



# Universidade do Minho

Licenciatura em Engenharia Informática

## Processamento de Linguagens

TP1: Conversor de ficheiros *CSV*

### **Grupo 48**

Ana Murta (A93284)

Ana Henriques (A93268)

Rui Coelho (A58898)

março, 2022

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>5</b>
<b>2</b>	<b>Análise e especificação do problema</b>	<b>6</b>
2.1	Descrição do problema . . . . .	6
2.2	Especificação dos requisitos . . . . .	7
2.2.1	Requisitos adicionais . . . . .	7
2.2.2	<i>Output</i> esperado . . . . .	8
<b>3</b>	<b>Proposta de solução</b>	<b>10</b>
3.1	Estruturas de dados utilizadas . . . . .	10
3.2	CSV . . . . .	10
3.2.1	Processamento inicial . . . . .	10
3.2.2	Processamento do cabeçalho . . . . .	10
3.2.3	Processamento do corpo . . . . .	11
3.3	JSON . . . . .	11
3.3.1	Conversão . . . . .	11
3.4	Menu . . . . .	13
<b>4</b>	<b>Testes de conversão</b>	<b>14</b>
4.1	Ficheiros <i>CSV</i> de teste . . . . .	14
4.2	Ficheiros <i>JSON</i> . . . . .	15
<b>5</b>	<b>Conclusão</b>	<b>17</b>
<b>A</b>	<b>Código</b>	<b>18</b>
A.1	Conversão de um ficheiro <i>CSV</i> em <i>JSON</i> . . . . .	18
A.2	Visualização de ficheiros . . . . .	19
A.3	Menu principal de execução . . . . .	19
A.4	Processamento inicial do <i>input</i> . . . . .	19

A.5	Processamento do cabeçalho do <i>input</i> . . . . .	20
A.6	Processamento do corpo do <i>input</i> . . . . .	20
A.7	Cálculo das funções de agregação . . . . .	21
A.8	Criação de dicionários . . . . .	21
A.9	Preparação do ficheiro <i>JSON</i> . . . . .	21

# Lista de Figuras

2.1	Ficheiro: <i>alunos1.csv</i> . . . . .	6
2.2	Ficheiro: <i>alunos2.csv</i> . . . . .	7
2.3	Ficheiro: <i>alunos3.csv</i> . . . . .	7
2.4	Ficheiro: <i>alunos4.csv</i> . . . . .	7
2.5	Ficheiro: <i>alunos5.csv</i> . . . . .	8
2.6	Ficheiro: <i>alunos6.csv</i> . . . . .	8
2.7	<i>Output</i> esperado após a conversão do tipo de ficheiro. . . . .	9
3.1	Exemplo de <i>output JSON</i> . . . . .	12
3.2	Exemplo de <i>output JSON</i> . . . . .	12
3.3	Menu inicial. . . . .	13
3.4	Menu de visualização. . . . .	13
4.1	Ficheiro: <i>alunos7.csv</i> . . . . .	14
4.2	Ficheiro: <i>alunos8.csv</i> . . . . .	14
4.3	Mensagem de erro na conversão do ficheiro <i>alunos8.csv</i> . . . . .	15
4.4	Ficheiro <i>JSON</i> gerado para <i>alunos7.csv</i> . . . . .	16

# Lista de Tabelas

2.1 Formato tabelar dos dados do ficheiro *CSV* . . . . . 6

# Capítulo 1

## Introdução

O presente projeto centra-se no desenvolvimento de uma aplicação que permita efetuar a conversão de ficheiros *CSV* (*Comma Separated Values*) em ficheiros *JSON* (*JavaScript Object Notation*). Deste modo, o relatório aqui apresentado pretende dar conhecer o trabalho desenvolvido, assim como explicitar as decisões e estratégias adotadas pelo grupo aquando da criação da referida aplicação.

Em suma, a aplicação desenvolvida permite a leitura de ficheiros em formato *CSV*, com listas e funções de agregação, operando posteriormente sobre os dados lidos de modo a criar um ficheiro *JSON* válido.

## Capítulo 2

# Análise e especificação do problema

### 2.1 Descrição do problema

O conversor de ficheiros a desenvolver, tal como foi anteriormente mencionado, deve ser capaz de processar e converter ficheiros *CSV* em ficheiros de formato *JSON*.

