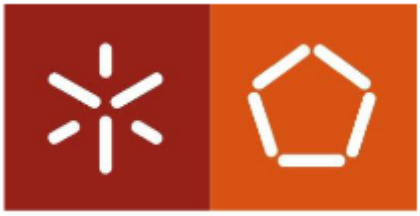

Processamento de Linguagens
(3º ano LEI)

Trabalho Prático 1
Enunciado 1 (CSV)

Relatório de Desenvolvimento

Grupo 45

a93241 Francisco Reis Izquierdo
a89526 Duarte Augusto Rodrigues Lucas
a83920 Afonso Trindade de Araújo Pascoal Faria

24 de Março de 2022

Conteúdo

Listings

Lista de Figuras

1 Introdução

No âmbito da unidade curricular de Processamento de Linguagens, foi proposto ao grupo de trabalho a escolha e consequente implementação de um dos enunciados de trabalho dispostos pela equipa docente. O enunciado escolhido pelo grupo tem como especificação a conversão de ficheiros **CSV** com listas e funções de agregação para ficheiros **JSON**. Alado a isto, os enunciados disponíveis tinham como principal foco em comum o desenvolvimento de um filtro de texto para fazer o reconhecimento dos padrões identificados de forma a proceder à transformação pedida no enunciado respetivo, recorrendo à linguagem de programação em Python. Posto isto, o presente relatório tem como objetivo dar a conhecer toda a informação acerca da abordagem tomada para com o problema em estudo, bem como estratégias adotadas e focar o desenvolvimento e implementação face ao enunciado escolhido e descrito acima. Além disso, o grupo de trabalho adotou algumas das metodologias abordadas nas aulas teóricas e práticas de forma a compactuar com o pedido no enunciado e aplicar os conceitos prestados pela equipa docente, de entre os quais podemos referir o uso das expressões regulares como o principal conceito que foi abordado em detalhe ao longo do projeto respetivo ao relatório em questão. Por fim, na linha de raciocínio previamente mencionada, foram usadas ferramentas no auxílio da implementação do projeto, nomeadamente módulos que permitissem a utilização dos conceitos previamente referidos. Além disto, o grupo de trabalho criou diversos ficheiros de entrada coerentes de forma a realizar testes que permitissem tirar conclusões assertivas acerca do que podia ser melhorado e se estávamos a ir de encontro ao pedido pelas especificações do enunciado em estudo.

2 Análise do problema

2.1 Descrição do Problema

Tal como referido anteriormente, o enunciado escolhido pelo grupo de trabalho tem como especificação a conversão de ficheiros característicos em formato CSV em ficheiros cujo formato final é JSON. O cerne do problema consiste no detalhe prestado aquando das especificações do enunciado, no qual as mesmas fazem referência às características presentes nos ficheiros de entrada, isto é nos ficheiros em formato CSV, no qual os mesmo podem ter lista e/ou funções de agregação. Alado a isto, o formato presente no corpo do ficheiro CSV é descrito no cabeçalho do mesmo, sendo que este cabeçalho corresponde à primeira linha do ficheiro. Assim, um dos principais objetivos é criar um filtro de texto capaz de analisar e filtrar a informação proveniente do cabeçalho de forma a compactuar com a posterior leitura do restante ficheiro e conversão do mesmo, de forma coerente e respeitando o teor dos exemplos prestados no enunciado.

2.2 Formato CSV e Formato JSON

É também importante salientar as diferentes e possíveis características descritas no cabeçalho que dizem respeito à coluna a que pertencem, no qual podemos ter, atributo, listas com tamanho definido, listas com intervalo de tamanhos e listas com funções de agregação. Assim, para cada elemento mencionado e que se encontra declarado no cabeçalho do ficheiro, convém criar um mecanismo que através de um filtro de texto a desenvolver pelo grupo, consegue facilitar no processo de conversão previamente mencionado. Seguem-se abaixo, exemplos prestados no enunciado relativos a cada elemento descrito acima, no qual é feita a correlação com um ficheiro de entrada **CSV** relativo a informações de alunos, bem como o respetivo *output* esperado para cada exemplo no ficheiro de saída **JSON**.

- Atributo: Para este elemento, apenas é utilizada a identificação do campo.

