

Processamento de Linguagens

Engenharia Informática (3º ano)

Trabalho Prático 1

4 de Março de 2022 (10h00)

Dispõe de **2 semanas** para desenvolver este trabalho, a entrega deverá ser feita a **20 de Março**.

1 Objectivos e Organização

Este trabalho prático tem como principais objectivos:

- aumentar a experiência de uso do ambiente Linux e de algumas ferramentas de apoio à programação;
- aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões em streams de texto;
- desenvolver, a partir de ER, sistemática e automaticamente, Processadores de Linguagens Regulares, que encontrem ou transformem textos com base no conceito de regras de produção Condição-Ação;
- utilizar o Python e os seus módulos `re` e `ply` para gerar os filtros de texto.

Na resolução dos trabalhos práticos desta UC, aprecia-se a imaginação/criatividade dos grupos em todo o processo de desenvolvimento!

Deve entregar a sua solução até Domingo dia 20 de Março. O ficheiro com o relatório e a solução deve ter o nome 'pl2022-tp1-grNN', em que NN corresponderá ao número de grupo. O número de grupo será atribuído por ordem de registo (será enviado um aviso sobre o processo de registo).

Cada grupo, **é livre** para escolher qual o enunciado que pretende desenvolver.

A submissão deverá ser feita por email com os seguintes dados:

to: jcr@di.uminho.pt

subject: PL2022::grNN::TP1::Enunciado

body: Colocar um ZIP com os ficheiros do TP1: relatório, código desenvolvido e datasets de teste.

Em cima, "Enunciado" é um dos valores: CSV, PPP ou EMD.

Na defesa, a realizar na semana de 21 a 25 de Março nas aulas práticas, o programa desenvolvido será apresentado aos membros da equipa docente, totalmente pronto e a funcionar (acompanhado do respectivo relatório de desenvolvimento) e será defendido por todos os elementos do grupo, em data a marcar. O relatório a elaborar, deve ser claro e, além do respectivo enunciado, da descrição do problema, das decisões que lideraram o desenho da solução e sua implementação, deverá conter exemplos de utilização (textos fontes diversos e respectivo resultado produzido). Como é de tradição, o relatório será escrito em LaTeX.

2 Enunciados

Para sistematizar o trabalho que se pede em cada uma das propostas seguintes, considere que deve, em qualquer um dos casos, realizar a seguinte lista de tarefas:

1. Especificar os padrões que quer encontrar no texto-fonte, através de ERs;
2. Identificar as acções semânticas a realizar como reacção ao reconhecimento de cada um desses padrões;
3. Identificar as Estruturas de Dados globais que possa eventualmente precisar para armazenar temporariamente a informação que vai extraindo do texto-fonte ou que vai construindo à medida que o processamento avança;
4. Desenvolver um Filtro de Texto para fazer o reconhecimento dos padrões identificados e proceder à transformação pretendida, com recurso aos módulos `re` e `ply`.

2.1 Enunciado: 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 poder 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":

```
Número, Nome, Curso
3162, Cândido Faísca, Teatro
7777, Cristiano Ronaldo, Desporto
264, Marcelo Sousa, Ciência Política
```

Corresponde à seguinte tabela:

Número	Nome	Curso
3162	Cândido Faísca	Teatro
7777	Cristiano Ronaldo	Desporto
264	Marcelo Sousa	Ciência Política

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

2.1.1 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"):

```
Número, Nome, Curso, Notas{5}, , , ,
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
```

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")

```
Número, Nome, Curso, Notas{3,5}, , , ,
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,
```

2.1.2 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"):

```
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,
```

```
Número, Nome, Curso, Notas{3,5}:::media,,,,,  
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,
```

etc.

2.1.3 Resultado esperado

O resultado final esperado é um ficheiro **JSON** resultante da conversão dum ficheiro **CSV**.

