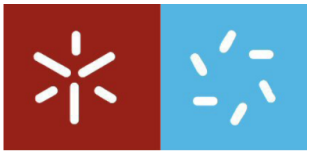

Licenciatura em Ciências da Computação

Processamento de Linguagens e Compiladores

(Grupo-6)

Trabalho Prático 1

Relatório de Desenvolvimento

Bruno Miguel Fernandes Araújo
(a97509)

Bruna Micaela Rodrigues Araújo
(a84914)

Filipe José Silva Castro
(a96156)

13 novembro 2022

Resumo

Neste relatório iremos colocar em prova os conceitos que nos foram ensinados e utilizaremos estes para resolver os desafios propostos. Iremos mostrar o nosso raciocínio por detrás de cada resolução, alguns exemplos e por fim indicar algumas dificuldades que enfrentamos.

Conteúdo

Capítulo 1

Introdução

No âmbito da cadeira de Processamento de Linguagens e Compiladores foi-nos proposto pelo educando um trabalho com objetivo de entendermos melhor a utilização de expressões regulares no contexto da programação, mais propriamente na linguagem Python.

Foram-nos proposto vários exercícios dentro dos quais teríamos de escolher pelo menos um.

Sendo assim, escolhemos o exercício/grupo 5 e decidimos não só expandir um pouco mais nesse grupo como criar um novo pequeno exercício não relacionado com os propostos.

O grupo 5 baseia-se resumidamente numa criação de um conversor de um ficheiro do tipo .csv num do tipo .json.

O exercício que criamos trata-se de uma criação de um programa que ,através de filtros, consiga indicar algumas características de um poema/texto fornecido.

Iremos então apresentar as nossas soluções a estes problemas, explorando as nossas decisões e mostrando alguns exemplos.

Estrutura do Relatório

Após uma pequena introdução no capítulo 1.

Temos a explicação e exploração do exercício em que nos foi pedido criar um conversor juntamente com alguns exemplos, capítulo 2

De seguida temos o exercício que criamos onde também falamos das decisões que fizemos e mostramos alguns exemplos , capitulo 3

Depois com capítulo 4 damos uma conclusão ao trabalho onde dá-mos uma pequena síntese, mencionamos o que foi concluído e apontamos as dificuldades que tivemos.

E por fim temos os códigos completos de cada exercício, primeiro do conversor e de seguida o programa que analisa poemas/textos em prosa.

Capítulo 2

Conversor .csv para .json (exercício/grupo 5)

Para realizarmos esta conversão temos que ler a primeira linha do ficheiro CSV que servirá como cabeçalho para definir o que cada coluna representa.

Temos então de primeiramente abrir o ficheiro de input e criar um ficheiro para eventualmente este ser escrito.

```
if inputFromUser!="":
   #Abrir o ficheiro quando o input    do tipo:<nome_do_ficheiro>
    f = open(inputFromUser+".csv", "r")
   (...)
   #Criar o ficheiro quando o input    do tipo:<nome_do_ficheiro>
    f = open(inputFromUser+".json", "w")
    f.write(texto)
```

Depois disto lemos o ficheiro e guardamos cada linha num dicionário onde a chave corresponde ao número da linha e o valor é o array com as palavras dessa linha.

Para isto demos split de cada linha por vírgula.

```
for linha in f:
        lista[i] = (re.split(r',', linha2))
        i += 1
```

(linha2 e a sua explicação aparecerá mais à frente) Por fim, para colocar no formato .json criamos uma variável string chamada de texto que vai sendo incrementada com a sintaxe que define este formato.

```
texto = "[\n"
   #Percorrer os dados guardados
   for j in range(1, i):
      valores = []
      texto += "\t{\n"
      for x in range(len(lista[j]) - 1):
             texto += "\t\t\"" + lista[0][x] + "\": \"" + lista[j][x] + "\",\n"
      texto += "\t},\n"
   texto += "]\n"
f.write(texto)
```

2.1 Listas

Os campos agora têm de ter a capacidade de ter listas com tamanho N definido. Quando um campo tem uma lista este tem no seu nome {N}.
Usamos a expressão

```
'([a-zA-Z]+)\{([\backslash d]+)(![\backslash d]+)?\}'
```

para verificar se existe um campo com {N}, o primeiro grupo será então a palavra e o segundo o dígito N (a parte

```
(![\backslash d]+)?
```

vai fazer sentido no próximo exercício.)
Além disso criamos um array onde serão guardados todos os valores que aparecem, além disso se este for o último elemento ele tem um $\backslash n$
junto a este e precisamos de então usar o .sub() para extrair apenas no número.

```
texto = "[\n"
   #Percorrer os dados guardados
   for j in range(1, i):
      valores = []
      texto += "\t{\n"
      for x in range(len(lista[j]) - 1):
       if re.search(r'([a-zA-Z]+){([\d]+)(![\d]+)?}',lista[0][x]) :
         #A vari vel m encontra um padr o que nos diz o nome da vari vel e os n meros
                                                        m nimo e m ximo de argumentos
         m = re.search(r'([a-zA-Z]+){([\d]+)(![\d]+)?}',lista[0][x])
         valores.append(int(re.sub(r'([0-9]+)\n',r'\1',lista[j][x])))
         z=x+1
         t=0
         #Se n o houver n mero m ximo de argumentos
         else:
          for t in range(0, int(m.group(2))):
             if z < len(lista[j]) and lista[j][z] != "\n" and lista[j][z]:
                  l = re.sub(r'([0-9]+)\n', r'\1', lista[j][z])
                  valores.append(int(l))
             z += 1
         texto+=escreve(m,funcao,valores)
       elif lista[0][x] == "": pass
       else:
             texto += "\t\t\"" + lista[0][x] + "\": \"" + lista[j][x] + "\",\n"
      texto += "\t},\n"
   texto += "]\n"
f.write(texto)
```

Depois de construído o necessário enviamos m (contém o nome do campo e o N) e o array com todos os valores (a "função"mencionada irá fazer sentido mais para a frente)
para uma função que encarrega-se de passar o array em string.

```
def escreve(m,funcao,valores):
   #quando n o tem fun   o
    if not funcao:
        return "\t\t\"" + m.group(1) + "\": " + "[" + ','.join(map(str,valores)) + "]\n"
    else:
        string = "\t\t\"" + m.group(1) + "_" + funcao.group(1) + "\": "
```

```
        resposta = func(funcao.group(1), valores)
        if resposta[0] == 1:
            string += str(resposta[1]) + "\n"
        else:
            string += "[" + ','.join(map(str, valores)) + "]\n"
        return string
```

2.2 Listas com intervalo

As listas previamente mencionadas agora têm a capacidade de ter um intervalo de tamanhos. Quando um campo tem uma lista com intervalo este tem no seu nome {N1,N2}. (Em que N1 é o limite menor e N2 é o limite maior.

Tivemos de substituir a vírgula dentro de {N1,N2} por um outro caracter porque quando fazemos .split() por vírgulas este iria separar as chavetas, sendo assim escolhemos um ! e usamos o .sub() com a expressão que apanha o pretendido (({[\d]+),([\d]+}))

E substituímos por \1!\2

Ou seja, grupo 1!grupo 2, em que o grupo 1 seria {N1 e o grupo 2 o N2}.

```
linha2 = re.sub(r'({[\d]+),([\d]+})', r'\1!\2', linha)
```

Para este caso , usamos a expressão

```
'([a-zA-Z]+)\{([\backslash d]+)(![\backslash d]+)?\}
```

desta temos no primeiro grupo o nome da da lista, no segundo grupo o limite minimo e no terceiro grupo temos um !junto do limite maior.

Para retirar este ! criamos uma variável nmax que usará o sub() no terceiro grupo e substituirá este ! por nada, eliminando-o (nmax fica então o o limite maior).

