



Universidade do Minho

UNIVERSIDADE DO MINHO

LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

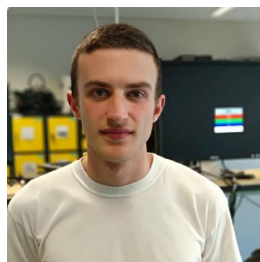
PLC - Trabalho Prático 1
Grupo nº14

Simão Pedro Batista Caridade Quintela
(A97444)

David José de Sousa Machado
(A91665)

Hugo Filipe de Sá Rocha
(A96463)

13 de novembro de 2022



Conteúdo

1	Introdução	3
2	Enunciados e Resoluções	4
2.1	Processador de Pessoas listadas nos Róis de Confessados . . .	4
2.1.1	Resolução do problema	4
2.2	Ficheiros CSV com listas e funções de agregação	9
2.2.1	Resolução do problema	12
3	Exemplos de funcionamento	16
3.1	Processador de Pessoas listadas nos Róis de Confessados . . .	16
3.1.1	Frequência de processos por ano	16
3.1.2	Frequência de nomes por século	17
3.1.3	Frequência de relações por século	17
3.1.4	Escrever as 20 primeiras linhas num JSON	18
3.2	Ficheiros CSV com listas e funções de agregação	18
3.2.1	Informação base sem listas	18
3.2.2	Informação com listas	19
3.2.3	Listas com um intervalo de tamanhos	20
3.2.4	Funções de agregação	21
4	Conclusão	23

Capítulo 1

Introdução

No âmbito da disciplina de Processamento de Linguagens e Compiladores foi-nos proposto pelo docente Pedro Rangel Henriques a realização de um trabalho prático visando colocar em prática a utilização de expressões regulares para a análise de ficheiros de texto.

O trabalho prático consiste na resolução de, pelo menos, um problema em cinco propostos. Analisados os problemas acabamos por resolver o problema 1 (Processador de Pessoas listadas nos Róis de Confessados) e o problema 5 (Ficheiros CSV com listas e funções de agregação).

Neste documento estão apresentadas as soluções utilizadas para a resolução dos problemas abordados, bem como a correspondente demonstração do funcionamento dos programas.

Capítulo 2

Enunciados e Resoluções

2.1 Processador de Pessoas listadas nos Róis de Confessados

Construa agora um ou vários programas Python para processar o texto 'processos.txt' com o intuito de calcular frequências de alguns elementos (a ideia é utilizar arrays associativos para o efeito) conforme solicitado a seguir.

- a) Calcula a frequência de processos por ano (primeiro elemento da data);
- b) Calcula a frequência de nomes próprios (o primeiro em cada nome) e apelidos (o último em cada nome) por séculos;
- c) Calcula a frequência dos vários tipos de relação: irmão, sobrinho, etc.
- d) Imprimir os 20 primeiros registos num novo ficheiro de output, mas em formato JSON.

2.1.1 Resolução do problema

Neste trabalho decidimos utilizar uma estratégia inicial semelhante em todas as alíneas que consiste em ler todas as linhas do ficheiro *processos.txt* e guardá-las num array para depois poderem ser lidas na resolução das diferentes alíneas.

- a. Na resolução da alínea a. utilizamos um dicionário *years* para guardar a frequência de processos por ano associando cada ano à frequência de processos. Para obtermos o ano e processo correspondente a cada linha do ficheiro utilizamos as funções **search** e **split**, e a seguinte expressão regular `([0-9]+)[:2]([0-9]4[0-9]2[0-9]2)`

```

1 def processFrequency():
2     years = {}
3
4     for line in lines:
5         process_and_date = re.search('([0-9]+)
6         [:]{2}([0-9]{4}\-[0-9]{2}\-[0-9]{2})', line)
7
8         if process_and_date != None:
9             process = process_and_date.group(1)
10            date = process_and_date.group(2)
11            dataSplitted = re.split(r'-', date)
12            year = dataSplitted[0]
13
14
15            if year not in years:
16                years[year] = 1
17            else:
18                years[year] += 1
19
20    return years
21

```

- b. Na resolução da alínea b. utilizamos o dicionário *centurys* que tem a seguinte estrutura:

```

1 # this is just an example:
2 centurys = {
3     19: {
4         "First": {"Candido": 10},
5         "Last": {"Faisca": 5}
6     },
7     20: {
8         "First": {"Ivone": 130},
9         "Last": {"Costa": 2000}
10    }
11 }
12 }
13

```

Utilizamos a expressão regular `([0-9]4)([0-9]2)([0-9]2)` com a função **search** para determinar a data da linha lida.

