Programação para a Web Semântica

Prof. Ivan Luiz Marques Ricarte
ricarte@unicamp.br

FT-Unicamp

27 de novembro de 2019

Sumário

Capítulo 1

Introdução

Apresentamos neste texto um sucinto resumo dos aspectos abordados na disciplina TT001 — Tópicos em Computação e Informática I que, na oferta realizada no segundo semestre de 2019, abordou o tema *Programação para a Web Semântica*. Organizamos o conteúdo abordado nesse semestre em três grandes blocos:

1. Obtendo informação da Web;
2. Organizando a informação para a Web Semântica; e
3. Utilizando a informação na Web Semântica.

No primeiro bloco, focamos em obter dados diretamente de servidores Web, seja extraindo dados de documentos Web, seja utilizando um serviço Web que já fornece dados em resposta a requisições, seja obtendo arquivos com conjuntos de dados desde um servidor Web.

No bloco sobre a organização da informação para a Web Semântica, o foco será na representação de dados em formato mais apropriado para o processamento do que para a visualização.

Por fim, abordamos os aspectos relacionados à integração de dados nesse formato, juntamente com os princípios da organização de dados abertos conectados *(linked open data)*.

Desenvolvemos os exemplos de programação usando a linguagem Python, na versão 3.7 [13], com apoio do ambiente de desenvolvimento PyCharm Community Edition [9]. Os exemplos de código apresentados buscam ser minimalistas, com foco central no aspecto sendo discutido e não na robustez ou qualidade, e buscam ficar dentro dos limites das margens do texto, motivo pelo qual muitas vezes são utilizadas variáveis que poderiam facilmente ser eliminadas em um código real. Mesmo assim, são exemplos operacionais e executáveis.

Capítulo 2

Obtendo informação da Web

Neste capítulo abordamos os aspectos relacionados à obtenção de dados na World Wide Web "tradicional", ou seja, na qual a informação não está organizada segundo os princípios da Web Semântica.

2.1 A World Wide Web

A World Wide Web adota uma arquitetura cliente-servidor, na qual tipicamente o cliente é um navegador *(browser)* Web e o servidor é um processo que executa em uma máquina cuja lozalização é identificada por um localizador uniforme de recursos (URL, de *Uniform Resource Locator*) [22]. O URL identifica não apenas a máquina e o processo do servidor, mas também qual recurso é solicitado desse servidor.

Numa interação típica entre um cliente e um servidor Web, o cliente (navegador) emite uma solicitação com o URL do recurso solicitado, o que pode ser feito pela digitação do URL na caixa de endereço do navegador ou pela seleção de uma hiperligação num arquivo sendo exibido. Essa solicitação segue o padrão definido no protocolo de transferência de hipertexto (HTTP, de *Hypertext Transfer Protocol*) [8]. O retorno dessa solicitação é composto por um cabeçalho, com informação sobre o sucesso (ou falha) no atendimento da solicitação e por um

corpo com o conteúdo do recurso solicitado, tipicamente (mas não restrita a) uma página codificada na linguagem de marcação de hipertexto (HTML, de *Hypertext Markup Language*) [23].

No contexto desta disciplina, no entanto, o cliente será sempre uma aplicação que desenvolveremos com o objetivo de obter, organizar e utilizar dados disponíveis na Web. Um dos pacotes básicos de Python para realizar uma interação HTTP e assim desenvolver um cliente Web é o pacote `requests` [16], que não faz parte da biblioteca padrão de Python e deve ser instalado pelo usuário. Apesar de haver algumas funcionalidades na biblioteca padrão para realizar interações HTTP, esse pacote simplifica a programação para esse tipo de necessidade.

O exemplo a seguir ilustra algumas das facilidades oferecidas no uso do pacote `requests`:

```
1  import requests
2
3  url = 'http://www.ft.unicamp.br/~ricarte/'
4  try:
5      response = requests.get(url)
6      print('Status: ', response.status_code)
7      rspenc = response.encoding
8      appenc = response.apparent_encoding
9      print('Encoding: %s (%s)' % (rspenc, appenc))
10     print('Bytes retornados: ', len(response.content))
11     contform = response.text.splitlines()
12     print('Linhas: ',len(contform))
13     for line in contform[1:20]:
14         print(line)
15 except Exception as e:
16     print(f'\nErro ao processar {url}:\n {e}')
```

O uso do pacote `requests` está sinalizado na linha 1 do código, que define, na linha 3, o endereço URL do recurso que será solicitado de um servidor Web. Na linha 5 é feita a solicitação do tipo GET, uma das alternativas em HTTP que

é usada para obter o conteúdo de um recurso, e o resultado está armazenado na variável `response`. Todas as informações retornadas pelo servidor são armazenadas nessa variável, como o código de status da resposta (por exemplo, 200 para okay ou 404 para recurso não encontrado) na linha 6, a codificação de caracteres declarada e aparente (linhas 7 e 8, respectivamente), o conteúdo do recurso em um arranjo de bytes (linha 10) e em formato texto (linha 11). Nesse exemplo, o conteúdo textual é dividido em linhas e as primeiras 20 linhas são apresentadas no console. As primeiras linhas do resultado da execução desse código são:

```
Status:  200
Encoding: UTF-8 (utf-8)
Bytes retornados:  24232
Linhas:  287
<html lang="pt-BR">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width">
<title>Início - Prof. Ivan Ricarte</title>
<link rel="profile" href="http://gmpg.org/xfn/11">
```

Nessa resposta, as quatro primeiras linhas (Status, Encoding, Bytes retornados e Linhas) apresentam os valores das variáveis extraídas da variável `response` e, a partir da tag `<html>`, o conteúdo do recurso (codificado em HTML) é apresentado.

2.2 'Raspando' a Web

A Web certamente tem muitos dados disponíveis, mas a maior parte desse conteúdo está em HTML. O problema com isso é que HTML, apesar de ser uma linguagem estruturada, está voltada principalmente para a apresentação de documentos. Assim, embora o consumo (visualização) por humanos seja favorecido, o consumo por agentes (código) é dificultado.

A extração de dados contidos em páginas HTML é uma tarefa comum, sendo usualmente denominada *Web scraping* — que, em tradução livre, chamamos de 'raspar a Web'. Embora essa extração possa ser realizada diretamente sobre o conteúdo da página, usando operações de strings e eventualmente expressões regulares, essa atividade pode ser facilitada ao explorar a estrutura do documento para localizar os dados desejados.

O pacote `BeautifulSoup` [17] oferece recursos exatamente para facilitar a programação em Python dessa atividade de extração de dados de páginas HTML. No exemplo a seguir, as funcionalidades desse pacote são utilizadas para localizar e extrair a informação sobre a ementa de uma disciplina a partir da página Web da Diretoria Acadêmica (DAC) da Unicamp.

```
import requests
from bs4 import BeautifulSoup
discipline = 'TT001'
dac = 'http://www.dac.unicamp.br/sistemas/'
urlbase = dac + 'catalogos/grad/catalogo2019/'
urlIndex = urlbase + 'disciplinas/tipo' +
    discipline[0:2] + '.html'
try:
  pageIndex = requests.get(urlIndex)
  soupIndex = BeautifulSoup(pageIndex.content, 'lxml')
  allDisc = soupIndex.find_all('a')
  oneDisc = [x for x in allDisc if discipline in x]
  elemA = oneDisc[0]
  urlDisc = urlbase + elemA['href'][3:]
  pageDisc = requests.get(urlDisc)
  soupDisc = BeautifulSoup(pageDisc.content, 'lxml')
  discTag = soupDisc.find(attrs={'name': discipline})
  discTagPar = discTag.parent
  discVetor = discTagPar.next_sibling.next_sibling
  ementa = repr(discVetor.next_sibling.find('p').
```

```
21                  next_element)[1:-1]
22    print(discTag.string)
23    print('Ementa: ' + ementa)
24  except:
25    print('Erro ao obter ementa de',discipline)
```

