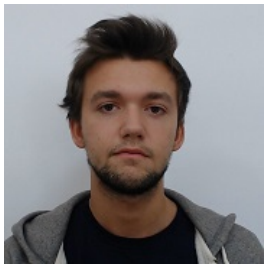
Processamento de Linguagens

Trabalho Prático 1
3 de Abril de 2021
Grupo 28


António Santos, A83700


Jorge Vieira, A84240


Pedro Fernandes, A84313

Conteúdo

Capítulo 1

Introdução

Área: Processamento de Linguagens

1.1 Contexto

No contexto da Unidade Curricular de Processamento de Linguagens, foi proposto ao grupo o desenvolvimento de um programa capaz de converter qualquer ficheiro em formato CSV(Comma Separated Values,tipicamente usado para descarregar uma Folha de Cálculo num ficheiro de texto) para o formato JSON (Um formato textual, baseado no conceito de um conjunto de pares (campo : valor)).

Para além disso, o programa também deveria ser capaz de representar listas aninhadas e aplicar, como operação de *fold*, 4 funções específicas a estas listas: *sum, avg, max, min.*

1.2 Problema

Como estamos no âmbito de Processamento de Linguagens, o desafio era utilizar expressões regulares para criar Filtros de Texto e desse modo realizar o parsing do ficheiro, assim como fazer uso das funções do módulo *re* da linguagem *python* (sub(), split(), search()) para manipular os dados de forma a termos o resultado pretendido, o que tornou o trabalho bastante mais desafiante.

1.3 Objetivo

Como este é um trabalho académico os seus principais objetivos são:

1. Aumentar a capacidade de escrever Expressões Regulares para descrição de padrões de frases dentro de textos
2. Desenvolver Filtros de Texto que filtrem ou transformem textos com base no conceito de regras de produção Condição-Ação.
3. Utilizar o módulo 'er' do *python* com suas funções de search(),split(),sub() para implementar os Filtros de Texto pedidos.

1.4 Estrutura do documento

Este relatório está dividido em 4 capítulos sendo o primeiro a Introdução ao projeto, o segundo um resumo da implementação de cada funcionalidade do programa, o terceiro apresenta uma reflexão sobre as alternativas que poderiam ser usadas no projeto tal como o resultado dos testes realizados. E o quarto uma reflexão sobre o desenvolvimento do programa no geral. Também está incluído em apêndice com todo o código necessário para o funcionamento do programa.

Capítulo 2

Desenvolvimento do programa

2.1 Inicialização

Numa primeira fase de desenvolvimento, para podermos começar a trabalhar com os dados é necessário enviar a localização do ficheiro *CSV* como argumento ao executar o programa, caso contrário este irá utilizar uma localização por defeito (neste caso test.csv). De seguida, como precisávamos de uma estrutura de dados que guardasse os vários elementos do ficheiro *CSV*, inicializamos uma lista *res* como lista vazia.

```
if(len(sys.argv) == 1):
    filePath = "test.csv"
else:
    filePath = sys.argv[1]

f = open(filePath,"r",encoding='utf8')
res = []
```

Após termos a estrutura *JSON* (*res*) pronta para ser construida necessitámos de ler a primeira linha do ficheiro *CSV* para obter a informação necessária para descodificar o resto do ficheiro.

O código seguinte separa todos os argumentos presentes no cabeçalho (separados por ';') e coloca-os numa lista para serem usados posteriormente. Caso o argumento apresente o caratér *newline* este é removido pois não é necessário para o funcionamento do programa e poderia eventualmente causar erros na execução do mesmo.

```
    args = [re.sub("\n","",x) if re.search("\n",x) else x for x in re.split(';',f.readline())]
```

2.2 Parsing

Para as restantes linhas do *CSV* é executada a seguinte linha de código que tem uma função idêntica ao codigo acima mas está adaptado para os elementos do ficheiro e não para o cabeçalho. De notar que a partir deste passo cada linha do ficheiro *CSV* é processado individualmente e sequencialmente.

```
    listaelem = [re.sub("\n","",x) if re.search("\n",x) else x for x in re.split(';',linha)]
```

Se o número de elementos no cabeçalho for diferente do número de elementos da linha *CSV* então estamos perante um ficheiro *CSV* inválido e por isso temos que lançar um erro e parar a execução do programa.

```
    if(len(args) != len(listaelem)):
        raise Exception("CSV Inválido! (Num de argumentos != Num de Valores)")
```

2.3 Conversão

Se o programa consegue entrar nesta etapa então podemos garantir que temos os elementos da linha *CSV* de acordo com o cabeçalho e por isso poderemos começar a sua conversão. Para isso é criado um dicionário para representar o elemento em *JSON* e é populado com os elementos da linha *CSV*.

