Relatório Trabalho Prático 1 - LEI 2020/21
Processadores e Filtros de Texto

Processamento de Linguagens

Grupo 7

Março 2022

(a) Pedro Ferreira a93282

(b) Joaquim Roque a93310

(c) Ana Gonçalves a93259

Figure 1: Autores do trabalho

Contents

1 Introdução

No âmbito da unidade curricular de Processamento de Linguagens foi nos apresentado o desafio de criar um filtro de texto, mais especificamente sobre o formato CSV para um formato JSON. Com isto em mente, apresentaremos neste documento o processo de desenvolvimento da nossa solução perante o desafio, constituída principalmente por uma interpretação do problema, seguida de uma sequência de descrições da solução para cada estilo de campo do ficheiro CSV. Finalmente apresentaremos exemplos teste de ficheiros CSV juntamente com o ficheiro JSON output, seguidos de uma conclusão relativa ao desenvolvimento deste projeto.

2 Processo de Desenvolvimento

O desafio que nos foi proposto na unidade curricular de Processamento de Linguagens consiste em desenvolver um filtro de ficheiros CSV em ficheiros JSON. E assim sendo, optamos por realizar os requisitos por ordem de prioridade com base em expressões regulares. Demos início ao trabalho fazendo um filtro simples de um ficheiro que continham campos independentes uns dos outros. Seguidamente, demos inicio a uma fase em que já era tido em conta campos do ficheiro CSV que correspondiam a listas de tamanho fixo e mais tarde com um intervalo de tamanhos. Finalmente optamos por adicionar as funções de agregação, inicialmente as apresentadas no enunciado e, complementarmente, funções extra definidas pelo grupo.

3 Expressões Regulares principais

Antes de dar início a explicação de cada uma das soluções, será mais proveitoso apresentar primeiro as expressões regulares que serviram de base para a resolução deste desafio.

3.1 *"campo"*

A variável *"campo"* tem como objetivo dar *match* ao conteúdo que é possível estar na primeira linha do ficheiro separado pelos delimitadores.

```
campo = r"([\w]+|\"[\w,;|]+\")((\{\d+\}|\{\d*,\d+\})(::\w+)?)?"
```

A seguir apresentar-se-á o significado de cada elemento da expressão regular:

```
([\w]+|\"[\w,;|]+\")
```

Corresponde à parte obrigatória que um campo não vazio deve ter, um conjunto de caracteres alfanuméricos incluindo o *underscore* ou, o mesmo conjunto mas entre aspas, que dá a possibilidade de adicionar os delimitadores como texto.

```
((\{\d+\}|\{\d*,\d+\})(::\w+)?)?
```

Corresponde à parte opcional de adicionar aos campos:

- Primeira parte relativa à inserção de intervalos de tamanho fixo
- Segunda parte relativa à inserção de intervalos de tamanho variável
- Terceira parte relativa a funções de agregação na existência de intervalos

3.2 *"is_valid"*

Além desta expressão regular, criamos uma outra capaz de validar a primeira linha inteira:

```
is_valid = r"^((" + campo + ")?[,;|\n])+$"
```

Esta expressão regular garante que a primeira linha do ficheiro tem de ser composta ou por pelo menos 1 campo (no formato da expressão regular anterior) ou por campos vazios.

4 Requisitos

Como anteriormente referido, a seguir será feita uma explicação concisa relativamente a cada um dos requisitos.