De resto este faz o mesmo processo que só em {N} , também recorre á função "escrever"e segue o mesmo caminho.

```
texto = "[\n"
   #Percorrer os dados guardados
   for j in range(1, i):
      valores = []
      texto += "\t{\n"
      for x in range(len(lista[j]) - 1):
       if re.search(r'([a-zA-Z]+){([\d]+)(![\d]+)?}',lista[0][x]) :
         #A vari vel m encontra um padr o que nos diz o nome da vari vel e os n meros
                                                    m nimo e m ximo de argumentos
         m = re.search(r'([a-zA-Z]+){([\d]+)(![\d]+)?}',lista[0][x])
         valores.append(int(re.sub(r'([0-9]+)\n',r'\1',lista[j][x])))
         z=x+1
         t=0
         #Se houver n mero m ximo de argumentos
         if(m.group(3)):
          nmax = re.sub(r'!','',m.group(3))
          for t in range(0,int(nmax)):
             if z<len(lista[j]) and lista[j][z]!="\n" and lista[j][z]:
                 l=re.sub(r'([0-9]+)\n',r'\1',lista[j][z])
                 valores.append(int(l))
             z+=1
         #Se n o houver n mero m ximo de argumentos
```

```
22          else:
23           for t in range(0, int(m.group(2))):
24              if z < len(lista[j]) and lista[j][z] != "\n" and lista[j][z]:
25                   l = re.sub(r'([0-9]+)\n', r'\1', lista[j][z])
26                   valores.append(int(l))
27              z += 1
28          texto+=escreve(m,funcao,valores)
29       elif lista[0][x] == "": pass
30       else:
31              texto += "\t\t\"" + lista[0][x] + "\": \"" + lista[j][x] + "\",\n"
32      texto += "\t},\n"
33   texto += "]\n"
```

2.3 Funções de agregação

Agora é possível adicionar funções das listas a uma lista. Quando um campo que tem uma lista quer ter uma função das listas associada,este tem no seu nome ::(nome da função).

Para este caso , usa-se o mesmo código anterior mas temos de guardar a função numa variável(neste caso chama-se funcao) através da expressão regular ::([a-z]+) , onde temos como o primeiro grupo, o nome da função.

```
(...)
          m = re.search(r'([a-zA-Z]+){([\d]+)(![\d]+)?}',lista[0][x])
          funcao = re.search(r'::([a-z]+)', lista[0][x]) #apenas acrescenta-mos esta
                                                         atribu    o
          valores.append(int(re.sub(r'([0-9]+)\n',r'\1',lista[j][x])))
(...)
```

De seguida enviamos para a função escrever, o nome desta função juntamente com as outra componentes (array com os valores e m(nome da lista, limite menor e se existir,o limite maior))

Nesta, encaminhamos o nome da função e o array para func.

```
def escreve(m,funcao,valores):
   #quando n o tem fun   o
    if not funcao:
        return "\t\t\"" + m.group(1) + "\": " + "[" + ','.join(map(str,valores)) + "]\n"
    else:
        string = "\t\t\"" + m.group(1) + "_" + funcao.group(1) + "\": "
        resposta = func(funcao.group(1), valores)
        if resposta[0] == 1:
            string += str(resposta[1]) + "\n"
        else:
            string += "[" + ','.join(map(str, valores)) + "]\n"
        return string
```

Por fim em func , dependendo do nome da função, é aplicada uma função diferente no array.

```
# Fun   o que executa as opera   es;
# Recebe o nome da opera   o
# Devolve uma lista com [ 1 caso o resultado seja um valor ou 0 caso o resultado seja uma
                                                    lista , resultado da opera   o ]
def func(nome,lista):
    if nome == 'sum':
        return [1,sum(lista)]
    elif nome == 'soma':
        return [1,sum(lista)]
```

```
    elif nome == 'media':
        return [1,sum(lista)/len(lista)]
    elif nome == 'ord':
        return[0,lista.sort()]
    elif nome == 'min':
        return [1,min(lista)]
    elif nome == 'max':
        return [1,max(lista)]
    elif nome == 'moda':
        ocorr = {}
        for m in lista:
            ocorr[m] = lista.count(m)
        return [1,max(ocorr)]
    elif nome == "mediana":
        lista.sort()
        if len(lista)%2 == 1:
            return [1,lista[int(len(lista)/2)]]
        else:
            return [1,(lista[int(len(lista)/2)]+lista[(int(len(lista)/2))-1])/2]
```

Nós acrescentamos a capacidade de poderem ser usadas funções que devolvem um array quando isto acontece o return tem de ter um [0,...] quando devolver um número o return tem de ter um [1,...]

Depois dependendo do que é enviado , na função escrever,este vai para um if ou else onde a mensagem é escrita da forma correta.

No caso de devolver um número esta apenas acrescenta-o à string, se devolver uma lista esta é transformada numa string e adicionada à string final.

```
def escreve(m,funcao,valores):
   #quando n o tem fun   o
    if not funcao:
        return "\t\t\"" + m.group(1) + "\": " + "[" + ','.join(map(str,valores)) + "]\n"
    else:
        string = "\t\t\"" + m.group(1) + "_" + funcao.group(1) + "\": "
        resposta = func(funcao.group(1), valores)
        #quando devolve um numero
        if resposta[0] == 1:
            string += str(resposta[1]) + "\n"
        else:
        #quando devolve um arrray
            string += "[" + ','.join(map(str, valores)) + "]\n"
        return string
```

2.4 Exercicios extra

Possibilidade de ser usada uma expressão regular para filtrar o resultado final.

Para ser chamada esta capacidade temos de colocar no input o seguinte formato nome do ficheiro expressão regular

Seguindo então esse raciocínio, temos então de separar este caso do normal, fazemos isto usando o .search() para detetar se existe um espaço , se existir este pega no nome do ficheiro usando a expressão

```
([^\s]*)\s
```

em que o nome encontra-se no primeiro grupo.

```
if inputFromUser!="":

   #Abrir o ficheiro quando o input    do tipo:<nome_do_ficheiro> <express o regular> ou
                                                do tipo <nome_do_ficheiro> <nome_do_campo>
                                                 <valor_que_est _nesse_campo>
   if re.search(r'\s', inputFromUser):
    ficheiro=re.search(r'([^\s]*)\s',inputFromUser)
    print(ficheiro)
    f = open(str(ficheiro.group(1))+".csv", "r")
```

Além disso na construção do dicionário acrescentamos este caso e que só são adicionados novos elementos a este ,caso tenham na sua linha a expressão regular.

Distinguimos este caso vendo se este tiver no input um espaço (len(re.findall(r' ',inputFromUser))==1).

Usamos .search(),no input, com a expressão

```
'[\s]+([\s]'
```

e pegamos no grupo 1, onde estará a string da expressão regular, se esta expressão se encontrar na linha (re.search(str(k.group(1)),linha2)) fazemos então a adição ao dicionário da mesma forma como no caso normal.

```
   lista = {}
   i = 0
   for linha in f:

     #Substituir v rgulas dentro de chavetas por pontos de exclama  o (ex: {3,5} -> {3!
                                                    5})
     linha2 = re.sub(r'({[\d]+),([\d]+})', r'\1!\2', linha)

     #Fazer split dos dados do ficheiro .csv usando as v rgulas
     if i==0:
        lista[i] = (re.split(r',', linha2))
        i += 1
     else:
     #Quando um input    do tipo <nome_do_ficheiro> <express o regular>
      if len(re.findall(r'\s',inputFromUser))==1:
       k = re.search(r'[\s]+([^\s]*)', inputFromUser)
       if re.search(str(k.group(1)),linha2):
        lista[i] = (re.split(r',', linha2))
        i += 1
```

Possibilidade de filtrar por campo com valor, ou seja colocando um campo e um valor este devolve todos os que contém esse valor no campo.

Para ser chamada esta capacidade temos de colocar no input o seguinte formato nome do ficheiro nome do campo valor do campo

A abertura do ficheiro neste tipo de input segue a mesma lógica que o exercício anterior, mas na construção do dicionário temos de acrescentar num novo caso para esta situação.

Distinguimos este caso vendo se este tiver no input dois espaços (len(re.findall(r' ',inputFromUser))==2).

Depois usaremos o .search() com a expressão

```
'[ ]+([^ ]*)[ ]([^ ]*)'
```

de forma a obtermos no primeiro grupo o nome do campo e no segundo grupo o valor do campo

Verificamos se o segundo grupo existe/foi apanhado , se o nome do campo pertence à primeira linha (linha que define o cabeçalho) e se o valor do campo está presente na linha onde o ciclo se encontra com uma vírgula atrás e à frente ("(?¡=[,])"+str(ok.group(2))+"(?=[,])").

