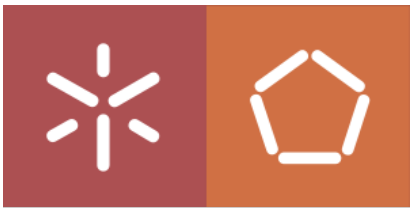

Universidade do Minho
Escola de Engenharia

# Universidade do Minho

## Licenciatura em Engenharia Informática

Processamento de Linguagens
PL1 - Ficheiros CSV com listas e funções de agregação

Diogo Casal Novo (a88276) João Lourenço (A93233)
João Ferreira (A93263)

março, 2022

# Conteúdo

**1 Introdução** **3**

**2 Análise do Problema** **4**

2.1 Descrição informal do problema . . . . . 4

2.2 Especificação . . . . . 4

2.2.1 Requisitos base . . . . . 4

2.2.2 Requisitos adicionais . . . . . 5

**3 Resolução** **6**

3.1 Algoritmo e Decisões . . . . . 6

3.1.1 readHeader . . . . . 6

3.1.2 readData . . . . . 7

3.1.3 writeData . . . . . 7

**4 Conclusão** **8**

# Capítulo 1

# Introdução

O projeto descrito no presente relatório encontra-se inserido no contexto da UC de Processamento de Linguagens.

A aplicação a desenvolver tem como objetivo a conversão de um ficheiro CSV (Comma separated values) para o formato JSON (JavaScript Object Notation). No entanto, o ficheiro original poderá conter algumas extensões, as quais requererão o preprocessamento dos dados associados à extensão antes da sua conversão para JSON.

O desenvolvimento deste projeto tem como objetivo a aplicação dos conhecimentos sobre Expressões Regulares e Python num caso concreto.

# Capítulo 2

# Análise do Problema

## 2.1 Descrição informal do problema

O sistema a desenvolver deverá ser um conversor de ficheiros gravados em formato Comma separated Values (CSV) para ficheiros em formato JavaScript Object Notation (JSON), um formato de ficheiro chave = valor. **O cabeçalho do ficheiro CSV descreverá os campos** que servirão como chaves no correspondente ficheiro JSON criado. O cabeçalho do ficheiro CSV suportará ainda **operações extra**:

1. **A presença de listas com tamanho definido**, através da notação $\{n\}$, sendo n um número natural;
2. **A presença de listas com intervalo de tamanhos**, através da notação $\{n,m\}$, sendo n e m números naturais, com n < m;
3. **O uso de funções de agregação**, aplicadas às listas já referidas, sendo estas funções a função *max*, *min*, *count*, *avg* e *sum*;

## 2.2 Especificação

Do enunciado referido na secção anterior é possível levantar um conjunto de requisitos que o projeto deverá cumprir. O grupo, no entanto, decidiu ir mais além, pelo que foi levantado um conjunto de requisitos adicionais que tem o objetivo de complementar o funcionamento da ferramenta.

### 2.2.1 Requisitos base

Os requisitos base foram obtidos através da análise do enunciado, presente de forma resumida na secção 2.1. Estes são de seguida enunciados.

1. Separar os vários campos do cabeçalho do ficheiro através do caractere ',';

2. Identificar os campos que possem a indicação de lista, através do conjunto de caracteres **"n"** ou **"n,m"**, com n, m $\in$ ao conjunto de números naturais e $\mathbf{n} < \mathbf{m}$;

3. Dos campos que possuem a indicação de lista, identificar os que possuem indicação para o uso de funções de agregação;

4. Em cada linha do corpo do CSV, separar os vários campos através do caractere ',';

5. Nos campos de uma linha afetados pela indicação de lista, levar essa indicação em conta no momento da leitura dos valores;

6. Aplicar a operação definida no cabeçalho à lista de valores, caso esta exista;

7. Converter os valores lidos para o correspondente par chave $\Longrightarrow$ valor, conjugando a chave lida no cabeçalho com o valor lido na linha;

8. Imprimir os pares chave $\Longrightarrow$ valor obtidos em notação *JSON*.

### 2.2.2 Requisitos adicionais

Embora os requisitos acima mencionados constituam uma base sólida para a aplicação, mas, tal como dito anteriormente, o grupo decidiu incluir mais alguns. Estes são:

1. No caso de não ser possível aplicar a operação à lista (por exemplo, se quisermos aplicar a operação max a uma lista que contém strings), o programa continua mas colocará a lista sem aplicar nenhuma função;

2. Sempre que possível, o programa tentará converter os números do tipo texto para reais;

3. O programa tem uma pequena interface que permite ao utilizador submeter o nome do ficheiro a converter.

# Capítulo 3

# Resolução

## 3.1 Algoritmo e Decisões

Como já foi mencionado antes, a primeira linha do ficheiro CSV é o cabeçalho e o restante documento é o corpo. Como o cabeçalho tem um formato específico e diferente do restante documento e o corpo do ficheiro depende do conteúdo do cabeçalho para poder ser corretamente interpretado, a estratégia mais razoável passa por dividir a interpretação do documento em duas fazes: interpretar o cabeçalho e interpretar o corpo.