4.1 Processamento de uma Linha

Por cada linha do ficheiro é realizado um *split*, que vai separar os campos por um delimitador (”,” , ”;” ou ”—”). No entanto, não é desejável que estes delimitadores separem campos quando estão integrados em *string*, por exemplo, ”hoje o dia esta bonito, não achas?”. Assim sendo, é preciso fazer uma ”limpeza” da linha. Esta limpeza ocorre na função *”limpaLinha”*

```
def limpaLinha(linhaSplit):
    listaLimpa = []
    concatenate = ""                                    # onde vai ser guardados campos que pertencem juntos
    i = 0

    while i < (len(linhaSplit)):                        # enquanto os campos não acabarem
        if re.search(r'\".*\"',linhaSplit[i]):          # se existirem 2 aspas no mesmo campo, o campo está correto
            listaLimpa.append(linhaSplit[i])

        elif re.search(r'^\"',linhaSplit[i]):           # se existir só 1 aspa no campo, o campo tem de ser completado
            while linhaSplit[i][-1] != "\"":            # enquanto não aparecer um campo a acabar com 1 aspas o campo é adicionado à string com o delimitador correto
                if sizeListComa+1 == len(split):        # se delimitador for ,
                    concatenate = concatenate + linhaSplit[i]+ ","
                elif sizeListSemiColon+1 == len(split):         # se delimitador for ;
                    concatenate = concatenate + linhaSplit[i]+ ";"
                else:                                   # se delimitador for |
                    concatenate = concatenate + linhaSplit[i]+ "|"
                i+=1
            concatenate = concatenate + linhaSplit[i]   # quando encontrar as aspas que faltam adiciona à string sem delimitador
            listaLimpa.append(concatenate[1:-1])        # campo adicionado à lista limpa

        else:                                           # se campo não tiver nenhuma aspas está correto
            listaLimpa.append(linhaSplit[i])
        i+=1

    return listaLimpa
```

Figure 2: Função *”limpaLinha”*

Como podemos verificar pelo código, existem 3 possíveis situações para limpar uma linha.

- **Quando um campo tem um par de aspas:**
  Significa que esse campo está correto, e por isso pode ser adicionado imediatamente à lista limpa
- **Quando um campo tem apenas umas aspas**
  Significa que esse campo está incompleto, o que será necessário agrupar numa string auxiliar todos os campos até ser encontrada as aspas correspondentes que faltam
- **Quando um campo não tem aspas**
  Significa que esse campo está correto, e por isso pode ser adicionado imediatamente à lista limpa também

Com esta função finalizamos a limpeza desejada dos dados e podemos avançar para a criação do ficheiro JSON, através do percorrer de cada campo de cada linha.

4.2 Campo Básico

Nesta solução, um campo básico, trata-se de um campo que não possui nenhum tipo de intervalo (e consequentemente nenhuma função de agregação). Assim sendo, este caso está apresentado na última condição *else*, que consiste unicamente em retirar o "\n" e escrever o campo juntamente com o cabeçalho correspondente.

```
else:
    listaLimpa[indCampo] = listaLimpa[indCampo].strip("\n")                    # retira o \n do ultimo campo
    if listaCabecalho[indCabecalho][0][0] != "\"":                             # verifica se cabecalho já tem ""
        fWrite.write("\t\""+listaCabecalho[indCabecalho][0]+"\" : \"" + listaLimpa[indCampo]+"\"")
    else:
        fWrite.write("\t"+listaCabecalho[indCabecalho][0]+" : \"" + listaLimpa[indCampo]+"\"")
    indCampo = indCampo + 1
    indCabecalho = indCabecalho + 1

if indCabecalho == len(listaCabecalho):                                        # se for o ultimo elemento da linha não vai colocar , depois dele
    fWrite.write("\n")
else:                                                                          # se nao for o ultimo elemento da linha vai colocar , depois dele
    fWrite.write(",\n")
```