Nesse código, duas páginas HTML são processadas em busca da informação desejada. A primeira página é um índice de disciplinas com o dado prefixo (no caso, 'TT'), que mantém uma lista de ligações (elementos `A` com atributo `href`) para a página que contém as descrições dos detalhes das disciplinas, como em:

```
...
<div class='texto'>
<center><table BORDER=0 CELLSPACING=0 CELLPADDING=0
                COLS=1 WIDTH='590'>
<div class='div10b'><a href=
   '../coordenadorias/0064/0064.html#TT001'>TT001</a></div>
<div class='div10b'><a href=
   '../coordenadorias/0064/0064.html#TT002'>TT002</a></div>
<div class='div10b'><a href=
   '../coordenadorias/0064/0064.html#TT003'>TT003</a></div>
...
```

O endereço da primeira página é definido, a partir da sigla da disciplina (linha 3), entre as linhas 4 e 7. Essa página é obtida na linha 9, usando o método `get` de `requests`. Na linha seguinte, o conteúdo HTML dessa página é processado pelo construtor do objeto de tipo `BeautifulSoup` usando o analisador `lxml` (que também precisa ser instalado, embora haja outros analisadores pré-instalados disponíveis). A partir dessa operação, uma representação interna da estrutura do documento é criada e mantida na variável `soupIndex`. Na linha 11, essa variável é utilizada para obter todos os elementos âncora (com rótulo HTML `A`) nessa estrutura, um dos quais (linhas 12 e 13) referencia a disciplina de interesse. Desse elemento selecionado buscamos o endereço URL do destino, no

seu atributo `href` (linha 14), substituindo a referência ao diretório pai (`..`) pelo endereço base do catálogo de disciplinas.

A segunda página contém a informação com os detalhes das disciplinas, como em:

```
...
<div class="ancora">
  <a name="TT001">TT001 - Tópicos em Computação
       e Informática I</a>
</div>
<p>
OF:S-6 T:002 P:000 L:000 O:000 D:000 HS:002
      SL:002 C:002 AV:N EX:S FM:75%
<br><strong>Pré-Req.: </strong> AA425
<div class="justificado">
<strong>Ementa: </strong>
<p>Estudo de temas [...] </p>
<br />
</div>
...
```

O conteúdo dessa página que contém a informação desejada é transferido na linha 15 e, novamente, a estrutura do documento é analisada na construção de um objeto na linha 16. Nessa estrutura, o elemento que inicia a descrição da disciplina (elemento `A` com atributo `name` igual à disciplina buscada) é localizado (linha 17). No entanto, a informação desejada (a ementa) não faz parte desse elemento, mas está no bloco de elementos seguinte. Para chegar até lá, utilizamos os métodos de navegação na estrutura (`parent`, `next_sibling`) entre as linhas 18 e 21 para, finalmente, buscar o conteúdo desejado no elemento `P` com o texto da ementa. São apresentados no console então o nome da disciplina (o conteúdo do elemento `A` com atributo `name`), na linha 22, e a ementa, na linha 23. A execução desse código resulta em:

```
TT001 - Tópicos em Computação e Informática I
```

```
Ementa: Estudo de temas relevantes em Computação e
Informática. A ementa desta disciplina será definida
por ocasião de seu oferecimento.
```

Já se alterarmos o código da disciplina na linha 3 para, por exemplo, SI200, outras páginas serão processadas, mas o resultado da execução será similar:

```
SI200 - Algoritmos e Programação de Computadores II
Ementa: Recursividade. Manipulação de arquivos.
Bibliotecas estáticas e dinâmicas.
Desenvolvimento de programas.
```

Para entender mais sobre a navegação em documentos estruturados, recomendamos ver a especificação XPath [21], que apresenta a linguagem usualmente utilizada para esse fim. Na verdade, o analisador `lxml`, por si, permite também a navegação em documentos usando essa linguagem por meio do método `xpath`, que pode ser usado como uma alternativa ao `BeautifulSoup`.

2.3 Dados de Serviços Web

Embora a maior parte dos conteúdos na Web seja disponibilizada em páginas HTML, outros modos de interação têm sido oferecidos para a obtenção de dados a partir de servidores. Um serviço Web é um desses modos.

Tipicamente, um serviço Web é associado a um endereço URL ao qual podem ser associados alguns parâmetros. Em alguns casos, o provedor do serviço solicita que o usuário seja registrado e, para utilizar o serviço, um dos parâmetros necessários será a chave pessoal *(key)* de acesso. O retorno pode ser um valor simples ou um objeto estruturado, tipicamente em XML (*eXtensible Markup Language*, a linguagem genérica para criação de linguagens de documentos estruturados) ou JSON (*JavaScript Object Notation*) [6].

Um exemplo de um serviço Web livre (sem necessidade de chave) é o serviço que retorna posição da Estação Espacial Internacional, oferecido no endereço `http://api.open-notify.org/iss-now.json`. Como o próprio

nome sugere, o objeto retornado está no formato JSON. Uma invocação direta do serviço a partir de um navegador Web apresentará uma página de texto com os dados solicitados nesse formato, como em:

```
{
 "timestamp": 1573230295,
 "message": "success",
 "iss_position": {
     "longitude": "-166.0736",
     "latitude": "46.6736"
     }
}
```

Em Python, a estrutura de dados mais apropriada para representar uma informação nesse formato é o *dict* (dicionário), que mapeia chaves (os nomes das propriedades) a valores. O pacote `requests` já tem um método `json` que realiza essa conversão. Por exemplo, o código a seguir utiliza esse método para obter apenas a latitude e a longitude da Estação Espacial:

```
1 import requests
2 pos=requests.get("http://api.open-notify.org/iss-now.json")
3 dados=pos.json()
4 print('lat:', dados['iss_position']['latitude'])
5 print('lon:', dados['iss_position']['longitude'])
```

Nesse exemplo, o objeto JSON é obtido do serviço Web (linha 2) e transformado em um dicionário, associado à variável `dados`, com o método `json` (linha 3). A informação desejada está na propriedade `iss_position` dessa variável, sendo que nessa propriedade há dois dados, `latitude` e `longitude`, que são obtidos e apresentados nas linhas 4 e 5, respectivamente. Uma execução desse código retorna algo como:

```
lat: 17.7410
lon: -126.6090
```

2.4 Conjuntos de dados abertos

Outra maneira usual de obter dados na Web é quando um provedor oferece um conjunto completo de dados como um recurso, associado a um endereço URL. Esse modo de disponibilização de dados está associado ao conceito de dados abertos, sendo que há muitos provedores de dados abertos na Web, tais como:

Portal brasileiro de dados abertos `http://dados.gov.br/`

Governo dos Estados Unidos `https://www.data.gov/`

Organização Mundial da Saúde `https://www.who.int/gho/database/en/`

Banco Mundial `https://data.worldbank.org/`

DataHub `https://datahub.io/collections`

Como seria de se esperar, há uma diversidade muito grande de formatos nos quais esses dados são disponibilizados e, consequentemente, níveis variados de facilidade de extrair e manipular informação a partir desses dados. Para capturar essa diversidade, Tim Berners-Lee (o criador da Web) propôs uma classificação para dados abertos de uma a cinco estrelas [3]. Em ordem crescente e cumulativa, os cinco níveis demandam:

1. Tornar os dados disponíveis na Web, independentemente do formato, com uma licença aberta;
2. Disponibilizar os dados em um formato estruturado, como uma planilha;
3. Utilizar um formato não-proprietário para disponibilizar os dados estruturados;
4. Identificar os itens de dados, de modo que outros possam apontar para eles;
5. Ligar seus dados aos de outros, para prover contexto.

Os níveis 4 e 5 estão diretamente relacionados diretamente à definição da Web Semântica e, portanto, serão abordados nos próximos capítulos. Os demais níveis serão tratados na sequência.

2.4.1 Dados abertos uma estrela

A classificação de uma estrela é dada aos dados que são disponibilizados com licença aberta, porém em formato não estruturado. É o que acontece, por exemplo, quando há dados em um arquivo em formato PDF.