Um ficheiro *CSV* permite o armazenamento de dados, de acordo com um formato estruturado de tabela. Assim sendo, a primeira linha presente neste tipo de ficheiros representa o cabeçalho, que permite identificar as diversas colunas da tabela, cuja separação é efetuada, tipicamente, através de vírgulas (,) – podendo, opcionalmente, usar-se outro caracter como separador. Seguidamente, as restantes linhas de um ficheiro *CSV* identificam as diversas entradas de dados da tabela. A Figura 2.1 apresenta uma porção de um ficheiro *CSV* – usado, posteriormente, para efeitos de teste do programa desenvolvido.

```
Número, Nome, Curso
12334, Ana Júlia, Desporto
96248, Ana Murta, Culinária
96236, Ana Henriques, Desporto
56471, Rui Coelho, Economia
```

Figura 2.1: Ficheiro: *alunos1.csv*.

A Tabela 2.1 apresenta a conversão dos dados presentes no *CSV* descrito na Figura 2.1.

Tabela 2.1: Formato tabelar dos dados do ficheiro *CSV*

Número	Nome	Curso
12334	Ana Júlia	Desporto
96248	Ana Murta	Culinária
96236	Ana Henriques	Desporto
56471	Rui Coelho	Economia

O formato acima apresentado representa o modelo mais simples de *CSV* considerado no desenho da aplicação. Para o desenvolvimento do conversor de ficheiros foram, adicionalmente, consideradas extensões do formato *CSV*, como o caso da incorporação de listas e de funções de agregação das listas.

A Figura 2.2 apresenta o formato *CSV* com a incorporação de listas de tamanho fixo. Tal como pode ser observado, uma coluna é descrita como uma lista de valores através do uso de  $\{N\}$ , onde  $N$  representa o tamanho, fixo, da lista.

```
Número,Nome,Curso,Notas{5}
12334,Ana Júlia,Desporto,10,12,11,16,17
96248,Ana Murta,Culinária,19,20,18,18,18
96236,Ana Henriques,Desporto,17,18,16,16,17
56471,Rui Coelho,Economia,15,18,17,15,16
```

Figura 2.2: Ficheiro: *alunos2.csv*.

Em alternativa, tal como pode ser observado na 2.3, as listas podem ter um tamanho variável, sendo, então, descritas por  $\{N,M\}$ , onde  $N$  representa o número mínimo de elementos da lista e  $M$  representa o número máximo.

```
Número,Nome,Curso,Notas{3,5},,,,,
12334,Ana Júlia,Desporto,10,12,11,,
96248,Ana Murta,Culinária,19,20,18,18,
96236,Ana Henriques,Desporto,17,18,16,,
56471,Rui Coelho,Economia,15,18,17,,
11247,João Mendes,Design,15,12,14,,
74215,Joana Oliveira,Psicologia,17,14,16,12,13
```

Figura 2.3: Ficheiro: *alunos3.csv*.

Por fim, e tal como foi referido anteriormente, as listas podem conter funções de agregação, que indicam operações a ser executadas à lista de valores presentes na lista. A descrição de funções de agregação é dada por  $::X$ , onde  $X$  corresponde ao nome da função a ser aplicada. A Figura 2.4 apresenta um ficheiro com as funções de agregação  $::media$  e  $::max$ , que sugerem o cálculo da média e do máximo da lista, respetivamente.

```
Número,Nome,Curso,Notas{3,5}::media,,,,,NotaFinal{2}::max,,Extracurricular,Universidade
12334,Ana Júlia,Desporto,10,12,11,,,16,17,Futsal,Universidade do Minho
96248,Ana Murta,Culinária,19,20,18,18,,,17,18,Patinagem Artística,Universidade do Minho
96236,Ana Henriques,Desporto,17,18,16,,,16,17,Voleibol,Universidade do Minho
56471,Rui Coelho,Economia,15,18,17,,,15,16,Futsal,Universidade do Minho
```

Figura 2.4: Ficheiro: *alunos4.csv*.