Este processo é simples para a maioria dos elementos a única exceção é caso o argumento no cabeçalho apresente o '*', neste caso teremos que tratar o elemento como uma lista, por isso teremos que confirmar que estamos perante uma lista válida e teremos que a converter com o seguinte código:

```
if not re.match("^\((\d+(?:\.\d+)*)(,(\d+(?:\.\d+)*))*\)$",listaelem[index]):
        raise Exception("CSV Inválido! (Lista inválida)")

    listaValores = [float(x) for x in re.findall("\d+(?:\.\d+)?",listaelem[index])]
```

Este código garante que a lista é composta por números inteiros ou decimais separados por uma vírgula tudo dentro de parentêses e coloca-os numa lista de *floats* pronta para ser processada caso seja necessário, temos como exemplo: (1,2,3,4,5)→ $[1.0, 2.0, 3.0, 4.0, 5.0]$

O cabeçalho também poderá pedir uma operação de agregação na lista. Nesse caso cada operação é adicionada a uma lista para ser executada prosteriormente.

```
listaAgr = re.split("\*",args[index])
```

As operações são as seguintes:

2.3.1 Agregação *AVG*

Esta operação pega em todos os elementos da lista e calcula a sua média:

```
    elem[listaAgr[0]+'_'+agr] = sum(listaValores) / len(listaValores)
```

O resultado desta operação será por exemplo: (1,2,3,4,5)→ 3.0

2.3.2 Agregação *SUM*

Esta operação calcula a soma de todos os elementos da lista:

```
    elem[listaAgr[0]+'_'+agr] = sum(listaValores)
```

O resultado desta operação será por exemplo: (1,2,3,4,5)→ 15.0

2.3.3 Agregação *MAX*

Esta operação devolve o elemento com maior valor da lista, ordenando esta e retirando o último elemento:

```
listaValores.sort()
elem[listaAgr[0]+'_'+agr] = listaValores[-1]
```

O resultado desta operação será por exemplo: (1,2,3,4,5)→ 5.0

2.3.4 Agregação *MIN*

Esta operação devolve o elemento com menor valor da lista, ordenando esta e retirando o primeiro elemento:

```
listaValores.sort()
elem[listaAgr[0]+'_'+agr] = listaValores[0]
```

O resultado desta operação será por exemplo: (1,2,3,4,5)→ 1.0

2.4 Exportação

Nesta última fase, como já temos o dicionário completamente preenchido com a informação do ficheiro *CSV*, apenas precisamos de gerar o ficheiro *JSON* com os mesmos dados. Para isso exportamos o dicionário para o ficheiro *JSON* criado utilizando a função *dump* do módulo *json* do *python*.

```
with open(filePath.replace("csv","json"), 'w', encoding='utf-8') as jsonFile:
json.dump(res, jsonFile,indent=2,ensure_ascii=False)
```

Capítulo 3

Codificação e Testes

3.1 Alternativas, Decisões e Problemas de Implementação

O grupo tomou como prioridade principal a produção de um código compacto, legível e conciso. Contudo, devido a esta decisão poderão não ter sido usadas as opções mais eficientes para algumas áreas do código, tal como na *Agregação MAX* e *Agregação MIN* onde usamos o processo complexo de ordenar a lista para retirar o maior (ou menor elemento) da mesma, isto resulta num código não tão eficiente mas mais compacto. O grupo acabou por adotar esta decisão devido á infíma diferença nos tempos de execução entre as duas alternativas.