Por exemplo, o ficheiro "alunos.csv" (original), deveria ser transformado no seguinte ficheiro "alunos.json":

```
[  
  {  
    "Número": "3612",  
    "Nome": "Cândido Faísca",  
    "Curso": "Teatro"  
  },  
  {  
    "Número": "7777",  
    "Nome": "Cristiano Ronaldo",  
    "Curso": "Desporto"  
  },  
  {  
    "Número": "264",  
    "Nome": "Marcelo Sousa",  
    "Curso": "Ciência Política"  
  }  
]
```

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

```
[  
  {  
    "Número": "3612",  
    "Nome": "Cândido Faísca",  
    "Curso": "Teatro",  
    "Notas": [12,13,14,15,16]  
  },  
  {  
    "Número": "7777",  
    "Nome": "Cristiano Ronaldo",  
    "Curso": "Desporto",  
    "Notas": [17,12,20,11,12]  
  },  
  {  
    "Número": "264",  
    "Nome": "Marcelo Sousa",  
    "Curso": "Ciência Política",  
    "Notas": [18,19,19,20,18]  
  }  
]
```

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"):

```
[  
  {  
    "Número": "3612",  
    "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
  }
]

```

2.2 Enunciado: Pré-processador para LaTeX ou HTML

Desenvolver um documento em LaTeX ou mesmo em HTML é uma actividade inteligente e intelectualmente interessante enquanto estruturante das ideias e sistematizante dos processos. Porém o acto de editar o respectivo documento é por vezes fastidioso devido ao peso das marcas (as tags) que têm de ser inseridas para anotar o texto com indicações de forma, conteúdo ou formato. Por isso apareceram editores sensíveis ao contexto que sabendo que se está a escrever um documento LaTeX ou HTML nos facilitam a vida inserindo as ditas marcas, ou anotações.

Uma alternativa mais simples mas também muito frequente é permitir o uso de anotações mais leves e simples (até de preferência independentes do tipo de documento final) e de pois recorrer ao pré-processamento para substituir essa notação ligeira, abreviada, pelas marcas finais correctas. Este é o caso do conhecido PPP (documento descritivo em anexo), desenvolvido há alguns anos por José Carlos Ramalho, ou mesmo da mais actual e bem conhecida linguagem Markdown para construção de páginas HTML.

O que se lhe pede neste trabalho é que, depois de investigar o PPP e a linguagem Markdown, ou outro análogo, especifique uma sua linguagem de anotação para abreviar a escrita de:

- formatação: negrito, itálico, sublinhado;
- vários níveis de títulos;
- listas de tópicos (items) não-numerados, numerados ou tipo entradas de um dicionário ;
- inclusão de imagens;
- inclusão e formatação de tabelas;
- todos os outros que achar necessário ou a sua imaginação vislumbrar.

Deve, depois e recorrendo à ferramenta Flex, criar um processador que transforme a sua notação em LATEX ou HTML. Preveja o aninhamento e a combinação dos componentes enumerados.

2.3 Enunciado: Processador de Registos de Exames Médicos Desportivos

Neste exercício pretende-se trabalhar com um dataset gerado no âmbito do registo de exames médicos desportivos.

Construa, então, um ou vários programas Python para processar o dataset "emd.csv" e produzir o solicitado nas alíneas seguintes:

- Criar um website com as seguintes características:
 1. Página principal: de nome "index.html", contendo os seguintes indicadores estatísticos:
 - (a) Datas externas dos registos no dataset;
 - (b) Distribuição por género em cada ano e no total;
 - (c) Distribuição por modalidade em cada ano e no total;
 - (d) Distribuição por idade e género (para a idade, considera apenas 2 escalões: < 35 anos e >= 35);
 - (e) Distribuição por morada;
 - (f) Distribuição por estatuto de federado em cada ano;
 - (g) Percentagem de aptos e não aptos por ano.

2. Página do indicador: clicando no indicador na página principal, devemos saltar para a página do indicador onde temos a informação que permitiu obter esse indicador. Por exemplo, para a distribuição por morada, a página deverá apresentar uma lista de moradas, ordenada alfabeticamente e para cada morada deverá apresentar uma sublista de registos, ordenada alfabeticamente por nome de atleta (com os dados: nome do atleta, modalidade).

Bom trabalho e boa sorte

A equipe docente