Processamento de Linguagens

Trabalho Prático Nº1
ER & Filtros de Texto

Conversor genérico de CSV para JSON

Joana Afonso Gomes
(a84912)

Susana Marques
(a84167)

18 de junho de 2021

Resumo

No âmbito da Unidade Curricular de Processamento de Linguagens, foi-nos proposta a realização de um primeiro trabalho prático com vista a aprofundar conhecimentos no que toca à escrita de Expressões Regulares para descrição de padrões de frases dentro de texto, desenvolvendo a partir das mesmas Filtros de Texto que transformem textos com base no conceito de regras de produção Condição-Ação.

Paralelamente, e de forma a permitir o correto desenvolvimento e implementar os Filtros de Texto pedidos, é utilizado o módulo `er` de Python.

O enunciado atribuído ao nosso grupo foi um conversor genérico de um qualquer ficheiro gravado em formato CSV para o formato JSON.

Conteúdo

Capítulo 1

Introdução

Área: Processamento de Linguagens

1.1 Enquadramento & Contexto

Este primeiro trabalho prático tem como objetivo aprofundar a escrita de **Expressões Regulares** para descrição de padrões de frases dentro de textos, desenvolvendo, a partir destas, **Filtros de Texto**, que filtrem ou transformem textos com base no conceito de regras de produção Condição-Ação.

Com tal fim em vista, focando-nos na correta implementação dos Filtros de Texto, recorremos ao módulo **er** do Python e às suas funções de *search()*, *split()* e *sub()*.

1.2 Problema & Objetivo

O problema proposto no enunciado que nos foi atribuído passa pela criação de um conversor de ficheiros CSV para um ficheiro JSON.

Os CSV utilizado possuem uma estrutura tal que podem conter listas aninhadas em algumas células, sendo essas colunas convertidas em listas no JSON final, ou ainda funções *sum, avg, max, min* que são aplicadas através de um *fold* à lista, sendo no JSON apresentados os resultados de acordo.

1.3 Estrutura do documento

Neste relatório apresentamos a nossa abordagem ao enunciado que nos foi atribuído. O mesmo está divido nos seguintes capítulos:

No presente capítulo 1, **Introdução**, é feito o enquadramento a contextualização do problema, sendo referido simultaneamente quais os objetivos pretendidos no projeto.

No capítulo 2, **Análise e Especificação**, é descrito o problema e os requisitos para a sua correta resolução.

Logo depois, no capítulo 3, **Concepção da Resolução**, são apresentadas as estruturas de dados usadas e os algoritmos implementados no desenvolvimento do projeto.

De seguida, no capítulo 4, **Codificação e Testes**, apresentamos os testes realizados, o código Assembly gerado bem como o programa a correr na máquina virtual VM.

Findamos o relatório com o capítulo 5, **Conclusão**, analisando e sintetizando aquilo que foi apreendido com o desenvolvimento deste projeto.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O problema em questão consiste, como referido previamente, na conversão de um ficheiro CSV num ficheiro em formato JSON, sendo que a primeira linha do ficheiro CSV corresponde a um descodificador daquilo a que correspondem os valores apresentados nas restantes linhas do ficheiro.

O *dataset* contido no CSV poderá conter:

- Listas aninhadas nalgumas células, sendo o campo correspondente no cabeçalho completado com um *, devendo essa coluna ser convertida em listas no ficheiro JSON;

Exemplo:

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

- Acrescentando a esse * poderá haver uma função de agregação *sum, avg, max* ou *min*, sendo a respetiva operação aplicada com um *fold* à lista, e o JSON criado em concordância com isto.

Exemplo:

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

2.2 Especificação do Requisitos

Deduzimos os requisitos fundamentais do enunciado em questão como:

- Associar cada campo do cabeçalho aos seus respetivos valores;
- Descobrir os índices dos campos com os valores onde as listas aninhadas se encontram (se exister * no cabeçalho);
- Obter o resultado dos campos aplicando a função de agregação (*sum, avg, max, min*) caso o campo corresponda a uma lista aninhada com uma função.