Por fim, teremos de ver se o índice do nome do campo coincide com o índice do valor do campo.(é necessário criar uma nova variável onde guardaremos a cada palavra da lista, para isto usamos o split() para as vírgulas).

Se forem verificadas estas condições é feita a adição ao dicionário como no caso normal e no caso anterior.

```
    lista = {}
    i = 0
    verifica=[[]]
    for linha in f:

      #Substituir v rgulas dentro de chavetas por pontos de exclama  o (ex: {3,5} -> {3!5})
      linha2 = re.sub(r'({[\d]+),([\d]+})', r'\1!\2', linha)

      #Fazer split dos dados do ficheiro .csv usando as v rgulas
      if i==0:
         lista[i] = (re.split(r',', linha2))
         i += 1
      else:
      #Quando um input    do tipo <nome_do_ficheiro> <express o regular>
       if len(re.findall(r'\s',inputFromUser))==1:
        k = re.search(r'[\s]+([^\s]*)', inputFromUser)
        if re.search(str(k.group(1)),linha2):
         lista[i] = (re.split(r',', linha2))
         i += 1
       #Quando um input    do tipo <nome_do_ficheiro> <nome_do_campo> <valor_que_est _nesse_campo>
       elif len(re.findall(r'\s',inputFromUser))==2:
        ok = re.search(r'[\s]+([^\s]*)[\s]([^\s]*)', inputFromUser)
        if ok.group(2) and str(ok.group(1)) in lista[0] and re.search("(?<=[,])"+str(ok.group(2))+"(?=[,])",linha2):
                verifica[0] = (re.split(r',', linha2))
                if lista[0].index(str(ok.group(1))) == verifica[0].index(str(ok.group(2))):
                    lista[i] = (re.split(r',', linha2))
                    i += 1
       #Quando o input    do tipo:<nome_do_ficheiro>
       else:
          lista[i] = (re.split(r',', linha2))
          i += 1
```

2.5 Exemplos

1. Exemplo 1: (Lista com intervalo e função que devolve um valor(sum)) Input: nome do ficheiro

```
Número,Nome,Curso,Notas{3,5}::sum,,,,,
3162,Cândido Faísca,Teatro,12,13,14,,
7777,Cristiano Ronaldo,Desporto,17,12,20,11,12
264,Marcelo Sousa,Ciência Política,18,19,19,20,
```

Converte o ficheiro .csv para .json e faz a operação sum na lista Notas com tamanho de intervalo de 3 a 5.


```
[
        {
                "Número": "3162",
                "Nome": "Cândido Faísca",
                "Curso": "Teatro",
                "Notas_sum": 39
        },
        {
                "Número": "7777",
                "Nome": "Cristiano Ronaldo",
                "Curso": "Desporto",
                "Notas_sum": 72
        },
        {
                "Número": "264",
                "Nome": "Marcelo Sousa",
                "Curso": "Ciência Política",
                "Notas_sum": 76
        },
]
```

2. Exemplo 2: (Lista sem intervalo e função que devolve um valor(media)) Input: nome do ficheiro

```
Número,Nome,Curso,Notas{5}::media,,,,,
3162,Cândido Faísca,Teatro,12,13,14,15,16
7777,Cristiano Ronaldo,Desporto,17,12,20,11,12
264,Marcelo Sousa,Ciência Política,18,19,19,20,18
```

Converte o ficheiro .csv para .json e faz a operação media na lista Notas com tamanho 5.


```
[
        {
                "Número": "3162",
                "Nome": "Cândido Faísca",
                "Curso": "Teatro",
                "Notas_media": 14.0
        },
        {
                "Número": "7777",
                "Nome": "Cristiano Ronaldo",
                "Curso": "Desporto",
                "Notas_media": 14.4
        },
        {
                "Número": "264",
                "Nome": "Marcelo Sousa",
                "Curso": "Ciência Política",
                "Notas_media": 18.8
        },
]
```

3. Exemplo 3: (Lista com intervalo e função que devolve um array(ord)) Input: nome do ficheiro

```
Número,Nome,Curso,Notas{3,5}::ord,,,,,
3162,Cândido Faísca,Teatro,12,13,14,,
7777,Cristiano Ronaldo,Desporto,17,12,20,11,12
264,Marcelo Sousa,Ciência Política,18,19,19,20,
```

Converte o ficheiro .csv para .json e faz a operação ord na lista Notas com tamanho de intervalo de 3 a 5.


```
[
        {
                "Número": "3162",
                "Nome": "Cândido Faísca",
                "Curso": "Teatro",
                "Notas_ord": [12,13,14]
        },
        {
                "Número": "7777",
                "Nome": "Cristiano Ronaldo",
                "Curso": "Desporto",
                "Notas_ord": [11,12,12,17,20]
        },
        {
                "Número": "264",
                "Nome": "Marcelo Sousa",
                "Curso": "Ciência Política",
                "Notas_ord": [18,19,19,20]
        },
]
```

4. Exemplo 4: (Ficheiro emd e uma expressão regular,primeiro exercício extra)

   Input: emd gay


```
[
        {
                "_id": "6045074cd77860ac9483d34e",
                "index": "0",
                "dataEMD": "2020-02-25",
                "nome/primeiro": "Delgado",
                "nome/último": "Gay",
                "idade": "28",
                "género": "F",
                "morada": "Gloucester",
                "modalidade": "BTT",
                "clube": "ACRroriz",
                "email": "delgado.gay@acrroriz.biz",
                "federado": "true",
        },
        {
                "_id": "6045087f4aaa6f9a5b10a4ec",
                "index": "98",
                "dataEMD": "2021-02-15",
                "nome/primeiro": "Douglas",
                "nome/último": "Gay",
                "idade": "29",
                "género": "F",
                "morada": "Lowgap",
                "modalidade": "Atletismo",
                "clube": "GDGoma",
                "email": "douglas.gay@gdgoma.net",
                "federado": "true",
        },
]
```


   Converte o ficheiro emd.csv e devolve todos que contêm a palavra gay.

5. Exemplo 5: (Ficheiro emd , nome do campo e valor do campo,segundo exercício extra)
   Input: emd index 4


```
[
        {
                "_id": "6045074c3319a0f9e79aad87",
                "index": "4",
                "dataEMD": "2019-09-01",
                "nome/primeiro": "Mckay",
                "nome/último": "Bolton",
                "idade": "29",
                "género": "F",
                "morada": "Chilton",
                "modalidade": "Futebol",
                "clube": "ACDRcrespos",
                "email": "mckay.bolton@acdrcrespos.me",
                "federado": "false",
        },
        {
                "_id": "6045080a94353bd5c1b93145",
                "index": "4",
                "dataEMD": "2019-01-30",
                "nome/primeiro": "Jennifer",
                "nome/último": "Morris",
                "idade": "26",
                "género": "F",
                "morada": "Oretta",
                "modalidade": "Dança",
                "clube": "AmigosMontanha",
                "email": "jennifer.morris@amigosmontanha.us",
                "federado": "false",
        },
        {
                "_id": "6045087fcc1396f35014c1c5",
                "index": "4",
                "dataEMD": "2019-05-29",
                "nome/primeiro": "Frieda",
                "nome/último": "Hansen",
                "idade": "32",
                "género": "F",
                "morada": "Allamuchy",
                "modalidade": "Ciclismo",
                "clube": "GDGoma",
                "email": "frieda.hansen@gdgoma.co.uk",
                "federado": "true",
        },
]
```


   Converte o ficheiro emd.csv e devolve todos que têm index 4.

Capítulo 3

Características de textos em poema e em prosa

Após receber um ficheiro de texto com um texto, um poema ou apenas um texto em prosa, este deve escrever num ficheiro de texto algumas características deste.

Abertura e leitura do ficheiro .txt do input.

