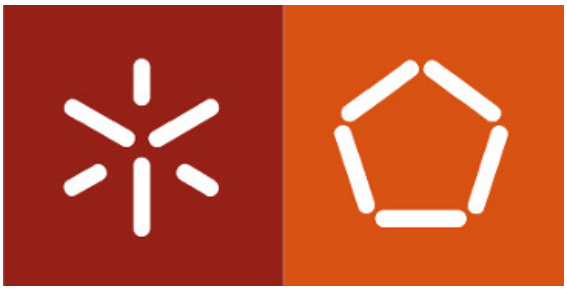

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Processamento de Linguagens e Conhecimento

Scripting no Processamento de Linguagem Natural

Trabalho Prático 1

Etienne Costa, A76089
Gonçalo Pinto, A83732
Luís Ribeiro, A85954

4 de abril de 2021

Conteúdo

Capítulo 1

Introdução

No 2º semestre do 1º ano do Mestrado em Engenharia Informática da Universidade do Minho, existe uma unidade curricular enquadrada no perfil de Processamento de Linguagens e Conhecimento denominada por Scripting no Processamento de Linguagem Natural, que tem como objetivo a agilizar a capacidade dos alunos no processo de automatização de uma série de tarefas ligadas a ficheiros, textos e informação em linguagem natural, desenvolver a aptidão em lidar com grandes quantidades de informação e potenciar a aprendizagem de uma programação documental heterogénea.

O presente trabalho enquadra-se nesta unidade curricular e pretende desenvolver competências aos alunos na escrita de *scripts* para automatização de uma variedade de tarefas e transformações, na capacidade de resolver problemas usando transformações via expressões regulares, na compreensão das vantagens e o funcionamento de sistemas guiados por regras de produção (condição-reação), na construção de DSLs concretas, dicionários eletrónicos e corpora, bem como, o seu uso e extração de informação diversa a partir desta última. Pretende ainda que os alunos desenvolvam a capacidade de construir pequenos protótipos para modelar linguagem natural.

Neste trabalho pretendia-se que cada grupo realizasse um trabalho de análise, planeamento e de implementação de *scripts* em **Python** que permitisse extrair o máximo de informação de um *Website* sobre uma árvore genealógica de uma determinada família, gerando como resultado final a informação encontrada mas no formato adequado à sua interpretação e de forma acessível.

Capítulo 2

Contextualização

Neste trabalho foi nos facultado um *Website* [1] com a informação da Família Antunes Guimarães (Briteiros). Para realizar a análise, planeamento e implementação foi necessário efetuar um levantamento de características e estudo das entidades envolvidas, com o objetivo de perceber as particularidades entre cada entidade a tratar. Seguidamente apresentamos algumas características:

- **Pessoas**, existem no total 146 pessoas. Podemos visualizar uma lista de nomes onde cada nome encaminha para a página pessoal de cada pessoa, onde nesta podemos encontrar:
  - Nome: sendo um elemento obrigatório;
  - Nascimento: pode ser opcional e caso exista é denotado com o símbolo *;
    - Local: pode ser opcional;
    - Data: pode ser opcional;
  - Falecimento: pode ser opcional e caso exista é denotado com o símbolo +;
    - Local: pode ser opcional;
    - Data: pode ser opcional;
  - Pais: pode ser opcional e caso exista possui os seguintes atributos:

    Pai: nome, data de nascimento e o seu identificador pois é um *link*;

    Mãe: nome, data de nascimento e o seu identificador pois é um *link*;
  - Casamentos: pode ser opcional e caso exista pode existir mais que um casamento associado à mesma pessoa, mas cada casamento possui as seguintes informações:
    - Local: pode ser opcional;
    - Data: pode ser opcional;
    - Parceiro: nome, data de nascimento e o seu identificador pois é um *link*;
  - Filhos: pode ser opcional e caso existam é apresentado uma lista de filhos e caso a pessoa tenha casado mais que uma vez pode aparecer divido por casamentos, nesta lista podemos encontrar:

    * Filho: nome, data de nascimento, caso tenha casado o nome do parceiro(a) e o seu identificador pois é um *link*;
  - Notas: pode ser opcional e caso exista é apresentado uma lista de notas.
- **Famílias**, existem no total 8 famílias. Podemos visualizar uma lista de nomes, onde cada nome encaminha para a uma página com acesso aos índices de nomes de pessoas. Cada índice, associado a um nome de uma pessoa, é um link para a página da mesma.