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

Figura 1: Exemplo de atributo (ficheiro de entrada)

```
[
    {
        "Número": "3612",
        "Nome": "Cândido Faísca",
        "Curso": "Teatro",
    },
    {
        "Número": "7777",
        "Nome": "Cristiano Ronaldo",
        "Curso": "Desporto",
    },
    {
        "Número": "264",
        "Nome": "Marcelo Sousa",
        "Curso": "Ciência Política",
    }
]
```

Figura 2: Exemplo de atributo (ficheiro de saída)

- Listas com tamanho definido: Para estes elementos, é mencionado o atributo bem como o tamanho N da lista que contém os vários elementos relativos ao atributo em questão, sendo que para o exemplo em questão temos $N = 5$. É importante mencionar que a presença de N vírgulas coincide com o número de colunas que abrange.

```
Número,Nome,Curso,Notas5,,,,,
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
```

Figura 3: Exemplo de lista com tamanho definido $N = 5$ (ficheiro de entrada)

```
[
    {
        "Número": "3612",
        "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]
    }
]
```

Figura 4: Exemplo de lista com tamanho definido $N = 5$ (ficheiro de saída)

- Listas com intervalo de tamanhos: Para estes elementos, é mencionado o atributo bem como o valor mínimo N e máximo M do tamanho da lista que contém os vários elementos relativos ao atributo em questão, sendo que para o exemplo em questão iremos utilizar $N = 3$ e $M = 5$. É importante mencionar que a presença de M vírgulas coincide com o número de colunas que abrange.

```
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,
```

Figura 5: Exemplo de lista com intervalos de tamanho $N = 3$ e $M = 5$ (ficheiro de entrada)

```
[
    {
        "Número": "3612",
        "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]
    }
]
```

Figura 6: Exemplo de lista com intervalos de tamanho $N = 3$ e $M = 5$ (ficheiro de saída)

- Funções de agregação: Para estes elementos, para além de mencionado o atributo e da lista que contém os vários elementos relativos ao atributo em questão, é mencionada a função de agregação, isto é a função a ser aplicada a cada elemento da lista, sendo que para o exempo em questão temos como função de agregação, a função *sum* que soma todos os elementos da lista.

```
Número,Nome,Curso,Notas3,5::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,
```

Figura 7: Exemplo de função de agregação com intervalos de tamanho $N = 3$ e $M = 5$ (ficheiro de entrada)

```
[
    {
        "Número": "3612",
        "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
    }
]
```

Figura 8: Exemplo de função de agregação com intervalos de tamanho $N = 3$ e $M = 5$ (ficheiro de saída)

2.3 Módulos

É importante também realçar aspetos ainda não mencionados relativos a informações prestadas no enunciado. De entre as várias informações, destaca-se o facto de, para a realização do projeto, não ser permitida a utilização dos módulos *csv* e *json* relativamente à leitura e escrita de ficheiros. Com isto, a leitura e escrita dos ficheiros em causa devem ser tidos em conta, no qual há que ter especial atenção relativamente a ambos, de forma a simplificar tanto quanto possível o trabalho. Primeiramente, iremos dar ênfase à leitura do ficheiro CSV, uma vez que a sua leitura não pode ser auxiliada pelo módulo csv. Assim, a leitura do ficheiro de entrada é efetuada através de um ciclo no qual é feito para

cada linha proveniente do ficheiro, o respetivo tratamento, de forma a posteriormente ser feita a escrita da mesma no ficheiro de saída. Por outro lado, a escrita do ficheiro de saída é feita através de *pretty print* de forma a que o resultado final coincida com os exemplos prestados. Além disto, convém mencionar o uso dos módulos *re* e *ply* no que concerne à identificação de padrões de texto ao longo do ficheiro de entrada e que auxiliaram na leitura do mesmo. Também podemos mencionar o uso de módulos tais como sys no qual este é usado para verificar a integridade do ficheiro de entrada, isto é se realmente o formato do mesmo é CSV, bem como o uso do módulo *statistics* o qual auxilia no uso de cálculos matemáticos relativos a algumas das funções de agregação.

```
1 import sys
2 import ply.lex as lex
3 import re
4 import statistics
```

Listing 1: Modulos

3 Estratégias Adotadas

Uma vez especificado o teor dos vários tipos de elementos que um cabeçalho do ficheiro de entrada pode conter por forma a caracterizar o seu conteúdo, o próximo passo consiste na descrição de todas as estratégias adotadas pelo grupo de trabalho com vista a alcançar um determinado objetivo. As várias estratégias adotadas visam a convergir para a solução proposta, para a qual cada etapa relativa às estratégias mencionadas brevemente, mostraram ser fulcrais.