Criação do ficheiro final .txt com nome de "Resultado"e escrita neste o pretendido.

```
import re

inputFromUser=input(">> ")

if inputFromUser!="":
    resultado=""
    texto=""
    f = open(inputFromUser + ".txt", "r")
    #passar de ficheiro para uma variavel com o texto
    for linha in f:
        texto+=linha
   (...)
    f.close()
    f = open("Resultado.txt", "w")
    f.write(texto+"\n\n"+resultado)
```

3.1 Numero de Palavras

O numero de Palavras encontradas no ficheiro de texto. Encontrando todas as ocorrencias de todas as palavras usando o findall() com a expressão [a-zA-Zz-ý]+, obtemos um array com todas as palavras, vemos o tamanho deste e obtemos o numero de palavras presente no texto.

```
    Palavras=re.findall(r'([a-zA-Zz-  ]+)',texto)  #z-   tem no seu intervalo todos os
                                                    caracteres com acento            etc
    resultado += "Numero de Palavras: " + str(len(Palavras)) + "\n\n"
```

3.2 Palavra que teve mais ocurrências

A palavra que teve mais ocorrências no ficheiro de texto.

Usamos um dicionário que vai ter como chave a palavra e como valor o número de ocorrências dele. Usamos um lookbehind e um lookahead para não serem contadas aquelas palavras que se encontram dentro de outras, a expressão (?¡![a-zA-Zz-ý])'+palavra+ ('(?![a-zA-Zz-ý]) faz que seja possível apanhar essa palavra sem letras à esquerda e a sua direta.

Depois vimos qual palavra no dicionário é que tinha o maior valor.

```
    palocr={}
    for palavra in Palavras:
        if palavra not in palocr:
            palocr[palavra]=len(re.findall('(?<![a-zA-Zz-  ])'+palavra+'(?![a-zA-Zz-  ])',
                                                                texto))
    maior=""
    for palavra in palocr:
        if palocr[palavra]==max(list(palocr.values())):
            maior=palavra
    resultado+="Palavra que ocorreu mais: "+maior+" com " +str(palocr[maior])+"
                                                    ocorr ncias.\n\n"
```

3.3 Tipo do texto (poema/prosa)

O tipo do texto que se encontra no ficheiro.

$\backslash n\backslash n$

ocorre quando há uma mudança de estrofe, por isso podemos encontrar todas as ocorrências desses e se o array vazio não for vazio temos um poema se for temos um texto em prosa.

```
Estrofes=len(re.findall("\n\n",texto))+1
if Estrofes>1:
 resultado += "Tipo do texto: Poema\n\n"
else:
 resultado += "Tipo do texto: Prosa\n"
```

Se este se encontrar em prosa, cria um ficheiro com o próprio texto mais as mensagens previamente mencionadas.

3.4 Poema

Se for um poema este explora mais características.

1. Numero de estrofes: Número de estrofes no poema.

```
resultado += "Numero de Estrofes: " + str(Estrofes)+"\n\n"
```

2. Tipo de estrofes: O tipo de cada estrofe.

   Depois de separarmos o texto por $\backslash n\backslash n$

(separar por estrofe), separamos a estrofe por $\backslash n$

o tamanho do array resultante devolve uma palavra dependendo de quanto for. (O máximo que colocamos foi até 7 versos ou seja uma Septilha).

```
        tiposquadra=["Nada","Mon stico","D stico","Terceto","Quarteto","Quintilha",
                                                    "Sextilha","Septilha"]
        l=1
```

```
        for a in estrofes:
                numero=0
                numero=len(re.split(r'\n',a))
                resultado +=str(l)+"    estrofe: "+tiposquadra[numero]+"\n\n"
                l+=1
```

3. Soneto? Se este for um soneto então deve indicá-lo.

   Um soneto é um poema com quatro estrofes, a primeira e segunda estrofe são quartetos e a terceira e quarta estrofe são tercetos.

   Então verificamos na mensagem que estivemos a construir até agora, se esta tem:

   Escrito 1 e Q (equivale a linha 1 e Quarteto, não há necessidade de procurar a Palavra toda quando só existe um tipo de estrofe com Q) (1[ª :a-z]+Q) .

   Escrito 2 e Q. (2[ª :a-z]+Q)

   Escrito 3 e T (equivale a linha 1 e Quarteto, não necessidade de procurar a Palavra toda quando só existe um tipo de estrofe com T) (3[ª :a-z]+T)

   Escrito 4 e T. (4[ª :a-z]+T)

   Temos um soneto.

```
        if Estrofes==4 and re.search(r'1[   :a-z]+Q',resultado) and re.search(r'2[
                                                    :a-z]+Q',resultado) and re.
                                                    search(r'3[   :a-z]+T',resultado
                                                    )and re.search(r'4[   :a-z]+T',
                                                    resultado) :
            resultado+="Este Poema    um Soneto.\n\n"
```

4. Tipos de Rimas: Decidimos considerar que duas palavras rimam quando a última palavra de cada verso tem os seus 2 últimos caracteres iguais.(Isto é verdade para algumas que rimam, mas infelizmente existe a possibilidade de duas palavras não rimarem e terem esta terminação igual assim como a existência de palavras que rimam por causa do seu tom.)

   Achamos que encontrar rimas num poema era uma ideia boa então apesar de ser algo que pensamos que não seja possível programar todas as possibilidades, decidimos seguir com aquela consideração.

   (a) Construção do texto com os padrões de cada rima: O texto original agora tem de incluir dígitos no início equivalentes a cada tipo de rima.

   Começamos por primeiro criar um array linhas que vai ter todas as linhas do poema, dando split() por $\backslash n$.

Depois através do search() e da expressão '[a-zA-Zz-ý]*([a-zA-Zz-ý]{2})[!,.?]*$\backslash n'$.

Usando um dicionário com chaves equivalentes à última terminação de cada linha e valores equivalentes a um dígito que é incrementado quando é encontrada uma nova chave, conseguimos evitar as ocorrências iguais o que faz com que palavras com terminações iguais, tenham o mesmo dígito.

Durante o passo anterior, um array guarda os valores do dicionário nele, depois do ciclo terminamos com um array em que cada índice dele equivale a uma linha e contém nessas posições o dígito atribuído à terminação dessa linha.

```
1         linhas = re.split(r'\n', texto)
2         rimas = {}
3         array = []
4         ns = 0
5         #Cria um array com com os digitos equivalente a cada rima, para depois
                                                    inserir no inicio de cada
                                                    verso
6         for i in linhas:
7             i+='\n'
8             rima = re.search(r'[a-zA-Zz-  ]*([a-zA-Zz-  ]{2})[!,.?;]*\n', i)
9             if rima and rima.group(1) not in rimas:
10                rimas[rima.group(1)] = ns
11                ns += 1
12                array.append(rimas[rima.group(1)])
13            elif rima:
14                array.append(rimas[rima.group(1)])
```

De seguida iremos então reescrever o texto original adicionando o dígito atribuído aos últimos 2 caracteres de cada última palavra de cada linha.

Aproveitamos e usamos este ciclo para guardar a última palavra de cada linha num array, através da expressão ([a-zA-Zz-ý]*)[!,.?]*e pegando no grupo 1.

```
1         x=0
2         texto=""
3         ultima=[]
4         #Guarda a ultima palavra de cada linha, atualiza o array com todas as
                                                    linhas, colocando o \n no
                                                    fim de cada linha e o digito
                                                     correspondente a cada rima.
5         #Al m disso tamb m reconstroi o poema de forma a que seja possivel
                                                    imprimi-lo no fim com os
                                                    padr es das rimas
6         for t in range(0,len(linhas)):
7           if linhas[t]:
8               linhas[t]+= '\n'
```

```
                texto += str(array[x]) + "- " + linhas[t]
                linhas[t]= str(array[x]) + "- " + linhas[t]
                x += 1
                ultima.append(re.search(r' ([a-zA-Zz-  ]*)[!,.?;]*\n', linhas[t]).
                                                                group(1))
            else:
                texto += "\n"
                ultima.append("vazio")
```

(b) Rima Emparelhada: Diz o número de rimas Emparelhadas juntamente com as palavras que rimam e a linha onde estas se encontram.

Uma rima Emparelhada é quando temos o padrão AA ou seja neste contexto é quando temos duas linhas seguidas com o mesmo dígito no inicio.