Figure 3: Fragmento de código que processa um campo simples

4.3 Listas

As listas neste problema vão ser solucionadas em conjunto, alterando apenas uma variável dependendo de que tipo a lista é (tamanho variável vs tamanho fixo).

```
while(indCabecalho < len(listaCabecalho)):                                     # enquanto os campos não acabarem
    valores = []                                                               # guarda os valores das listas
    strings = []                                                               # guarda as strings das listas
    listaTotal = []                                                            # guarda todo o conteudo da lista

    if intervaloVal := re.search(r'(?:\{(\d*),(\d+)\})|(?:\{(\d+)\})',listaCabecalho[indCabecalho][2]):     # se encontrar algum intervalo

        if (intervaloVal := re.search(r'(?:\{(\d*),(\d+)\})',listaCabecalho[indCabecalho][2])):             # se encontrar um intervalo variável
            intervaloFim = int(intervaloVal.groups()[1]) + indCampo                                        # descobre o indice na linha de informação onde acaba o intervalo
        else:
            intervaloVal = re.search(r'(?:\{(\d+)\})',listaCabecalho[indCabecalho][2])                      # se encontrar um intervalo fixo
            intervaloFim = int(intervaloVal.groups()[0]) + indCampo                                        # descobre o indice na linha de informação onde acaba o intervalo

        while intervaloFim > indCampo:                                                                     # Enquanto não chegar ao fim do intervalo:

            if indCampo >= len(listaLimpa) or (listaLimpa[indCampo] == "") or (listaLimpa[indCampo] == "\n"):   # se for o último elemento do intervalo ou da linha indice avança
                indCampo = intervaloFim - 1                                                                    # para o último valor do intervalo

            elif re.search(r'\d+',listaLimpa[indCampo]):                                                   # se não for vazio e for um numero:
                valores.append(listaLimpa[indCampo])                                                       # vai ser adicionado à lista dos valores
                listaTotal.append(listaLimpa[indCampo])                                                    # vai ser adicionado à lista de todo o conteudo da lista
            else:                                                                                          # se não for vazio e for uma string
                strings.append("\""+listaLimpa[indCampo]+"\"")                                             # vai ser adicionado à lista das strings
                listaTotal.append("\""+listaLimpa[indCampo]+"\"")                                          # vai ser adicionado à lista de todo o conteudo da lista

            indCampo = indCampo + 1
```

Figure 4: Fragmento de Código que processa listas no ficheiro input

Como podemos verificar pelo código, se existir algum tipo de listas, vai ser determinado que tipo, e obter o índice do último valor possível do intervalo. A partir daí, ambas as listas funcionam de igual forma. Isto é, se ainda não se chegou ao fim do intervalo e o campo não é vazio nem "\n", esse campo vai ser adicionado a um *array* de valores, se for um número, ou a um *array* de *strings*, se for uma *string*. E independentemente do seu formato, vai ser adicionado a uma lista com todos os valores dentro do intervalo.
Mais à frente, estes 3 *arrays* vão ser utilizados ou para funções de agregação ou para uma lista simples de valores no ficheiro JSON.

4.4 Funções de Agregação

4.4.1 *Sum*

Com base numa função descrita no enunciado do trabalho prático, decidimos adicioná-la à nossa solução. *Sum* trata-se de uma função simples, que calcula a soma todos os valores numéricos da lista, sendo esta apresentada no ficheiro JSON.

```
if listaCabecalho[indCabecalho][3] == "::sum":                                  # se for uma função de soma
    soma = sumArray(valores)                                                    # soma todos os valores guardados
    fWrite.write("\t\""+listaCabecalho[indCabecalho][0]+"_sum\" : " + str(soma)) # coloca no JSON com o prefixo _sum
```

Figure 5: Função de Agregação de soma

4.4.2 *Media*

Na mesma situação que a função anterior, a *Media* também nos foi apresentada no enunciado, como a soma, e calcula a média de todos os valores numéricos no ficheiro JSON

```
elif listaCabecalho[indCabecalho][3] == "::media":                                  # se for uma função de media
    soma = sumArray(valores)                                                        # soma todos os valores guardados
    fWrite.write("\t\""+listaCabecalho[indCabecalho][0]+"_media\" : " + str(soma/len(valores))) # coloca no JSON com o prefixo_media
```

Figure 6: Função de Agregação de média

4.4.3 *Sort*

Esta função de agregação foi escolhida visto que o grupo concluiu que seria uma funcionalidade útil, que permitia a ordenação tanto de valores numéricos como de *strings*. Esta função recorre a uma outra, *cleanValores*, que permite retirar todos os "'" de cada um dos valores.

```
elif listaCabecalho[indCabecalho][3] == "::sort":                                   # se for uma função de sort
    cleaned = cleanValores(listaTotal)                                              # vai retirar ' de todos os valores
    cleaned.sort()                                                                  # vai ordenar a lista de strings
    fWrite.write("\t\""+listaCabecalho[indCabecalho][0]+"_sort\" : " + str(cleaned).replace("\'",""))
```