Capítulo 3

Desenvolvimento

3.1 Extração das Pessoas do Website

Um dos *scripts* desenvolvidos extrai todas as pessoas e a informação disponibilizada de cada uma, devolvendo um dataset de pessoas em JSON. Contudo, existe a possibilidade de indicar que o dataset de pessoas seja em YAML, indicando-se apenas, na execução do *script*, a *tag yaml* respetiva. Assim, o grupo dividiu a construção deste *script* em três fases.

3.1.1 Obtenção dos identificadores de cada pessoa

Tal como foi apresentado no *Website*, é possível encontrar uma lista de nomes com a respetiva ligação à página de cada pessoa. Esta lista encontra-se dividia por paginação, isto é, apenas são apresentados 30 elementos de cada vez, assim iterou-se cerca de 5 vezes acedendo ao seguinte link:

```
for page in range(0, 150, 30):
    url = 'http://pagfam.geneall.net/3418/pessoas_search.php?start='
        + str(page) + '&orderby=&sort=&idx=0&search='
    ...
```

De forma a obter a lista de pessoas foi necessário recorrer ao *package requests* [2] que efetua um pedido HTTP ao Website. Posteriormente, com o *package bs4*, mais especificamente com o *BeautifulSoup* [3] foi possível fazer o *parsing* do HTML da resposta obtida. De seguida, com o auxílio deste último *package*, foi obtido uma lista de listas de todas as ligações através da *tag 'a'*, percorrendo cada sublista foi aplicada uma expressão regular. Para tal recorreu-se ao seguinte *package jjcli* [4] no sentido de encontrar o identificador em cada link, construindo assim uma lista de identificadores.

```
for page in range(0, 150, 30):
    ...
    list_hyperlinks.append(soup.find_all('a')[3:len(soup.find_all('a')) - 4])

for sublist in list_hyperlinks:
    for hyperlink in sublist:
        list_ids.append(findall(r'\?id=[0-9]+\s*\">', str(hyperlink))[0][4:11])
```

3.1.2 Construção do elemento pessoa no dataset

Após obtida uma lista de identificadores esta foi percorrida, onde para cada elemento foi efetuado um pedido HTTP igual ao que se fez previamente, mas neste o *link* acedido é o que permite aceder à página pessoal de cada elemento.

```
link = 'http://pagfam.geneall.net/3418/pessoas.php?id=' + str(id_page)
```

Seguidamente foi efetuado o mesmo procedimento de *parsing* do HTML com o mesmo *package*. Como já foi referido, uma página pessoal encontra-se dividida por *tags* podendo ser estas Pais, Casamentos, Filhos e Notas. De forma a dividir esta informação, cada página recorre a separadores do seguinte tipo *class="marcadorP"*. Assim, foi aplicado uma expressão regular que permite construir uma lista onde no primeiro elemento está toda a informação até à primeira ocorrência deste separador e nos restantes elementos a informação encontra-se separada por *class="marcadorP"*.

```
p1, *sec_cont = split(r'<div\s+class="marcadorP".*?>(.*?)</div>', pg)
```