Pegamos no primeiro dígito da linha (x) e da linha seguinte (x+1) através da expressão '([0-9])+ ', e se estes coincidirem acrescentamos ao array (onde vão ser guardadas as rimas emparelhadas) a palavra final da linha onde o ciclo se encontra juntamente com o número da linha e a palavra final da linha seguinte juntamente com o número dessa linha.

```
        RimEmp=[]
        numlinha=1
        for x in range(0,len(linhas)):
            if linhas[x]:
                linhax=int(re.search(r'([0-9])+\- ', linhas[x]).group(1))
                #Rima Emparelhada
                if x<len(linhas)-1 and linhas[x+1]:
                    linhax1= int(re.search(r'([0-9])+\- ', linhas[x + 1]).group(1
                                                                        ))
                    if linhax==linhax1:
                      RimEmp.append(ultima[x]+" (linha "+str(numlinha)+") - "+
                                                                        ultima[x+1]+"
                                                                        (linha "+str(
                                                                        numlinha+1)+")
                                                                        ")
            numlinha+=1
        resultado+="Tipos de Rimas: \n\nRimas Emparelhadas: "+str(len(RimEmp))+"\
                                                                n"+'\n'.join(map(str,RimEmp)
                                                                )+"\n\n"
```

(c) Rima Encadeada: Diz o número de rimas Interpoladas juntamente com as palavras que rimam e a linha onde estas se encontram.

Uma rima Encadeada é quando temos o padrão ABA ou seja neste contexto é quando temos duas linhas com o mesmo dígito no início separadas por uma linha com dígito diferente.

Pegamos no primeiro dígito da linha (x) e da linha que vem depois da seguinte (x+2) através da expressão '([0-9])+ ', e se estes coincidirem acrescentamos ao array (onde vão ser guardadas as rimas interpoladas) a palavra final da linha onde o ciclo se encontra juntamente com o número da linha desta e a palavra final da linha que vem depois da seguinte juntamente com o número dessa linha.

```
        RimEnc=[]
        numlinha=1
        for x in range(0,len(linhas)):
            if linhas[x]:
                linhax=int(re.search(r'([0-9])+\- ', linhas[x]).group(1))
                #Rima Encadeadas
                if x<len(linhas)-2 and linhas[x+2] and linhas[x+1]:
                    linhax2= int(re.search(r'([0-9])+\- ', linhas[x + 2]).group(1
                                                                        ))
```

```
                    if linhax==linhax2:
                      RimEnc.append(ultima[x]+" (linha "+str(numlinha)+") - "+
                                                                    ultima[x+2]+"
                                                                    (linha "+str(
                                                                    numlinha+2)+")
                                                                    ")
            numlinha+=1
        resultado+="Rimas Encadeadas: "+str(len(RimEnc))+"\n"+'\n'.join(map(str,
                                                        RimEnc))+"\n\n"
```

(É preciso ver se linhas[x+1] não se encontra vazio porque se este estiver, estamos num caso em que o último verso de uma estrofe e o primeiro verso da seguinte estrofe rimam e não queremos guardar esse caso.)

(d) Rima Alternada: Diz o número de rimas Alternadas juntamente com as palavras que rimam e a linha onde estas se encontram.

Uma rima Alternada é quando temos o padrão ABAB ou seja neste contexto é quando temos duas linhas com o mesmo dígito no início separadas por uma linha com dígito diferente que irá coincidir com o dígito da linha depois desta.

Seguimos o mesmo raciocínio das anteriores mas neste caso temos de ter o dígito da linha (x) onde se encontra o ciclo igual ao dígito da linha depois da seguinte (x+2) e o dígito da linha seguinte (x+1) da atual coincide com o dígito da linha depois desta (x+3). Além disso as linhas não podem ter uma seguinte com o mesmo dígito no início, ou seja se tivermos AAAA não é alternada.

```
        RimAlt=[]
        numlinha=1
        for x in range(0,len(linhas)):
             if linhas[x]:
                linhax=int(re.search(r'([0-9])+\- ', linhas[x]).group(1))
                #Rima Alternada
                if x<len(linhas)-3 and linhas[x+1] and linhas[x+2] and linhas[x+3
                                                                    ]:
                    linhax1= int(re.search(r'([0-9])+\- ', linhas[x + 1]).group(1
                                                                    ))
                    linhax2= int(re.search(r'([0-9])+\- ', linhas[x + 2]).group(1
                                                                    ))
                    linhax3= int(re.search(r'([0-9])+\- ', linhas[x + 3]).group(1
                                                                    ))
                    if linhax==linhax2 and linhax!=linhax1 and linhax1==linhax3
                                                                    and linhax1!=
                                                                    linhax2 and
                                                                    linhax2!=linhax3
                                                                    :
                      RimAlt.append(ultima[x] + " (linha " + str(numlinha) + ") -
                                                                     " + ultima[x
                                                                    + 2] + " (
                                                                    linha " + str(
                                                                    numlinha + 2)+
                                                                    ") - "+ ultima
                                                                    [x+1]+" (linha
                                                                     "+str(
                                                                    numlinha+1)+")
                                                                     - "+ultima[x+
                                                                    3]+" (linha "+
                                                                    str(numlinha+3
                                                                    )+")")
             numlinha+=1
        resultado+="Rimas Alternadas: "+ str(len(RimAlt))+ "\n" + '\n'.join(map(
                                                        str, RimAlt)) + "\n\n"
```

(e) Rima Interpolada: Diz o número de rimas Interpoladas juntamente com as palavras que rimam e a linha onde estas se encontram

Uma rima Interpolada é quando temos o padrão ABBA ou seja neste contexto é quando temos duas linhas com o mesmo dígito no início separadas por duas linhas com dígito diferente destas mas comum entre si.

Neste caso temos de ter o dígito da linha (x) onde se encontra o ciclo igual ao dígito da linha que vem depois de duas linhas (x+3) e o dígito da linha seguinte (x+1) da atual coincide com o dígito da linha depois da seguinte (x+2). Além disso, temos de ter em conta que o padrão AAAA não é interpolada.

```
        RimInt=[]
        numlinha=1
        for x in range(0,len(linhas)):
             if linhas[x]:
                linhax=int(re.search(r'([0-9])+\- ', linhas[x]).group(1))
               #Rima Interpolada
                if x<len(linhas)-3 and linhas[x+1] and linhas[x+2] and linhas[x+3
                                                                    ]:
                    linhax1= int(re.search(r'([0-9])+\- ', linhas[x + 1]).group(1
                                                                    ))
                    linhax2= int(re.search(r'([0-9])+\- ', linhas[x + 2]).group(1
                                                                    ))
                    linhax3= int(re.search(r'([0-9])+\- ', linhas[x + 3]).group(1
                                                                    ))
                    if linhax==linhax3 and linhax!=linhax1 and linhax1==linhax2
                                                                    and linhax2!=
                                                                    linhax3:
                      RimInt.append(ultima[x] + " (linha " + str(numlinha) + ") -
                                                                     " + ultima[x
                                                                    + 3] + " (
                                                                    linha " + str(
                                                                    numlinha + 3)+
                                                                    ") / "+ ultima
                                                                    [x+1]+" (linha
                                                                     "+str(
                                                                    numlinha+1)+")
                                                                     - "+ultima[x+
                                                                    2]+" (linha "+
                                                                    str(numlinha+2
                                                                    )+")")
             numlinha+=1
        resultado+="Rimas Interpoladas: " + str(len(RimInt)) + "\n" + '\n'.join(
                                                    map(str, RimInt)) + "\n\n"
```

(f) Sem Rima: Diz o número de últimas palavras de cada verso que não rimam, juntamente com a linha em que esta se encontra.