Mesmo com dados em um formato que não facilita o acesso por agentes (código), é possível extrair a informação desse tipo de arquivo com o apoio de alguns pacotes da linguagem Python. Para um arquivo PDF com texto, pacotes como pyPDF2 [12] oferecem facilidades para extrair os dados do arquivo. O exemplo a seguir ilustra a extração de alguns dados de um arquivo textual em PDF disponível na Web:

```
import PyPDF2
import requests
import io
url = 'https://www.ft.unicamp.br/sites/default/' +
      'files/institucional/congregacao/pautas/' +
      'Pauta_05_Congregacao_Extraordinaria_04-11-2019.pdf'
name = url[url.rfind('/')+1:]
req = requests.get(url)
content = io.BytesIO(req.content)
pdfRd = PyPDF2.PdfFileReader(content)
print(name, 'tem', pdfRd.getNumPages(), 'paginas')
# Informações básicas: DocumentInformation
docInfo = pdfRd.getDocumentInfo()
if docInfo is None:
print(name, 'nao tem DocumentInfo')
else:
```

```
17 print('DocInfo Autor:',docInfo.author)
18 print('DocInfo Title:',docInfo.title)
19 # Metadados: XmpMetadata
20 metaDoc = pdfRd.getXmpMetadata()
21 if metaDoc is None:
22 print(name,'nao tem XmpMetadata')
23 else:
24 print('Xmp Autor',metaDoc.dc_creator)
25 print('Xmp Titulo', metaDoc.dc_title)
26 print('Xmp Data',metaDoc.dc_date)
27 # Página: PageObject
28 page = pdfRd.getPage(1)
29 text = page.extractText()
30 print(text)
```

As primeiras linhas desse código têm o padrão já conhecido para transferir o conteúdo de um recurso da Web, usando o método `get` de `requests`. Como o método `PdfFileReader` (linha 10) espera ler o conteúdo PDF desde um arquivo local, o método `BytesIO` é utilizado para simular um arquivo com os dados transferidos desde a Web (linha 9). A linha 11 ilustra o uso de um método para ler propriedades do arquivo, nesse caso, o número de páginas. Ao executar a instrução nessa linha, a seguinte informação é apresentada no console:

```
Pauta_05_Congregacao_Extraordinaria_04-11-2019.pdf
tem 59 paginas
```

Arquivos PDF mantêm algumas informações além do conteúdo e, a depender de como o arquivo é gerado, essa informação pode estar nos campos de metadados `DocumentInfo`, como ilustrado nas linhas 13 a 18, ou nos campos `XmpMetadata`, como ilustrado nas linhas 20 a 26. No caso desse documento, a execução das instruções referentes ao campo `DocumentInfo` apresenta no console:

```
DocInfo Autor: Administrador
DocInfo Title: None
```

Já a execução das linhas 20 a 26 produz, para esse mesmo documento:

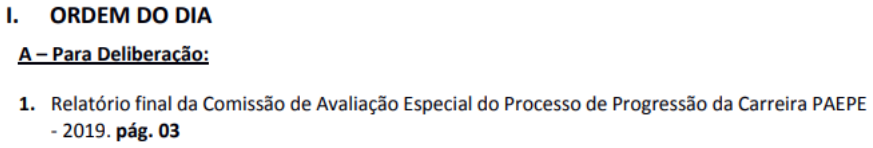
```
Xmp Autor ['Administrador']
Xmp Titulo {}
Xmp Data []
```

Como podemos observar, na geração desse arquivo apenas a informação sobre o autor foi armazenada.

Por fim, é possível obter uma página específica usando o método `getPage` e extrair seu conteúdo textual com o método `extractText`. Nesse exemplo, o uso desses métodos é ilustrado nas linhas 28 e 29 para extrair o conteúdo da terceira página do documento, resultando em suas primeiras linhas:

```
I. ORDEM DO DIA
A
Para Deliberação:
1. Relatório final da Comissão de Avaliação Especial
 do Processo de Progressão da Carreira PAEPE - 2019
. pág. 03
[...]
```

Esse conteúdo corresponde ao início da página representada na Figura 2.1, que uma vez assim obtido pode ser trabalhado como qualquer *string*.


I. ORDEM DO DIA

A – Para Deliberação:

1. Relatório final da Comissão de Avaliação Especial do Processo de Progressão da Carreira PAEPE - 2019. **pág. 03**

Figura 2.1: Fragmento de página PDF processado com pyPDF2

Em outros casos, o arquivo PDF pode conter tabelas com informação numérica. Nesses casos, o pacote `tabula-py` [1] pode ser útil. Por exemplo, considere o arquivo PDF contendo uma matriz de multiplicação, com um fragmento inicial apresentado na Figura 2.2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
3	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45
4	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60

Figura 2.2: Fragmento de página PDF processado com tabula-py

O código a seguir ilustra a utilização do pacote `tabula-py` para extrair a informação desse arquivo PDF, disponibilizado em um endereço URL:

```
from tabula import read_pdf
import requests
import io
url = 'https://www.georgebrown.ca/uploadedFiles/TLC/' +
      '_documents/Multiplication%20Table_15%20x%2015.pdf'
req = requests.get(url)
content = io.BytesIO(req.content)
tabPdfRd = read_pdf(content)
print(tabPdfRd)
```

Como no exemplo anterior, o pacote `io` é utilizado para lidar com o conteúdo transferido do servidor Web como se fosse um arquivo local. O método `read_pdf` é utilizado para traduzir o conteúdo da tabela apresentada nesse arquivo para um *data frame*, uma estrutura tabular provida no pacote *Python Data Analysis Library*, `pandas` [10]. A execução desse código resulta em:

```
Unnamed: 0  1  2  3  4  5  6 ...  9  10  11  12  13  14  15
0        1  1  2  3  4  5  6 ...  9  10  11  12  13  14  15
1        2  2  4  6  8 10 12 ... 18  20  22  24  26  28  30
2        3  3  6  9 12 15 18 ... 27  30  33  36  39  42  45
3        4  4  8 12 16 20 24 ... 36  40  44  48  52  56  60
4        5  5 10 15 20 25 30 ... 45  50  55  60  65  70  75
[...]
```

2.4.2 Dados abertos duas estrelas

Dados abertos com a classificação de duas estrelas têm por característica, além do que se espera de dados abertos de uma estrela, a apresentação dos dados em formato estruturado, mesmo que seja em um formato proprietário, como uma planilha em Excel.

A biblioteca `pandas`, a mesma usada internamente pelo pacote `tabula-py`, oferece recursos em Python para lidar com esse tipo de conjunto de dados. Como essa biblioteca consegue lidar diretamente com recursos na Web, não é necessário usar o método `get` de `requests`, como podemos ver no exemplo a seguir:

```
import pandas as pd
tabRd =  pd.read_excel('https://ufob.edu.br/' +
            'acessoainformacao/index.php/dados-abertos?' +
            'download=158:servidores-qualificados-e-' +
            'afastados-para-qualificacao',
            header=2)
print(tabRd)
```

Nesse exemplo, o método `read_excel` de `pandas` é usado diretamente para obter um recurso em Excel disponibilizado no Portal Brasileiro de Dados Abertos, e os dados lidos são traduzidos diretamente para o formato de *data frame*. Internamente, o pacote `xlrd` é utilizado para interpretar o conteúdo da planilha Excel. O parâmetro `header` indica as linhas que devem ser ignoradas no início do arquivo.

A execução desse código resulta, em suas primeiras linhas, em algo como (com parte dos nomes omitidos):

```
         SERVIDOR    LOTAÇÃO  ...      NÍVEL  UNIVERSIDADE
0    ADM...CHAVES       CCBS  ...  DOUTORADO          UFMG
1   ALIN...PESSOA       CEHU  ...  DOUTORADO      UNISINOS
[...]
[40 rows x 6 columns]
```

Os elementos de um data frame podem ser acessados pela posição (índice) usando o acessor `iloc`. Por exemplo, a instrução

```
print(tabRd.iloc[1,1])
```

resulta no valor 'CEHU', correspondente à segunda coluna da segunda linha, uma vez que o índice inicial é 0.

O método `read_excel` também pode ler arquivos em formato Open Document (ODF). Nesse caso, é preciso especificar o parâmetro `engine='odf'`.