## 2.2 Especificação dos requisitos

Atendendo às formatações acima descritas do ficheiro *CSV*, e das extensões apresentadas, é esperado que o programa desenvolvido em *Python* seja capaz de:

- Ler as diversas linhas do ficheiro *CSV*
- Preparar e processar os dados recolhidos
- Armazenar os dados gerados num ficheiro *JSON*

### 2.2.1 Requisitos adicionais

Para além dos requisitos acima apresentados, é ainda possível efetuar a conversão de ficheiros *CSV* cujo delimitador de colunas corresponde a um caracter diferente da vírgula – como, por exemplo, seria o caso de um delimitador ponto e vírgula (;). As Figuras 2.5 e 2.6 apresentam exemplos de ficheiros *CSV* que recorrem a delimitadores diferentes.



```
Número;Nome;Curso;Notas{3;5}::media;;;;;NotaFinal{2}::max;;Extracurricular;Universidade
12334;Ana Júlia;Desporto;10;12;11;;;16;17;Futsal;Universidade do Minho
96248;Ana Murta;Culinária;19;20;18;18;;17;18;Patinagem Artística;Universidade do Minho
96236;Ana Henriques;Desporto;17;18;16;;;16;17;Voleibol;Universidade do Minho
56471;Rui Coelho;Economia;15;18;17;;;15;16;Futsal;Universidade do Minho
```

Figura 2.5: Ficheiro: *alunos5.csv*.

```
Número|Nome|Curso|Notas{3|5}::media||| |NotaFinal{2}::max| |Extracurricular|Universidade
12334|Ana Júlia|Desporto|10|12|11| |16|17|Futsal|Universidade do Minho
96248|Ana Murta|Culinária|19|20|18|18| |17|18|Patinagem Artística|Universidade do Minho
96236|Ana Henriques|Desporto|17|18|16| |16|17|Voleibol|Universidade do Minho
56471|Rui Coelho|Economia|15|18|17| |15|16|Futsal|Universidade do Minho
```

Figura 2.6: Ficheiro: *alunos6.csv*.

Adicionalmente, foi desenvolvida uma interface de utilização em linha de comandos, com vista a proporcionar um modo intuitivo para o utilizador interagir com a aplicação criada.

### 2.2.2 *Output* esperado

Considerando o ficheiro *alunos1.csv*, parcialmente representado na Figura 2.1, seria expectável que o programa gerasse um ficheiro *JSON* com o formato apresentado na Figura 2.7.

```

1  [
2      {
3          "Número": "12334",
4          "Nome": "Ana Júlia",
5          "Curso": "Desporto"
6      },
7      {
8          "Número": "96248",
9          "Nome": "Ana Murta",
10         "Curso": "Culinária"
11     },
12     {
13         "Número": "96236",
14         "Nome": "Ana Henriques",
15         "Curso": "Desporto"
16     },
17     {
18         "Número": "56471",
19         "Nome": "Rui Coelho",
20         "Curso": "Economia"
21     },
22     {
23         "Número": "11247",
24         "Nome": "João Mendes",
25         "Curso": "Design"
26     },
27     {
28         "Número": "74215",
29         "Nome": "Joana Oliveira",
30         "Curso": "Psicologia"
31     },
32     {
33         "Número": "94521",
34         "Nome": "Mafalda Gomes",
35         "Curso": "Culinária"
36     },
37     {
38         "Número": "45681",
39         "Nome": "Gustavo Cerqueira",
40         "Curso": "Ciência Política"
41     },
42     {
43         "Número": "32187",
44         "Nome": "Alexandre Costa",
45         "Curso": "Economia"
46     }
47 ]

```

Figura 2.7: *Output* esperado após a conversão do tipo de ficheiro.

## Capítulo 3

# Proposta de solução

### 3.1 Estruturas de dados utilizadas

Para o desenvolvimento do conversor de ficheiros, foram utilizadas estruturas de dados como listas, dicionários e tuplos. Estas estruturas possibilitaram o tratamento dos dados presentes em ficheiros *CSV* de um modo simples e conveniente, facilitando, assim, o desenvolvimento da aplicação final.

### 3.2 CSV

#### 3.2.1 Processamento inicial

A função *cleanInput* permite efetuar algum pré-processamento do ficheiro *CSV* a ser convertido. Essencialmente, a função uniformiza os separadores das diversas colunas do ficheiro de entrada, usando a *regex* `r';|\||\t'` para capturar os separadores e substituí-los pelo carácter vírgula (,). O processamento inicial termina com a limpeza de linhas vazias que possam existir no *input* de entrada.