3.1 Estruturas de Dados

Posto o supramencionado acerca do ficheiro de entrada, por forma a caracterizar o seu corpo e o seu conteúdo, uma das estratégias adotadas foi a implementação de estruturas de dados que consistissem no armazenamento da informação aquando da leitura do ficheiro. Assim, para cada elemento acima descrito, foi criada uma estrutura de dados apropriada, com o intuito de armazenar toda a informação necessária, para a escrita do ficheiro de saída ser completa. Um aspeto em comum a todas as estruturas de dados desenvolvidas é o facto de estas terem sido implementadas como classes em Python. Com isto, segue-se a descrição das respetivas estruturas de dados.

- *ColumnId*: Esta estrutura de dados visa a representar o tipo de dados mais simples, isto é sem envolver listas ou funções de agregação, ou seja

apenas o atributo. A variável de instância *id* diz respeito ao atributo, isto é ao atributo identificado na coluna do cabeçalho e a variável de instância *parameter* corresponde ao "valor" ao longo da coluna do atributo.

```
1 # Global struture for ColumnId
2 class ColumnID:
3
4   # Function to initialize an object of this class
5   def __init__(self, id):
6     self.name = id
7     self.parameter = ""
```

Listing 2: ColumnId

- *DefinedList*: Esta estrutura de dados visa a representar o tipo de dados relativo às lista de tamanho definido. A variável de instância *id* diz respeito ao atributo do campo presente no cabeçalho, a variável de instância *size* corresponde ao tamanho definido da lista e por fim a variável de instância *list* diz respeito à lista que contêm os vários elementos.

```
1 # Global struture for DEFINEDLIST
2 class DefinedList:
3
4   # Function to initialize an object of this class
5   def __init__(self, id, value):
6     self.name = id
7     self.size = value
8     self.list = []
```

Listing 3: DefinedList

- *BetweenList*: Esta estrutura de dados visa a representar o tipo de dados relativo às lista com intervalo de tamanhos, no qual a variável de instância *id* diz respeito ao atributo do campo presente no cabeçalho, as variáveis de instância *min* e *max* representam o valor mínimo e máximo do intervalo respetivamente, e por fim a variável de instância *list* diz respeito à lista que contêm os vários elementos.

```
# Global struture for BETWEENLIST
class BetweenList:

  # Function to initialize an object of this class
  def __init__(self, id, min, max):
    self.name = id
    self.min = min
    self.max = max
    self.list = []
```

Listing 4: BetweenList

- *FunctionAgregBList*: Esta estrutura de dados visa a representar o tipo de dados relativo a funções de agregação aplicadas a listas com intervalos de tamanho, no qual a variável de instância *id* diz respeito ao atributo do campo presente no cabeçalho, as variáveis de instância *min* e *max* representam o valor mínimo e máximo do intervalo respetivamente, a variável de instância *function* diz respeito à função a aplicar aos valores que se encontram na variável de instância *list* que diz respeito à lista que contêm os vários elementos.

```
# Global struture for FUNCTIONAGREGBLIST
class FunctionAgregBList:

  # Function to initialize an object of this class
  def __init__(self, id, min, max, function):
    self.name = id
    self.min = min
    self.max = max
    self.function = function
    self.list = []
```

Listing 5: FunctionAgregBList

- *FunctionAgregDList*: Esta estrutura de dados visa a representar o tipo de dados relativo a funções de agregação aplicadas a listas de tamanho definido, no qual a variável de instância *id* diz respeito ao atributo do campo presente no cabeçalho, a variável de instância *size* representa o tamanho da variável de instância *list*, cuja função de agregação a aplicar a cada elemento é representada pela variável de instância *function*.

```
# Global struture for FUNCTIONAGREGDLIST
class FunctionAgregDList:

  # Function to initialize an object of this class
  def __init__(self, id, size, function):
    self.name = id
    self.size = size
    self.function = function
    self.list = []
```

