



Universidade do Minho
Escola de Engenharia

Processamento de Linguagens (3^o ano de Curso)
Trabalho Prático 1
Relatório de Desenvolvimento

David Duarte
(a93253)

Ema Dias
(a89518)

Leonardo Freitas
(a93281)

27 de março de 2022

Resumo

O presente relatório foi realizado no âmbito da Unidade Curricular, Processamento de Linguagens, e tem como objetivo identificar o problema e objetivos fornecidos pela equipa docente e respetivas decisões tomadas para atingir os mesmos, por parte da equipa de trabalho.

O mesmo refere-se ao primeiro trabalho prático, e corresponde ao enunciado: Ficheiros CSV com listas e funções de agregação, enunciado este escolhido pela equipa.

Pretende-se, com este projeto, aumentar a experiência de uso de algumas ferramentas de apoio à programação, aumentar a capacidade de escrita de Expressões Regulares para descrever padrões de texto, desenvolver Processadores de Linguagens Regulares, com o objetivo de transformar textos com base no conceito de regras de produção Condição-Ação. Além disso, dever-se-á utilizar Python e os módulos re e ply para gerar os filtros de texto.

Conteúdo

1	Introdução	2
1.1	Enquadramento e Contexto	2
1.2	Problema e Objetivos	2
1.3	Informações Adicionais	3
2	Análise e Especificação	6
2.1	Especificação de Requisitos	6
2.2	Decisões tomadas	6
3	Concepção/desenho da Resolução	8
3.1	Leitura do ficheiro CSV	8
3.2	Escrita do ficheiro json	9
4	Conclusão	10
4.1	Apresentação dos resultados obtidos	10
<i>Área: Processamento de Linguagens</i>		

Capítulo 1

Introdução

1.1 Enquadramento e Contexto

Um ficheiro CVS (Comma separated values), tal como o nome indica, é um ficheiro cujos valores estão separados por vírgulas. Além disso, cada linha desse mesmo ficheiro corresponde a uma entrada/registo, seja o mesmo de um artigo, um cliente ou qualquer outro tipo de entidade. Os ficheiros são produzidos em folhas de cálculo, nomeadamente em Excel. Em suma, o ficheiro CVS é usado como formato para armazenamento de dados e, tem como vantagem a simplicidade de importação e exportação de arquivos numa linguagem de fácil leitura.

O formato JSON, JavaScript Object Notation, é usado para estruturar dados em forma de texto, é uma alternativa ao modelo xml. O arquivo contém uma série de dados segundo um padrão específico. Os mesmos devem estar estruturados segundo uma coleção de pares (chave,valor), onde chave identifica o conteúdo, devendo ser constituído por uma string, e valor representa o conteúdo, podendo ser do tipo: string, array, object, number, boolean ou null, ou, alternativamente, uma lista ordenada de valores. O formato JSON pode ser utilizado em qualquer linguagem de programação e em qualquer plataforma.

A formatação do ficheiro deverá contemplar:

- { } - para delimitação do conteúdo
- [] - para identificar um array
- : - para separação chave/valor
- , - para separação dos elementos.

Em suma, o formato .json tem sido cada vez mais utilizado dada a sua simplicidade e facilidade de uso.

1.2 Problema e Objetivos

No enunciado selecionado pela equipa, tem-se como objetivo realizar um conversor de um ficheiro CSV para formato JSON, sabendo que na primeira linha do CVS encontrar-se-á um cabeçalho que indicará o que representa cada coluna.

A título de exemplo:

Um ficheiro “alunos.csv”:

Número, Nome, Curso
3162, Cândido Faísca, Teatro
7777, Cristiano Ronaldo, Desporto

Corresponde à tabela:

Número	Nome	Curso
3162	Cândido Faísca	Teatro
7777	Cristiano Ronaldo	Desporto

E deverá corresponder ao ficheiro .json:

```
[
  {
    "Número": "3612",
    "Nome": "Cândido Faísca",
    "Curso": "Teatro"
  },
  {
    "Número": "7777"
    "Nome": "Cristiano Ronaldo",
    "Curso": "Desporto"
  }
]
```

Tem-se como tarefas a especificação de padrões de procura, através de Expressões Regulares, a identificação das ações a realizar aquando da captura do padrão, a identificação das Estruturas de Dados necessárias ao armazenamento temporário da informação extraída do texto-fonte ou da informação construída no processamento e o desenvolvimento de um filtro de texto para reconhecimento de padrões identificados e proceder à conversão desejada, utilizando como recursos os módulos `re` e `ply`.