Com a expressão regular `:[A-Za-z]+(:)` e com a função **findall** conseguimos identificar o nome da pessoa processada, do pai e da mãe.

Por fim, utilizando de novo a função **findall** e a expressão regular `([A-Z] [A-Za-z]+), ([A-Za-z]+). ?(?i:(Proc.[0-9]+))` conseguimos identificar os nomes de pessoas que tiveram envolvidas noutros processos.

```

1 def year_to_century(year):

```

```

2     return -(-year // 100)
3
4 def nameFrequency():
5     centurys = {}
6     for line in lines:
7         date = re.search(r'([0-9]{4})\-([0-9]{2})
8 \-([0-9]{2})', line)
9         names_in_dots = re.findall('(:([A-Za-z ]+)(:)',
10 line)
11         names_with_procs = re.findall('([A-Z][A-Za-z ]+
12 ,([A-Za-z ]+). ?(?:i:(Proc.[0-9]+))', line)
13
14         names = names_in_dots + names_with_procs
15
16         if date:
17             year = int(date.group(1))
18             century = year_to_century(year)
19             if century not in centurys:
20                 centurys[century] = {}
21                 centurys[century]["First"] = {}
22                 centurys[century]["Last"] = {}
23
24             for name in names:
25                 person_name = name[0]
26                 name_splitted = re.split(" ", person_name)
27                 first_name = name_splitted[0]
28                 last_name = name_splitted[-1]
29                 if first_name not in centurys[century]["First"]
30 ]:
31                     centurys[century]["First"][first_name] = 1
32                 else:
33                     centurys[century]["First"][first_name] +=
34 1
35
36                 if last_name not in centurys[century]["Last"]:
37                     centurys[century]["Last"][last_name] = 1
38                 else:
39                     centurys[century]["Last"][last_name] += 1
40     return centurys

```

- c. Na resolução da alínea c utilizamos um dicionário *rel_freq* para guardar a frequência de relações existente no ficheiro de texto.

Ao analisar o ficheiro reparamos que estes dois padrões que se repetiam:

(a) ::Filho::Pai(opcional)::Mãe(Opcional)::

(b) nome, relação de parentesco, Proc.x

Para identificar o padrão (a) usamos a função **findall** e a expressão regular `:[A-Za-z|]+):` . De notar que, no dicionário, o **Pai** e a **Mãe** são ambos contabilizados na entrada **Progenitor** visto que em várias linhas, por vezes a ordem pela qual aparece o nome dos mesmos é trocada. Para o efeito, e para não arriscar recolher informação errada, optamos por colocá-los na mesma entrada.

Para identificar o padrão (b), utilizamos a função **findall** e a expressão regular `([A-Z][A-Za-z]+),([A-Za-z]+). ?(?i:(Proc.[0-9]+))` para identificar as restantes relações de parentesco com o processado.

```

1 def relationFrequency():
2     rel_freq = {}
3     rel_freq["Progenitores"] = 0
4     rel_freq["Filho"] = 0
5
6     for line in lines:
7         parents_and_son = re.findall(":[A-Za-z| ]+):",
8             line)
9
10        if parents_and_son:
11            parents = parents_and_son[1:]
12            rel_freq["Filho"] += 1
13            rel_freq["Progenitores"] += len(parents)
14
15        relations = re.findall("([A-Z][A-Za-z ]+),([A-Za-z
16        ]+). ?(?i:(Proc.[0-9]+))", line)
17        if relations:
18            for relation in relations:
19                if relation[1] not in rel_freq:
20                    rel_freq[relation[1]] = 1
21                else:
22                    rel_freq[relation[1]] += 1
23
24    return rel_freq

```

- d. Para fechar o exercício 1 falta imprimir as primeiras 20 linhas do ficheiro *processos.txt* em formato JSON.

Para a resolução desta alínea utilizamos duas funções, uma para recolher informação, e outra para escrever informação no ficheiro JSON pretendido.

