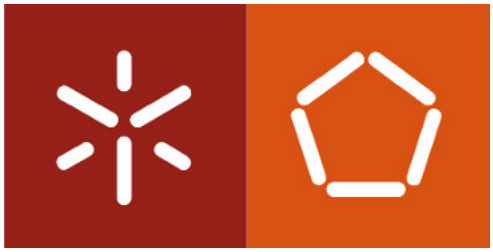

Universidade do Minho
Licenciatura em Engenharia Informática

Processamento de Linguagens

Trabalho Prático 1 - Grupo 52

Armando Silva - A87949 Joana Oliveira - A87956

Março 2022

Conteúdo

1 Introdução

O seguinte projeto foi elaborado no âmbito da UC de Processamento de Linguagens. Os seus principais objetivos são desenvolver a capacidade de escrever Expressões Regulares (ER), bem como sistemas capazes de transformar e filtrar texto e a utilização do módulo 're', juntamente com funções de search(), split(), findall(), etc. A linguagem de programação utilizada foi o Python.

A tarefa principal deste projeto era então o desenvolvimento de um sistema capaz de transformar ficheiros em formato CSV em ficheiros equivalentes no formato JSON. O ficheiro original poderá conter notação e operadores especifícos que indicam como terá de ser feita a leitura e processamento dos dados antes da sua conversão para JSON.

Ao longo do relatório irá ser explicado mais detalhadamente o problema, bem como todo o processo de implementação dos algoritmos responsáveis pela conversão de ficheiros.

2 Análise e Especificação do Problema

2.1 Descrição

Pretende-se então que, para qualquer ficheiro em formato CSV, seja feita a sua conversão para o formato JSON. Para tal, o cabelhaço do ficheiro original irá descrever quais os campos que o ficheiro destino irá conter. Alguns destes campos apresentam notações específicas relativas, por exemplo, à listagem de um certo número de elementos que se encontrem seguidos numa determinada linha do ficheiro. Estas listas poderão ter tamanho fixo ou poderão variar num determinado intervalo de valores. Além disso, é ainda possível a aplicação de funções de agregação a essas mesmas listas, tais como a soma ou a média dos seus valores, sendo estas representadas pela devida notação.

2.2 Requisitos

Ficheiro CSV

O *input* recebido pelo programa é um ficheiro CSV. Este possui os diversos dados separados por vírgulas. O cabeçalho, presente na primeira linha do ficheiro, permite identificar quais são os campos existentes e o que representam. Por exemplo, a utilização de chavetas indica que os elementos que se encontram no intervalo indicado entre elas terão de ser agrupados. Este intervalo poderá ser fixo ou variar entre dois valores distintos. Além disso, a utilização de '::' no seguimento da representação anterior determina a aplicação de funções a essas mesmas listas.

```
Número,Nome,Curso,Notas{3,5}::media,,,,,Notas{5},,,,
3162,Cândido Faísca,Teatro,12,13,14,,,15,16,13,12,10
7777,Cristiano Ronaldo,Desporto,17,12,20,11,12,15,16,13,12,9
264,Marcelo Sousa,Ciência Política,18,19,19,20,,15,16,13,12,8
```

Figura 1. Exemplo de um Ficheiro CSV

Ficheiro JSON

O *output* produzido pelo programa é um ficheiro JSON. Este ficheiro é inicializado e finalizado com parênteses retos. Cada linha do ficheiro CSV tem os seus campos representados dentro de chavetas, sendo que cada campo terá um valor associado. Estes campos poderão ser simples ou então listas, que serão representadas através do uso de parênteses retos.

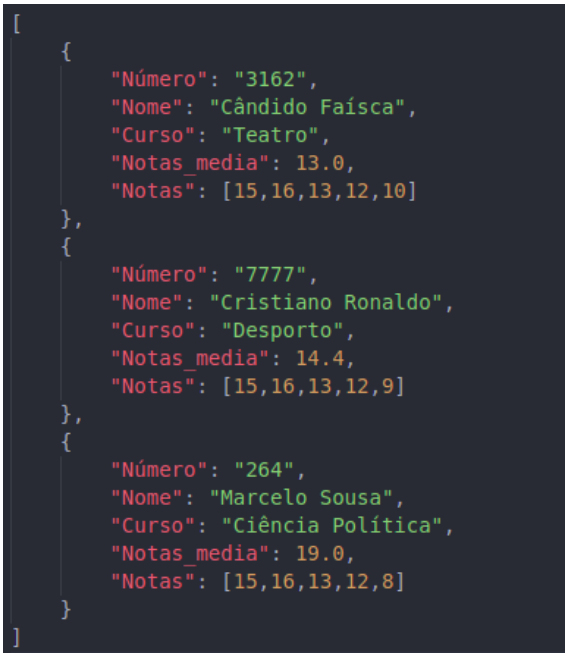

Figura 2. Exemplo de um Ficheiro JSON

3 Desenho da Solução

Concluída a fase de análise e especificação do problema, chegamos à fase do desenho da solução, onde serão apresentadas as estruturas de dados usadas e também a forma como o ficheiro CSV é lido e escrito para o formato JSON. De notar que todo o código explicado neste tópico poderá ser encontrado nos anexos.