Para tornar o programa mais flexivel o grupo decidiu tratar todos os valores numéricos recebidos nas listas como *floats*, isso faz com que seja possivel processar valores com casas decimais e faz com que os métodos de agregação consigam calcular com mais precisão.

Apesar de não ser a solução mais elegante foi decidido que o programa não iria armazenar as listas em memória devido a diversas complicações resultantes das implementações introduzidas no programa anteriormente, como solução foi implementada a opção de executar várias agregações sobre uma lista ao concatenar todas as operações no argumento do cabeçalho.

3.2 Testes realizados e Resultados

3.3 Teste 1

CSV

```
nome;posicoes*min*max;tempos*avg**min
Usain Bolt;(1,2,1,1,1);(10,10.70,9.58,9.63)
Francis Obikwelu;(2,3,1,1,4);(10.10,67,9.86,10.84)
Yohan Blake;(3,1,3,4,2);(10.30,9.69,11.20,11.09,10.53)
Justin Gatlin;(2,5,3,3,4);(10.14,12.31,11.65,11.32,12.04)
```

↓

JSON


```
[
  {
    "nome": "Usain Bolt",
    "posicoes_min": 1.0,
    "posicoes_max": 2.0,
    "tempos_avg": 9.977500000000001,
    "tempos": [
      9.58,
      9.63,
      10.0,
      10.7
    ],
    "tempos_min": 9.58
  },
  {
    "nome": "Francis Obikwelu",
    "posicoes_min": 1.0,
    "posicoes_max": 4.0,
    "tempos_avg": 24.45,
    "tempos": [
      9.86,
      10.1,
      10.84,
      67.0
    ],
    "tempos_min": 9.86
  },
  {
    "nome": "Yohan Blake",
    "posicoes_min": 1.0,
    "posicoes_max": 4.0,
    "tempos_avg": 10.562000000000001,
    "tempos": [
      9.69,
```

```
      10.3,
      10.53,
      11.09,
      11.2
    ],
    "tempos_min": 9.69
  },
  {
    "nome": "Justin Gatlin",
    "posicoes_min": 2.0,
    "posicoes_max": 5.0,
    "tempos_avg": 11.492,
    "tempos": [
      10.14,
      11.32,
      11.65,
      12.04,
      12.31
    ],
    "tempos_min": 10.14
  }
]
```

3.4 Teste 2

CSV

```
nome;numero;equipa;golospjogo**avg*max*sum
Cristiano Ronaldo;7;Juventus;(2,1,1,3,3,1)
Edin Dzeko;9;Roma;(1,2,1,0,1,1)
Romelu Lukaku;9;Inter de Milão;(3,2,1,0,1,2)
Lautaro Martinez;10;Inter de Milão;(0,2,1,3,1,0)
```

↓

JSON

```
[
  {
    "nome": "Cristiano Ronaldo",
    "numero": "7",
    "equipa": "Juventus",
    "golospjogo": [
      2.0,
      1.0,
      1.0,
      3.0,
      3.0,
      1.0
```

```
    ],
    "golospjogo_avg": 1.8333333333333333,
    "golospjogo_max": 3.0,
    "golospjogo_sum": 11.0
  },
  {
    "nome": "Edin Deko",
    "numero": "9",
    "equipa": "Roma",
    "golospjogo": [
      1.0,
      2.0,
      1.0,
      0.0,
      1.0,
      1.0
    ],
    "golospjogo_avg": 1.0,
    "golospjogo_max": 2.0,
    "golospjogo_sum": 6.0
  },
  {
    "nome": "Romelu Lukaku",
    "numero": "9",
    "equipa": "Inter de Milão",
    "golospjogo": [
      3.0,
      2.0,
      1.0,
      0.0,
      1.0,
      2.0
    ],
    "golospjogo_avg": 1.5,
    "golospjogo_max": 3.0,
    "golospjogo_sum": 9.0
  },
  {
    "nome": "Lautaro Martnez",
    "numero": "10",
    "equipa": "Inter de Milão",
    "golospjogo": [
      0.0,
      2.0,
      1.0,
      3.0,
      1.0,
      0.0
    ],
    "golospjogo_avg": 1.1666666666666667,
    "golospjogo_max": 3.0,
    "golospjogo_sum": 7.0
  }
]
```

Capítulo 4

Conclusão

Em jeito de conclusão, o grupo acredita que cumpriu com todos os objetivos principais do trabalho e conseguiu produzir um programa eficiente que converte com sucesso qualquer ficheiro de texto em formato *CSV* para um ficheiro *JSON* com as respetivas operações de *fold*, cumprindo assim os requisitos do enunciado.