A função **info_to_json** tem como objetivo recolher informação linha a linha (assegurando-se que não lê duas vezes a mesma linha), utilizando expreesões regulares mostradas anteriormente, na seguinte forma:

```

1 # dada a seguinte linha tem-se
2 # 569::1867-05-23::Abel Alves Barroso::Antonio Alves
   Barroso::Maria Jose Alvares Barroso::Bento Alvares

```

```

3      Barroso,Tio Paterno. Proc.32057.    Domingos Jose
4      Alvares Barroso,Tio Materno. Proc.32235.:
5
6      json_info = {
7          575::1894-11-08 :{
8              "Processo": "575",
9              "Data": "1894-11-08",
10             "Pessoa processada": "Abel Alves Barroso",
11             "Pai": "Antonio Alves Barroso",
12             "Mae": "Maria Jose Alvares Barroso",
13             "Tio Paterno": "Bento Alvares Barroso",
14             "Tio Materno": "Domingos Jose Alvares Barroso"
15         }
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

1 def info_to_json():
2     json_info = {}
3
4     valid_lines = 0
5     i = 0
6     while valid_lines < 20:
7         line = lines[i]
8
9         if line != '':
10             process_and_date = re.search('([0-9]+)
11             [:]{2}([0-9]{4}\-[0-9]{2}\-[0-9]{2})', line)
12             both = process_and_date.group(0)
13
14             if both not in json_info:
15                 process = process_and_date.group(1)
16                 date = process_and_date.group(2)
17
18                 json_info[both] = {"Processo": process, "
19 Data": date}
20
21                 son_and_parents = re.findall('(:([A-Za-z|
22 ]+)(:)', line)
23
24                 json_info[both]["Pessoa processada"] =
25 son_and_parents[0][0]
26
27                 if len(son_and_parents) == 2:
28                     json_info[both]["Mae"] =
29 son_and_parents[1][0]
30                 else:
31                     json_info[both]["Pai"] =
32 son_and_parents[1][0]
33                     json_info[both]["Mae"] =
34 son_and_parents[2][0]
35
36                 relations = re.findall('([A-Z][A-Za-z ]+
37 ,([A-Za-z ]+). ?(?i:(Proc.[0-9]+))', line)

```



```

31         if relations:
32             for relation in relations:
33                 json_info[both][relation[1]] =
relation[0]
34
35
36         valid_lines+=1
37
38         i+=1
39     return json_info
40

```

Posto isto, e tendo toda a informação necessária, basta utilizar a função definida por nós **write_on_json** para escrever toda a informação num ficheiro JSON.

```

1 def write_on_json():
2     json_info = info_to_json()
3     f = open('res.json', 'w')
4
5     f.write('[\n')
6
7
8     for (i,entry) in enumerate(json_info):
9         f.write('    {\n')
10        data = json_info[entry]
11        for (j, key) in enumerate(data):
12            f.write(f'        \"{key}\": \"{data[key]}\"')
13
14        if j == len(data)-1:
15            f.write('\n')
16        else:
17            f.write(',\n')
18
19
20        if i == len(json_info)-1:
21            f.write('    }\n')
22        else:
23            f.write('    },\n')
24
25    f.write(']\n')
26    f.close()
27

```

2.2 Ficheiros CSV com listas e funções de agregação

Neste enunciado pretende-se fazer um conversor de um ficheiro **CSV** (*Comma separated values*) para o formato **JSON**. Para se realizar a conversão pretendida, é importante saber que a primeira linha do **CSV** dado funciona

como cabeçalho que define o que representa cada coluna.
 Por exemplo, o seguinte ficheiro "alunos.csv":

```
1 Numero, Nome, Curso
2 3162, Candido Faisca, Teatro
3 7777, Cristiano Ronaldo, Desporto
4 264, Marcelo Sousa, Ciencia Politica
5
```

Corresponde à seguinte tabela:

Numero	Nome	Curso
3162	Candido Faisca	Teatro
7777	Cristiano Ronaldo	Desporto
264	Marcelo Sousa	Ciencia Politica

No entanto, neste trabalho, os CSV recebidos têm algumas extensões.

Listas

Nestes datasets, poderemos ter conjuntos de campos que formam listas.