3.1 Estuturas de Dados

São utilizados 4 arrays, definidos como variáveis globais:

- ***headers***, que armazena o nome dos cabeçalhos
- ***operations***, que armazena as funções de agregação a serem aplicadas
- ***intervals***, onde são guardados o tamanho ou intervalos de tamanho das listas
- ***defined_op***, que contém o nome das funções de agregação que são suportadas pelo nosso programa

3.2 Leitura do Ficheiro CSV

O programa recebe como *input* um ficheiro CSV indicado pelo utilizador. No entanto, caso este não seja indicado, o ficheiro a converter por *default* é o "teste.csv". O ficheiro JSON será então criado com o mesmo nome do ficheiro input, ou seja, o ficheiro *default* resultará no ficheiro "teste.json".

Primeiramente é feita a leitura do cabeçalho, fazendo a filtragem dos mesmos através de uma expressão regular. De seguida, toda a informação relativa a cada um dos campos é armazenada nas váriaveis globais acima mencionadas, respeitando a ordem pela qual aparecem no cabeçalho. O array *headers* irá guardar os nomes de todos os campos. O array *operations* irá conter as funções de agregação a serem aplicadas, sendo "list"utilizada quando o objetivo é apenas listar os elementos e "none"quando não é suposto aplicar nenhuma função. Por fim, o array *intervals* irá armazenar o tamanho ou os intervalos de tamanho das listas, tendo o valor "0"caso o elemento não seja uma lista. Por exemplo, suponhamos as informações relativas ao campo "Notas{3,5}::media". Neste caso, o array *headers* iria conter o nome "Notas_media", o array *operations* iria conter a operação "media"e o array *intervals* iria conter a lista [3,5].

Posteriormente é então feita a leitura linha a linha do ficheiro, sendo cada uma dividida por vírgulas utilizando o método *split* do módulo 're', e os seus elementos guardados num array. Através de um ciclo são percorridos todos os índices dos arrays guardados como váriáveis globais e que contêm informações relativas a nomes, operações a efetuar e intervalos. A decisão sobre como lidar com cada campo é feita tendo em conta a operação contida no índice em que nos encontramos no array relativo às operações. O resultado de cada campo vai então sendo guardado num array relativo a cada linha que será depois retornado pela função *read_line*.

3.3 Escrita para Ficheiro JSON

O programa irá produzir como *output* um ficheiro JSON. Para tal, é chamada a função *converter* que irá para cada linha do ficheiro CSV invocar a função *read_line* e obter os resultados de todos os seus campos conforme o cabeçalho. Toda a notação necessária em cada campo é já processada nesta função de modo a evitar percorrer as informações de cada linha mais do que uma vez. Esta será depois apenas inserida no local adequado do ficheiro JSON.

4 Implementação e Testes

Nesta última fase apresentaremos toda a implementação e tratamento de erros efetuado. De modo a provar que o programa funciona corretamente apresentaremos alguns testes feitos pelo grupo.

4.1 Implementação e Tratamento de Erros

O programa está estruturado em três fases distintas.

A primeira fase é a leitura e *parsing* do *header*, onde é armazenada toda a informação nas variáveis globais definidas. Isto permite facilitar a atribuição e diferenciação dos cabeçalhos aquando da leitura de todas as linhas dos ficheiro CSV.

A segunda fase é a leitura linha a linha do restante ficheiro e escrita dos respetivos resultados em arrays de strings. Esta leitura recorre às variáveis globais que foram preenchidas durante a leitura do cabeçalho.

A última fase é a formatação e inserção dos diversos campos das diferentes listas de strings, provenientes da fase anterior, no ficheiro JSON.

De forma a completar um pouco mais o trabalho, foi implementado um sistema de controlo de erros que tem como objetivo impedir a conversão de um ficheiro CSV inválido para JSON. É tido em conta o número de vírgulas em cada linha, intervalos de tamanhos de listas inválidos, funções de agregação inexistentes e também incoerência dos elementos de uma lista, isto é, a existência de listas que contenham simultaneamente números e palavras.