Listing 6: FunctionAgregDList

3.2 Processamento de informação

Tal como referido anteriormente, para os ficheiros de entrada, há que ter especial atenção para a primeira linha, uma vez que a mesma é o cabeçalho que caracteriza o corpo do ficheiro. Este cabeçalho é lido de forma a perceber o que cada campo do mesmo representa. Assim, a estratégia adotada no que concerne ao processamento do cabeçalho foi criar uma lista de objetos, cujos objetos são as estruturas de dados já mencionadas. Assim, para cada campo do cabeçalho que coincida com alguma das estruturas de dados previamente descrita, é criado o objeto respetivo e posteriormente o mesmo é guardado na variável global *formatHeader*. Desta forma, à medida que se vão lendo as linhas cuja informação é providencial, para cada elemento da mesma, conseguimos comparar ao formato do cabeçalho processado. Posto isto, é importante mencionar que a identificação de cada campo do cabeçalho com a respetiva estrutura de dados é feita através do uso das expressões regulares, no qual as mesmas tiveram um papel fulcral no processamento do cabeçalho, bem como na separação de cada elemento de cada linha do ficheiro, sendo este tema abordado brevemente.

3.2.1 Processamento do cabeçalho

Tal como referido anteriormente, para cada campo do cabeçalho bem como para a separação de cada elemento da linha a ser lida, é necessário o desenvolvimento de um filtro de texto capaz de identificar cada elemento descrito acima. Para tal, através do já mencionado módulo *ply* conseguimos desenvolver os *tokens* que caracterizam os elementos descritos. Além disso, como foi anteriormente mencionado o processamento do cabeçalho é feito lendo cada campo do mesmo, sendo que alado ao desenvolvimento dos *tokens* para cada campo, é efetuado uma procura pelo *token* correspondente, através da função *buildHeader*.

```
# Function to build the format header
def buildHeader():
  global header
  with open(fileCSV, "r") as f:
    cabecalho = f.readline().strip()
    lexer = lex.lex()

    lexer.input(cabecalho)
    while True:
      for token in lexer:
      break
  f.close()
```

Listing 7: Função *buildHeader*

3.2.2 Tokens

Posto isto, é importante mencionar um componente fulcral na identificação dos *tokens*, sendo este componente o *lexer*, permitindo atribuir um valor sintático às *strings* presentes em cada campo do cabeçalho. Com isto, os *tokens* desenvolvidos pelo grupo são *ID*, representando os atributos, *DEFINEDLIST*, representando as listas de tamanho definido, *BETWEENLIST*, representando as listas de intervalos de tamanhos e *FUNCTIONAGREGBLIST* representando as funções de agregação associadas às listas com intervalos de tamanho e *FUNCTIONAGREGDLIST* representando as funções de agregação associadas às listas de tamanho definido. Além destes *tokens* convém dar também ênfase ao *token COMMA* no qual este é utilizado não para o processamento do cabeçalho, mas sim para o processamento das restantes linhas do ficheiro, de forma a fazer a separação entre vírgulas e posteriormente individualizar cada elemento da mesma.

```
###################### Defining the tokens ##################

# List of tokens for the syntax

tokens = [ 'ID', 'DEFINEDLIST', 'BETWEENLIST', 'FUNCTIONAGREGBLIST'
    , "FUNCTIONAGREGDLIST", "COMMA"]

#############################################################
```

Listing 8: Lista de *tokens*

3.2.3 Expressões Regulares

A escolha das expressões regulares revelou-se bastante desafiante e interessante devido à necessidade de uma recolha especifica de dados do ficheiro de origem.

Para tal, foi necessário um estudo prévio da estrutura do ficheiro de origem com o objetivo de percebermos de que forma os dados estariam dispostos e como os iriamos interpretar. Posto isto, o grupo de trabalho foi desenvolvendo expressões regulares na vertente de especificar cada componente presente nos tokens. Assim, temos as seguintes expressões regulares que caracterizam componentes presentes nos *tokens* previamente referidos.

```
1 ###################### Regular Expressions ########################
2
3 columnID = r'(?:[a-zA-Z\u00C0-\u017F\/_]\s*\d*)+'
4 size = r'\{(\d+)\}'
5 interval = r'\{(\d+)\,(\d+)\}'
6 function = r'\:\:(\w+)'
7 readCell = r'([^,\n\t]*)(,|\n|\t?)'
8
9 ###################################################################
```