2.4.3 Dados abertos três estrelas

Os dados abertos com classificação de três estrelas, além do que oferecem os dados abertos de duas estrelas, utilizam formatos não proprietários para representar os conjuntos de dados. Tipicamente, os dados abertos de três estrelas utilizam o formato CSV (valores separados por vírgulas) ou o já citado JSON.

A biblioteca `pandas` também oferece recursos para manipular dados nesses formatos. Por exemplo, para interpretar um arquivo CSV disponibilizado no Portal Brasileiros de Dados Abertos, no qual os valores estão separados por ponto e vírgula e a codificação utilizada foi ISO-8859-1, o método `read_csv` pode ser diretemente utilizado:

```
1 import pandas as pd
2 tabRd = pd.read_csv('http://dominios.governoeletronico.gov' +
3              '.br/dados-abertos/Dominios_GovBR_basico.csv',
4              sep=';', encoding='iso-8859-1')
5 print(tabRd)
```

A execução desse código cria um *data frame* com os domínios gov.br, e parte desses dados é apresentada no console:

```
                              dominio  ...   ticket
0                   eletrobras.gov.br  ...     5023
1     premioconservacaoenergia.gov.br  ...  5379045
2                        procel.gov.br  ...   453438
```

```
3                      proinfa.gov.br  ...   784297
4                        reluz.gov.br  ...   453442
...                                ...  ...      ...
1611                  cefetsvs.gov.br  ...  1033880
1612                   cefetbg.gov.br  ...  1038981
1613                      prr4.gov.br  ...    12880
1614                veracruz-rs.gov.br  ...   352374
1615              telebrasilia.gov.br  ...        0

[1616 rows x 9 columns]
```

O método `read_json` opera de modo semelhante. Utilizando como ilustração o mesmo arquivo JSON resultante do serviço Web com a posição da posição atual da Estação Espacial Internacional, o acesso com esse método seria como em:

```
1 import pandas as pd
2 tabRd = pd.read_json('http://api.open-notify.org/' +
3                      'iss-now.json')
4 print(tabRd)
```

O resultado dessa execução é similar a:

```
           iss_position  message            timestamp
latitude        -38.1610  success 2019-11-10 23:24:59
longitude       -86.0796  success 2019-11-10 23:24:59
```

Observamos que, como consequência do "achatamento" dos dados (transformação de uma estrutura hierárquica em tabular), parte dos dados (message, timestamp) é repetida nas linhas.

Capítulo 3

Organizando a informação para a Web Semântica

A classificação de dados abertos determina que, para que um conjunto de dados obtenha a classificação de quatro estrelas, é preciso que, além de atender os requisitos para a classificação de três estrelas, seja possível "identificar os itens de dados, de modo que outros possam apontar para eles." Neste capítulo veremos o que isso quer dizer.

3.1 O modelo de dados da Web Semântica

A título de exemplo, considere hipoteticamente que o site da Faculdade de Tecnologia oferece um conjunto de dados com o histórico de disciplinas oferecidas na graduação. No momento, esse é um conjunto de dados três estrelas, em formato CSV, e um fragmento desse arquivo contém:

```
Ano,Código,Nome,Professor
2019,TT001,Tópicos em computação e informática,Ivan Ricarte
2019,SI400,Programação orientada a objetos II,Ivan Ricarte
2019,ST008,Metodologia do trabalho científico,Ivan Ricarte
2018,SI405,Análise orientada a objetos II,Ivan Ricarte
```

```
2018,SI400,Programação orientada a objetos II,Ivan Ricarte
[...]
```

A evolução desse conjunto de dados para quatro estrelas envolve a possibilidade de identificar, individualmente, seus itens de dados. Para iniciar, cada linha do arquivo seria um item de dado identificando uma oferta de uma disciplina. Adicionalmente, há elementos dentro desse conjunto que também podem ser identificados individualmente, como disciplinas (que têm código e nome) e professores.

Na Web Semântica, todas essas informações são representadas por meio de afirmações com três elementos: um sujeito, um predicado e um valor. Voltando ao exemplo, suponha que a segunda linha recebeu um identificador `o2` e a quinta linha o identificador `o5`, que a disciplina SI400 recebeu um identificador `d25` e o professor Ivan Ricarte, o identificador `p98`. Os dados dessas mesmas linhas, expressos em termos das triplas de afirmações básicas, seria algo como:

```
o2   ano  '2019'  .
o2   disciplina  d25  .
o2   professor  p98  .
o5   ano  '2018'  .
o5   disciplina d25  .
o5   professor p98  .
d25  código  'SI400'  .
d25  nome  'Programação orientada a objetos II'  .
p98  nome  'Ivan Ricarte'  .
```

Dentro do conjunto de tecnologias utilizadas para a Web Semântica, os itens identificados são recursos, esse modelo de representação por triplas é o *framework* de representação de recursos (RDF, de *Resource Description Framework*) e os rótulos utilizados para indicar os recursos individualmente são os identificadores universal de recursos (URI, de *Universal Resource Identifier*).

É importante destacar que RDF estabelece um modelo conceitual, que pode ser expresso de diferentes maneiras. A mesma informação descrita anteriormente por meio de um conjunto de frases de três elementos pode ser expresso como um

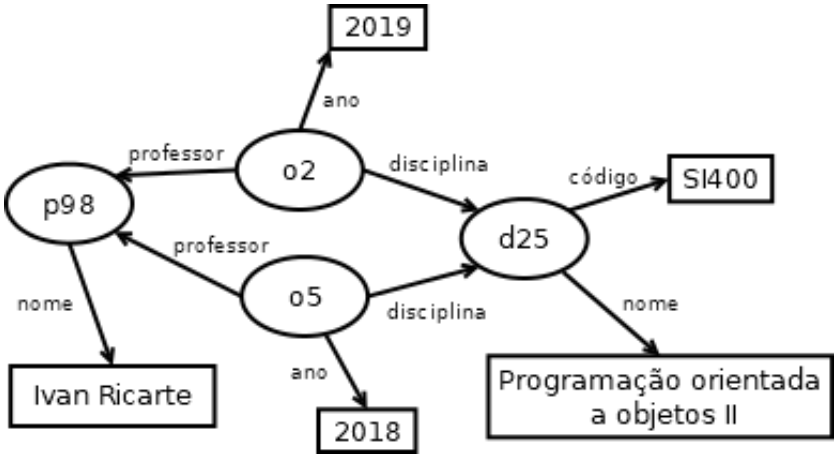


Figura 3.1: Exemplo de grafo com fatos ao estilo RDF

grafo, como apresentado na Figura 3.1. Podemos observar nessa figura três tipos de elementos: os recursos, representados por elipses; valores literais, representados por retângulos; e as propriedades, representadas pelos arcos.

O pacote RDFLib [15] oferece as funcionalidades para representar e manipular informação expressa em RDF na linguagem Python. Nesse pacote, o conjunto de fatos é representado por um objeto da classe `Graph`, que contém triplas com recurso, propriedade e valor. Recursos são criados como objetos `BNode`, quando o rótulo de sua identificação não é relevante, ou como objetos `URIRef`, quando o URI do recurso é conhecido. Propriedades são objetos `URIRef` e valores podem tanto ser recursos (`BNode` ou `URIRef`) como literais, representados como objetos da classe `Literal`.

O exemplo de código a seguir ilustra como os fatos do exemplo anterior poderiam ser criados em um programa Python usando RDFLib, com os fatos sendo adicionados ao grafo com o método `add`:

```
1 from rdflib import Graph, BNode, URIRef, Literal
2 g1 = Graph()
3 o2 = BNode()
4 o5 = BNode()
5 d25 = BNode()
```

```
6  p98 = BNode()
7  cod = URIRef(':codigo')
8  nome = URIRef(':nome')
9  ano = URIRef(':ano')
10 disc = URIRef(':disciplina')
11 prof = URIRef(':professor')
12 si400 = Literal('SI400')
13 poo2 = Literal('Programacao orientada a objetos II')
14 ilmr = Literal('Ivan Ricarte')
15 a2018 = Literal('2018')
16 a2019 = Literal('2019')
17 g1.add((d25, cod, si400))
18 g1.add((d25, nome, poo2))
19 g1.add((p98, nome, ilmr))
20 g1.add((o2, ano, a2019))
21 g1.add((o2, disc, d25))
22 g1.add((o2, prof, p98))
23 g1.add((o5, ano, a2018))
24 g1.add((o5, disc, d25))
25 g1.add((o5, prof, p98))
```

Nesse código, inicialmente é criado um grafo vazio (linha 2). Na sequência, os elementos do grafo são criados: recursos (linhas 3 a 6), propriedades (linhas 7 a 11) e literais (linhas 12 a 16). Por fim, as triplas são adicionadas ao grafo (linhas 17 a 25).