#### 3.2.2 Processamento do cabeçalho

Tal como observado nas Figuras 2.5 e 2.6, o cabeçalho do ficheiro *CSV* contém informação sobre cada campo do mesmo: o nome da coluna, a definição do tamanho fixo ou variável de uma lista e, ainda, a possível aplicação de uma função de agregação aos elementos de uma lista. O conjunto de funções de agregação processadas pelo conversor implementado é o seguinte: `["sum", "media", "min", "max", "count"]`.

Assim sendo, o processamento do cabeçalho começa por usar a *regex* `r'([~;:,{}+](?:{(.*)})?(?:\:\:(.*)?(?:;|,))?)'` para efetuar a capturação de 3 grupos por coluna: o nome do campo, o tamanho do campo se este for uma lista e a identificação da função de agregação aplicada a essa lista. Quando o campo não corresponde a uma lista ou não é aplicada nenhuma função de agregação, é capturada uma string vazia ("). Para tornar estes resultados mais legíveis, se tivermos uma string vazia para o tamanho da lista e para a função de agregação aplicada, essa string é convertida para um 0 e para um "none", respetivamente.

Deste modo, na situação de termos, por exemplo, este cabeçalho:

```
Número, Nome, Curso, Notas{3,5}::media,,, ,NotaFinal{2}::max,, Universidade
```

A função responsável por processar o cabeçalho – *header* – retorna o seguinte resultado:

```
[('Número', 0, 'none'),  
 ('Nome', 0, 'none'),  
 ('Curso', 0, 'none'),
```

```
(('Notas', '3,5', 'media'),
 ('NotaFinal', '2', 'max'),
 ('Universidade', 0, 'none'))]
```

### 3.2.3 Processamento do corpo

O corpo do ficheiro *CSV* dado como *input* é processado iterativamente, linha a linha, pela função *processLine*. Para isso, esta necessita de receber dois parâmetros: a linha a ser processada e a lista devolvida pela função *header – columnOperations*, em que cada elemento é um tuplo que contém a informação relativa a cada campo do cabeçalho (*Name, Size if List, Function if Applied*).

Tendo já sido inferido que as colunas estão separadas por vírgulas, são extraídas todas as componentes que constituem a linha, guardando-as numa lista – *components*. Feito isto, esta lista é percorrida elemento a elemento à medida que se iteram os tuplos da *columnOperations*. Para cada iteração, são efetuados os seguintes passos:

1. Se o campo a iterar não for uma lista, processa-se apenas o nome dessa coluna e o seu respetivo valor na linha em questão.
2. Se o campo corresponder a uma lista – *Size != 0* –, é preciso ter em atenção que o seu tamanho pode ser fixo ou variável. Como tal, o tamanho é redefinido para o valor máximo que este pode tomar, possibilitando distinguir com sucesso os elementos de *components* que pertencem a esse campo.
3. Através da *regex* `r'^-?\d+(?:\.\d+)?\$'`, os valores que pertencem a esse campo são guardados numa lista auxiliar, sendo primeiramente convertidos de *string* para *int*.
4. Sendo identificada a aplicação de uma função de agregação, a função *executeFunction* efetua a sua execução e retorna `f'"{columnName}_functionName": {result}'`, sendo *result* o resultado da execução. Caso contrário, se não existir função de agregação, a *executeFunction* retorna apenas `f'"{columnName}": {values}'`, sendo *values* os elementos da lista desse campo.

No final, se a linha a processar for, por exemplo:

```
"32187,Alexandre Costa,Economia,16,11,15,,,12,13,Universidade do Minho"
```

O output retornado pela *processLine* é o seguinte:

```
[{"Numero": 32187',
  "Nome": Alexandre Costa',
  "Curso": Economia',
  "Notas_media": 14.0',
  "NotaFinal_max": 13',
  "Universidade": Universidade do Minho']
```

## 3.3 JSON

### 3.3.1 Conversão

A conversão dos dados é efetuada com recurso à função *convertFile*, que permite a geração de um ficheiro *JSON* com o formato desejado, tal como pode ser observado nas Figuras 3.1 e 3.2. A função opera sobre as diversas linhas do ficheiro *CSV*, através de funções descritas anteriormente – como *cleanInput*, *header*.