O módulo `re` é um módulo que fornece diversas funções que permitem capturar um texto, tais como `findall`, `search`, ou até mesmo `split`, `sub` para manipular strings. Desta forma, é possível trabalhar com as expressões regulares.

O módulo `ply` é um pacote que agrega em forma de módulo a ferramenta `lex`. Esta ferramenta gera um analisador léxico, utilizado para criar tokens no fluxo de entrada.

1.3 Informações Adicionais

Além do exposto anteriormente, no enunciado do projeto ainda é referido que poder-se-ão ser recebidas extensões adicionais, como:

- Listas: com tamanho definido ou com um intervalo de tamanhos

A título de exemplo,

Um ficheiro “alunos2.csv” :

Número	Nome	Curso	Notas{5}
3162	Cândido Faísca	Teatro	12, 13, 14, 15, 16
7777	Cristiano Ronaldo	Desporto	17, 12, 20, 11, 12

onde caso sobrem campos se coloca os mesmos a vazio

Corresponde ao ficheiro .json:

```
[
  {
    "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]
  }
]
```

Ou um ficheiro “alunos3.csv”:

Número	Nome	Curso	Notas{3,5}
3162	Cândido Faísca	Teatro	12, 13, 14
7777	Cristiano Ronaldo	Desporto	17, 12, 20, 11, 12

Corresponde ao ficheiro .json:

```
[
  {
    "Número": "3612",
    "Nome": "Cândido Faísca",
    "Curso": "Teatro"
    "Notas": [12,13,14]
  },
  {
    "Número": "7777"
    "Nome": "Cristiano Ronaldo",
    "Curso": "Desporto"
    "Notas": [17,12,20,11,12]
  }
]
```

- Funções de agregação, tais como sum ou média

A título de exemplo,

Número, Nome, Curso, Notas3,5, , , , ::sum 3162, Cândido Faísca, Teatro, 12, 13, 14 7777, Cristiano Ronaldo, Desporto, 17, 12, 20, 11, 12

Um ficheiro “alunos4.csv”:

Corresponde ao ficheiro .json:

```
[
  {
    "Número": "3612",
    "Nome": "Cândido Faísca",
    "Curso": "Teatro"
    "Notas_sum": 39
  },
  {
    "Número": "7777"
    "Nome": "Cristiano Ronaldo",
    "Curso": "Desporto"
    "Notas_sum": 72
  }
]
```

Capítulo 2

Análise e Especificação

2.1 Especificação de Requisitos

Tal como referido anteriormente, com este projeto, pretende-se criar um mecanismo de leitura e captura de dados, de forma a organizá-los e escrevê-los num ficheiro json.

No enunciado do projeto, foram facultados alguns exemplos dos objetivos, também apresentados neste relatório, no entanto, a equipa docente indicou que o programa desenvolvido deveria ser capaz de suportar qualquer tipo de ficheiro CSV e, não apenas, aqueles dados como modelo.

Desta forma, a equipa estabeleceu como requisitos para alcançar os objetivos:

- A possibilidade de se escolher qual é o ficheiro CSV que se pretende utilizar para leitura, indicando-o através do terminal.
- A seleção de um ficheiro json que tenha como nome o mesmo do csv.
- A capacidade de leitura dos campos passados no ficheiro csv delimitados por vírgulas.
- A capacidade de suportar valores vazios inseridos no ficheiro csv.
- A capacidade de agregação de uma lista de parâmetros dados como dígitos.
- A capacidade de processar uma função de agregação associada a essa mesma lista.
- A capacidade de poder incluir essa mesma lista em qualquer lugar do cabeçalho, seja na posição final ou não.

Em suma, pretende-se que o programa desenvolvido seja o mais flexível possível.