Listas com tamanho definido

No cabeçalho, cada campo poderá ter um número N que representará o número de colunas que esse campo abrange. Por exemplo, imaginemos que ao exemplo anterior se acrescentou um campo **Notas**, com $N=5$ ("alunos2.csv"):

```
1 Numero, Nome, Curso, Notas{5}, , , , ,
2 3162, Candido Faisca, Teatro, 12, 13, 14, 15, 16
3 7777, Cristiano Ronaldo, Desporto, 17, 12, 20, 11, 12
4 264, Marcelo Sousa, Ciencia Politica, 18, 19, 19, 20, 18
5
```

Isto significa que o campo **Notas** abrange 5 colunas. (Reparem que temos de meter os campos que sobram a vazio, para o **CSV** bater certo).

Listas com um intervalo de tamanhos

Para além de um tamanho único, podemos também definir um intervalo de tamanhos N, M , significando que o número de colunas de um certo campo pode ir de N até M . ("alunos3.csv")

```
1 Numero, Nome, Curso, Notas{3,5}, , , , ,
2 3162, Candido Faisca, Teatro, 12, 13, 14, , ,
3 7777, Cristiano Ronaldo, Desporto, 17, 12, 20, 11, 12
4 264, Marcelo Sousa, Ciencia Politica, 18, 19, 19, 20, ,
5
```

Funções de agregação

Para além de listas, podemos ter funções de agregação, aplicadas a essas listas. Veja os seguintes exemplos ("alunos4.csv" e "alunos5.csv"):

```
1 Numero, Nome, Curso, Notas{3,5}::sum,,,,,
2 3162, Candido Faisca, Teatro, 12, 13, 14,,
3 7777, Cristiano Ronaldo, Desporto, 17, 12, 20, 11, 12
4 264, Marcelo Sousa, Ciencia Politica, 18, 19, 19, 20,
```

```
1 Numero, Nome, Curso, Notas{3,5}::media,,,,,
2 3162, Candido Faisca, Teatro, 12, 13, 14,,
3 7777, Cristiano Ronaldo, Desporto, 17, 12, 20, 11, 12
4 264, Marcelo Sousa, Ciencia Politica, 18, 19, 19, 20,
```

Resultado esperado

```
1 [
2   {
3     "Numero": "3612",
4     "Nome": "Candido Faisca",
5     "Curso": "Teatro"
6   },
7   {
8     "Numero": "7777",
9     "Nome": "Cristiano Ronaldo",
10    "Curso": "Desporto"
11  },
12  {
13    "Numero": "264",
14    "Nome": "Marcelo Sousa",
15    "Curso": "Ciencia Politica"
16  }
17 ]
```

No caso de existirem listas, os campos que representam essas listas devem ser mapeados para listas em **JSON** ("alunos2.csv"):

```
1 [
2   {
3     "Numero": "3612",
4     "Nome": "Candido Faisca",
5     "Curso": "Teatro",
6     "Notas": [12, 13, 14, 15, 16]
7   },
8   {
9     "Numero": "7777",
10    "Nome": "Cristiano Ronaldo",
11    "Curso": "Desporto",
12    "Notas": [17, 12, 20, 11, 12]
13  },
14  {
15    "Numero": "264",
16    "Nome": "Marcelo Sousa",
17    "Curso": "Ciencia Politica",
18    "Notas": [18, 19, 19, 20, 18]
19  }
20 ]
```

20]

No caso em que temos uma lista com uma função de agregação, o processador deve executar a função associada à lista, e colocar o resultado no **JSON**, identificando na chave qual foi a função executada ("alunos4.csv"):

```
1 [
2   {
3     "Numero": "3612",
4     "Nome": "Candido Faisca",
5     "Curso": "Teatro",
6     "Notas_sum": 39
7   },
8   {
9     "Numero": "7777",
10    "Nome": "Cristiano Ronaldo",
11    "Curso": "Desporto",
12    "Notas_sum": 72
13  },
14  {
15    "Numero": "264",
16    "Nome": "Marcelo Sousa",
17    "Curso": "Ciencia Politica",
18    "Notas_sum": 76
19  }
20 ]
```

2.2.1 Resolução do problema

Na seguinte secção vamos apresentar a resolução do problema. A apresentação está estruturada da seguinte forma:

1. Estrutura do programa
2. Estruturas de dados
3. Processamento dos cabeçalhos
4. Processamento das linhas do ficheiro
5. Escrita do resultado em formato JSON

Estrutura do programa

Após analisar o problema e exemplos fornecidos pelo professor, decidimos criar este conversor da seguinte forma.