Figure 7: Função de Agregação de ordenação

4.4.4 *SortByLength*

Semelhante à função *Sort*, acreditamos que ordenar pelo tamanho de cada elemento poderia vir a ser útil em algumas situações bastante especificas, e por isso, o grupo optou por adiciona-la. Em semelhança à função *Sort*, recorre também à função *cleanValores*, com exatamente o mesmo motivo, retirar "'" de cada um dos valores.

```
elif listaCabecalho[indCabecalho][3] == "::sortByLength":                           # se for uma função de sortByLength
    cleaned = cleanValores(listaTotal)                                              # vai retirar ' de todos os valores
    cleaned.sort(key=myLength)                                                      # vai ordenar a lista de strings pelo comprimento do valor
    fWrite.write("\t\""+listaCabecalho[indCabecalho][0]+"_sortByLength\" : " + str(cleaned).replace("\'",""))
```

Figure 8: Função de Agregação de ordenação por tamanho

5 Testes

Finalizando a apresentação da solução desenvolvida, vamos apresentar a seguir os testes que serviram para confirmar o resultado que era pretendido.

5.1 Campo Básico

Neste subtema, estarão apresentados ficheiros de input simples, juntamente com o seu output.

```
Número,Nome,Curso
3162,Cândido Faísca,Teatro
7777,Cristiano Ronaldo,Desporto
264,Marcelo Sousa,Ciência Política
```

Figure 9: Ficheiro input simples com delimitador ”,”

```
Número;Nome;Curso
3162;Cândido Faísca;Teatro
7777;Cristiano Ronaldo;Desporto
264;Marcelo Sousa;Ciência Política
```

Figure 10: Ficheiro input simples com delimitador ”;”

```
Número|Nome|Curso
3162|Cândido Faísca|Teatro
7777|Cristiano Ronaldo|Desporto
264|Marcelo Sousa|Ciência Política
```

Figure 11: Ficheiro input simples com delimitador ”|”

```
[
  {
    "Número" : "3162",
    "Nome" : "Cândido Faísca",
    "Curso" : "Teatro"
    },
  {
    "Número" : "7777",
    "Nome" : "Cristiano Ronaldo",
    "Curso" : "Desporto"
  },
  {
    "Número" : "264",
    "Nome" : "Marcelo Sousa",
    "Curso" : "Ciência Política"
  }
]
```

Figure 12: Ficheiro output comum das figuras 9, 10 e 11

5.2 Campo Com Delimitador

Aqui é onde serão apresentados os resultados de ficheiros input, cujos campos contêm algum dos 3 delimitadores

```
Número,"Nome, BI ou CC",Curso
3162,"Cândido Faísca, 23432, 23423",Teatro
7777,Cristiano Ronaldo,Desporto
264,Marcelo Sousa,Ciência Política
```

Figure 13: Ficheiro input cujo campo contêm o delimitador ”,”

```
[
  {
    "Número" : "3162",
    "Nome, BI ou CC" : "Cândido Faísca, 23432, 23423",
    "Curso" : "Teatro"
  },
  {
    "Número" : "7777",
    "Nome, BI ou CC" : "Cristiano Ronaldo",
    "Curso" : "Desporto"
  },
  {
    "Número" : "264",
    "Nome, BI ou CC" : "Marcelo Sousa",
    "Curso" : "Ciência Política"
  }
]
```

Figure 14: Ficheiro output da figura 13

```
Número;"Nome; BI ou CC";Curso
3162;"Cândido Faísca; 23432; 23423";Teatro
7777;Cristiano Ronaldo;Desporto
264;Marcelo Sousa;Ciência Política
```

Figure 15: Ficheiro input cujo campo contêm o delimitador ”;”

```
[
  {
    "Número" : "3162",
    "Nome; BI ou CC" : "Cândido Faísca; 23432; 23423",
    "Curso" : "Teatro"
  },
  {
    "Número" : "7777",
    "Nome; BI ou CC" : "Cristiano Ronaldo",
    "Curso" : "Desporto"
  },
  {
    "Número" : "264",
    "Nome; BI ou CC" : "Marcelo Sousa",
    "Curso" : "Ciência Política"
  }
]
```


