Universidade do Minho
Escola de Engenharia

Processamento de Linguagens (3º ano de MIEI)

Trabalho Prático I

Relatório - Grupo 78

Ana Luísa Lira Tomé Carneiro (A89533)

Ana Rita Abreu Peixoto (A89612)

Luís Miguel Lopes Pinto (A89506)

Abril 2021

Resumo

Este primeiro trabalho prático no âmbito da unidade curricular de processamento de linguagens consistiu na elaboração de um projeto em *python* onde através de expressões regulares e filtros de texto foi desenvolvido um conversor genérico de ficheiros CSV para ficheiros JSON.

No presente relatório explicamos como desenvolvemos os filtros de texto de forma a extrair informação pertinente do ficheiro CSV, como implementamos expressões regulares e como aplicamos métodos do *python* de forma a converter a formatação do ficheiro CSV em ficheiro JSON. Finalmente, incluímos exemplos de utilização do programa desenvolvido, assim como versões alternativas para a implementação, decisões tomadas e problemas ultrapassados.

Conteúdo

Listings

Capítulo 1

Introdução

1.1 Enquadramento e Contextualização

Os ficheiros CSV são ficheiros sem formatação muito utilizados por softwares como o *Microsoft Excel*, onde cada linha representa um registo distinto. Neste formato, os valores de cada linha estão separados por delimitadores como a vírgula ou o ponto e vírgula. Os ficheiros JSON têm um formato compacto muito usado em sistemas de exportação/transferência de dados entre aplicações para assegurar a interoperabilidade como *web services*, sendo um substituto dos ficheiros XML.

Com este trabalho pretendemos implementar um conversor genérico de CSV para JSON, onde se converte a informação presente em CSV para um ficheiro JSON. Assim, torna-se fácil e rápida a criação dos ficheiros JSON.

1.2 Problema e Objetivos

Em geral, o projeto pretende aprofundar a aprendizagem de temáticas e conceitos já abordados nas aulas, tais como:

• aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases dentro de textos;

• desenvolver sistematicamente, a partir de ER, Processadores de Linguagens Regulares, ou Filtros de Texto (FT), que filtrem ou transformem textos com base no conceito de regras de produção Condição-Ação;

• utilizar o módulo 'er' com suas funções de search(), split(), sub() etc.. do *Python* para implementar os FT pedidos.

De um conjunto de cinco exercícios foi nos proposto a realização de apenas um, determinado através da expressão (NrGrupo % 5) + 1. Assim, foi nos atribuído o exercício 4, já que (78 % 5) + 1 = 4.

Em particular, neste exercício 4, pretende-se fazer um conversor de um qualquer ficheiro gravado em formato CSV (*Comma Separated Values*, original e tipicamente usado para descarregar uma Folha de Cálculo num ficheiro de texto) para o formato JSON (um formato textual neutro e muito simples, baseado no conceito de um

conjunto de pares {"campo": "valor"}, concorrente do XML enquanto sistema de exportação/transferência de dados entre aplicações para assegurar a interoperabilidade).
Para poder realizar a conversão pretendida, é importante saber que a primeira linha do CSV dado funciona como cabeçalho que descodifica a que correspondem os valores que vêm nas linhas seguintes. Até aqui nada de novo, mas é claro que leva mais uns ingredientes. O dataset dado poderá ter listas aninhadas nalgumas células. Mas nesse caso o cabeçalho terá um asterisco '*' a seguir ao nome do respetivo campo. Se nada mais for colocado o conversor deverá converter cada valor dessa coluna numa lista em JSON. Mas a seguir ao asterisco pode haver uma função de agregação: sum, avg, max, min. Aí o conversor terá de aplicar a operação de fold respetiva sobre a lista e produzir o JSON de acordo.

Resumindo, através da exploração de expressões regulares e respetivas funções do *python*, pretende-se converter um ficheiro CSV (*Comma Separated Values*) para JSON.

1.3 Estrutura do documento

O presente relatório tem como objetivo ilustrar o trabalho realizado. Para isso, estruturamos o relatório em diferentes capítulos:

O primeiro capítulo é a **Introdução**. Aqui serão abordados tópicos como o enquadramento e contextualização do tema proposto, o problema que se pretende resolver e o seu objetivo e também será exposto o modo de estruturação do relatório.