Primeiramente vai ser lido o ficheiro, depois vão ser processados os cabeçalhos através da criação de uma estrutura em código que espelha a estrutura do

ficheiro. A seguir, a estrutura referida anteriormente vai ser utilizada para processar as linhas do ficheiro, transformando-as e guardando-as num array. Para finalizar esse array é convertido em texto no formato JSON.

Estrutura de dados

As estruturas de dados são um array de cabeçalhos, um array de linhas e um dicionário funções agregadoras.

O array de cabeçalhos é um array de dicionários que vai guardar toda a informação dos cabeçalhos do ficheiro CSV e organizá-la da seguinte forma:

```
1 {  
2     "nome": # nome do campo do csv  
3     "min":  # comprimento minimo do campo caso seja uma lista  
4     "start": # indice inicial  
5     "end":  # indice final  
6     "aggregate_func": # nome da funcao a executar para agregar  
                       a lista  
7 }
```

O array de linhas é um array de dicionários também, mas estes têm a estrutura da linha com a chave a ser um cabeçalho e o valor a ser a informação desse cabeçalho já processada.

```
1 {  
2     "Numero": "264",  
3     "Nome": "Marcelo Sousa",  
4     "Curso": "Ciencia Politica",  
5     "Notas_media": 19.0,  
6     "Soma_prod": 2496  
7 }
```

O dicionário de funções agregadoras contém entradas em que a chave é o nome da função e o valor é a função em si.

```
1 {  
2     "sum": sum,  
3     "media": lambda x : sum(x)/len(x),  
4     "prod": prod  
5 }
```

Processamento de cabeçalhos

Para processar os cabeçalhos executamos um **findall** que nos permite tornar a linha num array de tuplos que tem a seguinte forma (nome, intervalo de colunas, função agregadora). De seguida percorremos esse array e convertemo-lo na primeira estrutura mencionada anteriormente.

```

1 def rangeToTuple(range):
2     res = re.findall("(?:[0-9])+", range)
3     nums = [int(num) for num in res]
4
5     if len(nums) == 0:
6         return [1, 1]
7     if len(nums) == 1:
8         return [1, nums[0]]
9     return nums
10
11 headers = re.findall("([A-Za-z0-9 @]+)
12     +((?:[0-9]+,?)+)?(?:[:[A-Za-z]+]?)", lines[0])
13
14 # Build column structure in dictionaries
15 padding = 0
16 cols = []
17
18 for i, (nome, range, func) in enumerate(headers):
19     min, max = rangeToTuple(range)
20     entry = {
21         "nome": nome,
22         "min": min,
23         "start": padding+i,
24         "end": padding+i+max,
25         "aggregate_func": func[2:]
26     }
27     padding += max - 1
28     cols.append(entry)

```

Processamento das linhas do ficheiro

Para processar as linhas utilizamos a estrutura criada anteriormente, verificamos que tipo de transformações têm de ser efectuadas para cada conjunto de colunas e preenchemos o dicionário com a informação de cada linha.

```

1 aggregators = {
2     "sum": sum,
3     "media": lambda x : sum(x)/len(x),
4     "prod": prod
5 }
6
7 def rowToDict(row, cols):
8     fields = re.split(",", re.sub("\n", "", row))
9     dic = dict()
10
11     for entry in cols:
12         range = fields[entry["start"]:entry["end"]]
13
14         if len(range) <= 1:
15             dic[entry["nome"]] = range[0]
16             continue
17

```

```

18         converted = [int(n) for n in range if n != ""]
19         if entry["aggregate_func"] == "":
20             dic[entry["nome"]] = converted
21         else:
22             dic[f'{entry["nome"]}_{entry["aggregate_func"]}'] =
                aggregators[entry["aggregate_func"]](converted)
23
24     return dic
25
26 body = lines[1:] if len(lines) > 1 else []
27
28 json = [rowToDict(row, cols) for row in body]