2.2 Decisões tomadas

Para atingir o sucesso dos requisitos, no ponto anterior apresentados, a equipa decidiu tomar algumas ações como padrão de forma a regularizar a leitura do ficheiro. Ou seja, uma vez que a primeira linha é aquela que indica quais as colunas e cujos parâmetros que irão ser apresentados nas linhas seguintes, a análise dessa linha é feita através de um split, cujo delimitador é a vírgula. No entanto, como se sabe que

podem existir listas no ficheiro, e como estas mesmas listas poderão ter um tamanho variável apresentado da seguinte forma: { *tamanho mínimo*, *tamanho máximo* }, encontra-se um obstáculo à delimitação por vírgulas. Foi então necessário arranjar uma alternativa para que se pudesse guardar tanto o tamanho da lista, assim como qual o nome da coluna que corresponde à mesma.

A equipa, após várias tentativas, percebeu que a forma mais eficiente de o fazer era, sabendo que uma lista é apresentada com recurso às chavetas, aquilo que permitiria identificar essa mesma lista seria substituir com recurso à função *sub* aquilo que se encontra nas chavetas, não sem antes guardar esses mesmos valores em variáveis, pela palavra lista. Assim, é possível utilizar uma expressão regular que procure a palavra lista nesta linha já regularizada, de forma a sabermos qual é o nome deste campo.

Além disto, foi necessário realizar uma pesquisa sobre as funções de agregação, percebendo-se que as mesmas são funções aritméticas que permitem reduzir uma lista de valores a um único valor. Como tal, considerou-se uma lista de funções de agregação válidas, sendo as mesmas *max*, que deverá retornar o valor maior da lista, *min*, que deverá retornar o valor menor da lista, *avg*, que deverá realizar a média dos valores, *count*, retornando o número de elementos da mesma e, por fim, *sum* que corresponde ao somatório de todos os elementos.

Esta mesma função é, de facto, identificada pelo nome que está seguido à lista por ::, o que faz com que também seja possível utilizar uma expressão regular **re.search('(::)(\w+)(,)?',firstline2)**, para procurar o nome da função e, e esse nome será incluído no nome da lista através da substituição dos :: por _ na linha que é iterada no corpo do ciclo.

Por fim, tendo em consideração que o problema apresentado se poderia resolver de forma coerente utilizando apenas o módulo re, a equipa decidiu avançar com este módulo, ao invés de utilizar o módulo ply.

Capítulo 3

Concepção/desenho da Resolução

A realização deste problema, não ultrapassa mais do que a separação em duas etapas: a leitura de um ficheiro e a escrita noutra.

3.1 Leitura do ficheiro CSV

Para a leitura do ficheiro estamos perante uma primeira linha que retrata os campos apresentados, onde esta primeira linha já está devidamente regularizada e separada com as decisões apresentadas no capítulo anterior e o restante ficheiro.

Para a isso, utilizou-se a função *readlines()* do ficheiro cujo nome foi inserido no terminal e ao qual se aplicou a função *next* para não se ler a linha de cabeçalho.

De seguida, realizaram-se dois ciclos, um referente a cada linha, na forma de *for* e outro na forma de *while*. O primeiro ciclo tem como função iterar as linhas de dados, já o segundo pretende iterar a lista de chaves retiradas do split do cabeçalho. Para trabalhar de forma apropriada, utilizou-se o conceito de dicionário ensinado seja nas aulas teóricas como nas aulas práticas da Unidade Curricular.

Dessa forma, criou-se um dicionário cujas keys representam uma entrada do ficheiro em termos de iterador (de 0 até ao número de linhas de conteúdo do ficheiro) e, cujos values são dicionários de cada uma dessas entradas, onde a key representa o nome do cabeçalho e o value o seu respetivo valor. Ou seja, para cada entrada é criado um dicionário onde se associa a cada elemento do segundo ciclo (cada elemento do cabeçalho) o respetivo valor lido, e, no final do segundo ciclo, esse dicionário é inserido no dicionário cuja key representa a linha lida - 1.