Existem alguns formatos padronizados para representar textualmente um conjunto de afirmações em RDF. O formato mais comum é a representação XML, que é eficiente para o processamento por um programa. Para esses fatos do exemplo, o modelo RDF em um arquivo XML seria algo como:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:ns1=":"
```

```
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
 <rdf:Description rdf:nodeID="N0a0b2e91dab744488b6e3e174">
  <ns1:professor rdf:nodeID="N551c224434d64a6e98c4748cb"/>
  <ns1:disciplina rdf:nodeID="N862d63a768af4d778e59a7547"/>
  <ns1:ano>2018</ns1:ano>
 </rdf:Description>
 <rdf:Description rdf:nodeID="N87748123009e4786b1bdc61db">
  <ns1:disciplina rdf:nodeID="N862d63a768af4d778e59a7547"/>
  <ns1:ano>2019</ns1:ano>
  <ns1:professor rdf:nodeID="N551c224434d64a6e98c4748cb"/>
 </rdf:Description>
 <rdf:Description rdf:nodeID="N551c224434d64a6e98c4748cb">
  <ns1:nome>Ivan Ricarte</ns1:nome>
 </rdf:Description>
 <rdf:Description rdf:nodeID="N862d63a768af4d778e59a7547">
  <ns1:codigo>SI400</ns1:codigo>
  <ns1:nome>Programacao orientada a objetos II</ns1:nome>
 </rdf:Description>
</rdf:RDF>
```

Nesse arquivo, os fatos sobre cada recurso são agrupados em um elemento `Description`, com um atributo `nodeID` criado internamente pelo programa. Há quatro desses elementos, correspondentes às quatro elipses da Figura 3.1. Os valores literais são representados como os conteúdos dos elementos correspondentes às propriedades definidas no grafo.

Outro formato usual para armazenar RDF em arquivos textuais é o *turtle*, que também agrupa os fatos sobre cada recurso, separando as diferentes propriedades e valores por ';' e concluindo com '.', como pode ser visto nesse exemplo, associado aos mesmos fatos:

```
@prefix ns1: <:> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
```

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[] ns1:ano "2018" ;
ns1:disciplina _:N673d43e7ddb04f3b943643079ed6 ;
ns1:professor _:Nd427943e49e246b8b9f66915acf5 .

[] ns1:ano "2019" ;
ns1:disciplina _:N673d43e7ddb04f3b943643079ed6 ;
ns1:professor _:Nd427943e49e246b8b9f66915acf5 .

_:N673d43e7ddb04f3b943643079ed6 ns1:codigo "SI400" ;
ns1:nome "Programacao orientada a objetos II" .

_:Nd427943e49e246b8b9f66915acf5 ns1:nome "Ivan Ricarte" .
```

Como no exemplo anterior, com o arquivo XML, são quatro os recursos representados nesse arquivo turtle. Para cada recurso, o primeiro elemento é o sujeito, sendo que os pares subsequentes correspondem, cada um, a um par propriedade-valor. Os nós em branco não referenciados nesse conjunto de fatos foram representados pelo programa como `[]`, enquanto que os nós que foram referenciados receberam a atribuição de uma identificação arbitrária.

O modelo RDF, com base nesse princípio simples de representação por meio de fatos elementares, permite criar estruturas de informação arbitrariamente complexas. Devemos lembrar que tais estruturas são adequadas para o processamento automático, não sendo seu objetivo principal a compreensão por seres humanos.

3.2 Os vocabulários da Web Semântica

Ao criarmos um conjunto de fatos para representar alguma informação em nossa aplicação, definimos um vocabulário de termos que são utilizados para descrever

esses fatos. No exemplo das disciplinas, por exemplo, decidimos que uma disciplina tem um *código* e um *nome*, mas poderíamos, sem alterar os fatos, dizer que uma disciplina tem uma *sigla* e um *título*. Esse processo não é muito diferente de fazer um modelo de classes para uma aplicação de software: os nomes de classes e de seus atributos podem ter variações para diferentes desenvolvedores de uma mesma aplicação, mas estruturalmente os modelos deveriam ser equivalentes.

A Web Semântica, no entanto, tem por objetivo conectar todas as informações da Web usando essa representação básica. Se cada desenvolvedor decidir "batizar" suas propriedades com nomes diferentes, seria muito difícil estabelecer qualquer conexão entre conjuntos de dados desenvolvidos separadamente, mesmo que versassem sobre um mesmo domínio de conhecimento.

Para evitar esse problema, existem alguns vocabulários padronizados, que são amplamente aceitos e adotados pela comunidade de desenvolvedores de aplicações para a Web Semântica. Cada vocabulário define classes (tipos de recursos) e propriedades para domínios de conhecimento específicos, e uma mesma aplicação pode combinar tantos vocabulários quantos forem necessários.

Cada termo de um desses vocabulários, seja classe ou propriedade, tem um URI padronizado. Tipicamente, esse URI adota o formato de uma referência HTTP, mas não necessariamente representa um endereço URL válido para um recurso da Web. No entanto, em alguns casos o URI pode ser um endereço URL válido e um ser humano que decida ver o recurso nesse endereço vai receber uma página HTML válida, tipicamente com a descrição do termo ou do projeto que definiu o vocabulário. O prefixo comum de todos os termos de um vocabulário recebe o nome de *namespace*. Alguns exemplos de *namespace* estavam presentes nos arquivos XML (definidos no elemento RDF, após `xmlns:`) e turtle (definidos no início do arquivo, com `@prefix`) apresentados anteriormente.

Cabe ao desenvolvedor de uma aplicação para a Web Semântica conhecer esses vocabulários e, antes de criar novos termos para descrever as propriedades e tipos de objetos, verificar se há termos já definidos nesses vocabulários que possam representar adequadamente a informação. Desse modo, o seu conjunto de

dado será mais facilmente compreensível por outras aplicações, facilitando a reutilização dos dados.

Na sequência, apresentamos brevemente alguns desses vocabulários padronizados. Por padrão, termos que representam nomes de classes iniciam com letra maiúscula e nomes de propriedades com letras minúsculas.

3.2.1 rdf e rdfs

O vocabulário padronizado mais básico para representar informação na Web Semântica é aquele que define os termos necessários para criar vocabulários em RDF, conhecido como *RDF Vocabulary Description Language* [14]. Além do rdf, há também o vocabulário para esquemas RDF, o rdfs.

Nesses vocabulários são definidas, entre outras, as classes rdfs:Resource, rdf:Property e rdfs:Literal. A classe rdfs:Class permite representar um tipo ou categoria de recurso, e a propriedade rdf:type é usada para indicar que um recurso é membro de uma classe. Duas propriedades são voltadas a incorporar ao RDF textos voltados para seres humanos, que são o rdfs:label (o nome do recurso) e rdfs:comment (uma descrição do recurso), podendo essas propriedades ter associadas um sufixo de rótulo de linguagem, como "@en" ou "@pt". São também definidas nesses vocabulários as propriedades rdfs:subClassOf, usada para representar a especialização de uma classe, e rdfs:subPropertyOf, para especializações de propriedades. Há ainda termos para indicar o domínio (rdfs:domain) e o contradomínio (rdfs:image) de uma propriedade.

Esses termos são básicos para a construção de todos os outros vocabulários, e conhecê-los é fundamental para compreender as demais definições na Web Semântica.

O *namespace* para o vocabulário rdf é `http://www.w3.org/1999/02/22-rdf-syntax-ns` e para o vocabulário rdfs (RDF Schema) é `http://www.w3.org/2000/01/rdf-schema`.