Listing 9: Expressões Regulares

- *columnID*: A expressão regular desenvolvida visa a capturar qualquer conjunto de caracteres, uma vez que queremos capturar a identificação do atributo.
- *size*: A expressão regular desenvolvida visa a capturar qualquer número que represente um tamanho, uma vez que queremos capturar o tamanho associado a uma lista.
- *interval*: A expressão regular desenvolvida visa a capturar qualquer intervalo de valores (números) que represente um intervalo de tamanhos associado a uma lista.
- *function*: A expressão regular desenvolvida visa a capturar qualquer função, sendo que neste caso queremos capturar o operador relevante a estas, sendo este '::', bem como a função.
- *readCell*: A expressão regular desenvolvida visa a capturar qualquer palavra seguida de uma ou mais vírgulas, por forma a facilitar o tratamento dos diversos elementos de uma linha.

3.2.4 Lexer

Como anteriormente referido umas das componentes importantes deste trabalho é o *lexer*, na medida em que este é responsável pela transformação de uma string sem sentido em uma lista de simples coisas, sendo também capaz de reconhecer identificadores reservados e descartar espaços em branco. Com isto, foram definidas várias funções de forma que o *lexer* atribua o valor sintático a cada token

anteriormente descrito. Além disso, para cada conjunto de regras, após estas é também efetuado a criação do objeto correspondente ao *token* identificado, convertendo-o à estrutura de dados respetiva, sendo posteriormente acrescentado à lista global *formatHeader*.

```
# Rule for token FUNCTIONAGREGBLIST
def t_FUNCTIONAGREGBLIST(t):

  r'(?:[a-zA-Z\u00C0-\u017F\/_]\s*\d*)+\{\d+\,\d+\}\:\:\w+'

  regex = fr'({columnID})({interval})({function})'
  regexExp = re.compile(regex)

  obj = FunctionAgregBList(regexExp.search(t.value).group(1), int(
    regexExp.search(t.value).group(3)),
   int(regexExp.search(t.value).group(4)), regexExp.search(t.value)
    .group(6))

  global formatHeader
  formatHeader.append(obj)
```

Listing 10: Regra sintática para *FUNCTIONAGREGBLIST*

```
# Rule for token FUNCTIONAGREGDLIST
def t_FUNCTIONAGREGDLIST(t):

  r'(?:[a-zA-Z\u00C0-\u017F\/_]\s*\d*)+\{\d+\}\:\:\w+'

  regex = fr'({columnID})({size})({function})'
  regexExp = re.compile(regex)

  obj = FunctionAgregDList(regexExp.search(t.value).group(1), int(
    regexExp.search(t.value).group(3)),
    regexExp.search(t.value).group(5))

  global formatHeader
  formatHeader.append(obj)
```

Listing 11: Regra sintática para *FUNCTIONAGREGDLIST*

```
# Rule for token BETWEENLIST
def t_BETWEENLIST(t):

  r'(?:[a-zA-Z\u00C0-\u017F\/_]\s*\d*)+\{\d+\,\d+\}'

  regex = fr'({columnID})({interval})'

  regexExp = re.compile(regex)

  obj = BetweenList(regexExp.search(t.value).group(1), int(regexExp
    .search(t.value).group(3)),
   int(regexExp.search(t.value).group(4)))

  global formatHeader
  formatHeader.append(obj)
```

Listing 12: Regra sintática para *BETWEENLIST*

```
# Rule for token DEFINEDLIST
def t_DEFINEDLIST(t):

  r'(?:[a-zA-Z\u00C0-\u017F\/_]\s*\d*)+\{(\d+)\}'

  regex = fr'({columnID})({size})'

  regexExp = re.compile(regex)

  obj = DefinedList(regexExp.search(t.value).group(1), int(regexExp
    .search(t.value).group(3)))

  global formatHeader
  formatHeader.append(obj)
```

Listing 13: Regra sintática para *DEFINEDLIST*

```
# Rule for token ID
def t_ID(t):

  r'(?:[a-zA-Z\u00C0-\u017F\/_]\s*\d*)+'
  global formatHeader

  obj = ColumnID(t.value)

  formatHeader.append(obj)
```