De seguida, no capítulo 2 o foco será na **Análise e Especificação** do problema, onde será efetuada uma descrição informal do problema seguida da especificação dos requisitos, que permitirá abordar em detalhe a estrutura e conteúdo dos ficheiros CSV e JSON.

No terceiro capítulo apresentamos o **Desenho da Resolução**. De forma a exemplificar a solução obtida, esta secção está subdividida em duas: **Estruturas de Dados** e **Tratamento do Documento CSV**, onde irão ser expostas as estruturas que sustentaram o programa e o modo de leitura e tratamento dos dados do ficheiro CSV, respetivamente.

O capítulo 4 assenta na **Codificação e Testes**, que se subdivide em **Alternativas, Decisões e Problemas de Implementação** e em **Testes Realizados e Resultados em JSON** onde estão presentes exemplos de testes, através do fornecimento de um ficheiro CSV como input e obtenção de um ficheiro JSON.

No capítulo 5 é efetuada uma **Conclusão** e análise crítica do trabalho efetuado, realçando aspetos positivos da implementação e aspetos a melhorar.

Por fim, existe também uma última secção **Apêndice A** onde está presente o **Código do Programa**.

Capítulo 2

Análise e Especificação

2.1 Descrição Informal do Problema

Para um qualquer ficheiro gravado em formato CSV, independentemente do número de linhas e colunas, pretende-se extrair a primeira linha que funciona como cabeçalho para as restantes linhas do ficheiro. Posteriormente, vamos executar a conversão para um novo ficheiro JSON, ou seja, nada mais que um formato textual neutro e simples, baseado no conceito de um conjunto de pares {"campo": "valor"}, onde o "campo"será retirado do cabeçalho e o "valor"corresponde ao respetivo conteúdo presente na linha a fazer a correspondência para JSON. Contudo, este conteúdo terá de ser analisado, caso o cabeçalho contenha um asterisco seguido de uma função de agregação (max, min, sum ou avg) será efetuado um fold sobre a lista (o conteúdo) e produzido o JSON de acordo.

2.2 Especificação dos Requisitos

Como forma de cumprir com o objetivo do problema apresentado é necessário analisar e especificar os dados e requisitos do projeto. Para isso, é fundamental ter em consideração a estrutura e características dos ficheiros CSV e JSON usados para implementação da solução.

2.2.1 Ficheiro CSV

O *input* que o programa criado recebe é um ficheiro CSV. Trata-se de um ficheiro simples, em que os dados aparecem separados por um delimitador. Para o programa em questão e dado o contexto do problema, consideramos o ponto e vírgula como separador. Além disso, de forma a identificar cada campo presente em cada linha, existe um *header* na primeira linha do ficheiro. Este cabeçalho do ficheiro CSV contém os nomes que identificam cada campo. Caso esse nome seja seguido de um asterisco, significa que esse campo é uma lista. Seguido do asterisco pode haver uma função de agregação como max, min, avg ou sum.

2.2.2 Ficheiro JSON

O *output* que o programa produz é um ficheiro JSON. Estes ficheiros iniciam e terminam com parêntesis retos e cada registo está dividido em vários campos onde cada campo tem um valor associado cumprindo com o formato {"campo": "valor"}. No caso em que os valores sejam listas estes são representados por [ ] onde cada elemento, de forma a cumprir com a formatação JSON, tem de estar entre aspas.

Capítulo 3

Desenho da Resolução

Dada por concluída a fase de análise e especificação, chegamos agora a etapa de implementação, ou por outras palavras, desenho da resolução. Nesta fase é importante realçar as estruturas de dados utilizadas, bem como os algoritmos por detrás da implementação.

3.1 Estruturas de Dados

Para iniciarmos a leitura do ficheiro CSV é necessário armazenar os campos presentes no cabeçalho numa lista. Assim, implementou-se um *array, header,* que guarda os nomes dos campos especificados na primeira linha do ficheiro.

Em adição, de forma a tornar o tratamento e leitura do ficheiro mais eficiente, armazenamos no *array, indiceList* os índices do *array header* onde estão presentes os campos do tipo lista, isto é, campos com asteriscos. Além disso, também foram armazenados os nomes desses campos no *array, lista,*.