Capítulo 3

Concepção da Resolução

3.1 Estruturas de Dados

As estruturas de dados usadas neste trabalho foram bastante simples, consistindo apenas em múltiplos *arrays* para guardar informações intermédias e um *array* final que irá guardar toda a informação para depois conseguir escrever para o ficheiro *json* criado. Só escreverá no final, o que em termos de eficiência se revela bastante melhor, já que que precisa de chamar a função uma única vez, ainda que, em termos de memória guardada em *cache*, poderá ser problemático se o ficheiro *.csv* for muito grande. Apesar deste *trade-off*, após investigação, o grupo achou por bem apenas escrever para o ficheiro no final, por ser mais eficiente em todos os testes feitos.

Estruturas de dados usadas:

func: *Array* que permite verificar qual ou quais são as funções (*sum, avg, max, min*) que se encontram no ficheiro *.csv* ou se não tem função (apresenta apenas a lista) ; somente para os campos que se encontram com *

listas_aninhadas: *Array* que contém o índice de cada um dos campos onde se encontra uma lista (em conjunto com o array *func* funciona como dicionário de *Python*)

campos: *Array* que contém cada um dos campos da primeira linha do ficheiro *.cvs*

linhas: *Array* com todas as linhas do ficheiro excluindo a primeira

valor: *Array* com os valores de cada linha do ficheiro que serão processados e tratados

json_finall: *Array* que irá conter o conteúdo final através de substituição para posteriormente poder escreve-lo no ficheiro *.json*

3.2 Algoritmos

Como as estruturas neste trabalho são consideradas simples, os algoritmos implementados são mais robustos mas na mesma de igual eficiência, uma vez que em programação existe sempre o conceito de que o uso de estruturas de dados é inversamente proporcional ao uso de algoritmos, ou seja quanto mais simples forem as estruturas, mais complicados os algoritmos serão e vice-versa.

A nossa implementação consistiu em preencher o array **json_finall** através de *regex*, usando a biblioteca do *Python:* ***re*** e substituindo usando para cada linha do ficheiro *.csv* (com a exceção da primeira linha) um

ciclo que coloca o *array* **campos** e o *array* **valor** já alterados previamente no lugar pedido, com a identação e símbolos pedidos pelo ficheiro *.json*. Para lermos cada linha do ficheiro, usamos também *regex* e um ciclo pelo ficheiro, e alteramos os valores inicias nos campos com listas aninhadas através das estruturas **func** e **listas_aninhadas**, que nos permite descobrir os índices dos campos com os valores onde as listas aninhadas se encontram e a seguir conseguirmos realizar uma função (*max,min,avg,sum*), caso o campo corresponda a uma lista aninhada com uma função. De forma concreta, o grupo implementou a sua versão de dicionários em *Python*, usando apenas os *arrays* **func** e **listas_aninhadas** obtendo a mesma eficiência.

Acerca dos campos que têm * na primeira linha, ainda existiu um aperfeiçoamento feito pelo grupo, para que no ficheiro *.json* esses mesmo campos fossem alterados para substituir o * por _, ou simplesmente se eliminasse o *, caso apenas se quisesse obter uma lista e não realizar uma função (*max,min,avg,sum*). Para todos os outros valores onde não se encontrem listas, apenas se colocou aspas antes e a seguir ao valor que se encontrava no ficheiro *.csv*, conforme pedido no enunciado.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

A nível de problemas de implementação, quisemos ser o mais gerais possíveis, para que seja possível ler qualquer ficheiro *.csv* com o formato pedido. Numa implementação inicial, apesar do uso de *regex* ser constante, ainda existia muitos passos do trabalho que estavam a ser feitos puramente em *Python* como por exemplo a eliminação da primeira linha do ficheiro *.csv* que depois se alterou usando a seguinte linha de código:

```
linhas = re.findall(r'(?<=[\n]).+',file)
```

A escrita no array **json_finall** também estava puramente em *Python*, sendo depois alterada para usar *regex*, concretamente a função *sub*.