Listing 14: Regra sintática para *ID*

Além destas regras, foram definidas duas regras de controlo, de forma a prevenir e também facilitar o processamento de informação, no qual se definiu a regra sintática associada à informação a ignorar, nomeadamente vírgulas e espaços, como também a regra associada a erro, caso nenhuma informação conseguisse ser processada pelas regras previamente descritas.

```
# Rule to ignore
t_ignore = r',| "'
```

Listing 15: Regra sintática para ignorar

```
# Throw error if unkown token
def t_error(t):
  print("Error: Unknown token: ", t)
  sys.exit("")
```

Listing 16: Regra sintática para erro

3.2.5 Processamento das linhas

Através do cabeçalho processado, a leitura das restantes linhas devem seguir o padrão descrito por este, no qual para cada linha, é utilizada a regra descrita para o *token COMMA*, sendo que através de um iterador, conseguimos realizar o tratamento devido ao elemento da linha. Este tratamento passa pela conversão do elemento ao objeto correspondente, isto é através da comparação do cabeçalho processado, conseguimos perceber qual o teor do elemento em causa, criando a estrutura de dados respetiva e povoando as variáveis de instância com as características respetivas a esse elemento. Cada objeto é criado através da função *buildLine* e é guardado numa lista para posterior criação do dicionário respetivo.

```
# Function that converts the CSV file to the JSON file
def csv2json():
  with open(fileCSV, "r") as file:
    maxLine = 0
    allLines = []
    regexExp = re.compile(readCell)

    for line in file:
      list2JSON = []
      line = line.strip()

      if maxLine == 0:
        maxLine += 1

      else:
        listOfCells = []
        for element in regexExp.finditer(line):
          listOfCells.append(element.group(1))

        list2JSON = buildLine(listOfCells)
```

Listing 17: Função que converte o ficheiro **CSV** em **JSON**

3.3 Conversão de informação

3.3.1 Criação de dicionários

Após o processamento de cada linha do ficheiro de entrada, o seguinte passo, consiste na criação do dicionário respetivo de forma a guardar a informação relevante e de forma simples. Uma vez que, e tal como foi explicado, para cada linha é criada uma lista de objetos que são a estrutura de dados apropriada para cada elemento da linha em questão, o dicionário respetivo é criado em volta da mesma temática, no qual é iterado pela lista de objetos, comparando o tipo de cada um, procedendo à conversão do dicionário respetivo. Além disto, convém mencionar que é criado um dicionário para cada linha, no qual é feito um *update* à informação do mesmo, ao iterar-se sobre a lista de objetos supramencionada e comparando a mesma, de forma a tornar o dicionário coerente. Com isto, cada dicionário é posteriormente guardado numa lista de dicionários para, após a leitura de todas as linhas, proceder à escrita dos mesmos para o ficheiro de saída.

> **Nota:** No que concerne às funções de agregação, o grupo de trabalho abordou as funções propostas nos exemplos do enunciado, bem como a elaboração de novas funções, sendo estas funções *sum*, a soma de todos os elementos da lista, *media*, a média ponderada dos elementos da lista, *maisrecorrente*, o elemento mais recorrente da lista.

3.3.2 Escrita do ficheiro de saída

Com os dicionários prontos a serem escrito para o ficheiro de saída, devemos portanto voltar a atenção para tal. Assim, a escrita do ficheiro de saída é feita através da iteração sobre a lista contendo todos os dicionários, no qual para cada dicionário é feito o conceito de *pretty print*, no qual consiste em escrever a informação proveniente de cada dicionário de forma a que seja coerente com o expectável. Podemos também mencionar que o nome do ficheiro de saída coincide obviamente como o nome do ficheiro de entrada, havendo apenas a alteração à extensão do mesmo.