No interior do segundo ciclo, foi necessário estabelecer duas expressões regulares, sendo que a primeira, tal como foi dito anteriormente, é responsável por procurar a existência de uma lista. Posto isto, inseriu-se uma condição que verifica essa mesma existência e, que no seu interior realiza um ciclo que adiciona os dígitos encontrados daí adiante na linha até ao tamanho máximo dado da lista. É importante referir que também cada linha é separada por vírgulas e como tal, para cada uma é usada uma variável que itera desde 0. Existem duas condições inseridas para garantir que a leitura dos elementos da lista é realizada de forma correta, a primeira permite que seja inserido qualquer outro dígito que não pertença à lista a seguir à mesma. A segunda prevê o final da lista correspondente à linha e, nessa situação, não se pode efetuar mais comparações com dígitos e, por isso, é realizado um *break*.

Com a inserção de uma lista de tamanho variável, não foi possível desenvolver nenhum raciocínio que prevê a não inserção de um elemento a mais no caso de se ler um dígito não pertencente à lista. Ou seja, é possível, a título de exemplo, inserir uma idade caso o tamanho da lista seja fixo, no entanto, caso esse tamanho seja variável não se prevê a capacidade de não inserção desse mesmo dígito na lista, já que o programa apenas inclui a condição de verificar que já se alcançou o tamanho máximo, não havendo capacidade de saber o tamanho que a mesma pode tomar quando o mesmo é variável.

A segunda expressão regular que foi necessário incluir foi a expressão que captura valores vazios, pois sabemos que podem ser inseridos valores nulos quando se trabalha com listas de tamanho variável. Para este caso, deve-se avançar na leitura da linha.

Por fim, achou-se pertinente manter os valores da lista, tanto sejam inferiores ao número mínimo do tamanho da mesma, como caso sejam superiores, já que não possuímos nenhum conhecimento de qual o valor errado, ou, por outro lado, consideramos mais benéfico apresentar os valores inseridos, mesmo que estes não completem o tamanho pretendido. No entanto, deixamos claro que quando isto acontece, deve ser emitida uma mensagem de erro que informa que aquela lista não está correta, da forma de uma nova inserção no dicionário com a key erro.

Para terminar, verifica-se a existência de uma função associada à lista, e para caso exista é realizada essa mesma função com recurso às funções fornecidas pelo python e o resultado é substituído pela apresentação de todos os valores dos elementos. Deixou-se em aberto a possibilidade de ter sido submetido uma função que não pertence à lista das válidas e, para esse caso, apenas se mantém o resultado como lista dos elementos do ficheiro.

3.2 Escrita do ficheiro json

A escrita no ficheiro também envolve a realização de dois ciclos, o primeiro permite iterar todas as keys do dicionário e para cada uma delas itera-se, com recurso a outro ciclo todas as keys que se encontram no value do primeiro dicionário e são estas que são inseridas no ficheiro, acompanhadas pelos seus values.

É de realçar que se realizaram condições para evitar a escrita de vírgulas tanto para a última key de cada entrada como para a última entrada em si no ficheiro. Além disso, foram adicionados os parênteses retos e as chavetas com as respetivas identações apresentadas nos modelos apresentados pela equipa docente.

Por fim, resta indicar que para realizar a escrita no ficheiro, primeiramente foi necessário abrir o mesmo no modo de escrita, e além disso, como a equipa decidiu que esse ficheiro teria como nome o mesmo do csv, é necessário verificar se o ficheiro json com esse nome possui conteúdo, e tomamos a liberdade de o remover para não acontecer sobreposições de informação, já que à partida o utilizador terá o conhecimento que o nome do ficheiro csv será aquele que será fornecido ao ficheiro json.