Após o processamento inicial, a função *geraDicionario* permite construir uma lista onde cada elemento corresponde a um dicionário que descreve os dados de cada linha do ficheiro de *input*: para cada dicionário, é efetuado o mapeamento entre as diversas colunas (chaves) e os seus dados (valores). Em cada iteração, e com recurso à função *processLine*, é calculada a lista com os dados já devidamente processados – lista que é usada para a criação do dicionário para a linha correspondente. Cada entrada do dicionário é descrita pelo par chave-valor,

```

1  [
2    {
3      "Número": "12334",
4      "Nome": "Ana Júlia",
5      "Curso": "Desporto",
6      "Notas_media": 11.0,
7      "NotaFinal_max": 17,
8      "Extracurricular": "Futsal",
9      "Universidade": "Universidade do Minho"
10   },
11   {
12     "Número": "96248",
13     "Nome": "Ana Murta",
14     "Curso": "Culinária",
15     "Notas_media": 18.75,
16     "NotaFinal_max": 18,
17     "Extracurricular": "Patinagem Artística",
18     "Universidade": "Universidade do Minho"
19   },
20   {
21     "Número": "11247",
22     "Nome": "João Mendes",
23     "Curso": "Design",
24     "Notas_media": 13.67,
25     "NotaFinal_max": 14,
26     "Extracurricular": "Voleibol",
27     "Universidade": "Universidade do Minho"
28   }
29 ]

```

Figura 3.1: Exemplo de *output JSON*.

```

1  [
2    {
3      "Número": "59876",
4      "Nome": "Francisco Rodrigues",
5      "Curso": "Arquitetura",
6      "Notas": [13,17,11,15,14]
7    },
8    {
9      "Número": "93214",
10     "Nome": "Teresa Marques",
11     "Curso": "Psicologia",
12     "Notas": [14,16,11]
13   },
14   {
15     "Número": "24610",
16     "Nome": "Bruna Gomes",
17     "Curso": "Design",
18     "Notas": [19,16,17,20]
19   }
20 ]

```

Figura 3.2: Exemplo de *output JSON*.

onde a chave corresponde ao nome da coluna (calculado aquando do processamento do cabeçalho) e o valor corresponde a dados já processados, sendo capturado através da expressão regular `r'(?!"([a-zâ-ü]+":))(.*)'`. No decurso desta função, se uma chave se encontrar mapeada para um valor vazio, i.e., , esse mapeamento não é introduzido no dicionário gerado, uma vez que dados do valor para a chave/coluna é inexistente.

Finda a criação da lista de dicionários, a função recorre à função auxiliar *prepareJSON*. Em suma, esta função itera sobre os vários dicionários, formatando os diversos campos segundo o modelo *JSON* desejado. Aquando do processo iterativo, é verificado:

- se a coluna representa uma lista – seguindo, neste caso, uma formatação genérica de lista (e.g.,  $[1,2,3]$ );
- se a coluna representa uma função de agregação – não introduzindo aspas em torno do resultado da calculado;
- a posição da coluna – uma vez que a última entrada do dicionário detém uma formatação diferente das restantes.

Uma vez terminada a execução desta função auxiliar, os dados são finalmente escritos no ficheiro *JSON* de *output*.

## 3.4 Menu

Como foi previamente mencionado, foi desenvolvida uma simples interface para possibilitar que o utilizador use o programa de um modo mais cómodo. O menu apresentado na 3.3 apresenta o menu de entrada da aplicação, que demonstra as opções de conversão de ficheiros e de visualização de ficheiros. Uma vez selecionada a opção de conversão, o utilizador introduz a informação necessária referente ao ficheiro *CSV* de *input* e ao nome desejado para o ficheiro *JSON* a ser gerado.

```
**** CSV TO JSON CONVERTER ****
1 :: Convert file
2 :: View file
0 :: Exit
> Option: █
```

Figura 3.3: Menu inicial.

```
* Select file extension *
1 :: CSV file
2 :: JSON file
0 :: Back
> Option: █
```

Figura 3.4: Menu de visualização.

A visualização do conteúdo de ficheiros é também possível, sendo, para tal, necessário selecionar a opção no menu inicial. O menu apresentado na Figura 3.4 representa as opções disponíveis para a visualização de ficheiros, podendo o utilizador consultar o conteúdo dos ficheiros *CSV* de *input* ou dos ficheiros *JSON* de *output* à sua escolha.

Em qualquer momento de acesso a ficheiros, seja para leitura aquando do início da conversão ou para consulta posterior, é gerada uma mensagem de erro na eventualidade de o documento a que o utilizador quer aceder não seja acessível.