Aquando da realização do trabalho, o grupo encontrou duas maneiras de extrair informação/valor de cada campo do ficheiro, usando-as em situações diferentes, para inferir a eficiência e os problemas de cada uma. No primeiro caso optamos por obter os valores da primeira linha através da função **findall**, que se encontra na biblioteca *re* de *Python*, como se pode verificar de seguida:

```
campos = re.findall(r'[^;]+', linha1.group())
```

No segundo caso, como no enunciado é pedido o uso de funções **split**, optamos por obter no projeto os diferentes valores de todas as outras linhas do ficheiro com a função **split**, que também se encontra na biblioteca *re* de *Python*:

```
valorr = re.split(r';',linha)
```

Neste último caso, se o último campo do ficheiro se encontrar com o carater **;** no final, a função **split** vai acabar por produzir um array onde existirá uma lista vazia no último campo do mesmo. Para contornar este problemas, usou-se a função *filter* que permite eliminar todas as listas vazias de um *array*:

```
valorr =list(filter(bool,valorr))
```

Com o objetivo referido de generalizar o projeto ao máximo, optamos primeiro por aceitar que o utilizador possa colocar o nome do ficheiro que deseja converter, tendo em conta os casos em que o ficheiro possa não existir ou não se encontre com a extensão correta (*.csv*), fazendo com que o ficheiro convertido se encontre com o mesmo nome que o ficheiro original, apenas com a sua extensão sendo *.json*. Ao longo do ficheiro,

tivemos o cuidado de tratar casos de exceção, como por exemplo a existência de campos nulos no ficheiro, ou, no caso do campo ser listas, existirem carateres nesse campo entre parêntesis e separado por vírgulas que não sejam números no caso de se pedir uma função (como por exemplo a soma dos números, uma vez que é impossível somar letras ou outros caráteres dando um resultado plausível, por outro lado, se for apenas pedido a lista, ela irá ser produzida com qualquer tipo de caráter que exista separado por vírgulas).

Tivemos aida em conta verificar se em qualquer linha, incluindo a primeira, se usar o caráter **;** ou não se usar no final dessa linha, não irá afetar o nosso programa, e se quisermos ter várias funções iguais (por exemplo, várias somas de listas diferentes na mesma linha do *.csv*) isso é possível, sem afetar nenhum resultado.

4.2 Testes realizados e Resultados

Apresentamos de seguida alguns testes feitos (valores introduzidos) e os respectivos resultados obtidos, que se encontram na pasta enviada junto com o trabalho com o mesmo nome:

Teste: example1.csv:

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

Resultado: example1.json:

```
[
        {
                "numero": "A71823"
                "nome": "Ana Maria"
                "curso": "MIEI"
                "notas": [12,14,15,18]
        },

        {
                "numero": "A89765"
                "nome": "Joao Martins"
                "curso": "LCC"
                "notas": [11,16,13]
        },

        {
                "numero": "A54321"
                "nome": "Paulo Correia"
                "curso": "MIEFIS"
                "notas": [17]
        }
]
```

Teste: example2.csv:

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

Resultado: example2.json:

```
[
        {
                "numero": "A71823"
                "nome": "Ana Maria"
                "curso": "MIEI"
                "notas_avg": 14.75
        },

        {
                "numero": "A89765"
                "nome": "Joao Martins"
                "curso": "LCC"
                "notas_avg": 13.33
        },

        {
                "numero": "A54321"
                "nome": "Paulo Correia"
                "curso": "MIEFIS"
                "notas_avg": 17.0
        }
]
```

Nestes primeiros dois testes, usamos os ficheiros provenientes do enunciado, com objetivo de nos ajudar a perceber que toda a implementação está feita de forma correta.

Teste: example3.csv:

```
    numero;nome;curso;notas*;notas*avg;notas*sum;notas*max;notas*min;notas*mddn;notas
        *min
    A71823;Ana Maria;MIEI;(12,14,15,18);(12,14,15,18);(12,14,15,18);(12,14,15,18)
        ;(12,14,15,18);(12,14,15,18);(12,14,15,18);
    A89765;Joao Martins;LCC;(11,16,13);(11,16,13);(11,16,13);(11,16,13);(11,16,13)
        ;(11,16,13);(11,16,13)
    A54321;Paulo Correia;MIEFIS;(17);(17);(17);(17);(17);(17);(17)
    A53212;Mariana Carvalho;MIEI;(17,18.4);(17,18.4);(17,18.4);(17,18.4);(17,18.4)
        ;(17,18.4);(17,18.4)
    A87192;Francisco Couto;LCC;12;12;12;12;12;12;12
```