Figure 16: Ficheiro output da figura 15


```
Número|"Nome| BI ou CC"|Curso
3162|"Cândido Faísca| 23432| 23423"|Teatro
7777|Cristiano Ronaldo|Desporto
264|Marcelo Sousa|Ciência Política
```


Figure 17: Ficheiro input cujo campo contêm o delimitador "|"


```
[
  {
    "Número" : "3162",
    "Nome| BI ou CC" : "Cândido Faísca| 23432| 23423",
    "Curso" : "Teatro"
  },
  {
    "Número" : "7777",
    "Nome| BI ou CC" : "Cristiano Ronaldo",
    "Curso" : "Desporto"
  },
  {
    "Número" : "264",
    "Nome| BI ou CC" : "Marcelo Sousa",
    "Curso" : "Ciência Política"
  }
]
```


Figure 18: Ficheiro output da figura 17

5.3 Lista De Tamanho Fixo

Nesta secção serão apresentados os vários ficheiros testes que contêm listas de tamanho fixo, tanto de strings como valores numéricos

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

Figure 19: Ficheiro input com uma lista de tamanho fixo de valores numéricos

```
[
  {
    "Número" : "3162",
    "Nome" : "Cândido Faísca",
    "Curso" : "Teatro",
    "Notas" : [12, 13, 14, 15, 16]
  },
  {
    "Número" : "7777",
    "Nome" : "Cristiano Ronaldo",
    "Curso" : "Desporto",
    "Notas" : [17, 12, 20, 11, 12]
  },
  {
    "Número" : "264",
    "Nome" : "Marcelo Sousa",
    "Curso" : "Ciência Política",
    "Notas" : [18, 19, 19, 20, 18]
  }
]
```

Figure 20: Ficheiro output da figura 19

```
Número,Nome,Curso,Morada{3},,,
3162,Cândido Faísca,Teatro,Portugal,Braga,Martim
7777,Cristiano Ronaldo,Desporto,Portugal,Porto,Arouca
264,Marcelo Sousa,Ciência Política,Portugal,Braga,Cabreiros
```

Figure 21: Ficheiro input com uma lista de tamanho fixo de Strings

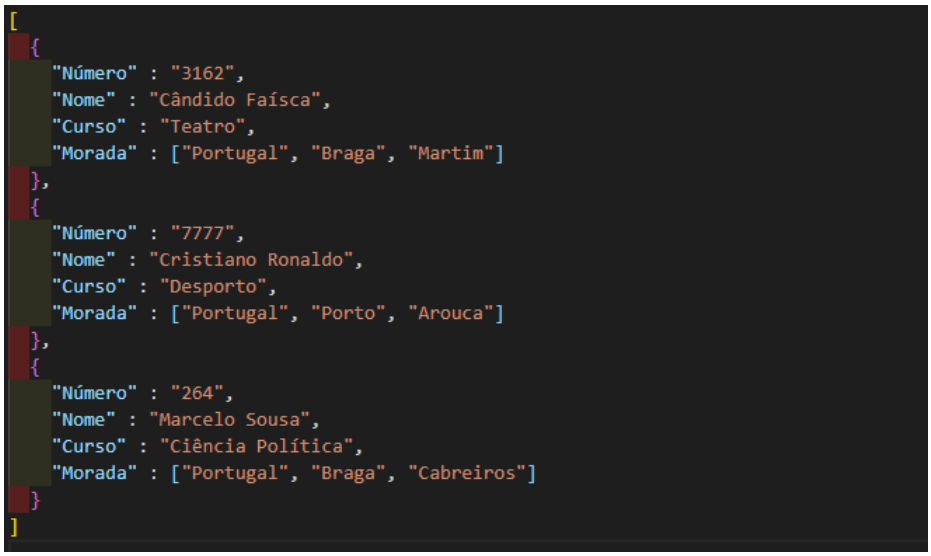

Figure 22: Ficheiro output da figura 21

5.4 Lista de Tamanho Variável

Nesta secção serão apresentados os vários ficheiros testes que contêm listas de tamanho variável, tanto de strings como valores numéricos