No primeiro elemento da lista, identificado por *p1*, podemos encontrar a informação sobre o Nome, Nascimento e Falecimento, sendo necessário aplicar algumas expressões regulares no sentido de extrair esta informação. Para os restantes elementos da lista foi percorrido cada elemento, ou seja, cada elemento representa uma *tag* das acima referidas. Tiramos partido dos vários quantificadores e filtros padrão que as expressões regulares oferecem, aplicando uma expressão regular geral com o objetivo de cobrir grande parte das particularidades encontradas. Assim, foi possível construir listas com vários elementos que representam a informação pretendida.

```
for sec in sec_cont:
    if sec in tags:
        ...
    else:
        familiares = findall(
            r'((<div.*?class="txt2".*?>\n)?((<div|<ul|<DIV).*?>)\s*
            (<b>(.*?)</b>)?\s*([^\<]+)?\s*(<nobr>(.*?)</nobr>)?.*?)?
            <a\s+href=(.*?id=(\d+)"?)>(.*?)</a>', sec)
        ...
        for ignore, yet_Another_ignore, ignore_div, ignore_div,ignore_tag,
            tag, localC, ignore_data, dataC, url, find_id, nome in familiares:
            ...
```

Dada a natureza do `findall` e dos filtros padrão das expressões regulares, algumas *tags* são ignoradas, pois servem apenas como auxiliares na identificação das particularidades acima referidas.
Como é apresentado, cada elemento da variável *familiares* possui a informação respetiva de cada *tag*. Tendo em conta sempre o separador que estamos a processar, é possível conhecer a informação de cada elemento.

3.1.3 Construção do dataset de pessoas

No sentido de armazenar a informação de cada pessoa, o grupo decidiu recorrer às estruturas de dados denominadas **dicionários**, que a linguagem utilizada disponibiliza. Cada pessoa vai ser representada por um dicionário, sendo este composto por mais dicionários, bem como, uma lista de filhos. Na chave de cada dicionário representou-se o elemento principal a tratar (Nascimento, Falecimento, Pais, Casamentos, Filhos e Notas) e no valor respetivo um dicionário, exceto no caso dos filhos onde o valor é uma lista que pode ser dividida em dicionários se os filhos estiverem divididos por casamentos. Cada dicionário que seja um valor, possui uma chave para a informação considerada relevante a guardar, e no valor a informação que se extraiu após o procedimento acima referido. De seguida apresentamos um esboço da estrutura de uma pessoa.

```
{
    Casamentos: [{data: _, idParceiro: _, local: _, parceiro: _}],
    Falecimento: {data: _, local: _},
    Filhos: [{id: _, nome: _}],
    Nascimento: {data: _, local: _},
    Nome: _,
    Pais: [{Pai: {id: _, nome: _}}, {Mãe: {id: _, nome: _}}],
    id: _
}
```

Após construído cada dicionário sobre a informação de uma pessoa, este é adicionado a uma lista. Terminada a iteração sobre todas as pessoas, é construído um novo dicionário onde a chave é a palavra 'Pessoas' e no valor a lista construída. Decidiu-se utilizar esta abordagem pois torna-se útil para um dos *scripts* construídos.

Como um dicionário em Python é algo muito específico, o grupo achou por bem transformar em algo que fosse agradável para o utilizador. Assim, com a utilização do *package json* [5] é retornado um *dataset* em formato JSON, por ser um formato compacto, de padrão aberto independente, de troca de dados simples e rápida (parsing) entre sistemas. Tendo em conta algumas preferências dos utilizadores, o grupo acrescentou a possibilidade que, aquando o utilizador executa o *script* pode indicar um formato alternativo. Nesta fase apenas foi acrescentado a possibilidade de ser em YAML pois é uma linguagem de marcação rápida em vez de XML para isso recorreu-se ao *package yaml* [6].

3.2 Extração das Famílias do Website

Outro dos *scripts* construídos tem como função extrair todas as famílias e as pessoas que fazem parte destas, devolvendo um dataset de famílias em JSON. Para isso o grupo dividiu a construção deste *script* em três fases.