Resultado: example3.json:

```
[
        {
                "numero": "A71823"
                "nome": "Ana Maria"
                "curso": "MIEI"
                "notas": [12,14,15,18]
                "notas_avg": 14.75
                "notas_sum": 59.0
                "notas_max": 18.0
                "notas_min": 12.0
                "notas": [12,14,15,18]
                "notas_min": 12.0
        },
```

```
14
15          {
16                  "numero": "A89765"
17                  "nome": "Joao Martins"
18                  "curso": "LCC"
19                  "notas": [11,16,13]
20                  "notas_avg": 13.33
21                  "notas_sum": 40.0
22                  "notas_max": 16.0
23                  "notas_min": 11.0
24                  "notas": [11,16,13]
25                  "notas_min": 11.0
26          },
27
28          {
29                  "numero": "A54321"
30                  "nome": "Paulo Correia"
31                  "curso": "MIEFIS"
32                  "notas": [17]
33                  "notas_avg": 17.0
34                  "notas_sum": 17.0
35                  "notas_max": 17.0
36                  "notas_min": 17.0
37                  "notas": [17]
38                  "notas_min": 17.0
39          },
40
41          {
42                  "numero": "A53212"
43                  "nome": "Mariana Carvalho"
44                  "curso": "MIEI"
45                  "notas": [17,18.4]
46                  "notas_avg": 17.7
47                  "notas_sum": 35.4
48                  "notas_max": 18.4
49                  "notas_min": 17.0
50                  "notas": [17,18.4]
51                  "notas_min": 17.0
52          },
53
54          {
55                  "numero": "A87192"
56                  "nome": "Francisco Couto"
57                  "curso": "LCC"
58                  "notas": [12]
59                  "notas_avg": 12.0
60                  "notas_sum": 12.0
61                  "notas_max": 12.0
62                  "notas_min": 12.0
63                  "notas": [12]
64                  "notas_min": 12.0
65          }
66  ]
```

Neste terceiro teste, foi-nos possível verificar casos que foram tratados no programa para o tornar o mais geral possível, como por exemplo:

1) Se existir **;** no final de certas linhas ou não, o programa continua a funcionar normalmente

2) Se depois do * a palavra não for reconhecida como uma função (*sum,avg,min,max*), tudo o que está depois do * é truncado e apenas devolve a lista, tal e qual como se não estivesse nada a seguir ao * (neste caso: *notas_mddn*)

3) Poderá existir dois campos que executem a mesma função (neste caso: *notas_min*)

Teste: example4.csv:

```
1  numero;nome;curso;notas*avg
2  A71823;Ana Maria;MIEI;(w,14,15,18)
3  A89765;Joao Martins;LCC;
4  A54321;Paulo Correia;MIEFIS;(17)
```

Resultado: example4.json:

```
1
2 (Ficheiro vazio)
```

Este exemplo consiste na verificação de casos de exceção onde é impossível gerar o ficheiro, não por ele não existir ou por ter uma extensão diferente (apesar desse caso também ser tratado), mas porque existem valores nulos no ficheiro, como se pode verificar pelo resultado obtido no terminal após tratamento de exceções:

```
O ficheiro csv encontra-se com carateres que não são números dentro de listas onde se
pretende que tenham números, por favor tente de novo depois de compor o ficheiro .csv
Existem valores nulos no ficheiro.csv!
```

Com esta resposta conseguimos observar que é possível verificar que existem listas onde os caráteres não são todos números. Apesar deste aviso, se não fosse por haver valores nulos, estas listas seriam na mesma escritas no ficheiro, apesar de não se conseguir fazer a função pretendida e apenas serem escritas como uma lista normal com caráteres diferentes de números dentro delas, tal e qual como se fosse apenas pedido para escrever a lista (* sem mais nada a seguir).

Capítulo 5

Conclusão

Este trabalho contribuiu amplamente para a aprendizagem do grupo no que toca à elaboração de Filtros de Texto (com base em regras Condição-Ação) e para a consolidação do conhecimento prático e teórico sobre produção de Expressões Regulares adquirido nas aulas.