4.2 Testes realizados

Para provar o bom funcionamento do programa foi criado um ficheiro de teste CSV válido e outro inválido, detalhando o erro presente no ficheiro inválido.

```
Nome,Pontuação,Amigos{0,5},,,,,Golos{0,5},,,,
Armando,100,Joana,André,João,Gonçalo,,1,4,0,3,1
Joana,99,Armando,André,João,Catarina,Helena,5,6,2,1,0
André,101,Armando,João,Joana,Gouveia,,2,5,1,3,0
João,98,Armando,Joana,André,,,2,5,6,8,1
Gouveia,0,,,,,,,,,
```

Figura 3. Ficheiro de teste CSV válido


```
[
    {
        "Nome": "Armando",
        "Pontuação": "100",
        "Amigos": ["Joana","André","João","Gonçalo"],
        "Golos": [1,4,0,3,1]
    },
    {
        "Nome": "Joana",
        "Pontuação": "99",
        "Amigos": ["Armando","André","João","Catarina","Helena"],
        "Golos": [5,6,2,1,0]
    },
    {
        "Nome": "André",
        "Pontuação": "101",
        "Amigos": ["Armando","João","Joana","Gouveia"],
        "Golos": [2,5,1,3,0]
    },
    {
        "Nome": "João",
        "Pontuação": "98",
        "Amigos": ["Armando","Joana","André"],
        "Golos": [2,5,6,8,1]
    },
    {
        "Nome": "Gouveia",
        "Pontuação": "0",
        "Amigos": [],
        "Golos": []
    }
]
```


Figura 4. Resultado JSON do ficheiro de teste CSV válido

```
Nome,Pontuação,Amigos{0,5},,,,,Golos{0,5},,,,
Armando,100,Joana,André,João,Gonçalo,,1,4,0,3,1,2
Joana,99,Armando,André,João,Catarina,Helena,5,6,2,1,0
André,101,Armando,João,Joana,Gouveia,,2,5,1,3,0
João,98,Armando,Joana,André,,,2,5,6,8,1
Gouveia,0,,,,,,,,,
```

Figura 5. Ficheiro de teste CSV inválido

```
armando@ASUS:~/Documents/LEI/3ano/PL/TP/PL2122/TP1$ python3 convertCSVtoJSON.py teste2.csv
Traceback (most recent call last):
  File "convertCSVtoJSON.py", line 180, in <module>
    main()
  File "convertCSVtoJSON.py", line 175, in main
    result = converter(lines[1:])
  File "convertCSVtoJSON.py", line 148, in converter
    l = read_line(line)
  File "convertCSVtoJSON.py", line 96, in read_line
    raise NameError("Numero de elementos invalido!\n")
NameError: Numero de elementos invalido!
```

Figura 6. Resultado do ficheiro de teste CSV inválido

5 Conclusão

Terminada a realização deste trabalho prático, consideramos pertinente fazer uma apreciação crítica, realçando os seus pontos positivos e negativos, demonstrando as dificuldades e obstáculos que surgiram durante a realização do mesmo.

Começando pelos pontos positivos, consideramos que um deles tenha sido a capacidade de cumprir todos os requisitos pedidos, usando maioritamente ERs para a resolução de problemas, com o auxílio do módulo 're'. Além disso, cada linha do ficheiro CSV é lida apenas uma vez, com exceção do cabeçalho, devido à necessidade de confirmar que este cumpre com a sintaxe estipulada. Por fim, foi implementado um sistema de controlo de erros que garante que a informação contida no ficheiro original está correta e permite que a conversão seja feita sem qualquer tipo de problema.

Foram encontradas algumas dificuldades no que toca a estratégias de procura de padrões para o uso eficiente de ERs. Também a procura por possíveis erros que poderiam ocorrer durante a leitura do ficheiro se mostrou como um grande desafio, uma vez que estes requerem uma análise mais a fundo do programa, bem como uma realização intensiva dos mais variados testes.

Em suma, consideramos que a realização deste projeto permitiu aprimorar o nosso conhecimento sobre expressões regulares e filtros de texto e, com isto, ultrapassar todos os obstáculos anteriormente mencionados de forma eficaz, dando ao grupo um sentimento de realização.