Finalmente, criamos um *array, option,* onde ficam armazenados o nome das funções de agregação (max, min, avg e sum). Tanto as funções como o nome dos campos encontram-se armazenadas nos mesmos índices, mas em *arrays* diferentes (as funções no *array option* e o nome do campo no *array lista*). Assim, associamos as funções de agregação aos respetivos campos que as implementam.

3.2 Tratamento do Documento CSV

O programa implementado recebe como *input* um ficheiro CSV. Primeiramente, procede-se à abertura do ficheiro para posterior leitura. De seguida, é analisada a primeira linha do ficheiro: o cabeçalho. A partir desta primeira linha é possível extrair informações, tais como: os campos de cada linha e quais denotam listas e, para esses, se existe ou não uma função de agregação. Com o auxílio da função *split* foi possível separar os valores do cabeçalho e armazena-los no *array header*. De seguida, foram populados os *arrays indiceList* e *option*. Para o primeiro caso, de modo a cumprir com o objetivo de apenas armazenar o índice dos campos que são listas, utilizou-se a função *search* com uma expressão regular muito simples '*'. No caso do preenchimento do *array option* o procedimento passou por recorrer às funções *split* e *sub*. A primeira utilizou a expressão regular '*'

para aceder apenas à primeira parte da *string*. No caso da função *sub*, as expressões regulares utilizadas tiveram como propósito eliminar o caracter '\n'.

Posteriormente ao armazenamento nas estruturas de dados, efetuou-se a restante leitura do ficheiro com recurso à função *readlines()* do *python* e guardou-se o resultado numa variável. Para esta variável que possui o conteúdo do ficheiro, foi criado um ciclo *for* que itera linha a linha. O corpo deste ciclo trata da análise e conversão de CSV para JSON. De modo a concretizar tal feito, começou-se por separar a linha nos seus diferentes campos, com o auxílio da função *split*. De seguida, é efetuado um ciclo *for* interior que percorre todos os campos de cada linha. Neste ciclo interno verifica-se se o campo atual se trata de uma lista. Em caso afirmativo, recorre-se à função *findall* para extrair todos os valores numéricos da *string*. Dado que o valor de retorno do *findall* é uma lista de *string*, foi necessário converte-la para lista de inteiros, para cumprir com o formato JSON. Nesta fase são realizadas as operações correspondentes às funções de agregação sum, min, max e avg, caso existam. Estas operações foram realizadas de forma simples recorrendo às funções do *python* que calculam os valores mínimo, máximo e soma dos valores de uma lista. Para a média, efetuou-se a divisão da soma dos elementos da lista pelo seu comprimento.

De modo a escrever corretamente em formato JSON, foi necessário ter em conta a sua estrutura e ,desta forma, eliminar ou substituir determinados caracteres recorrendo à função *sub*. Também por motivos estruturais do JSON, teve de ser acrescentado no inicio e no final do ficheiro parênteses retos.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Durante a realização do trabalho surgiram alguns casos que necessitavam de ser tratados como exceções à norma, tais como, listas vazias e listas com outros caracteres sem ser números. Para estes casos decidiu-se não aplicar nenhuma função agregadora uma vez que como se tratam de casos excecionais impediam o utilizador de obter um resultado apropriado e correto. Assim, usou-se o método *sub* como forma de eliminar os parêntesis curvos. De seguida, com o auxílio do método *split* e da expressão regular '[()]', dividiu-se a *string* resultante pelas vírgulas de forma a obter um *array* com os caracteres ou com uma string vazia entre '' (plicas). Finalmente, como forma de obter a formatação do JSON decidiu-se utilizar o método *sub* para substituir as ' \'' (plicas) por '\"' (aspas). No final, o *array* resultado encontra-se na formatação JSON, onde as listas vazias () equivalem a [""] e as listas com caracteres como (a,b,c,d) equivalem a ["a", "b", "c", "d"].

No que toca às decisões tomadas na implementação, houve algumas que merecem destaque. Entre as diversas opções para a implementação das estruturas de dados, optamos pelas listas devido à sua simplicidade. Uma alternativa seria utilizar conjuntos em vez de duas lista. Além disso, também optamos por fazer escrita no ficheiro JSON através de funções de escrita do *python* ao invés da utilização da biblioteca JSON. Em relação aos ficheiros de *input* do programa, consideramos que o delimitador mais adequado seria o ponto e vírgula, e como tal, os ficheiros CSV de teste criados obedecem a este princípio.