Capítulo 4

Conclusão

4.1 Apresentação dos resultados obtidos

Para terminar este relatório, iremos apresentar os resultados obtidos de alguns exemplos da utilização do programa desenvolvido.


```
input >  alunos.csv
1 id_aluno,nome,curso,idade,Notas{2,5}
2 "a1","Aysha Melanie Gilberto","LEI",21,12,,,,
3 "a2","Igor André Cantanhede","ENGFIS",22,12,16,,,
4 "a3","Laurénio Narciso","ENGFIS",24,8,14,15,14,
5 "a4","Jasnoor Casegas","LCC",20,14,20,17,11, |
```

Figura 4.1: Ficheiro Alunos Original

```

input > {} alunos.json > ...
1  [
2    {
3      "id_aluno": "a1",
4      "nome": "Aysha Melanie Gilberto",
5      "curso": "LEI",
6      "idade": 21,
7      "Erro": "Erro no tamanho da lista",
8      "Notas": [12]
9    },
10   {
11     "id_aluno": "a2",
12     "nome": "Igor Andr  Cantanhede",
13     "curso": "ENGFIS",
14     "idade": 22,
15     "Notas": [12, 16]
16   },
17   {
18     "id_aluno": "a3",
19     "nome": "Laur nio Narciso",
20     "curso": "ENGFIS",
21     "idade": 24,
22     "Notas": [8, 14, 15, 14]
23   },
24   {
25     "id_aluno": "a4",
26     "nome": "Jasnoor Casegas",
27     "curso": "LCC",
28     "idade": 20,
29     "Notas": [14, 20, 17, 11]
30   }
31 ]

```

Figura 4.2: Ficheiro Alunos Convertido para json

```

input > count.csv
1  id_aluno,nome,curso,idade,Notas{2,5}::count
2  "a1","Aysha Melanie Gilberto","LEI",21,12,13,,,
3  "a2","Igor Andr  Cantanhede","ENGFIS",22,12,16,,,
4  "a3","Laur nio Narciso","ENGFIS",24,8,14,15,14,
5  "a4","Jasnoor Casegas","LCC",20,14,20,17,11,

```

Figura 4.3: Ficheiro Alunos com a fun  o de Agrega  o Count

```

input > {} count,json > ...
1  [
2  {
3      "id_aluno":"a1",
4      "nome":"Aysha Melanie Gilberto",
5      "curso":"LEI",
6      "idade":21,
7      "Notas_count":2
8  },
9  {
10     "id_aluno":"a2",
11     "nome":"Igor Andr  Cantanhede",
12     "curso":"ENGFIS",
13     "idade":22,
14     "Notas_count":2
15  },
16  {
17     "id_aluno":"a3",
18     "nome":"Laur nio Narciso",
19     "curso":"ENGFIS",
20     "idade":24,
21     "Notas_count":4
22  },
23  {
24     "id_aluno":"a4",
25     "nome":"Jasnoor Casegas",
26     "curso":"LCC",
27     "idade":20,
28     "Notas_count":4
29  }
30 ]

```

Figura 4.4: Ficheiro Alunos com a fun  o de Agregac  o Count Convertida

```

1  id_aluno,nome,curso,idade,Notas{2,5}::avg|
2  "a1","Aysha Melanie Gilberto","LEI",21,12,13,,,
3  "a2","Igor Andr  Cantanhede","ENGFIS",22,12,16,,,
4  "a3","Laur nio Narciso","ENGFIS",24,8,14,15,14,
5  "a4","Jasnoor Casegas","LCC",20,14,20,17,11,

```

Figura 4.5: Ficheiro Alunos com a fun  o de Agregac  o Media

```
[
  {
    "id_aluno": "a1",
    "nome": "Aysha Melanie Gilberto",
    "curso": "LEI",
    "idade": 21,
    "Notas_avg": 12.5
  },
  {
    "id_aluno": "a2",
    "nome": "Igor André Cantanhede",
    "curso": "ENGFIS",
    "idade": 22,
    "Notas_avg": 14.0
  },
  {
    "id_aluno": "a3",
    "nome": "Laurénio Narciso",
    "curso": "ENGFIS",
    "idade": 24,
    "Notas_avg": 12.75
  },
  {
    "id_aluno": "a4",
    "nome": "Jasnoor Casegas",
    "curso": "LCC",
    "idade": 20,
    "Notas_avg": 15.5
  }
]
```

Figura 4.6: Ficheiro Alunos com a função de Agregação Media Convertida

```
input > max.csv
1 id_aluno,nome,curso,idade,Notas{2,5}::max
2 "a1","Aysha Melanie Gilberto","LEI",21,12,13,,
3 "a2","Igor André Cantanhede","ENGFIS",22,12,16,,
4 "a3","Laurénio Narciso","ENGFIS",24,8,14,15,14,
5 "a4","Jasnoor Casegas","LCC",20,14,20,17,11,
```