A função *readHeader* é responsável por interpretar o conteúdo da primeira linha do ficheiro.

A função *readData* é responsável por interpretar cada uma das linhas do corpo do ficheiro.

Por sua vez ainda é necessário escrever o conteúdo interpretado num documento *JSON*. Para isso utilizamos a função *writeData* que recebe o resultado da função *readData* e escreve o conteúdo no documento.

### 3.1.1 readHeader

O cabeçalho de um ficheiro *CSV* contem informação sobre cada campo presente no ficheiro. Para alem do nome do campo, este pode conter algumas informações adicionais tais como se aquele campo é uma lista, qual o tamanho da lista ou entre que valores é que este pode variar, e adicionalmente, poderá ainda conter que tipo de função de agregação é que devemos aplicar aos elementos da lista.

Cada campo esta separado dos restantes por meio de uma virgula, mas, como se for fornecido um intervalo de comprimento de uma lista a virgula também é usada para separar os dois números, foi necessário usar a função *findall* e a seguinte expressão regular para separar os diversos campos numa lista.

```
([^{,]+{\d+}(?:::\w+)?|[^{,]+{\d+,\d+}(?:::\w+)?|[^{,]*)[\n,]
```

Por sua vez, apesar dos campos estarem separados ainda é necessário distinguir o que cada campo precisa. para isso, utilizamos um ciclo for que, para cada elemento da lista criada, utiliza uma função *match* com a expressão regular apresentada a seguir para descodifica o conteúdo de cada campo.

```
(?P<value>[^{,]*)({(?P<dig1>\d+)(,(?P<dig2>\d+))?}(::(?P<op>\w+))?)?
```

Esta expressão nomeia alguns grupos para que seja possível aceder diretamente aos conteúdos relevantes para o problema no caso de estes estarem presentes. Utilizando *group('value')* podemos aceder ao nome do campo, da mesma forma que *group('dig1')* e *group('dig2')* permitem aceder aos primeiro e segundo números fornecido, respetivamente, e *group('op')* permite aceder à função de agregação.

Finalmente, esta função guarda esta informação no array *headerList* para posteriormente ser utilizado.

### 3.1.2 readData

As linhas do corpo do documento são strings com os diversos campos separado por virgulas, logo, é possível usar uma função *split* para facilmente separar os diversos campos em substrings. Estes vão ser depois interpretados, ordenadamente, de acordo com a informação contida no array *headerList*.

Se o campo possuir mais que um único elemento, esta função agrupa os elementos deste numa lista e, no caso de ser solicitado aplicar uma função de agregação, também é responsabilidade desta função interpretar qual a funções de agregação pedida, e realizar os cálculos necessários. Só é possível executar estas operações caso seja possível interpretar os valores como inteiros. As funções de agregação possíveis são:

- **count:** quantos valores existem na lista
- **min:** qual o menor valor presente na lista
- **max:** qual o maior valor presente na lista
- **avg:** qual a media dos valores presentes na lista
- **sum:** qual o valor da soma de todos os elementos da lista

Esta função apenas recebe uma linha do documento de cada vez e retorna um dicionário correspondente ao conteúdo interpretado dessa mesma linha. O dicionário retornado já está com o formato no qual deve ser escrito.

### 3.1.3 writeData

Esta função é responsável por criar o ficheiro de output e de o preencher com o conteudo do ficheiro *CSV* mas em formato *JSON*. Para isso, esta invoca a função *readHeader* para esta intrepretar o cabeçalho e, em seguida e para cada linha do corpo, aplica a função *readData* que retorna um dicionário com o conteudo da linha intrepretado e a função *dictToFile* que escreve no ficheiro o conteudo do dicionário

#### dictToFile

Esta é uma função auxiliar que recebe um dicionario e um ficheiro, e escreve nesse ficheiro o conteudo do dicionario em formato *JSON*.

# Capítulo 4

# Conclusão

Conclui-se que que o projeto foi desenvolvido com sucesso, atingindo todas as metas e objetivos inicialmente propostos. Na realização do mesmo, foi aplicado e consolidado o conhecimento adquirido, até ao momento, na UC de Processamento de Linguagens, não só no âmbito de Expressões Regulares, mas também no uso da linguagem Python.