Com este trabalho o grupo aprendeu que tinha em sua disposição uma ferramenta potente e fléxivel para manipular dados. As expressões regulares estão presentes em todas as linguagem modernas de programação e por isso podem ser uma mais valia para qualquer tipo de projeto no futuro.

Em termos pedagógicos, todos os elementos concluem que o trabalho foi bastante enriquecedor e que aprofundou o nosso conhecimento na área da UC, principalmente na parte das expressões regulares.

Apêndice A

Código do Programa

```
# Library json para a conversão
import json
#Library re para fazer as expresses regulares
import re
#Library sys para podermos receber e ler argumentos na linha de comandos
import sys

#Se recebermos o nome do ficheiro como argumento então converte esse ficheiro, caso contrário
    usa o caminho por defeito
if(len(sys.argv) == 1):
    filePath = "test.csv"
else:
    filePath = sys.argv[1]

#Abre o ficheiro e inicializa a lista JSON
f = open(filePath,"r",encoding='utf8')
res = []

#Separa todos os argumentos delimitados por ; na primeira linha do csv removendo os \n se
    existirem
args = [re.sub("\n","",x) if re.search("\n",x) else x for x in re.split(';',f.readline())]

#Começa a ler o resto das linhas do csv
for linha in f.readlines():

    #Separa todos as caracteristicas dos elementos delimitadas por ; removendo os \n se existirem
    listaelem = [re.sub("\n","",x) if re.search("\n",x) else x for x in re.split(';',linha)]

    #Se a quantidade de valores e a quantidade de argumentos for diferente atiramos um erro e o
        programa para
    if(len(args) != len(listaelem)):
        raise Exception("CSV Inválido! (Num de argumentos != Num de Valores)")

    #Cria um dicionario para o elemento (formato usado pelo json)
    elem = {}

    #Itera sobre a lista de argumentos para os relacionar com as caracteristicas
    for index in range(len(args)):
```

```
        #Verifica se o argumento contem uma lista
        if re.search("\*",args[index]):

            #Se o elemento não for uma lista de formato (num,...) então atiramos um erro e o
                programa para
            if not re.match("^\(\d+(?:\.\d+)*)(,(\d+(?:\.\d+)*))*\)$",listaelem[index]):
                raise Exception("CSV Inválido! (Lista inválida)")

            #Cria uma lista com o array de valores
            listaValores = [float(x) for x in re.findall("\d+(?:\.\d+)?",listaelem[index])]

            #Cria uma lista com todas as operaçes sobre a lista do CSV
            listaAgr = re.split("\*",args[index])

            #Itera cada operação para executar
            for agr in listaAgr[1:]:
                #Agregação avg
                if agr == "avg":
                    elem[listaAgr[0]+'_'+agr] = sum(listaValores) / len(listaValores)
                #Agregação sum
                elif agr == "sum":
                    elem[listaAgr[0]+'_'+agr] = sum(listaValores)
                #Agregação max
                elif agr == "max":
                    listaValores.sort()
                    elem[listaAgr[0]+'_'+agr] = listaValores[-1]
                #Agregação min
                elif agr == "min":
                    listaValores.sort()
                    elem[listaAgr[0]+'_'+agr] = listaValores[0]
                #Se não tiver nenhuma agregação
                else:
                    elem[re.sub("\*\w*","",args[index])] = listaValores
        #Se não for uma lista
        else:
            elem[args[index]] = listaelem[index]

    #Adiciona o elemento álista json
    res.append(elem)

#Escreve o json num novo ficheiro
with open(re.sub("csv$","json",filePath), 'w', encoding='utf-8') as jsonFile:
    json.dump(res, jsonFile,indent=2,ensure_ascii=False)
```