## Capítulo 4

# Testes de conversão

### 4.1 Ficheiros *CSV* de teste

De modo averiguar o correto funcionamento do programa desenvolvido, foram criados diversos ficheiros *CSV* de teste, tendo alguns destes ficheiros sido previamente apresentados – cf. Figuras 2.1, 2.2, 2.3, 2.4, 2.5 e 2.6. Estes ficheiros foram testados, individualmente, com o conversor de formato – tendo, em todos os casos, sido gerado o output em formato *JSON* esperado.

Para além dos ficheiros previamente apresentados, foram criados dois ficheiros *CSV* adicionais para conduzir mais testes às funcionalidades implementadas. Estes ficheiros permitiram avaliar o comportamento da aplicação quando: (i) diversas colunas do ficheiro *CSV* de *input* encontram-se vazias; e (ii) alguma coluna detém uma função de agregação não reconhecida pelo programa.

A Figura 4.1 apresenta o caso usado para testar a condição (i) acima descrita. Adicionalmente, este ficheiro foi criado de modo a permitir um teste mais robusto para as funcionalidades implementadas, particularmente, para a incorporação de listas (com tamanho fixo e tamanho variável) e a execução de funções de agregação.

```
Número|Nome|Curso|Notas{3|5}::count|||||NotaFinal{2}|Extracurricular|Universidade
12334|Desporto|10|12|11|||16|17|Futsal|Universidade do Minho
96248|Ana Murta|19|20|18|18|17|18|Patinagem Artística|
96236|Ana Henriques|Desporto|16|17|Voleibol|Universidade do Minho
56471|Rui Coelho|15|18|17|11|12|15|16|Futsal|Universidade do Minho
74215|Joana Oliveira|Psicologia|17|14|16|12|13|13|14|Patinagem Artística|Universidade do Minho
94521|Mafalda Gomes|Culinária|14|16|18||14|15|Universidade do Minho
```

Figura 4.1: Ficheiro: *alunos7.csv*.

A Figura 4.2 apresenta um caso semelhante ao descrito anteriormente, distinguindo-se apenas pela incorporação de uma função de agregação na coluna “*NotaFinal*”, a função *fakeFun*, que não integra o conjunto de funções de agregação incorporadas na aplicação, pelo que é esperado que o ficheiro não seja convertido.

```
Número|Nome|Curso|Notas{3|5}::count|||||NotaFinal{2}::fakeFun|Extracurricular|Universidade
12334|Desporto|10|12|11|||16|17|Futsal|Universidade do Minho
96248|Ana Murta|19|20|18|18|17|18|Patinagem Artística|
96236|Ana Henriques|Desporto|16|17|Voleibol|Universidade do Minho
56471|Rui Coelho|15|18|17|11|12|15|16|Futsal|Universidade do Minho
74215|Joana Oliveira|Psicologia|17|14|16|12|13|13|14|Patinagem Artística|Universidade do Minho
94521|Mafalda Gomes|Culinária|14|16|18||14|15|Universidade do Minho
```

Figura 4.2: Ficheiro: *alunos8.csv*.

## 4.2 Ficheiros *JSON*

A conversão do ficheiro *CSV alunos8.csv*, tal como foi mencionado previamente, levantaria uma situação de erro, devido à existência da função *fakeFun* que não é reconhecida pelo programa. A Figura 4.3 apresenta a mensagem fornecida pelo programa aquando da tentativa de conversão do ficheiro referido.

```
[CSV] Insert file name (extension included): alunos8.csv
[FILE] Opened successfully.
[PRESS ENTER TO CONTINUE]
[JSON] Insert file name (extension included): alunos8.json
[ERROR] Unsupported function on CSV file.
[PRESS ENTER TO CONTINUE]
```

Figura 4.3: Mensagem de erro na conversão do ficheiro *alunos8.csv*.