3.2.2 owl

A *Web Ontology Language* (OWL) [11] tem por objetivo descrever ontologias que, na área de informática, são representações de fatos e regras sobre um domínio de conhecimento. O vocabulário de owl também define termos básicos para a Web Semântica, porém com foco em um nível mais conceitual do que os vocabulários rdf e rdfs.

Os termos de owl que são normalmente utilizados na construção de outros vocabulários são a classe owl:Thing (a base para representar qualquer coisa) e a propriedade owl:sameAs, usada para indicar que dois termos de vocabulários diferentes são equivalentes.

O *namespace* para os termos de OWL é `http://www.w3.org/2002/07/owl`.

3.2.3 dc

O vocabulário *Dublin Core* (dc) [2] define termos para descrever metadados (dados sobre dados) referentes a recursos digitais.

Apesar deste vocabulário definir também classes e tipos de recursos, tipicamente são mais utilizadas nas aplicações em gerais suas propriedades, tais como dc:title para o título de uma obra ou trabalho, dc:creator para identificar seu autor, dc:contributor para contribuidores, dc:date para a data, dc:subject para seus assuntos, dc:identifier para identificadores e dc:language para identificar a linguagem de publicação.

O vocabulário dc está associado ao *namespace* `http://purl.org/dc/elements/1.1/`.

3.2.4 foaf

O vocabulário *Friend of a friend* (foaf) [5] define termos para descrever pessoas, suas atividades e seus relacionamentos com outras pessoas e objetos.

Foaf especifica categorias como Agent, Person, Group, Organization, Project, Document e Image, bem como propriedades como name (firstname, lastname), nick, phone, birthday, mbox, homepage, knows, img (para a imagem de uma pessoa) e depiction (para imagens de outros elementos).

Este vocabulário está associado ao *namespace* `http://xmlns.com/foaf/0.1/`.

3.2.5 skos

SKOS é o *Simple Knowledge Organization System* [20], um vocabulário para descrever conceitos e seus relacionamentos, à maneira de um tesaurus.

SKOS define termos como definition, example, preflabel, altlabel e hiddenlabel e relações como broader, narrower e related.

O *namespace* deste vocabulário é `http://www.w3.org/2004/02/skos/core#`.

3.2.6 doap

O vocabulário DOAP [24] foi criado para a descrição de projetos, com classes tais como Project, Specification, Version, Repository, SVNRepository, GitRepository e propriedades como name, homepage, shortdesc, description, created, license, developer, documenter, tester, translator, helper e programming-language

Está associado ao *namespace* `http://usefulinc.com/ns/doap#`

3.2.7 sioc

O vocabulário SIOC é utilizado para descrever comunidades online como fóruns e blogs, com classes como Community, Site (tipo de Space), Forum (tipo de Container), Post (tipo de Item), Thread, User, Usergroup, Role e propriedades como content, about, name, administrator_of (reversa: has_administrator), creator_of (reversa: has_creator), email, function_of (reversa: has_function), topic.

Seu *namespace* é `http://rdfs.org/sioc/ns#`.

3.2.8 Vocabulários em RDFLib

RDFLib tem um módulo `namespace` que, além das facilidades para criar um namespace e associá-lo a um prefixo, já traz alguns dos principais vocabulários pré-definidos: RDF, RDFS, OWL, XSD, FOAF, SKOS, DOAP, DC, DCTERMS e VOID.

Para ilustrar o uso desses vocabulários, voltamos aos mesmos fatos do exemplo apresentado na Seção 3.1, onde foram realizados as seguintes substituições de propriedades:

ano foi utilizada a propriedade dc:date para representar o ano de oferta de uma displina por um professor;

código foi utilizada a propriedade dc:identifier para representar a sigla da disciplina;

nome para o nome de uma disciplina, foi utilizada a propriedade rdfs:label;

nome para o nome de uma pessoa, foi utilizada a propriedade foaf:name.

Adicionalmente, foi agregado um fato de que um professor é uma pessoa, ou seja, que tem a propriedade rdf:type com um valor foaf:Person.

Com essas substituições, o código ficou como segue:

```
1 from rdflib import Graph, BNode, URIRef, Literal
2 from rdflib.namespace import DC, RDF, RDFS, FOAF
3 g1 = Graph()
4 o2 = BNode()
5 o5 = BNode()
6 d25 = BNode()
7 p98 = BNode()
8 disc = URIRef(':disciplina')
```

```
9  prof = URIRef(':professor')
10 si400 = Literal('SI400')
11 poo2 = Literal('Programacao orientada a objetos II')
12 ilmr = Literal('Ivan Ricarte')
13 a2018 = Literal('2018')
14 a2019 = Literal('2019')
15 g1.add((d25, DC.identifier, si400))
16 g1.add((d25, RDFS.label, poo2))
17 g1.add((p98, FOAF.name, ilmr))
18 g1.add((o2, DC.date, a2019))
19 g1.add((o2, disc, d25))
20 g1.add((o2, prof, p98))
21 g1.add((o5, DC.date, a2018))
22 g1.add((o5, disc, d25))
23 g1.add((o5, prof, p98))
24 g1.add((p98, RDF.type, FOAF.Person))
```

O correspondente arquivo de triplas em formato turtle é:

```
@prefix ns1: <:> .
@prefix ns2: <http://purl.org/dc/elements/1.1/> .
@prefix ns3: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[] ns1:disciplina _:N75227b09a65340f9be2efb2517c6c819 ;
   ns1:professor _:Ned2a6a1cfa6246a28baa625c4352b5fd ;
   ns2:date "2018" .

[] ns1:disciplina _:N75227b09a65340f9be2efb2517c6c819 ;
   ns1:professor _:Ned2a6a1cfa6246a28baa625c4352b5fd ;
   ns2:date "2019" .
```

```
_:N75227b09a65340f9be2efb2517c6c819
   rdfs:label "Programacao orientada a objetos II" ;
   ns2:identifier "SI400" .

_:Ned2a6a1cfa6246a28baa625c4352b5fd a ns3:Person ;
  ns3:name "Ivan Ricarte" .
```

Podemos observar nesse arquivo de triplas que ainda é utilizado um *namespace* anônimo (prefixo ns1 nesse caso, definido na primeira linha do arquivo) para identificar os elementos (recursos e propriedades) definidos localmente. Há também o uso de uma abreviação, `a`, para a propriedade rdf:type, como podemos observar na penúltima linha do arquivo. Por fim, apesar dos vocabulários DC e FOAF serem conhecidos pelo RDFLib, como podemos verificar pelos *namespaces* definidos na segunda e terceira linha, os prefixos utilizados são genéricos (ns2 e ns3), o que não facilita a compreensão do arquivo de triplas por um ser humano.

Para resolver esse último problema, é possível utilizar a funcionalidade oferecida pelo *Namespace manager* de RDFLib para associar um prefixo mais familiar a um *namespace*. Nesse caso, definimos os prefixos desejados para os *namespaces* em um novo gerenciador e associamos esse gerenciador ao grafo, como em:

```
1 from rdflib import Graph, BNode, URIRef, Literal
2 from rdflib.namespace import NamespaceManager, \
3    DC, RDF, RDFS, FOAF
4 nsmgr = NamespaceManager(Graph())
5 nsmgr.bind('dc', DC)
6 nsmgr.bind('foaf',FOAF)
7 g1 = Graph()
8 g1.namespace_manager = nsmgr
```

Com esse novo gerenciador de *namespaces*, o arquivo de triplas usa os prefixos mais familiares para DC e FOAF:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

```
@prefix ns1: <:> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[] ns1:disciplina _:Nbf2b9bf46310460a81f2bb6ac7b4f6a8 ;
  ns1:professor _:N2a40aa3981484647be957dc1b0a3e842 ;
  dc:date "2018" .

[] ns1:disciplina _:Nbf2b9bf46310460a81f2bb6ac7b4f6a8 ;
  ns1:professor _:N2a40aa3981484647be957dc1b0a3e842 ;
  dc:date "2019" .

_:N2a40aa3981484647be957dc1b0a3e842 a foaf:Person ;
  foaf:name "Ivan Ricarte" .

_:Nbf2b9bf46310460a81f2bb6ac7b4f6a8
  rdfs:label "Programacao orientada a objetos II" ;
  dc:identifier "SI400" .
```

3.3 Estratégias para manter os dados

As triplas criadas durante a execução de um programa e adicionadas a um grafo em RDFLib são mantidas, por padrão, na memória volátil do processo, o que significa que, após a execução, todos esses dados são descartados.