Figura 4.7: Ficheiro Alunos com a função de Agregação Max

```
input > {} max.json > ...
1  [
2  {
3      "id_aluno": "a1",
4      "nome": "Aysha Melanie Gilberto",
5      "curso": "LEI",
6      "idade": 21,
7      "Notas_max": 13
8  },
9  {
10     "id_aluno": "a2",
11     "nome": "Igor Andr  Cantanhede",
12     "curso": "ENGFIS",
13     "idade": 22,
14     "Notas_max": 16
15 },
16 {
17     "id_aluno": "a3",
18     "nome": "Laur nio Narciso",
19     "curso": "ENGFIS",
20     "idade": 24,
21     "Notas_max": 15
22 },
23 {
24     "id_aluno": "a4",
25     "nome": "Jasnoor Casegas",
26     "curso": "LCC",
27     "idade": 20,
28     "Notas_max": 20
29 }
30 ]
```

Figura 4.8: Ficheiro Alunos com a fun  o de Agrega  o Max Convertida


```

input > min.csv
1 id_aluno, nome, curso, idade, Notas{2,5}::min
2 "a1", "Aysha Melanie Gilberto", "LEI", 21, 12, 13, , ,
3 "a2", "Igor André Cantanhede", "ENGFIS", 22, 12, 16, , ,
4 "a3", "Laurénio Narciso", "ENGFIS", 24, 8, 14, 15, 14,
5 "a4", "Jasnoor Casegas", "LCC", 20, 14, 20, 17, 11,

```

Figura 4.9: Ficheiro Alunos com a função de Agregação Min

```

input > {} min.json > ...
1 [
2   {
3     "id_aluno": "a1",
4     "nome": "Aysha Melanie Gilberto",
5     "curso": "LEI",
6     "idade": 21,
7     "Notas_min": 12
8   },
9   {
10    "id_aluno": "a2",
11    "nome": "Igor André Cantanhede",
12    "curso": "ENGFIS",
13    "idade": 22,
14    "Notas_min": 12
15  },
16  {
17    "id_aluno": "a3",
18    "nome": "Laurénio Narciso",
19    "curso": "ENGFIS",
20    "idade": 24,
21    "Notas_min": 8
22  },
23  {
24    "id_aluno": "a4",
25    "nome": "Jasnoor Casegas",
26    "curso": "LCC",
27    "idade": 20,
28    "Notas_min": 11
29  }
30 ]

```

Figura 4.10: Ficheiro Alunos com a função de Agregação Min Convertida

```

input > sum.csv
1 id_aluno,nome,curso,idade,Notas{2,5}::sum
2 "a1","Aysha Melanie Gilberto","LEI",21,12,13,,,
3 "a2","Igor André Cantanhede","ENGFIS",22,12,16,,,
4 "a3","Laurénio Narciso","ENGFIS",24,8,14,15,14,
5 "a4","Jasnoor Casegas","LCC",20,14,20,17,11,

```

Figura 4.11: Ficheiro Alunos com a função de Agregação Sum

```

input > {} sum.json > ...
1 [
2   {
3     "id_aluno":"a1",
4     "nome":"Aysha Melanie Gilberto",
5     "curso":"LEI",
6     "idade":21,
7     "Notas_sum":25
8   },
9   {
10    "id_aluno":"a2",
11    "nome":"Igor André Cantanhede",
12    "curso":"ENGFIS",
13    "idade":22,
14    "Notas_sum":28
15  },
16  {
17    "id_aluno":"a3",
18    "nome":"Laurénio Narciso",
19    "curso":"ENGFIS",
20    "idade":24,
21    "Notas_sum":51
22  },
23  {
24    "id_aluno":"a4",
25    "nome":"Jasnoor Casegas",
26    "curso":"LCC",
27    "idade":20,
28    "Notas_sum":62
29  }
30 ]

```

Figura 4.12: Ficheiro Alunos com a função de Agregação Sum Convertida