A Figura 4.4 apresenta o *output* gerado pela aplicação após receber o ficheiro *CSV* apresentado na Figura 4.1. Como pode ser observado, o *output* gerado é congruente com as especificações enunciadas previamente – particularmente, no que diz respeito ao processamento das listas (com tamanho fixo ou tamanho variável) e à execução de funções de agregação.



```

1  [
2      {
3          "Número": "12334",
4          "Curso": "Desporto",
5          "Notas_count": 3,
6          "NotaFinal": [16,17],
7          "Extracurricular": "Futsal",
8          "Universidade": "Universidade do Minho"
9      },
10     {
11         "Número": "96248",
12         "Nome": "Ana Murta",
13         "Notas_count": 4,
14         "NotaFinal": [17,18],
15         "Extracurricular": "Patinagem Artística"
16     },
17     {
18         "Número": "96236",
19         "Nome": "Ana Henriques",
20         "Curso": "Desporto",
21         "NotaFinal": [16,17],
22         "Extracurricular": "Voleibol",
23         "Universidade": "Universidade do Minho"
24     },
25     {
26         "Número": "56471",
27         "Nome": "Rui Coelho",
28         "Notas_count": 5,
29         "NotaFinal": [15,16],
30         "Extracurricular": "Futsal",
31         "Universidade": "Universidade do Minho"
32     },
33     {
34         "Número": "74215",
35         "Nome": "Joana Oliveira",
36         "Curso": "Psicologia",
37         "Notas_count": 5,
38         "NotaFinal": [13,14],
39         "Extracurricular": "Patinagem Artística",
40         "Universidade": "Universidade do Minho"
41     },
42     {
43         "Número": "94521",
44         "Nome": "Mafalda Gomes",
45         "Curso": "Culinária",
46         "Notas_count": 3,
47         "NotaFinal": [14,15],
48         "Universidade": "Universidade do Minho"
49     }
50 ]

```

Figura 4.4: Ficheiro *JSON* gerado para *alunos7.csv*.

## Capítulo 5

# Conclusão

O presente trabalho centrou-se no desenvolvimento de uma aplicação capaz de converter ficheiros em formato *CSV* para o formato *JSON*. No decurso da concetualização e criação do programa previamente apresentado, a equipa conseguiu desenvolver e fortalecer competências no domínio da criação de expressões regulares, no processamento de linguagens e na utilização da linguagem de programação *Python* como ferramenta de desenvolvimento de *software*. Face aos objetivos estabelecidos e ao trabalho apresentado, a equipa considera ter alcançado com sucesso as metas esperadas, criando um programa funcional e simples que preenche os requisitos estipulados. Adicionalmente, o programa encontra-se apto para converter ficheiros genéricos de *CSV*, transformando-os em ficheiros *JSON* válidos.

# Apêndice A

## Código

### A.1 Conversão de um ficheiro *CSV* em *JSON*

```
def convertFile():
    lines = openFile(1)
    if lines == - 1:
        return -1
    if lines:
        output_name = input("[JSON] Insert file name (extension included): ")

        # PROCESSAMENTO INICAL DO INPUT
        lines = cleanInput(lines)

        # PROCESSAR O HEADER
        try:
            columnOperations = header(lines[0])
            lines.remove(lines[0])
        except NameError: # SE A FUNÇÃO DE AGREGAÇÃO NÃO EXISTIR, É LANÇADA UMA EXCEÇÃO
            print("[ERROR] Unsupported function on CSV file.")
            input("[PRESS ENTER TO CONTINUE]")
            clear()
            return -2

        # PROCESSAR AS RESTANTES LINHAS DO FICHEIRO
        full_dic = geraDicionario(columnOperations, lines)

        # PREPAR O OUTPUT PARA JSON
        outData = prepareJSON(full_dic)

        # GUARDAR O OUTPUT
        outputFile = open("../output/"+output_name, 'w')
        outputFile.write(outData)
        outputFile.close
        print("[FILE] Converted successfully.")
        input("[PRESS ENTER TO CONTINUE]")
        clear()
        return 0
```

## A.2 Visualização de ficheiros

```
def viewFile():
    clear()
    run = True
    while(run):
        print("\n\n\t* Select file extension *\n")
        print_menu(menu_file)
        opt = input("\n> Option: ")
        if (str(opt) == '1' or str(opt) == '2'):
            lines = openFile(int(opt))
            if lines == - 1: # CHECKING FOR FILE NOT FOUND
                pass
            elif lines:
                clear()
                for l in lines:
                    print(l)
                input("[PRESS ENTER TO CONTINUE]")
                clear()
        elif str(opt) == '0':
            run = False
        else:
            pass
```

## A.3 Menu princial de execução

```
def runMenu():
    run = True
    while(run):
        clear()
        print("\n\n\t**** CSV TO JSON CONVERTER ****\n")
        print_menu(menu_initial)
        option = input('\n> Option: ')
        print()
        if str(option) == '1':
            val = convertFile()
            if val == - 2: # CHECKING FOR ERRORS ON FUNCTIONS
                pass
        elif str(option) == '2':
            viewFile()
        elif str(option) == '0':
            run = False
            clear()
            print("\n\n\t**** LEAVING ****\n")
        else:
            pass
```