Inicialmente foi implementada uma versão B do problema sem tratamentos de exceções e sem ter em consideração casos de eficiência do código. Esta versão, apresentada nos Anexos A.2, foi implementada de forma a que o ficheiro JSON seja apresentado ao utilizador através do terminal. Para que o *output* seja encaminhado para o ficheiro JSON é necessário executar o programa com o comando:

```
python3.9 versaoB.py > file.json
```

Este programa está implementado de forma dividir o cabeçalho do ficheiro CSV através do ';', armazenando o nome dos seus campos num *array fields*. De seguida percorremos todos os valores de cada linha aplicando

cada um dos campos do cabeçalho de forma a constituir o formato JSON {"campo": "valor"}. Para cada campo é necessário verificar se é uma lista e caso o seja, é chamada a função *parseList* que irá fazer o tratamento e leitura do campo aplicando, caso exista, a respetiva função de agregação. Para realizar esse tratamento e leitura utilizou-se diversas ERs de forma a que seja aplicada a formatação do JSON, como por exemplo selecionar a função de agregação adequada ou aspetos de estrutura como o uso de parênteses retos ([ ]) e das aspas ("").

Na versão A, presente nos Anexos A.1, o raciocínio de implementação é semelhante, contudo nesta versão A tentou-se avaliar qual seria a melhor opção de leitura e tratamento do ficheiro CSV de forma a melhorar a eficiência e de manter a formatação JSON em casos de exceção.

4.2 Testes Realizados e Resultados em JSON

Como forma de testar o programa implementado criaram-se diversos ficheiros CSV de teste com ajuda de um site de geração de conteúdo aleatório[1] para ficheiros CSV. Em baixo encontra-se um ficheiro CSV gerado por esse site com 10 campos e 6 linhas de registos e a sua respetiva conversão para JSON. No *listing* 4.1 podemos que alguns campos nos registos encontram-se vazios ou com caracteres diferentes de números e a sua conversão para JSON no *listing* 4.2.

```
id;curso;nome;apelido;email;notas*;notas*sum;notas*avg;notas*max;notas*min
100;MIETI;Margette;Rodmann;Margette.Rodmann@plmail.com;([]);(a,b,c,d,e);(14,2,11,19)
    ;(14,2,11,19);(14,2,11,19)
101;MIEI;Noelle;Natica;Noelle.Natica@plmail.com;(19,11,17,18,12,10);(a,b,c)
    ;(19,11,17,18,12,10);(19,11,17,18,12,10);(19,11,17,18,12,10)
102;MIEI;Phedra;Elvyn;Phedra.Elvyn@plmail.com;(8,10,10,17,9,12);(8,10,10,17,9,12)
    ;(8,10,10,17,9,12);(oisrgofiwrog);(8,10,10,17,9,12)
103;MIEFIS;Donnie;Grobe;Donnie.Grobe@plmail.com;(12,12,15,8,13,4);(12,12,15,8,13,4)
    ;(12,12,15,8,13,4);(12,12,15,8,13,4);(12,12,15,8,13,4)
104;MIEI;Hope;Felecia;Hope.Felecia@plmail.com;(16,19,16,13,19,19);(16,19,16,13,19,19)
    ;(16,19,16,13,19,19);(16,19,16,13,19,19);(16,19,16,13,19,19)
105;MIEFIS;Helena;Ball;Helena.Ball@plmail.com;(14,18,18,12,7,18);(14,18,18,12,7,18)
    ;(14,18,18,12,7,18);(14,18,18,12,7,18);(14,18,18,12,7,18)
```

Listing 4.1: Teste CSV

```
[
   {
    "id": "100",
    "curso": "MIETI",
    "nome": "Margette",
    "apelido": "Rodmann",
    "email": "Margette.Rodmann@plmail.com",
    "notas": ["[]"],
    "notas": ["a", "b", "c", "d", "e"],
    "notas_avg": 11.5,
    "notas_max": 19,
    "notas_min": 2
```