3.2.1 Obtenção dos identificadores de cada família

Como foi apresentado previamente, existe uma página disponível que apresenta uma lista de nomes de cada família onde cada nome é uma ligação à página desta. Assim, efetou-se um pedido HTTP de forma a obter essa página HTML extraindo as ligações com *tag* 'a' foi construído uma lista de ligações, onde cada elemento pode ou não possuir o identificador. O algoritmo executado para esta tarefa encontra-se descrito abaixo:

```
pedido = reqs.get('http://pagfam.geneall.net/3418/familias_search.php').text
list_hyperlinks = BeautifulSoup(pedido, 'html.parser').find_all('a')[3:]
for hyperlink in list_hyperlinks :
        list_familias.append({"id": findall(r'\?id=[0-9]+\s*\">',
            str(hyperlink))[0][4:8], "nome": hyperlink.text})
```

3.2.2 Obtenção dos nomes de cada pessoa de uma família

Posteriormente acedeu-se à página que disponibiliza todos os nomes das pessoas de cada família, onde nessa lista apenas se achou relevante retirar o nome da pessoa e o seu identificador. Assim, aplicando algumas expressões de forma a manipular os links que são apresentados, foi possível retirar essa informação. O procedimento executado é seguidamente apresentado.

```
for familia in list_familias:
    ...
    pedido = reqs.get('http://pagfam.geneall.net/3418/fam_names.php?id=' + familia["id"]).text
    list_pessoas = BeautifulSoup(pedido, 'html.parser').find_all('td', {'class': 'txt2'})
    for linha in list_pessoas:
        if len(linha.find_all('a')) > 0:
            for elem in linha.find_all('a', href=True):
                individuo = {
                    "id": findall(r'\?id=[0-9]+', str(elem))[0][4:],
                    "nome": elem.text
                }
                ...
    ...
```

3.2.3 Construção do dataset de famílias

No sentido de construir um *dataset* das famílias, o grupo seguiu a mesma abordagem apresentada no capítulo 3.1, baseado num dicionário em Python e posteriormente recorrendo ao *package json* para transformar num formato de fácil de leitura. Este *dataset* é composto por um objeto cuja chave mapeia uma lista de objetos que representa cada família; cada objeto é composto por um identificador da família, nome da família e uma lista de indivíduos onde cada indivíduo é composto pelo seu identificador e nome.

3.3 Extração de informação de uma pessoa do Website

O terceiro *script* criado por nós extrai a informação de uma determinada pessoa, devolvendo a informação em YAML, análogo a um dos exemplos que o professor disponibilizou. Tendo em conta que previamente já foram desenvolvidos dois *scripts*, onde um permite a construção de um dicionário de pessoas, decidiu-se aproveitar o trabalho realizado no mesmo (*script* apresentado em 3.1).

Desta forma foram obtidos os identificadores de cada pessoa. Construiu-se os elementos que representam pessoas, colocando cada um num dicionário. A este *script*, foi acrescentado mais uma fase que foi converter a informação presente no dicionário em YAML. Como esperado, apenas foi possível efetuar este *script* pois existe o *package yaml* que permite facilmente efetuar a conversão.

Este *script* apenas faz sentido ser executado se o utilizador souber e indicar um identificador válido e existente.

3.3.1 Conversão da informação de uma pessoa

Numa primeira fase é percorrido o *dataset* no sentido de encontrar o identificador que o utilizador indicou. Caso exista, é verificado se o valor associado aos campos considerados têm informação, se possuírem é construído uma nova estrutura com a informação a retirar e acrescentada a uma lista de informações sobre a pessoa em questão. Terminada toda a extração é construído um documento YAML com a seguinte estrutura:

1. *'I' + identificador da pessoa*: nome, informação do nascimento e falecimento, referências aos seus casamentos, pais e notas;
2. *'Fp' + identificador do pai e mãe*: nome do pai, nome da mãe e filhos com referência à pessoa;
3. *'F' + identificador da pessoa e parceiro*: nome do parceiro(a), a lista de nomes dos filhos da pessoa e referência à pessoa;
4. *'D' + identificador incremental e identificador da pessoa*: possui um campo de texto sobre as notas que a pessoa possui;

Desta forma é apresentado a informação de uma maneira acessível e fácil de interpretar sobre uma determinada pessoa num formato muito comum.