```

Escrita do resultado em formato JSON

Na escrita do resultado temos tabulação parametrizável e aquilo que é feito é a conversão para texto, utilizando **fstrings** para converter cada par do dicionário num par de JSON e o dicionário em si num objeto, depois todo esse texto é concatenado de forma a apresentar o resultado final correctamente.

```

1 json = [rowToDict(row, cols) for row in body]
2
3 # Convert to JSON
4 tab = "    "
5
6 def renderPair(key, value):
7     n_value = ('"' + value + '"') if type(value) is str else
        value
8     return f'"{key}": {n_value}'
9
10 def renderObj(entry):
11     outElem = f'{tab}{{\n{tab*2}'
12     outElem += f',\n{tab*2}'.join([renderPair(key, value) for
        key, value in entry.items()])
13     outElem += f'\n{tab}}}'
14     return outElem
15
16 elems = [renderObj(entry) for entry in json]
17 out = "[\n" + ",\n".join(elems) + "\n]"
18
19
20 # Write result
21 with open("res.json", "w") as f:
22     f.write(out)

```

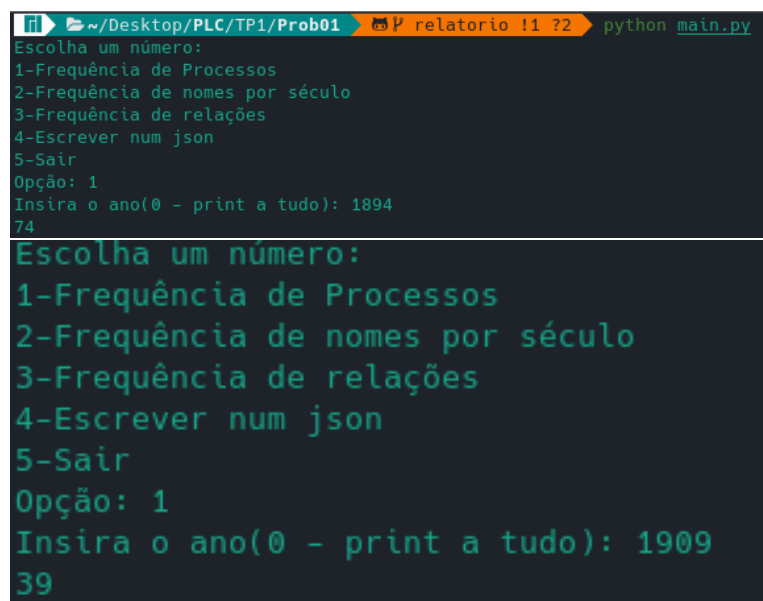
Capítulo 3

Exemplos de funcionamento

3.1 Processador de Pessoas listadas nos Róis de Confessados

Nesta secção vamos mostrar o funcionamento do programa bem como a informação recolhida pelas funções previamente apresentadas.

3.1.1 Frequência de processos por ano



```
~/Desktop/PLC/TP1/Prob01 relatorio !1 ?2 python main.py
Escolha um número:
1-Frequência de Processos
2-Frequência de nomes por século
3-Frequência de relações
4-Escriver num json
5-Sair
Opção: 1
Insira o ano(0 - print a tudo): 1894
74
Escolha um número:
1-Frequência de Processos
2-Frequência de nomes por século
3-Frequência de relações
4-Escriver num json
5-Sair
Opção: 1
Insira o ano(0 - print a tudo): 1909
39
```

Como podemos ver, o programa faz a contagem do número de processos por ano.