[1] https://extendsclass.com/csv-generator.html

```
  },
  {
   "id": "101",
   "curso": "MIEI",
   "nome": "Noelle",
   "apelido": "Natica",
   "email": "Noelle.Natica@plmail.com",
   "notas": [19, 11, 17, 18, 12, 10],
   "notas": ["a", "b", "c"],
   "notas_avg": 14.5,
   "notas_max": 19,
   "notas_min": 10
  },
  {
   "id": "102",
   "curso": "MIEI",
   "nome": "Phedra",
   "apelido": "Elvyn",
   "email": "Phedra.Elvyn@plmail.com",
   "notas": [8, 10, 10, 17, 9, 12],
   "notas_sum": 66,
   "notas_avg": 11.0,
   "notas": ["oisrgofiwrog"],
   "notas_min": 8
  },
  {
   "id": "103",
   "curso": "MIEFIS",
   "nome": "Donnie",
   "apelido": "Grobe",
   "email": "Donnie.Grobe@plmail.com",
   "notas": [12, 12, 15, 8, 13, 4],
   "notas_sum": 64,
   "notas_avg": 10.666666666666666,
   "notas_max": 15,
   "notas_min": 4
  },
  {
   "id": "104",
   "curso": "MIEI",
   "nome": "Hope",
   "apelido": "Felecia",
   "email": "Hope.Felecia@plmail.com",
   "notas": [16, 19, 16, 13, 19, 19],
   "notas_sum": 102,
   "notas_avg": 17.0,
   "notas_max": 19,
   "notas_min": 13
  },
  {
```

```
    "id": "105",
    "curso": "MIEFIS",
    "nome": "Helena",
    "apelido": "Ball",
    "email": "Helena.Ball@plmail.com",
    "notas": [14, 18, 18, 12, 7, 18],
    "notas_sum": 87,
    "notas_avg": 14.5,
    "notas_max": 18,
    "notas_min": 7
   }
]
```

Listing 4.2: Resultado JSON

Capítulo 5

Conclusão

Dada por concluída a realização do trabalho prático, consideramos boa prática fazer uma apreciação crítica realçando não só os aspetos positivos como também as dificuldades que surgiram e o modo como estas foram colmatadas.

No que diz respeito aos pontos fortes, destacamos a universalidade do trabalho, uma vez que funciona para qualquer tipo de ficheiro CSV cujo delimitador seja ";", i.e, funciona independentemente do número de colunas ou linhas e podemos usar quantas vezes queiramos as funções de agregação. Destacamos também a eficiência que trouxe a estrutura de dados implementada, que se refletiu na leitura e análise do cabeçalho apenas uma vez, e permitiu diminuir significativamente o número de linhas lidas.

Durante a realização deste trabalho surgiram algumas dificuldades, tais como: lidar com os casos de erro no formato do ficheiro CSV e decidir qual a melhor opção de implementação. Apesar disso, consideramos que os problemas foram ultrapassados com sucesso.

Desta forma, pretendemos explicitar que a realização deste projeto foi um aspeto essencial para aprimorar e melhor cimentar os conhecimentos sobre Expressões Regulares e Filtros de Texto. Para além disso, uma vez que todas as dificuldades foram ultrapassadas de forma eficaz concluímos que o balanço do resultado final foi positivo.