Para este caso basta verificarmos se a última palavra da linha onde o ciclo se encontra não está em nenhum dos outros arrays associados a rimas.(Rima Alternada, Rima Emparelhada, Rima Alternada,Rima Interpolada).

```
        SemRima=[]
        numlinha=1
        for x in range(0,len(linhas)):
            if linhas[x]:
                #Sem Rima
                if not re.search(ultima[x],','.join(map(str, RimEmp))) and not re
                                                                .search(ultima[x],',
                                                                '.join(map(str,
```

```
                                                                RimEnc)))and not re.
                                                                search(ultima[x],','
                                                                .join(map(str,
                                                                RimAlt)))and not re.
                                                                search(ultima[x],','
                                                                .join(map(str,
                                                                RimInt))):
                        SemRima.append(ultima[x]+" (linha "+str(numlinha)+")")
            numlinha+=1
        resultado+="Sem Rima: " + str(len(SemRima)) +"\n"+ '\n'.join(map(str,
                                                        SemRima)) + "\n\n"
```

3.5 Exemplos

(Na escrita de palavras com dígitos com sinais por exemplo Ó etc estas ao serem escritas aparecem em formato utf-8 e depois de algumas tentativas de correção não conseguimos encontrar uma solução).
No input colocamos apenas o nome do ficheiro.

1. Poema 1: Mar Português de Fernando Pessoa No input colocamos apenas o nome do ficheiro.

```
Ó mar salgado, quanto do teu sal
São lágrimas de Portugal!
Por te cruzarmos, quantas mães choraram,
Quantos filhos em vão rezaram!
Quantas noivas ficaram por casar
Para que fosses nosso, ó mar!

Valeu a pena? Tudo vale a pena
Se a alma não é pequena.
Quem quer passar além do Bojador
Tem que passar além da dor.
Deus ao mar o perigo e o abismo deu,
Mas nele é que espelhou o céu.
```

Cria um ficheiro texto Resultado que depois de escrito vai conter:

```
0- Ã\ mar salgado, quanto do teu sal
0- SÃ£o lÃ¡grimas de Portugal!
1- Por te cruzarmos, quantas mÃ£es choraram,
1- Quantos filhos em vÃ£o rezaram!
2- Quantas noivas ficaram por casar
2- Para que fosses nosso, Ãª mar!

3- Valeu a pena? Tudo vale a pena
3- Se a alma nÃ£o Ã© pequena.
4- Quem quer passar alÃ©m do Bojador
4- Tem que passar alÃ©m da dor.
5- Deus ao mar o perigo e o abismo deu,
6- Mas nele Ã© que espelhou o cÃ©u.


Numero de Palavras: 74

Palavra que ocorreu mais: o com 3 ocorrências.

Tipo do texto: Poema

Numero de Estrofes: 2

1ª estrofe: Sextilha
```

```
2ª estrofe: Sextilha

Tipos de Rimas:

Rimas Emparelhadas: 5
sal (linha 1) - Portugal (linha 2)
choraram (linha 3) - rezaram (linha 4)
casar (linha 5) - mar (linha 6)
pena (linha 8) - pequena (linha 9)
Bojador (linha 10) - dor (linha 11)

Rimas Encadeadas: 0


Rimas Alternadas: 0


Rimas Interpoladas: 0


Sem Rima: 2
deu (linha 12)
cÃ©u (linha 13)
```

2. Poema 2: Amor é fogo que arde sem se ver de Luís Vaz de Camões

```
Amor é fogo que arde sem se ver;
É ferida que dói, e não se sente;
É um contentamento descontente;
É dor que desatina sem doer.

É um não querer mais que bem querer;
É um andar solitário entre a gente;
É nunca contentar-se de contente;
É um cuidar que se ganha em se perder.

É querer estar preso por vontade;
É servir a quem vence, o vencedor;
É ter com quem nos mata, lealdade.

Mas como causar pode seu favor
Nos corações humanos amizade,
Se tão contrário a si é o mesmo Amor?
```

Cria um ficheiro texto Resultado que depois de escrito vai conter:

```
0- Amor Ã© fogo que arde sem se ver;
```

1- Ã‰ ferida que dÃ³i, e nÃ£o se sente;
1- Ã‰ um contentamento descontente;
0- Ã‰ dor que desatina sem doer.

0- Ã‰ um nÃ£o querer mais que bem querer;
1- Ã‰ um andar solitÃ¡rio entre a gente;
1- Ã‰ nunca contentar-se de contente;
0- Ã‰ um cuidar que se ganha em se perder.

2- Ã‰ querer estar preso por vontade;
3- Ã‰ servir a quem vence, o vencedor;
2- Ã‰ ter com quem nos mata, lealdade.

3- Mas como causar pode seu favor
2- Nos coraÃ§Ãµes humanos amizade,
3- Se tÃ£o contrÃ¡rio a si Ã© o mesmo Amor?

Numero de Palavras: 95

Palavra que ocorreu mais: Ã com 10 ocorrências.

Tipo do texto: Poema

Numero de Estrofes: 4

1ª estrofe: Quarteto

2ª estrofe: Quarteto

3ª estrofe: Terceto

4ª estrofe: Terceto

Este Poema é um Soneto.

Tipos de Rimas:

Rimas Emparelhadas: 2
sente (linha 2) - descontente (linha 3)
gente (linha 7) - contente (linha 8)

Rimas Encadeadas: 2
vontade (linha 11) - lealdade (linha 13)
favor (linha 15) - Amor (linha 17)

Rimas Alternadas: 0

```
Rimas Interpoladas: 2
ver (linha 1) - doer (linha 4) / sente (linha 2) - descontente (linha 3)
querer (linha 6) - perder (linha 9) / gente (linha 7) - contente (linha 8)

Sem Rima: 2
vencedor (linha 12)
amizade (linha 16)
```

3. Poema 3: Autopsicografia de Fernando Pessoa

```
O poeta é um fingidor
Finge tão completamente
Que chega a fingir que é dor
A dor que deveras sente.

E os que leem o que escreve,
Na dor lida sentem bem,
Não as duas que ele teve,
Mas só a que eles não têm.

E assim nas calhas de roda
Gira, a entreter a razão,
Esse comboio de corda
Que se chama coração.
```

Cria um ficheiro texto Resultado que depois de escrito vai conter:

```
0- O poeta Ã© um fingidor
1- Finge tÃ£o completamente
0- Que chega a fingir que Ã© dor
1- A dor que deveras sente.

2- E os que leem o que escreve,
3- Na dor lida sentem bem,
2- NÃ£o as duas que ele teve,
4- Mas sÃ³ a que eles nÃ£o tÃªm.

5- E assim nas calhas de roda
6- Gira, a entreter a razÃ£o,
5- Esse comboio de corda
6- Que se chama coraÃ§Ã£o.


Numero de Palavras: 64

Palavra que ocorreu mais: que com 6 ocorrências.
```

```
Tipo do texto: Poema

Numero de Estrofes: 3

1ª estrofe: Quarteto

2ª estrofe: Quarteto

3ª estrofe: Quarteto

Tipos de Rimas:

Rimas Emparelhadas: 0


Rimas Encadeadas: 5
fingidor (linha 1) - dor (linha 3)
completamente (linha 2) - sente (linha 4)
escreve (linha 6) - teve (linha 8)
roda (linha 11) - corda (linha 13)
razÃ£o (linha 12) - coraÃ§Ã£o (linha 14)

Rimas Alternadas: 2
fingidor (linha 1) - dor (linha 3) / completamente (linha 2) - sente (linha 4)
roda (linha 11) - corda (linha 13) / razÃ£o (linha 12) - coraÃ§Ã£o (linha 14)

Rimas Interpoladas: 0


Sem Rima: 2
bem (linha 7)
tÃªm (linha 9)
```

4. Texto em Prosa:

```
Temos um grande texto, muito grande.Gostava que fosse pequeno mas infelizmente é grande.
Não entendo como é que utf-8 não imprime direito que tristeza.
```

Cria um ficheiro texto Resultado que depois de escrito vai conter:

```
Temos um grande texto, muito grande.Gostava que fosse pequeno mas infelizmente Ã© grande.
NÃ£o entendo como Ã© que utf-8 nÃ£o imprime direito que tristeza.

Numero de Palavras: 25

Palavra que ocorreu mais: que com 3 ocorrências.

Tipo do texto: Prosa
```

Capítulo 4

Conclusão

Este trabalho foi útil, no sentido em que nos ajudou bastante na consolidação da matéria, forçou-nos a experimentar diferentes expressões regulares e explorar alguns comandos do python. Regx101 ajudou bastante nesta componente, fez poupar bastante tempo pois é uma forma mais rápida de verificar o que as expressões regulares estão a apanhar.