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

Figure 23: Ficheiro input com uma lista de tamanho fixo de valores numéricos

```
[
  {
    "Número" : "3162",
    "Nome" : "Cândido Faísca",
    "Curso" : "Teatro",
    "Notas" : [12, 13, 14]
  },
  {
    "Número" : "7777",
    "Nome" : "Cristiano Ronaldo",
    "Curso" : "Desporto",
    "Notas" : [17, 12, 20, 11, 12]
  },
  {
    "Número" : "264",
    "Nome" : "Marcelo Sousa",
    "Curso" : "Ciência Política",
    "Notas" : [18, 19, 19, 20]
  }
]
```


Figure 24: Ficheiro output da figura 23


```
Número,Nome,Curso,Morada{3,5},,,,,
3162,Cândido Faísca,Teatro,Portugal,Braga,Martim,,
7777,Cristiano Ronaldo,Desporto,Portugal,Porto,Arouca,,
264,Marcelo Sousa,Ciência Política,Portugal,Braga,Cabreiros,Rua De S.Antonio,
```


Figure 25: Ficheiro input com uma lista de tamanho fixo de Strings


```
[
  {
    "Número" : "3162",
    "Nome" : "Cândido Faísca",
    "Curso" : "Teatro",
    "Morada" : ["Portugal", "Braga", "Martim"]
  },
  {
    "Número" : "7777",
    "Nome" : "Cristiano Ronaldo",
    "Curso" : "Desporto",
    "Morada" : ["Portugal", "Porto", "Arouca"]
  },
  {
    "Número" : "264",
    "Nome" : "Marcelo Sousa",
    "Curso" : "Ciência Política",
    "Morada" : ["Portugal", "Braga", "Cabreiros", "Rua De S.Antonio"]
  }
]
```


Figure 26: Ficheiro output da figura 25

5.5 Funções de Agregação

Finalmente, serão apresentados por último os ficheiros testes e resultado de todas as funções de agregação criadas.

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

Figure 27: Função de Agregação Sum


```
[
  {
    "Número" : "3162",
    "Nome" : "Cândido Faísca",
    "Curso" : "Teatro",
    "Notas_sum" : 39
  },
  {
    "Número" : "7777",
    "Nome" : "Cristiano Ronaldo",
    "Curso" : "Desporto",
    "Notas_sum" : 72
  },
  {
    "Número" : "264",
    "Nome" : "Marcelo Sousa",
    "Curso" : "Ciência Política",
    "Notas_sum" : 76
  }
]
```


Figure 28: Ficheiro output da figura 27

```
Número,Nome,Curso,Notas{3,5}::sum,,,,,
3162,Cândido Faísca,Teatro,12,13,faltou,,
7777,Cristiano Ronaldo,Desporto,faltou,12,20,11,12
264,Marcelo Sousa,Ciência Política,18,19,19,20,
```

Figure 29: Função de Agregação Sum com valores mistos

```
[
  {
    "Número" : "3162",
    "Nome" : "Cândido Faísca",
    "Curso" : "Teatro",
    "Notas_sum" : 25
  },
  {
    "Número" : "7777",
    "Nome" : "Cristiano Ronaldo",
    "Curso" : "Desporto",
    "Notas_sum" : 55
  },
  {
    "Número" : "264",
    "Nome" : "Marcelo Sousa",
    "Curso" : "Ciência Política",
    "Notas_sum" : 76
  }
]
```


Figure 30: Ficheiro output da figura 29

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

Figure 31: Função de Agregação Media


```
[
  {
    "Número" : "3162",
    "Nome" : "Cândido Faísca",
    "Curso" : "Teatro",
    "Notas_media" : 13.0
  },
  {
    "Número" : "7777",
    "Nome" : "Cristiano Ronaldo",
    "Curso" : "Desporto",
    "Notas_media" : 14.4
  },
  {
    "Número" : "264",
    "Nome" : "Marcelo Sousa",
    "Curso" : "Ciência Política",
    "Notas_media" : 19.0
  }
]
```