3.4 Obter informação do Dataset construído

Por fim, o último *script* construído permite efetuar *queries* ao *dataset* de pessoas criado previamente em 3.1, com auxílio da *framework json-server* [7], que foi apresentada na unidade curricular de Processamento e Representação de Informação que consta do mesmo perfil da presente unidade curricular. Esta *framework* permite construir um servidor baseado num ficheiro JSON com um formato específico, permitindo ainda interagir com este, através de pedidos numa determinada porta, obtendo respostas no formato JSON. Assim, o *script* criado interage com o utilizador de forma local (uso das funcionalidades do *package sys*), [8] onde é efetuado pedidos ao servidor construído, seguido do retorno da respetiva resposta.

Algumas operações disponibilizadas:

- Procurar por nome;
- Procurar por identificador;
- Procurar por Nascimento ou Falecimento;
- Procurar em todos os campos;
- Sair do programa;

Ao executar uma determinada operação é criado um histórico com o *package readline* [9], assim o utilizador não necessita de escrever novamente uma determinada *query*.

Caso a procura seja no sentido de obter um identificador ou um nome, a procura é direta, isto é, o valor que foi introduzido é procurado no campo devido do *dataset*. Caso seja por Nascimento ou Falecimento, é verificado se o valor introduzido possui números (pesquisa sobre a data) ou se é uma palavra (pesquisa sobre um determinado local) com *package re* [10]. Por fim, uma pesquisa global verifica-se se o valor introduzido existe no *dataset* ou não.

Capítulo 4

Conclusão

O presente relatório descreveu, de forma sucinta, o trabalho de análise, planeamento e de implementação de *scripts* em **Python** que permitisse extrair o máximo de informação de um *Website* sobre uma árvore genealógica de uma determinada família, gerando como resultado final a informação encontrada mas no formato adequado à sua interpretação e de forma acessível.

Durante a elaboração deste trabalho surgiram algumas dificuldades que, com o esforço e entusiasmo do grupo pelo resultado, acabaram por ser ultrapassadas. Entre as quais destacam-se o desafio que foi cobrir todas as particularidades que as pessoas desta família apresentam.

Numa perspetiva futura, considera-se que seria interessante a expansão do *script* de extração de informação de uma pessoa do Website para algo mais completo, isto é, em vez de se apresentar apenas a informação de uma pessoa, apresentar as pessoas com que essa pessoa se relaciona, contudo o grupo tentou implementar essa solução, no entanto obteve problemas com o quantidade de recursividade que a linguagem utilizada permite.

Após a realização deste trabalho, o grupo ficou consciente da importância de automatização de uma série de tarefas no processamento de grandes quantidades de informação.

Consideramos que os objetivos propostos com a realização deste trabalho foram cumpridos, bem como, a consolidação dos conhecimentos em *scripting* no processamento de linguagem natural.

Por fim, o grupo espera que os conhecimentos obtidos e consolidados sejam de enorme utilidade tendo em conta uma perspetiva futura.

Bibliografia

[1] *Família Antunes Guimarães (Briteiros)*. 2004. URL: http://pagfam.geneall.net/3418/ (ver p. 2).

[2] *Requests: HTTP for Humans*. 2021. URL: https://docs.python-requests.org/en/master/ (ver p. 3).

[3] *Beautiful Soup Documentation*. 2020. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (ver p. 3).

[4] *Python module for command-line filter*. 2020. URL: https://pypi.org/project/jjcli/ (ver p. 3).

[5] *JSON encoder and decoder*. 2021. URL: https://docs.python.org/3/library/json.html (ver p. 5).

[6] *PyYAML Documentation*. 2021. URL: https://pyyaml.org/wiki/PyYAMLDocumentation (ver p. 5).

[7] *JSON Server*. 2020. URL: https://www.npmjs.com/package/json-server (ver p. 8).

[8] *System-specific parameters and functions*. 2021. URL: https://docs.python.org/3/library/sys.html (ver p. 8).

[9] *GNU readline interface*. 2021. URL: https://docs.python.org/3/library/readline.html (ver p. 8).

[10] *Regular expression operations*. 2021. URL: https://docs.python.org/3/library/re.html (ver p. 8).