3.1.2 Frequência de nomes por século

```

~/Desktop/PLC/TP1/Prob01  P P relatorio !! ?3  python main.py
Escolha um número:
1-Frequência de Processos
2-Frequência de nomes por século
3-Frequência de relações
4-Escriver num json
5-Sair
Opção: 2
Século(0 - print a tudo): 18
({'First': {'Acacio': 2, 'Bento': 1188, 'Josefa': 397, 'Adao': 5, 'Domingos': 3695, 'Mariana':
oa': 123, 'Francisco': 5424, 'Isabel': 1179, 'Mecia': 14, 'Angelica': 64, 'Paulo': 384, 'Dom
', 'Pedro': 1399, 'Felicia': 50, 'Angela': 308, 'Adriao': 10, 'Alexandre': 420, 'Margarida':
eotnio': 5, 'Geraldo': 71, 'Senhorinha': 230, 'Antonio': 7520, 'Catarina': 920, 'Gaspar':
tania': 22, 'Teresa': 579, 'Agostinho': 253, 'Helena': 155, 'Bras': 119, 'Luís': 1489, 'Toma
Inacio': 200, 'Antonia': 601, 'Pascoal': 156, 'Cecilia': 35, 'Rosa': 511, 'Jeronimo': 693,
16, 'Inocencia': 28, 'Luísa': 676, 'Cristina': 24, 'Tinoteo': 2, 'Marcos': 101, 'Apolonia':
', 'Last': {'Carvalho': 2200, 'Maria': 383, 'Azevedo': 946,
', 'Araujo': 2260, 'Barbosa': 1073, 'Duarte': 195, 'Jorge
Sousa': 1887, 'Bacelar': 178, 'Costa': 2437, 'Pereira':
5, 'Brandao': 282, 'Gomes': 1202, 'Afonseca': 337, 'Cape
186, 'Falcao': 92, 'Filipe': 17, 'Mendonca': 100, 'Sot
jes': 282, 'Ribeiro': 1388, 'Almeida': 389, 'Coelho': 614
zerra': 25, 'Santo': 16, 'Veloso': 216, 'Murca': 2, 'Mar
', 'Leitao': 154, 'Joao': 200, 'Cruz': 308, 'Cabecas': 18

```

Como podemos ver, o programa faz a contagem do número de nomes e apelidos por século.

3.1.3 Frequência de relações por século

```
Relação(0 - print a tudo): 0
{'Progenitores': 75607, 'Filho': 38225, 'Tio Paterno': 1853, 'Tio Materno': 1853, 'Sobrinho Paterno': 1635, 'Irmaos': 686, 'Sobrinhos Maternos': 98, 'Irmão Paterno': 98, 'Primo': 638, 'Primo Materno': 225, 'Tio Avo Paterno': 98, 'Irmão Materno': 98, 'Tio Avo Materno': 162, 'Sobrinho Neto Materno': 145, 'Avo Materno': 39, 'Filhos': 39, 'Primos': 13, 'Parente': 4, 'Primos Paternos': 1, 'Irmaos Maternos': 4, 'Tio Avo Paterno': 1, 'Sobrinhos Netos Maternos': 5, 'Sobrinho Bisneto Materno': 3, 'Irmão Paterno': 1, 'Irmão Materno': 1}

Escolha um número:
1-Frequência de Processos
2-Frequência de nomes por século
3-Frequência de relações
4-Escrever num json
5-Sair
Opção: 3
Relação(0 - print a tudo): Irmaos
686
```

Como podemos, ver o programa faz a contagem do número de relações.

3.1.4 Escrever as 20 primeiras linhas num JSON

```
[
  {
    "Processo": "575",
    "Data": "1894-11-08",
    "Pessoa processada": "Aarao Pereira Silva",
    "Pai": "Antonio Pereira Silva",
    "Mãe": "Francisca Campos Silva"
  },
  {
    "Processo": "582",
    "Data": "1909-05-12",
    "Pessoa processada": "Abel Almeida",
    "Pai": "Antonio Manuel Almeida",
    "Mãe": "Teresa Maria Sousa"
  },
  {
    "Processo": "569",
    "Data": "1867-05-23",
    "Pessoa processada": "Abel Alves Barroso",
    "Pai": "Antonio Alves Barroso",
    "Mãe": "Maria Jose Alvares Barroso",
    "Tio Paterno": "Bento Alvares Barroso",
    "Tio Materno": "Domingos Jose Alvares Barroso"
  },
]
```

Como podemos ver o programa está a escrever corretamente no ficheiro JSON.

3.2 Ficheiros CSV com listas e funções de agregação

3.2.1 Informação base sem listas

Para o seguinte input:

```
1 Numero, Nome, Curso
2 3162, Candido Faisca, Teatro
3 7777, Cristiano Ronaldo, Desporto
4 264, Marcelo Sousa, Ciencia Politica
```

Obtemos o seguinte output:

```

1  [
2      {
3          "Número": "3162",
4          "Nome": "Cândido Faísca",
5          "Curso": "Teatro"
6      },
7      {
8          "Número": "7777",
9          "Nome": "Cristiano Ronaldo",
10         "Curso": "Desporto"
11     },
12     {
13         "Número": "264",
14         "Nome": "Marcelo Sousa",
15         "Curso": "Ciência Política"
16     }
17 ]