Figure 32: Ficheiro output da figura 31

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

Figure 33: Função de Agregação Media com valores mistos

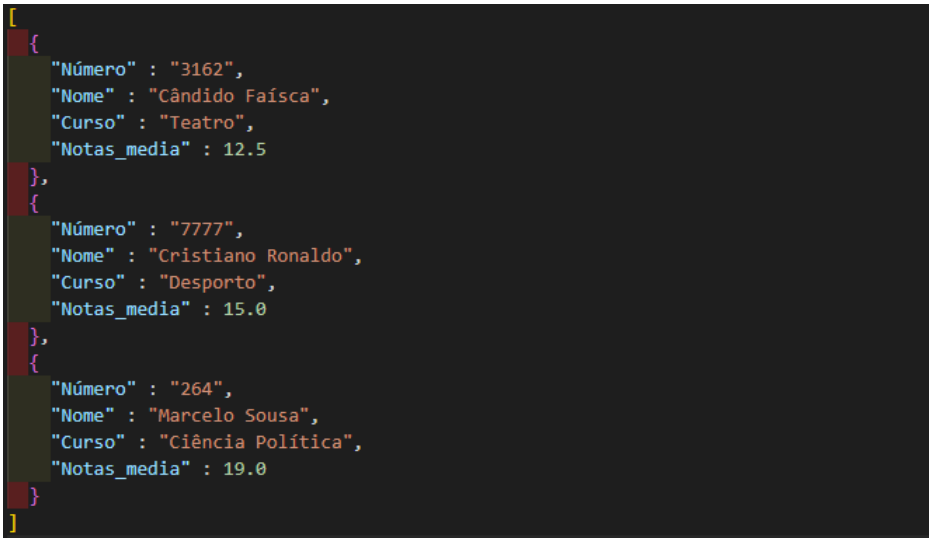

Figure 34: Ficheiro output da figura 33

```
Número,Nome,Curso,Notas{3,5}::sort,,,,,
3162,Cândido Faísca,Teatro,12,13,faltou,,
7777,Cristiano Ronaldo,Desporto,17,faltou,20,11,12
264,Marcelo Sousa,Ciência Política,18,19,19,20,
```

Figure 35: Função Agregação Sort com valores mistos

```
[
  {
    "Número" : "3162",
    "Nome" : "Cândido Faísca",
    "Curso" : "Teatro",
    "Notas_sort" : ["faltou", 12, 13]
  },
  {
    "Número" : "7777",
    "Nome" : "Cristiano Ronaldo",
    "Curso" : "Desporto",
    "Notas_sort" : ["faltou", 11, 12, 17, 20]
  },
  {
    "Número" : "264",
    "Nome" : "Marcelo Sousa",
    "Curso" : "Ciência Política",
    "Notas_sort" : [18, 19, 19, 20]
  }
]
```


Figure 36: Ficheiro output da figura 35

```
Número,Nome,Curso,Notas{3,5}::sortByLength,,,,,
3162,Cândido Faísca,Teatro,12,13,faltou,,
7777,Cristiano Ronaldo,Desporto,17,faltou,20,11,12
264,Marcelo Sousa,Ciência Política,18,19,19,20,
```

Figure 37: Função Agregação SortByLength com valores mistos


```
[
  {
    "Número" : "3162",
    "Nome" : "Cândido Faísca",
    "Curso" : "Teatro",
    "Notas_sortByLength" : [12, 13, "faltou"]
  },
  {
    "Número" : "7777",
    "Nome" : "Cristiano Ronaldo",
    "Curso" : "Desporto",
    "Notas_sortByLength" : [17, 20, 11, 12, "faltou"]
  },
  {
    "Número" : "264",
    "Nome" : "Marcelo Sousa",
    "Curso" : "Ciência Política",
    "Notas_sortByLength" : [18, 19, 19, 20]
  }
]
```


Figure 38: Ficheiro output da figura 37

6 Conclusão

Finalizando assim, a apresentação do processo de desenvolvimento do projeto relativamente a filtros de texto, o grupo concorda que este desafio permitiu um maior à-vontade em torno das expressões regulares. Isto é, o processo de ser apresentado a um desafio seguido de uma interpretação e finalmente criação de expressões regulares juntamente com desenvolvimento de código em *Python* permitiu gerar uma maior capacidade de resolução de problemas com ferramentas novas.