheiros ou do input do utilizador.

Neste projeto tivemos também oportunidade de praticar e aprofundar o nosso conhecimento do módulo '*re*' da linguagem de programação *Python*, com a aplicação de métodos, tais como a *split()* e a *sub()*. Foi também uma boa experiência desenvolver esta aplicação em *Python*, uma vez que a boa documentação encontrada na Internet facilita bastante a utilização dos módulos desta linguagem. Sem grande experiência prévia no uso desta linguagem, não sentimos que a dificuldade do trabalho veio pela aprendizagem de uma nova linguagem, o que nos deixa concentrar completamente em resolver o problema em mãos, o que é algo muito positivo.

De uma forma geral, finalizamos este trabalho prático seguros de que o seu objetivo foi cumprido, sendo este um projeto no qual nos empenhamos e que realizamos com interesse.