Tendo em conta o enunciado atribuído, consideramos que foram alcançados os objetivos propostos, julgando, portanto, na sua globalidade, um desfecho positivo na resolução do problema em questão.

Apêndice A

Código do Programa

Lista-se a seguir o código do programa que foi desenvolvido.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import re
def csv2json(file, filename):

    # array que permite verificar qual ou quais sao
    # as funçoes (sum, avg, max, min) que se encontram no csv ou se nao tem funçao ('')
    func=[]
    # array que contém o indice de cada um dos campos do array func
    listas_aninhadas =[]

    #create a json file
    #lê o nome do ficheiro .csv e cria um ficheiro .json com o mesmo nome
    json_f = re.search(r'(.*).csv',filename).group(1)
    js_f = json_f + '.json'
    json_final = open(js_f, "w+")


    #linha 1 do ficheiro
    linha1 = re.search(r'((.*))',file)
    #tambem poderia ser definido com split
    campos = re.findall(r'[^;]+', linha1.group())

    # campos é um array que poderá ter um numero infinito
    # onde se encontra cada um dos campos do ficheiro da linha 1

    for campo in campos:
        if(re.search(r'\*(avg|sum|min|max)',campo)):
            # coloca o nome da funçao no array func e o indice
            # do campo onde essa funcao se encontra no array listas_aninhadas
            func.append(re.search(r'\*(.+)',campo).group(1))
            listas_aninhadas.append(campos.index(campo))
```

```
        # substitui notas*avg por notas_avg
        campo1 = re.sub(r'\*','_',campo)
        campos[campos.index(campo)] = campo1

    # coloca o nome da funcao (neste caso '', ja que nao existe) no array func e o
    # indice do campo onde essa funcao se encontra no array listas_aninhadas
    elif(re.search(r'\*.*',campo)):
        # ao indicar apenas '' colocamos como se fosse uma flag que nos
        # permite saber que nao iremos usar nenhuma função,
        # apenas mostrar as listas alinhadas
        func.append('')
        listas_aninhadas.append(campos.index(campo))
        # substitui notas* por notas
        campo1=re.sub(r'\*.*','',campo)
        campos[campos.index(campo)] = campo1


# array de array de linhas do ficheiro, excluindo a primeira
linhas = re.findall(r'(?<=[\n]).+',file)

## (?<=[\n]).* verifica se tudo o que esta precedente do .* é
## uma nova linha. (positive look-behind) caso seja vai ser encontrado
## tudo exluindo a primeira linha. usamos .+ e não .* devido a espaços brancos##


# criação de um array que irá ter o conteúdo final do json criado
json_finall =''

# primeira sibstutiçao no json final pelo primeiro carater de um ficheiro json: [
json_finall = re.sub('\Z','[',json_finall)

# por cada linha do ficheiro .csv
for linha in linhas:
    # array com todos os valores de cada campo da linha
    valorr = re.split(r';',linha)

    # ao invês da função filter() poderiamos tb usar o findall() como feito em cima
    # para produzir o mesmo resultado com regex; mas como no
    # enunciado nos é pedido o uso de funções split
    # no trabalho, optamos por resolver estes dois problemas
    # idênticos de forma diferente para abordarmos
    # uma forma mais geral da resolução

    # filtra todos as listas vazias no array
    valorr =list(filter(bool,valorr))

    # se existir um campo em branco
    if len(valorr)!=len(campos):
```

```
    print('Existem valores nulos no ficheiro.csv!')
    return


# acrescentar aspas no inicio e final de cada valor,
# criando um novo array com todos os valores
valor = [re.sub(r'^','"',v) for v in valorr]
valor = [re.sub(r'$','"',v) for v in valor]
for i in range(len(func)):
    if(func[i] =='sum'):
        # substitui a lista onde o campo é primeira linha é 'sum' pela sua soma
        valor[listas_aninhadas[i]] = re.sub(r'"|\(|\)','',valor[listas_aninhadas[i]])
        # cria lista para usar o sum
        li = list(valor[listas_aninhadas[i]].split(","))
        try:
            li=[float(i) for i in li]
            valor[listas_aninhadas[i]]= str(round(sum(li),2))
        except Exception:
             print('O ficheiro csv encontra-se com carateres que não são números
                dentro de listas onde se pretende que tenham números, por favor
                tente de novo depois de compor o ficheiro .csv')