4 Exemplos de utilização

Uma vez elaborado e explicado todo o projeto que foi planeado, desenvolvido e implementado pelo grupo de trabalho, convém demonstrar a aplicação do mesmo, com exemplos de ficheiros **CSV** de entrada e a demonstração do respetivo ficheiro de saída **JSON**. Primeiramente iremos explicar o processo de como

"correr"o programa gerado através do projeto implementado. Assim, basta correr no terminal a invocação do programa da seguinte maneira:

```
>> python3 CSV.csv exemplo.csv
```

Para o qual será impresso em caso de sucesso, a seguinte mensagem na conversão para o ficheiro pretendido.

```
>> File exemplo.json sucessfully created!
```

4.1 Exemplos de ficheiros de entrada

Iremos agora mencionar alguns dos ficheiros de entrada utilizados/criados pelo grupo de trabalho na vertente de inspecionar possíveis falhas, mas também averiguar os resultados obtidos. Assim, destacam-se o ficheiro *emd.csv* no qual o seu conteúdo é referente a exames médicos desportivos, o ficheiro *alunos.csv*, no qual o seu conteúdo faz referência a informações relativas a alunos e também o ficheiro *jogadores.csv* no qual o seu conteúdo faz referência a informações relativas a jogadores de futebol. Para cada um destes ficheiros, iremos apresentar o seu conteúdo inicial (ou parte dele, uma vez que podem ser de grandes dimensões) e o resultado da conversão dos mesmos.

4.1.1 Ficheiro CSV - Exames médicos desportivos

```
_id,index,dataEMD,nome/primeiro,nome/último,idade,género,morada,modalidade,clube,email,federado,resultado
6045074cd77860ac9483d34e,0,2020-02-25,Delgado,Gay,28,F,Gloucester,BTT,ACRroriz,delgado.gay@acrroriz.biz,true,true
6045074ca6adebd591b5d239,1,2019-07-31,Foreman,Prince,34,M,Forestburg,Ciclismo,ACDRcrespos,foreman.prince@acdrcrespos.org,false,true
6045074c221e2fdf430e9ef0,2,2021-01-06,Cheryl,Berger,21,M,Umapine,Basquetebol,Vitoria,cheryl.berger@vitoria.biz,false,true
```

Figura 9: Ficheiro *emd.csv*

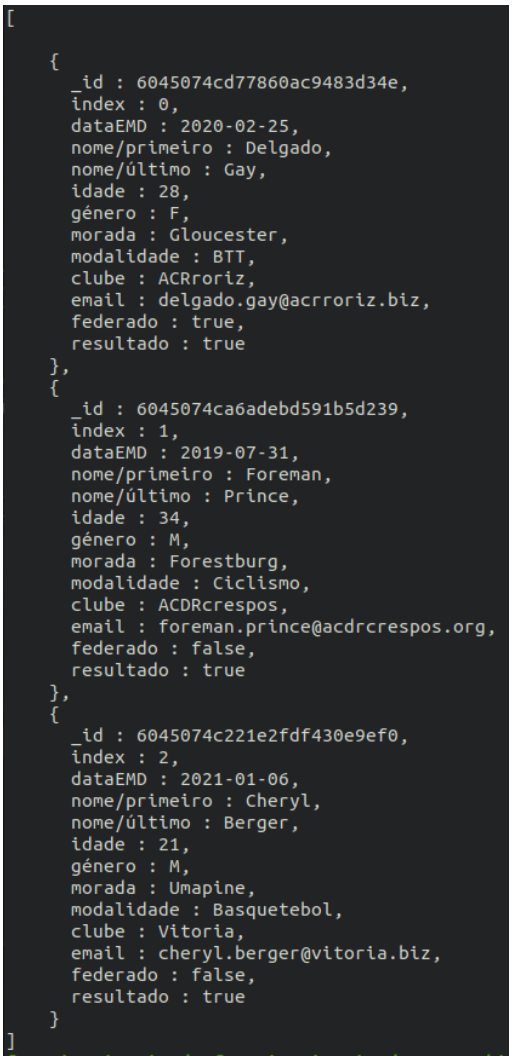

Figura 10: Ficheiro *emd.json*

4.1.2 Ficheiro CSV - Alunos

```
Número,Nome,Disciplinas{3},,,Emails{1,3},,,Notas{1,3}::sum,,
a93241,Francisco Reis Izquierdo,Processamento de Linguagens,Computação Gráfica,Base de Dados,a93241@alunos.uminho.pt,aluno1@hotmail.com,,18,18,17
a89745,António Martins,Processamento de Linguagens,Computação Gráfica,Investigação Operacional,a89745@alunos.uminho.pt,,,15,16,15
a97367,Timótio Tibúrcio Penedo da Rocha Calhau,Base de Dados,MNOL,Sistemas da Computação,a97367@alunos.uminho.pt,timotio@hotmail.com,penedocalhau@gmail.com,11,11,12
```