É possível manter os dados RDF em um repositório persistente. Há sistemas de armazenamento e bancos de dados especializados para esse tipo de informação, como os *triplestores* e o bancos de dados de grafos (*graph databases*).

No caso do RDFLib, é possível também ter um armazenamento persistente, usando para esse fim um banco de dados. Para tanto, é preciso instalar inicial-

mente o Oracle Berkeley Database[1] e, no ambiente de Python, o pacote `bsddb3`. Com essas opções podemos, ao criar um grafo em RDFLib, especificar a opção do repositório Sleepycat, que utiliza esses recursos para manter as triplas no banco de dados:

```
g1 = Graph(store='Sleepycat')
g1.open(path, create=True)
# Operações para adicionar triplas ao grafo
# ...
g1.close()
```

Neste exemplo, *path* é o caminho para um diretório que irá armazenar o banco de dados.

Para fazer a leitura das triplas previamente armazenadas em um banco de dados:

```
from rdflib import ConjunctiveGraph
g1 = ConjunctiveGraph(store='Sleepycat')
g1.open(path, create=False)
# Operações para manipular ou adicionar triplas ao grafo
# ...
g1.close()
```

A documentação de RDFLib esclarece que o `ConjunctiveGraph` é uma união anônima de todos os grafos armazenados no repositório especificado.

Outra alternativa é salvar o grafo em um arquivo ao final da execução e, ao início da execução seguinte, ler o grafo de um arquivo. Para exportar o grafo para um arquivo, utilizamos o método serialize:

```
g1.serialize('rdf3.xml')
```

O formato padrão para esse arquivo é XML.

Para ler um arquivo de triplas, utilizamos o método parse:

[1] Disponível em `https://www.oracle.com/database/technologies/related/berkeleydb-downloads.html`

```
g1.parse('rdf3.xml')
```

Mais uma vez, quando um formato não é especificado, o padrão é XML. O método `parse` aceita como argumento tanto arquivos locais como arquivos na Web, identificados por seu URL.

Outros formatos podem ser usados. Por exemplo, para usar o formato turtle, podemos gravar um arquivo como:

```
g1.serialize('rdf3.ttl', format='turtle')
```

Similarmente, para a leitura:

```
g1.parse('rdf3.ttl', format='turtle')
```

3.4 Comentários

Vimos neste capítulo como se dá, por meio da criação de recursos com identificadores únicos (URI), a identificação dos dados abertos disponibilizados na Web, permitindo assim que um conjunto de dados atinja o nível de classificação de quatro estrelas.

Veremos, no próximo capítulo, como esses conjuntos de dados podem ser utilizados e, ainda mais importante, conectados, de modo a atingir a classificação de cinco estrelas em dados abertos.

Capítulo 4

Utilizando a informação na Web Semântica

Uma vez que os dados de uma aplicação estejam representados, armazenados e disponibilizados em RDF, essa informação pode ser utilizada e incorporada a outras aplicações na Web Semântica. Reciprocamente, a própria aplicação pode obter e utilizar fatos armazenados em triplas em outras aplicações.

4.1 Consultas a dados na Web Semântica

O grafo RDF pode ser consultado por meio de uma linguagem de consulta específica, SPARQL, ou por meio de métodos do pacote RDFLib.

Inicialmente, vamos considerar apenas os próprios dados da aplicação disponibilizados em formato RDF. Esses dados podem estar armazenados em um repositório local, seja em formato de arquivo, seja em um *triple store*.

Posteriormente, veremos como realizar a consulta envolvendo os dados localizados em um servidor remoto.

4.1.1 Métodos de consulta de RDFLib

O pacote RDFLib oferece diversos métodos para extrair informação do grafo RDF em memória, tenha sido ele construído em memória, carregado de um arquivo local ou de um arquivo remoto.

Um grafo pode ser completamente percorrido por meio do iterador em um comando `for`:

```
from rdflib import Graph
g = Graph()
filename = 'curso94.ttl'
g.parse(filename, format='turtle')

print('Todos os fatos:')
for s, p, o in g:
    print(s, p, o)
print(len(g),'triplas')
```

Nesse exemplo, um grafo RDF de uma aplicação que mantém fatos sobre as disciplinas de um curso da Unicamp é obtido a partir de um arquivo em formato turtle e depois todas as suas triplas são apresentadas. Sua execução, omitindo os namespaces e truncando as string longas para melhorar a legibilidade, resulta em algo como:

```
Todos os fatos:
:SI305 :identifier SI305
:ST211 :descripion Estatística descritiva. Probabilidade...
:TT350 :identifier TT350
:SI350 :label Teoria Geral de Sistemas
:ST567 :creditos 4
:ST562 :label Estruturas de Arquivos
:ST562 :prereq :SI201
[...]
:SI918 :label Atividades Complementares
208 triplas
```

Devemos observar que, como o grafo é uma estrutura não hierárquica, a ordem de apresentação da resposta pode variar a cada execução.

Além de poder percorrer todas as triplas de um grafo, o método `triples` permite obter apenas as triplas que seguem um determinado padrão. Por exemplo, considere este código:

```
1 from rdflib import Graph
2 from rdflib.namespace import RDFS
3 g = Graph()
4 g.parse('curso94.ttl', format='turtle')
5 for s, p, o in g.triples( (None, RDFS.label, None) ):
6     print(s,':',o)
```

Neste caso, apenas as triplas cujo predicado é rdfs:label serão selecionadas do grafo, pois esse foi o único elemento da tripla que foi especificado. Como essa propriedade foi usada nesse arquivo para identificar o nome de uma disciplina, o resultado dessa execução é apresentar os identificadores de recursos para cada disciplina do curso e seu título:

```
:SI801 : Introdução a Auditoria e Segurança de Sistemas
:TT060 : Gestão de Projetos
:SI703 : Governança e Planejamento Estratégico de TI
:SI350 : Teoria Geral de Sistemas
:EB101 : Cálculo I
:SI800 : Empreendedorismo e Inovação
[...]
```

No exemplo seguinte, o padrão tem mais de um elemento da tripla especificado para o mesmo grafo:

```
for s, p, o in g.triples( (None,
                           URIRef(':prereq'),
                           URIRef(':SI100')) ):
    print(s)
```

O resultado da execução, nesse caso, é apresentar os identificadores dos recursos que representam as disciplinas que têm como pré-requisito a disciplina identificada pelo recurso ':SI100':

```
:ST266
:SI200
:SI305
:SI201
:ST567
:SI300
```

Se os três elementos da tripla são especificados, ou seja, nenhum dos três elementos é `None`, o método `triples` apenas verifica se a tripla especificada existe no grafo.

Além do método `triples`, RDFLib tem métodos utilitários que retornam apenas parte da tripla, como `subjects()`, `predicates()` e `objects()`, além de métodos que retornam pares de elementos, como `subject_objects()`, `subject_predicates()` e `predicates_objects()`. Assim, no exemplo anterior, que apresenta apenas o primeiro elemento da tripla, o método `subjects()` poderia ter sido utilizado:

```
for s in g.subjects( (URIRef(':prereq'),
                      URIRef(':SI100')) ):
    print(s)
```

Os recursos apresentados como resultado da execução seriam os mesmos do exemplo anterior.

Quando o retorno de qualquer um desse métodos é garantidamente um único valor, ou seja, quando a relação envolvida entre o sujeito e seu valor é uma propriedade funcional, o método `value` pode ser usado, como em

```
cod = Literal('SI200')
disc = g.value(None, DC.identifier, cod)
print(disc)
```

Nesse caso, há um único recurso cujo identificador é o literal ‘SI200’ e, portanto, a referência para esse recurso seria retornada na variável `disc` e sua representação (:SI200) apresentada na sequência.