```

3.2.2 Informação com listas

Para o seguinte input:

```

1 Numero, Nome, Curso, Notas{5}, , , , ,
2 3162, Candido Faísca, Teatro, 12, 13, 14, 15, 16
3 7777, Cristiano Ronaldo, Desporto, 17, 12, 20, 11, 12
4 264, Marcelo Sousa, Ciencia Politica, 18, 19, 19, 20, 18

```

Obtemos o seguinte output:

```

1  [
2      {
3          "Número": "3162",
4          "Nome": "Cândido Faísca",
5          "Curso": "Teatro",
6          "Notas": [12, 13, 14, 15, 16]
7      },
8      {
9          "Número": "7777",
10         "Nome": "Cristiano Ronaldo",
11         "Curso": "Desporto",
12         "Notas": [17, 12, 20, 11, 12]
13     },
14     {
15         "Número": "264",
16         "Nome": "Marcelo Sousa",
17         "Curso": "Ciência Política",
18         "Notas": [18, 19, 19, 20, 18]
19     }
20 ]

```

3.2.3 Listas com um intervalo de tamanhos

Para o seguinte input:

```

1 Numero, Nome, Curso, Notas{3,5},,,,
2 3162, Candido Faísca, Teatro, 12, 13, 14,,
3 7777, Cristiano Ronaldo, Desporto, 17, 12, 20, 11, 12
4 264, Marcelo Sousa, Ciencia Politica, 18, 19, 19, 20,

```

Obtemos o seguinte output:

```

1  [
2      {
3          "Número": "3162",
4          "Nome": "Cândido Faísca",
5          "Curso": "Teatro",
6          "Notas": [12, 13, 14]
7      },
8      {
9          "Número": "7777",
10         "Nome": "Cristiano Ronaldo",
11         "Curso": "Desporto",
12         "Notas": [17, 12, 20, 11, 12]
13     },
14     {
15         "Número": "264",
16         "Nome": "Marcelo Sousa",
17         "Curso": "Ciência Política",
18         "Notas": [18, 19, 19, 20]
19     }
20 ]

```

3.2.4 Funções de agregação

Para o seguinte input:

```

1 Numero, Nome, Curso, Notas{3,5}::media,,,,, Soma{5}::prod,,,,,
2 3162, Candido Faísca, Teatro, 12, 13, 14, , , 12, 13, 14, ,
3 7777, Cristiano Ronaldo, Desporto, 17, 12, 20, 11, 12, 12, 13, 15, ,
4 264, Marcelo Sousa, Ciencia Politica, 18, 19, 19, 20, , 12, 13, 16, ,

```

Obtemos o seguinte output:

```

1  [
2      {
3          "Número": "3162",
4          "Nome": "Cândido Faísca",
5          "Curso": "Teatro",
6          "Notas_media": 13.0,
7          "Soma_prod": 2184
8      },
9      {
10         "Número": "7777",
11         "Nome": "Cristiano Ronaldo",
12         "Curso": "Desporto",
13         "Notas_media": 14.4,
14         "Soma_prod": 2340
15     },
16     {
17         "Número": "264",
18         "Nome": "Marcelo Sousa",
19         "Curso": "Ciência Política",
20         "Notas_media": 19.0,
21         "Soma_prod": 2496
22     }
23 ]

```

Capítulo 4

Conclusão

Fazendo uma retrospectiva referente ao trabalho prático, todos os objetivos foram cumpridos com certeza e clareza. A aposta em realizar 2 dos 5 problemas propostos fez com que todo o grupo conseguisse estar dentro do trabalho prático e adquirir todos os conhecimentos inerentes ao mesmo.

A realização deste trabalho prático fez com que entedêssemos melhor certas funções do módulo **re**, como, por exemplo, **search**, **split**, **findall** e **sub**. Também nos deu sensibilidade no que toca a lidar com ficheiros de grandes dimensões, visto que nem sempre sabíamos de que forma a informação presente nos mesmos estava formatada.

Em suma, todos os objetivos foram concluídos sem grande dificuldade pelo grupo e consideramos que este trabalho foi uma boa preparação para o resto do semestre, no qual teremos certamente desafios nos quais utilizaremos o conhecimento aqui adquirido.