Figura 11: Ficheiro *alunos.csv*

```
[
   {
     Número : a93241,
     Nome : Francisco Reis Izquierdo,
     Disciplinas : ['Processamento de Linguagens', 'Computação Gráfica', 'Base de Dados'],
     Emails : ['a93241@alunos.uminho.pt', 'aluno1@hotmail.com'],
     Notas_sum : 53
   },
   {
     Número : a89745,
     Nome : António Martins,
     Disciplinas : ['Processamento de Linguagens', 'Computação Gráfica', 'Investigação Operacional'],
     Emails : ['a89745@alunos.uminho.pt'],
     Notas_sum : 46
   },
   {
     Número : a97367,
     Nome : Timótio Tibúrcio Penedo da Rocha Calhau,
     Disciplinas : ['Base de Dados', 'MNOL', 'Sistemas da Computação'],
     Emails : ['a97367@alunos.uminho.pt', 'timotio@hotmail.com', 'penedocalhau@gmail.com'],
     Notas_sum : 34
   }
]
```

Figura 12: Ficheiro *alunos.json*

4.1.3 Ficheiro CSV - Jogadores

```
Nome,Posições{1,3},,,Clubes{1,4},,,,ValoresdeTransferências{0,4}::media,,,
Cristiano Ronaldo dos Santos Aveiro,LW,ST,ME,Sporting Clube de Portugal,Manchester United,Real Madrid,Juventus,18000000,90000000,,
Lionel Andrés Messi Cuccittini,RW,CF,ST,Newell's Old Boys,Futebol Club Barcelona,Paris Saint Germain,,0,,,,
NeyTerra da Silva Santos Sénior,LW,ST,,Santos Futebol Clube,Futebol Club Barcelona,Paris Saint Germain,,88000000,222000000,108000000,25000000
Luva de Pedreiro,CF,ST,,Club de Regatas Vasco da Gama,,,,400000000,,,
```

Figura 13: Ficheiro *jogadores.csv*

```
[

    {
      Nome : Cristiano Ronaldo dos Santos Aveiro,
      Posições : ['LW', 'ST', 'ME'],
      Clubes : ['Sporting Clube de Portugal', 'Manchester United', 'Real Madrid', 'Juventus'],
      ValoresdeTransferências_media : 54000000.0
    },
    {
      Nome : Lionel Andrés Messi Cuccittini,
      Posições : ['RW', 'CF', 'ST'],
      Clubes : ["Newell's Old Boys", 'Futebol Club Barcelona', 'Paris Saint Germain'],
      ValoresdeTransferências_media : 0
    },
    {
      Nome : NeyTerra da Silva Santos Sénior,
      Posições : ['LW', 'ST'],
      Clubes : ['Santos Futebol Clube', 'Futebol Club Barcelona', 'Paris Saint Germain'],
      ValoresdeTransferências_media : 98000000.0
    },
    {
      Nome : Luva de Pedreiro,
      Posições : ['CF', 'ST'],
      Clubes : ['Club de Regatas Vasco da Gama'],
      ValoresdeTransferências_media : 400000000
    }
]
```

Figura 14: Ficheiro *jogadores.json*

5 Conclusão

A resolução deste trabalho prático demonstrou ser bastante desafiante e uma mais-valia para o nosso conhecimento, pois foi possível consolidar a matéria lecionada ao longo das aulas teóricas e práticas, permitindo explorar um pouco mais sobre a linguagem de programação Python. Por outro lado, foi possível praticar e aprofundar um pouco mais o nosso conhecimento sobre expressões regulares, dicionários em Python, e a utilização dos módulos *re* e *ply* e todas as suas funções. Além disso, o trabalho prático mostrou ser bastante crítco na vertente em que o mesmo exigiu ao grupo de trabalho todo um foco no planeamento relativamente ao modo como poderiamos ler e escrever os ficheiros pretendidos, bem como que abordagem poderiamos ter face ao uso de estruturas de dados. Por último, os *datasets* desenvolvidos e utilizados pelo grupo, mostraram ser uma mais valia, na medida em que os mesmos permitiram melhorar e refinar a implementação de todo o projeto desenvolvido.