    if(func[i] =='avg'):
        # substitui a lista onde o campo da primeira linha é 'avg' pela sua média
        valor[listas_aninhadas[i]] = re.sub(r'"|\(|\)','',valor[listas_aninhadas[i]])
        # cria lista para calcular a média
        li = list(valor[listas_aninhadas[i]].split(","))
        try:
            li=[float(i) for i in li]
            '''average'''
            valor[listas_aninhadas[i]]= str(round(sum(li)/len(li),2))
        except Exception:
             print('O ficheiro csv encontra-se com carateres que não são números
                dentro de listas onde se pretende que tenham números, por favor
                tente de novo depois de compor o ficheiro .csv')


    if(func[i] =='max'):
        # substitui a lista onde o campo da primeira linha é 'max' pelo seu valor máximo
        valor[listas_aninhadas[i]] = re.sub(r'"|\(|\)','',valor[listas_aninhadas[i]])
        # cria lista para usar o max
        li = list(valor[listas_aninhadas[i]].split(","))
        try:
            li=[float(i) for i in li]
            valor[listas_aninhadas[i]]= str(round(max(li),2))
        except Exception:
                print('O ficheiro csv encontra-se com carateres que não são números
```

```
                    dentro de listas onde se pretende que tenham números, por favor
                    tente de novo depois de compor o ficheiro .csv')


        if(func[i] =='min'):
            # substitui a lista onde o campo da primeira linha é 'min' pelo seu valor minimo
            valor[listas_aninhadas[i]] = re.sub('"|\(|\)','', valor[listas_aninhadas[i]])
            # cria lista para usar o min
            li = list(valor[listas_aninhadas[i]].split(","))
            try:
                li=[float(i) for i in li]
                valor[listas_aninhadas[i]]= str(round(min(li),2))
            except Exception:
                 print('O ficheiro csv encontra-se com carateres que não são números
                    dentro de listas onde se pretende que tenham números, por favor
                    tente de novo depois de compor o ficheiro .csv')

        elif(func[i]==''):
            valor[listas_aninhadas[i]]= re.sub(r'"','',valor[listas_aninhadas[i]])
            valor[listas_aninhadas[i]] = re.sub('\(|^','[', valor[listas_aninhadas[i]])
            valor[listas_aninhadas[i]] = re.sub('\)|$',']', valor[listas_aninhadas[i]],1)

    # substitui no array json final no final da string por {
    # para cada linha do ficheiro .csv
    json_finall = re.sub('\Z','\n\t{\n',json_finall)
    i=0
    while(i!=len(valor)):
        # para cada linha do .csv substitui no array
        # do json_finall no final pelos valores dessa linha e
        # campos já alterados previamente
        json_finall = re.sub('\Z','\t\t"'+campos[i]+'": '+valor[i]+'\n',json_finall)

        i=i+1
    # substitui por }, para cada linha
    json_finall = re.sub('\Z','\t},\n',json_finall)

#no final transformar a ultima virgula e \n por \n e ]
json_finall = re.sub(',\n\Z','\n]',json_finall)

# print para verificar que o ficheiro lê corretamente.
# retirar quando o ficheiro for demasiado grande por questões de performance
print(json_finall)

# escreve o array json_final para o ficheiro json criado e depois fecha-o
json_final.write(json_finall)
json_final.close()
```

```
    ##PROGRAMA COMEÇA AQUI##
while(True):
    print("Inserir ficheiro .csv como parâmetro pra ser convertido num ficheiro .json")
    inputFromUser = input("> ")
    #verifica se o input termina em .csv
    if(re.search('.csv$',inputFromUser)):
        try:
            #abre o ficheiro se ele existir e caso exista lê o conteúdo
            with open(inputFromUser) as f:
                content = f.read()
                #função que permite converter o ficheiro .csv em .json
                csv2json(content, inputFromUser)
        except getattr(__builtins__,'FileNotFoundError', IOError):
            print("Ficheiro não encontrado!")
    else: print('Ficheiro com extensão diferente de .csv!')
```