## A.4 Processamento inicial do *input*

```
def cleanInput(lines):
    lines = list(map(lambda st: re.sub(r';|\||\t',r',',st), lines))
    # REMOVER LINHAS VAZIAS DO INPUT
    for line in lines:
        if line == '':
            lines.remove(line)
    return lines
```

## A.5 Processamento do cabeçalho do *input*

```
def header(line):

    columnOperations = []
    functions = ["sum", "media", "min", "max", "count"]

    elements = re.findall(r'([^;:,{}+](?:{(.*)})?(?:\:\:\:(.*)?(?:;|,))?', line)

    for i in elements:
        if len(list(filter(None,i))) == 1:
            t = (i[0],0,"none")
            columnOperations.append(t)
        elif len(list(filter(None,i))) == 2:
            t = (i[0],i[1],"none")
            columnOperations.append(t)
        else:
            # a função de agregação passada não é reconhecida
            if i[2] not in functions:
                raise NameError
            else:
                t = (i[0],i[1],i[2])
                columnOperations.append(t)

    return columnOperations
```

## A.6 Processamento do corpo do *input*

```
def processLine(columnOperations: List[str], line: str):

    result = []
    pos = 0

    components = line.split(",")

    # i : (Column Name, Length if List, Fuction Name)
    for op in columnOperations:

        length = int(calculateLength(str(op[1])))
        list = []

        if length > 0:
            for i in components[pos:(pos+length)]:
                if re.match(r'^-?\d+(?:\.\d+)?$', i):
                    list.append(i)

            values = [int(value) for value in list]
            res = executeFunction(op[0],op[2],values)
            result.append(res)
            pos = pos + length
        else:
            result.append(f'"{op[0]}": {components[pos]}')
            pos = pos + 1

    return result
```

## A.7 Cálculo das funções de agregação

```
def executeFunction(columnName:str, function: str, values: List[int]):
    if len(values) != 0:
        if function == "sum":
            res = f'"{columnName}_sum": {sum(values)}'
        elif function == "media":
            result = round(sum(values)/len(values),2)
            res = f'"{columnName}_media": {result}'
        elif function == "min":
            res = f'"{columnName}_min": {min(values)}'
        elif function == "max":
            res = f'"{columnName}_max": {max(values)}'
        elif function == "count":
            res = f'"{columnName}_count": {len(values)}'
        elif function == "none":
            res = f'"{columnName}": {values}'
    else:
        res = f'"{columnName}": '
    return res
```

## A.8 Criação de dicionários

```
def geraDicionario(columnOperations, lines):

    rule = re.compile(r'(?!"([a-zà-ü]+:))(.*)')
    full_dic = [] # Lista para guardar todos os dicionários gerados

    # CRIAÇÃO ITERATIVA DE DICIONÁRIOS
    for line in lines:
        dicionario = {}
        res = processLine(columnOperations,line)
        for i in range(len(columnOperations)):
            m = re.search(rule,res[i])
            if m:
                if m.group()[1:] != "":
                    if columnOperations[i][2] != 'none':
                        dicionario[columnOperations[i][0]+"_"+columnOperations[i][2]]
                            = m.group()[1:]
                    else:
                        dicionario[columnOperations[i][0]] = m.group()[1:]
        full_dic.append(dicionario.copy())
    return full_dic
```

## A.9 Preparação do ficheiro *JSON*

```
def prepareJSON(dicionario):
    res = "[\n" # Incio do ficheiro JSON
    for dic_entry in dicionario:
        res += "\t{\n" # Incio de um dicionário
        items = dic_entry.items()
        size = 0;
        for key in dic_entry:
            res += "\t\t\""+ key + "\": "
            if "," in dic_entry[key]:
                res += dic_entry[key].replace(" ",",")
```

```

        size += 1
    elif '_' in key:
        res += dic_entry[key]
        size += 1
    else:
        res += "\"" + dic_entry[key] + "\""
        size += 1
    if size == len(dic_entry):
        res += "\n"
    else:
        res += ",\n"
    res += "\t},\n" # Fim de um dicionário
res = res[:-2]
res += "\n]" # Final do ficheiro JSON
return res

```