Apesar de vários dos exercícios realizados poderem ter sido realizados sem o uso de expressões regulares, estas tornaram o código, em certos casos, muito mais eficiente. Por exemplo, na procura de uma certa palavra,sem estas,seríamos capazes de ter de recorrer a ciclos de procura, mas usando-as é algo mais rápido.

Acabamos com um trabalho com o que nos foi pedido, não só conseguimos realizar o exercício pedido como conseguimos fazê-lo usando as funções .split() .search() e .sub() .findall(). Além disso decidimos criar um exercício novo cujo o programa resultante tem utilidade na análise de poemas , no sentido das rimas não está completamente correto pois não há uma forma de programar rimas cujo o tom é que as faz rimar, apenas aquelas com terminações (sufixos) iguais. Em ambos conseguimos implementar as funções mais relevantes da biblioteca re que usamos várias vezes nas aulas e também fazer uso do lookbehind e lookahead.

Enfrentamos algumas dificuldades, maioritariamente na realização do relatório, latex não foi uma ferramenta que algum de nós estava muito envolvido, o que levou-nos a ter de investir algum tempo no entendimento deste, mas contamos isto como um ponto positivo pois saímos deste trabalho com um novo conhecimento.

Apêndice A

Código do Programa

A.1 Conversor

Lista-se a seguir o código completo do programa desenvolvido para criar o conversor (Capítulo 2) . Contém alguma documentação para ajudar no entendimento deste.

```
import re

inputFromUser=input(">> ")

# Fun   o que executa as opera  es;
# Recebe o nome da opera  o
# Devolve uma lista com [ 1 caso o resultado seja um valor ou 0 caso o resultado seja uma
                                                    lista , resultado da opera  o ]
def func(nome,lista):
    if nome == 'sum':
        return [1,sum(lista)]
    elif nome == 'soma':
        return [1,sum(lista)]
    elif nome == 'media':
        return [1,sum(lista)/len(lista)]
    elif nome == 'ord':
        return[0,lista.sort()]
    elif nome == 'min':
        return [1,min(lista)]
    elif nome == 'max':
        return [1,max(lista)]
    elif nome == 'moda':
        ocorr = {}
        for m in lista:
            ocorr[m] = lista.count(m)
        return [1,max(ocorr)]
    elif nome == "mediana":
        lista.sort()
        if len(lista)%2 == 1:
            return [1,lista[int(len(lista)/2)]]
        else:
            return [1,(lista[int(len(lista)/2)]+lista[(int(len(lista)/2))-1])/2]

# Fun   o que passa lista de valores e a fun   o para string
# Recebe o nome do da lista de valores, a opera   o a executar e a lista de valores (ex:
                                                    escreve(Notas,sum,[12,13,14])
# Devolve o texto formatado
```

```
def escreve(m,funcao,valores):
    if not funcao:
        return "\t\t\"" + m.group(1) + "\": " + "[" + ','.join(map(str,valores)) + "]\n"
    else:
        string = "\t\t\"" + m.group(1) + "_" + funcao.group(1) + "\": "
        resposta = func(funcao.group(1), valores)
        if resposta[0] == 1:
            string += str(resposta[1]) + "\n"
        else:
            string += "[" + ','.join(map(str, valores)) + "]\n"
        return string


if inputFromUser!="":

   #Abrir o ficheiro quando o input    do tipo:<nome_do_ficheiro> <express o regular> ou
                                                do tipo <nome_do_ficheiro> <nome_do_campo>
                                                 <valor_que_est _nesse_campo>
   if re.search(r'\s', inputFromUser):
    ficheiro=re.search(r'([^\s]*)\s',inputFromUser)
    f = open(str(ficheiro.group(1))+".csv", "r")

   #Abrir o ficheiro quando o input    do tipo:<nome_do_ficheiro>
   else:
    f = open(inputFromUser+".csv", "r")

   lista = {}
   i = 0
   verifica=[[]]
   for linha in f:

     #Substituir v rgulas dentro de chavetas por pontos de exclama  o (ex: {3,5} -> {3!
                                                          5})
     linha2 = re.sub(r'({[\d]+),([\d]+})', r'\1!\2', linha)

     #Fazer split dos dados do ficheiro .csv usando as v rgulas
     if i==0:
        lista[i] = (re.split(r',', linha2))
        i += 1
     else:
     #Quando um input    do tipo <nome_do_ficheiro> <express o regular>
      if len(re.findall(r'\s',inputFromUser))==1:
       k = re.search(r'[\s]+([^\s]*)', inputFromUser)
       if re.search(str(k.group(1)),linha2):
        lista[i] = (re.split(r',', linha2))
        i += 1
      #Quando um input    do tipo <nome_do_ficheiro> <nome_do_campo> <
                                                       valor_que_est _nesse_campo>
      elif len(re.findall(r'\s',inputFromUser))==2:
       ok = re.search(r'[\s]+([^\s]*)[\s]([^\s]*)', inputFromUser)
       if ok.group(2) and str(ok.group(1)) in lista[0] and re.search("(?<=[,])"+str(ok.
                                                        group(2))+"(?=[,])",linha2):
              verifica[0] = (re.split(r',', linha2))
              if lista[0].index(str(ok.group(1))) == verifica[0].index(str(ok.group(2))):
                  lista[i] = (re.split(r',', linha2))
                  i += 1
      #Quando o input    do tipo:<nome_do_ficheiro>
      else:
         lista[i] = (re.split(r',', linha2))
```

```
90          i += 1
91
92      f.close()
93
94      #Criar o ficheiro .json quando o input    do tipo:<nome_do_ficheiro> <express o
                                                    regular>
95      if re.search(r'\s', inputFromUser):
96          f = open(str(ficheiro.group(1)) +".json", "w")
97
98      #Criar o ficheiro quando o input    do tipo:<nome_do_ficheiro>
99      else:
100         f = open(inputFromUser+".json", "w")
101
102     texto = "[\n"
103     #Percorrer os dados guardados
104     for j in range(1, i):
105        valores = []
106        texto += "\t{\n"
107        for x in range(len(lista[j]) - 1):
108         if re.search(r'([a-zA-Z]+){([\d]+)(![\d]+)?}',lista[0][x]) :
109
110           #A vari vel m encontra um padr o que nos diz o nome da vari vel e os n meros
                                                          m nimo e m ximo de argumentos
111           m = re.search(r'([a-zA-Z]+){([\d]+)(![\d]+)?}',lista[0][x])
112           funcao = re.search(r'::([a-z]+)', lista[0][x])
113           valores.append(int(re.sub(r'([0-9]+)\n',r'\1',lista[j][x])))
114           z=x+1
115           t=0
116
117           #Se houver n mero m ximo de argumentos
118           if(m.group(3)):
119            nmax = re.sub(r'!','',m.group(3))
120            for t in range(0,int(nmax)):
121               if z<len(lista[j]) and lista[j][z]!="\n" and lista[j][z]:
122                   l=re.sub(r'([0-9]+)\n',r'\1',lista[j][z])
123                   valores.append(int(l))
124               z+=1
125
126           #Se n o houver n mero m ximo de argumentos
127           else:
128            for t in range(0, int(m.group(2))):
129               if z < len(lista[j]) and lista[j][z] != "\n" and lista[j][z]:
130                    l = re.sub(r'([0-9]+)\n', r'\1', lista[j][z])
131                    valores.append(int(l))
132               z += 1
133           texto+=escreve(m,funcao,valores)
134        elif lista[0][x] == "": pass
135        else:
136              texto += "\t\t\"" + lista[0][x] + "\": \"" + lista[j][x] + "\",\n"
137        texto += "\t},\n"
138
139     texto += "]\n"
140     f.write(texto)
```

A.2 Poema/Prosa

Lista-se a seguir o código completo do programa desenvolvido para descobrir certas características de um poema ou um texto em prosa (Capítulo 3) .