A Código do Programa

A.1 Leitura do cabeçalho do Ficheiro CSV

```
def head_reader(header):
    hs = re.findall(r'([^,\n]+{\d,*\d*}:*:*\w*)|([^,\n]+)',header)
    for h in hs:
        if (h[0]) == '': h = h[1]
        else: h = h[0]
        if re.search(r'{\d,*\d*}:*:*\w*',h):
            interval = re.findall(r'\d',h)

            #Error Handling dos intervalos
            if  (len(interval) == 1 and isdigit(interval[0])) or ( len(interval) ==
    2 and isdigit(interval[0]) and isdigit(interval[1]) and int(interval[0]) < int(
    interval[1]) ):
                intervals.append(interval)
            else:
                raise NameError("Intervalo invalido!\n")

            #Error Handling das virgulas
            interval = [int(num) for num in interval]
            maxInt = max(interval)
            new_s = h.replace("{","\{")
            new_s = new_s.replace("}","\}")
            er = r'(' + new_s + r',{' + str(maxInt-1) + r'}$)|(' + new_s + r',{' +
    str(maxInt) + r'}\w)'

            if not re.search(er, header):
                raise NameError("Cabecalho invalido!\n")

            if re.search(r'::\w+',h):
                op = re.findall(r':\w+',h)[0][1:]

                #Error Handling das operacoes
                if (str(op).lower() in defined_op):
                    operations.append(op)
                else:
                    raise NameError("Operacao inexistente!\n")

                two_headers = re.findall(r'[^{,}:\d]+',h)
                headers.append(two_headers[0] + "_" + two_headers[1])

            else:
                name = re.findall(r'\w+',h)[0]
                headers.append(name)
                operations.append("list")

        else:
            headers.append(h)
            operations.append("none")
            intervals.append(0)
```

Listagem 1. Código da leitura do cabeçalho do ficheiro CSV

A.2 Processamento de cada linha do Ficheiro CSV

```
def read_line(line):
    i = 0
    l = re.split(',',line)
    res = []
    for j in range(0,len(headers)):
        elements = []
        if operations[j] != "none":
            if len(intervals[j]) == 1: #Listas com tamanho definido
                it = int(intervals[j][0])
                while it > 0:

                    #Error handling das virgulas
                    if i >= len(l) or not re.search(r'\w', l[i]):
                        raise NameError("Numero de elementos invalido!\n")

                    elements.append(l[i])
                    i = i+1
                    it = it-1

                #Error handling das virgulas
                if i == len(l)-1 and not re.search(r'\n', l[i]):
                    raise NameError("Numero de elementos invalido!\n")

            else: #Listas com um intervalo de tamanhos
                it = int(intervals[j][1])

                while it > 0:

                    #Error handling das virgulas
                    if i >= len(l):
                        raise NameError("Numero de elementos invalido!\n")

                    if re.search(r'\w', l[i]):
                        elements.append(l[i])

                    i = i+1
                    it = it-1

                #Error handling das virgulas
                if i == len(l)-1 and not re.search(r'\n', l[i]):
                    raise NameError("Numero de elementos invalido!\n")

            if operations[j] != "list":  #Funcoes de agregacao

                #Error handling Funcoes de Agregacao
                for elem in elements:
                    for digit in elem:
                        if not isdigit(digit):
                            raise NameError("Impossivel aplicar funcao de agregacao
    !\n")

                elements = [int(num) for num in elements]
                if operations[j] == "sum": op_res = sum(elements)
                elif operations[j] == "media": op_res = sum(elements)/len(elements)
                elif operations[j] == "min": op_res = min(elements)
                elif operations[j] == "max": op_res = max(elements)
                res.append(str(op_res))

            else:
                # List correction
                if (len(elements) > 0):
                    is_number = False
                    is_word = False

                    #Error handling da coerencia das listas
```

```
                    for elem in elements:
                        for digit in elem:
                            if not isdigit(digit):
                                is_word = True
                            else:
                                is_number = True

                    if is_number and is_word:
                      raise NameError("Incoerencia nos parametros da lista\n")
                    if is_number:
                        elements = "[" + ','.join(map(str, elements)) + "]"
                        res.append(elements)
                    else:
                        elements = "[" + "\"" + '\",\"'.join(elements) + "\"" + "]"
                        res.append(elements)
                else:
                    res.append("[]")
        else:
            res.append("\"" + l[i] + "\"")
            i = i+1
    return res
```

Listagem 2. Código do processamento de cada linha do Ficheiro CSV

A.3 Escrita para Ficheiro JSON

```
def converter(lines):
    result = "[\n"
    i = 0
    for line in lines:
        result += "\t{\n"
        j = 0
        l = read_line(line)
        for h in headers:
            if (j == len(headers)-1):
                result += "\t\t"
                result += "\"" + h + "\": " + l[j] + "\n"
            else:
                result += "\t\t"
                result += "\"" + h + "\": " + l[j] + ",\n"
            j = j + 1
        if (i == len(lines)-1):
            result += "\t}\n"
        else:
            result += "\t},\n"
        i = i + 1
    result += "]\n"
    return result
```

Listagem 3. Código da escrita para Ficheiro JSON

A.4 Main

```
def main():
    if len(sys.argv) == 1: file = "teste.csv"
    elif len(sys.argv) > 2:
        raise NameError("Argumentos a mais na funcao principal\n")
    else: file = sys.argv[1]

    f = open(file)
    lines = f.read().splitlines()
    f.close()
    head_reader(lines[0])
    result = converter(lines[1:])
    f = open(file[:-4] + ".json","w+")
    f.write(result)
    f.close()
```

Listagem 4. Código da Main