Apêndice A

Código do Programa

A.1 Versão A - Principal

```
import sys
import re

f = open("output.json","w+")
f.write("[\n")

csv = open("csv_test_relatorio.csv")
first_line = csv.readline() #ler a primeira linha do ficheiro csv

header = re.split(r';',first_line) #separar a primeira linha no ;

indiceList = [] #guardar os indices das colunas que tem listas
option = [] #fun  o de agrega  o sobre as lista

for i in range(len(header)): #verificar se h  listas
    r = re.search(r'\*',header[i])
    if r:
        indiceList.append(i)

for i in indiceList:
    lista = re.split(r'\*',header[i])
    op = re.sub(r'\n',r'',lista[-1]) #capturar tudo exceto o \n
    if op == '': option.append("null")
    else: option.append(op)

CSVFile = csv.readlines()
CSVLen = len(CSVFile)

#processar uma linha de cada vez
for c,linha in enumerate(CSVFile):
    res = re.split(r';',linha)
```

```
32
33      f.write("\t {\n")
34
35      for i in range(len(res)):
36
37          if i in indiceList: #se aquele indice    uma lista
38              listStr = re.findall(r'(\d+)',res[i])
39
40              listNr = [int(nr) for nr in listStr] #converter lista de string para int
41
42              campo = re.split(r'\*', header[i])
43
44              ind = indiceList.index(i)
45              if len(listStr) != 0:
46
47                  if option[ind] == "sum":
48                      value = sum(listNr)
49                  elif option[ind] == "avg":
50                      value = sum(listNr) / len(listNr)
51                  elif option[ind] == "min":
52                      value = min(listNr)
53                  elif option[ind] == "max":
54                      value = max(listNr)
55                  else:
56                      value = listNr
57
58                  if (option[ind] != "null"):
59                      s = '\t\t"' + campo[0] + "_" + option[ind] + '": ' + str(value)
60                  else :
61                      s = '\t\t"' + campo[0] + '": ' + str(value)
62              else:
63                  res[i] = re.sub(r'[()]',r'',res[i])
64                  auxStr = re.split(r',',res[i])
65                  auxStr = re.sub(r'\'',r'"',str(auxStr)) #"
66                  s = '\t\t"' + campo[0] + '": ' + auxStr
67          else: # caso nao seja lista
68              header[i] = re.sub(r'\n',r'',header[i])
69              res[i] = re.sub(r'\n',r'',res[i]) #sub para remover \n
70              s = '\t\t"' + header[i] + '": "' + res[i] + '"'
71          if i != len(res) - 1:
72              s = s + ', '
73          f.write(s + '\n')
74      f.write("\t },\n") if c != CSVLen - 1 else f.write("\t }\n")
75  f.write("]\n")
```

Listing A.1: Programa em Python - Versão Principal

A.2 Versão B - Alternativa

```
import re

def parseList(y, res, i):

    if y.group(2) == "":
        lst1 = re.sub(r'\(', "[", res[i])
        lst2 = re.sub(r'\)', "]", lst1)
        print("\t\t"+ '"'+ y.group(1) + '"'+ ": " +lst2, end="")
    else:
        index = re.sub(r'\(', "",res[i])
        index2 = re.sub(r'\)', "", index)
        values = re.split(r',', index2)

        if y.group(2) == "sum":

            add = 0
            for value in values:
                number = int(value)
                if(isinstance(number,int))
                    add += number


            print("\t\t"+ '"'+ y.group(1) + "_" + y.group(2) + '"'+ ": " + str(add), end="")
        elif y.group(2) == "max":

            numbers = []
            for value in values:
                number = int(value)
                numbers.append(number)

            print("\t\t"+ '"'+ y.group(1)+ "_" + y.group(2) + '"'+ ": " +str(max(numbers)),
    end="")
        elif y.group(2) == "min":

            numbers = []
            for value in values:
                number = int(value)
                numbers.append(number)

            print("\t\t"+ '"'+y.group(1) + "_" + y.group(2) +'"'+ ": " + + str(min(numbers))
    , end="")
        elif y.group(2) == "avg":

            add = 0
            for value in values:
                number = int(value)
                add += number
            print("\t\t"+ '"'+y.group(1)+ "_" + y.group(2) + '"'+ ": " +str(add/len(values))
```

```
    , end="")
47          else:
48              lst1 = re.sub(r'\(', "[", res[i])
49              lst2 = re.sub(r'\)', "]", lst1)
50              print("\t\t"+ '"'+ y.group(1) + '"'+ ": " +lst2, end="")
51
52
53  f = open("csv_test_relatorio.csv", "r")
54  first_line = f.readline()
55  fields = re.split(r';',first_line)
56  print("[\n")
57  content = f.readlines()
58  limit = len(content)
59
60  for count,line in enumerate(content):
61      res = re.split(r';', line)
62      print("\t{")
63      i=0
64      length = len(fields)
65
66      for field in fields:
67          field = field.strip("\n")
68          y = re.search(r'(\w+)\*(\w*)',field)
69
70          if y:
71              parseList(y,res,i)
72          else:
73              res[i] = res[i].strip("\n")
74              print("\t\t"+ '"' + field + '"' + " : " + '"'+res[i]+ '"', end="")
75
76          i+=1
77          if i != length:
78              print(",\n")
79          else :
80              print("\n")
81
82      if count == (limit-1):
83          print("\t}\n")
84      else:
85          print("\t},\n")
86
87  print("]\n")
```

Listing A.2: Programa em Python - Versão Alternativa