Contém alguma documentação para ajudar no entendimento deste.

```
import re

inputFromUser=input(">> ")

if inputFromUser!="":
    resultado=""
    texto=""
    f = open(inputFromUser + ".txt", "r")
    #passar de ficheiro para uma variavel com o texto
    for linha in f:
        texto+=linha
    Estrofes=len(re.findall("\n\n",texto))+1
    #Todas as palavras
    Palavras=re.findall(r'([a-zA-Zz-  ]+)',texto)  #z-   tem no seu intervalo todos os
                                                   caracteres com acento            etc
                                                   exceto o    q temos de meter
    resultado += "Numero de Palavras: " + str(len(Palavras)) + "\n\n"
    #Palavra que teve mais ocorr ncias
    palocr={}
    for palavra in Palavras:
        if palavra not in palocr:
            palocr[palavra]=len(re.findall('(?<![a-zA-Zz-  ])'+palavra+'(?![a-zA-Zz-  ])',
                                                          texto))
    maior=""
    for palavra in palocr:
        if palocr[palavra]==max(list(palocr.values())):
            maior=palavra
    resultado+="Palavra que ocorreu mais: "+maior+" com " +str(palocr[maior])+"
                                                   ocorr ncias.\n\n"
    if Estrofes>1:
        #Tipo (Poema
        resultado += "Tipo do texto: Poema\n\n"
        # Numero de estrofes
        resultado += "Numero de Estrofes: " + str(Estrofes)+"\n\n"
        #Tipo de Estrofes
        estrofes = re.split(r'\n\n', texto)
        tiposquadra=["Nada","Mon stico","D stico","Terceto","Quarteto","Quintilha","
                                                     Sextilha","Septilha"]
        l=1
        for a in estrofes:
                numero=0
                numero=len(re.split(r'\n',a))
                resultado +=str(l)+"   estrofe: "+tiposquadra[numero]+"\n\n"
                l+=1
        #Verifica se    um Soneto
        if Estrofes==4 and re.search(r'1[   :a-z]+Q',resultado) and re.search(r'2[   :a-z
                                                     ]+Q',resultado) and re.search(r'3[
                                                     :a-z]+T',resultado)and re.search(r'4[
                                                        :a-z]+T',resultado) :
            resultado+="Este Poema    um Soneto.\n\n"
        #Rimas
        linhas = re.split(r'\n', texto)
```

```
45          rimas = {}
46          array = []
47          ns = 0
48          #Cria um array com com os digitos equivalente a cada rima, para depois inserir no
                                                        inicio de cada verso
49          for i in linhas:
50              i+='\n'
51              rima = re.search(r'[a-zA-Zz-  ]*([a-zA-Zz-  ]{2})[!,.?;]*\n', i)
52              if rima and rima.group(1) not in rimas:
53                  rimas[rima.group(1)] = ns
54                  ns += 1
55                  array.append(rimas[rima.group(1)])
56              elif rima:
57                  array.append(rimas[rima.group(1)])
58          x=0
59          texto=""
60          ultima=[]
61          #Guarda a ultima palavra de cada linha, atualiza o array com todas as linhas,
                                                        colocando o \n no fim de cada linha e
                                                         o digito correspondente a cada rima.
62          #Al m disso tamb m reconstroi o poema de forma a que seja possivel imprimi-lo no
                                                         fim com os padr es das rimas
63          for t in range(0,len(linhas)):
64            if linhas[t]:
65                linhas[t]+= '\n'
66                texto += str(array[x]) + "- " + linhas[t]
67                linhas[t]= str(array[x]) + "- " + linhas[t]
68                x += 1
69                ultima.append(re.search(r' ([a-zA-Zz-  ]*)[!,.?;]*\n', linhas[t]).group(1))
70            else:
71                texto += "\n"
72                ultima.append("vazio")
73          #Tipos de Rimas
74          RimEmp=[]
75          RimEnc=[]
76          SemRima=[]
77          RimAlt=[]
78          RimInt=[]
79          numlinha=1
80          for x in range(0,len(linhas)):
81              if linhas[x]:
82                  linhax=int(re.search(r'([0-9])+\- ', linhas[x]).group(1))
83                  #Rima Emparelhada
84                  if x<len(linhas)-1 and linhas[x+1]:
85                      linhax1= int(re.search(r'([0-9])+\- ', linhas[x + 1]).group(1))
86                      if linhax==linhax1:
87                        RimEmp.append(ultima[x]+" (linha "+str(numlinha)+") - "+ultima[x+1]+
                                                                  " (linha "+str(numlinha
                                                                  +1)+")")
88                  #Rima Encadeadas
89                  if x<len(linhas)-2 and linhas[x+2] and linhas[x+1]:
90                      linhax2= int(re.search(r'([0-9])+\- ', linhas[x + 2]).group(1))
91                      if linhax==linhax2:
92                        RimEnc.append(ultima[x]+" (linha "+str(numlinha)+") - "+ultima[x+2]+
                                                                  " (linha "+str(numlinha
                                                                  +2)+")")
93                  #Rima Alternada
94                  if x<len(linhas)-3 and linhas[x+1] and linhas[x+2] and linhas[x+3]:
95                      linhax1= int(re.search(r'([0-9])+\- ', linhas[x + 1]).group(1))
```

```
                    linhax2= int(re.search(r'([0-9])+\- ', linhas[x + 2]).group(1))
                    linhax3= int(re.search(r'([0-9])+\- ', linhas[x + 3]).group(1))
                    if linhax==linhax2 and linhax!=linhax1 and linhax1==linhax3 and
                                                                linhax1!=linhax2 and
                                                                linhax2!=linhax3:
                      RimAlt.append(ultima[x] + " (linha " + str(numlinha) + ") - " +
                                                                  ultima[x + 2] + " (
                                                                  linha " + str(numlinha
                                                                  + 2)+") / "+ ultima[x+1
                                                                  ]+" (linha "+str(
                                                                  numlinha+1)+") - "+
                                                                  ultima[x+3]+" (linha "+
                                                                  str(numlinha+3)+")")
                #Rima Interpolada
                if x<len(linhas)-3 and linhas[x+1] and linhas[x+2] and linhas[x+3]:
                    linhax1= int(re.search(r'([0-9])+\- ', linhas[x + 1]).group(1))
                    linhax2= int(re.search(r'([0-9])+\- ', linhas[x + 2]).group(1))
                    linhax3= int(re.search(r'([0-9])+\- ', linhas[x + 3]).group(1))
                    if linhax==linhax3 and linhax!=linhax1 and linhax1==linhax2 and
                                                                linhax2!=linhax3:
                      RimInt.append(ultima[x] + " (linha " + str(numlinha) + ") - " +
                                                                  ultima[x + 3] + " (
                                                                  linha " + str(numlinha
                                                                  + 3)+") / "+ ultima[x+1
                                                                  ]+" (linha "+str(
                                                                  numlinha+1)+") - "+
                                                                  ultima[x+2]+" (linha "+
                                                                  str(numlinha+2)+")")
                #Sem Rima
                if not re.search(ultima[x],','.join(map(str, RimEmp))) and not re.search(
                                                                ultima[x],','.join(map(str,
                                                                RimEnc)))and not re.search(
                                                                ultima[x],','.join(map(str,
                                                                RimAlt)))and not re.search(
                                                                ultima[x],','.join(map(str,
                                                                RimInt))):
                     SemRima.append(ultima[x]+" (linha "+str(numlinha)+")")
            numlinha+=1

        #Adiciona    mensagem final os tipos de Rima
        resultado+="Tipos de Rimas: \n\nRimas Emparelhadas: "+str(len(RimEmp))+"\n"+'\n'.
                                                        join(map(str,RimEmp))+"\n\n"
        resultado+="Rimas Encadeadas: "+str(len(RimEnc))+"\n"+'\n'.join(map(str,RimEnc))+"
                                                        \n\n"
        resultado+="Rimas Alternadas: "+ str(len(RimAlt))+ "\n" + '\n'.join(map(str,
                                                        RimAlt)) + "\n\n"
        resultado+="Rimas Interpoladas: " + str(len(RimInt)) + "\n" + '\n'.join(map(str,
                                                        RimInt)) + "\n\n"
        resultado+="Sem Rima: " + str(len(SemRima)) +"\n"+ '\n'.join(map(str, SemRima)) +
                                                        "\n\n"
    else:
        resultado += "Tipo do texto: Prosa\n"
    f.close()
    f = open("Resultado.txt", "w")
    f.write(texto+"\n\n"+resultado)
```