4.1.2 SPARQL

SPARQL [18] é uma linguagem de consulta para RDF, independente da linguagem de programação utilizada para desenvolver a aplicação para a Web Semântica. Além de poder ser utilizada embutida em aplicativos, pode ser utilizada para acesso remoto a repositórios com dados RDF, os chamados *sparql endpoints*.

Uma consulta em SPARQL segue o seguinte padrão básico:

```
SELECT <variáveis>
  WHERE {
    <padrões de triplas envolvendo variáveis> .
  }
```

As variáveis na consulta SPARQL são identificadas pelo símbolo ‘?’ seguido de um identificador. Quando há no grafo RDF triplas que combinam com o padrão especificado, os nós do grafo que combinaram com as variáveis da cláusula WHERE e que são listadas em SELECT são incluídas na resposta. Por exemplo, a consulta SPARQL mais ampla seria:

```
SELECT ?x ?y ?z
    WHERE {
      ?x ?y ?z .
    }
```

Nesse caso, os três elementos da cláusula WHERE são variáveis, ou seja, qualquer tripla satisfaz o padrão especificado. Como o SELECT inclui as três variáveis, todas as triplas do grafo são incluídas na resposta a essa consulta.

A cláusula WHERE pode incluir a especificação de diversas triplas que devem ser simultaneamente satisfeitas para incluir as variáveis na resposta. Por exemplo:

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema>
SELECT ?x ?z
WHERE {
   ?r dc:identifier ?x .
   ?r rdfs:label ?z .
}
```

Nesse caso, a resposta será composta pelos identificadores (dc:identifier) e rótulos (rdfs:label) de todos os recursos do grafo que tenham definidas essas duas propriedades. Os prefixos utilizados nos padrões podem ser definidos com as declarações PREFIX, especificados antes de SELECT.

Os resultados retornados na consulta podem ser ainda selecionados se for especificada uma cláusula FILTER após WHERE. Isso pode ser útil, por exemplo, para filtrar recursos em uma determinada linguagem, apenas.

4.1.3 Consulta SPARQL em RDFLib

No caso do RDFLib, o método `query()` recebe como argumento uma string com a consulta SPARQL e retorna os resultados. Por exemplo, uma consulta para apresentar todas as triplas do grafo com fatos sobre a disciplina do curso da Unicamp pode ser executada em Python como em:

```
1 from rdflib import Graph
2 g = Graph()
3 g.parse('curso94.ttl', format='turtle')
4 q1 = 'SELECT ?x ?y ?z WHERE { ?x ?y ?z . }'
5 resultado = g.query(q1)
6 print(f'Todas as triplas em {filename}:')
7 for stmt in resultado:
8     print("%s - %s - %s" % stmt)
```

Para especificar um *namespace* na consulta, o método `query` tem um parâmetro opcional que é o `initNS`, um dicionário (dict) com o prefixo e o identifi-

cador do namespace. Por exemplo, para obter desse grafo das disciplinas a sigla da disciplina (dc:identifier) e o seu nome (rdfs:label) para todas as disciplinas, a consulta poderia ser feita como em:

```
from rdflib import Graph
from rdflib.namespace import DC, RDFS
g = Graph()
g.parse('curso94.ttl', format='turtle')
q1 = """SELECT ?cod ?nom
WHERE { ?x  dc:identifier ?cod .
        ?x  rdfs:label ?nom .}"""
resultado = g.query(q1, initNs={'dc' : DC, 'rdfs' : RDFS})
for c, n in resultado:
    print(c,':',n)
```

O resultado dessa execução é similar a:

```
SI200 : Algoritmos e Programação de Computadores II
ST562 : Estruturas de Arquivos
SI800 : Empreendedorismo e Inovação
SI400 : Programação Orientada a Objetos II
SI220 : Matemática Discreta
[...]
```

No caso da consulta envolver termos que correspondem a variáveis do código, o parâmetro `initBindings`, um dicionário mapeando a variável da cláusula WHERE com a variável do código, pode ser utilizado. Por exemplo, a consulta das disciplinas que apresentam como pré-requisito a disciplina SI100, apresentada na seção anterior com o método `subjects()`, pode ser expressa em SPARQL como no código a seguir:

```
from rdflib import Graph, URIRef
g = Graph()
filename = 'curso94.ttl'
g.parse(filename, format='turtle')
```

```
preq = URIRef(':prereq')
disc = URIRef(':SI100')
q1 = """SELECT ?x
        WHERE { ?x ?p ?d . }"""
resultado = g.query(q1,
                    initBindings={'p' : preq, 'd': disc})
for x in resultado:
    print(x)
```

A cláusula `WHERE` pode envolver o relacionamento entre várias triplas, com os padrões especificados separados por pontos. Por exemplo, a consulta a seguir apresenta todas as cadeias de pré-requisitos de disciplinas com comprimento três:

```
from rdflib import Graph, URIRef
from rdflib.namespace import DC
g = Graph()
g.parse('curso94.ttl', format='turtle')
preq = URIRef(':prereq')
q1 = """SELECT ?d1 ?d2 ?d3
    WHERE { ?x ?pre ?y .
        ?y ?pre ?z .
        ?x dc:identifier ?d1 .
        ?y dc:identifier ?d2 .
        ?z dc:identifier ?d3 .}"""
resultado = g.query(q1, initBindings={'pre': preq},
                    initNs={'dc' : DC})
print(f'Cadeia de 3 prereqs:')
for x, y, z in resultado:
    print(x,'depende de',y,'que depende de',z)
```

O resultado dessa execução é similar a:

```
Cadeia de 3 prereqs:
SI700 depende de SI400 que depende de SI300
ST767 depende de ST567 que depende de SI100
```

```
SI801 depende de TT304 que depende de TT106
SI304 depende de ST266 que depende de SI100
TT060 depende de ST266 que depende de SI100
SI404 depende de ST266 que depende de SI100
ST562 depende de SI201 que depende de SI100
SI700 depende de SI201 que depende de SI100
SI400 depende de SI300 que depende de SI100
SI401 depende de SI300 que depende de SI100
SI405 depende de SI305 que depende de SI100
```

4.1.4 Consulta SPARQL a pontos remotos

Quando um provedor disponibiliza um *endpoint* para acesso a seus dados RDF, é possível fazer a consulta remotamente, seja por meio de uma interface interativa, seja por meio de código.

Um dos provedores de RDF mais utilizados na Web Semântica é a DBpedia, que representa os dados da Wikipedia em RDF. A DBpedia tem um *endpoint* disponibilizado em `https://dbpedia.org/sparql`, cuja aparência inicial é apresentada na Figura 4.1.

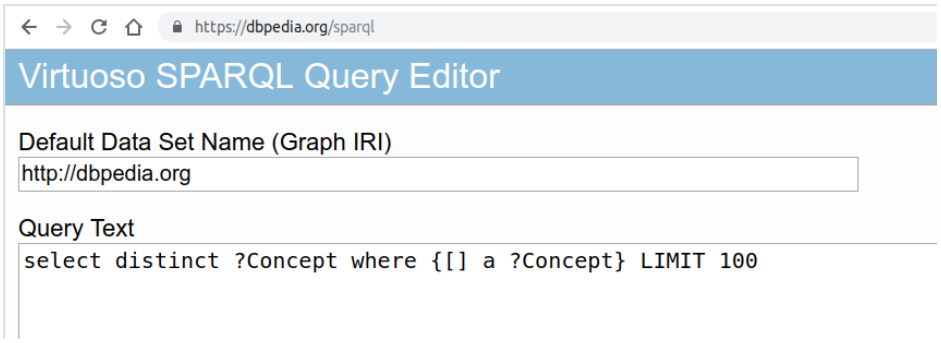


Figura 4.1: Endpoint SPARQL para DBpedia

A consulta apresentada ao acessar a página recupera 100 tipos (conceitos) usados nas representações de fatos na DBpedia, usando a forma abreviada `a` para a propriedade rdf:type. Uma execução dessa consulta gera respostas similares a:

```
http://xmlns.com/foaf/0.1/Organization
http://dbpedia.org/ontology/Company
http://xmlns.com/foaf/0.1/Person
http://dbpedia.org/ontology/Activity
http://dbpedia.org/ontology/Name
http://dbpedia.org/ontology/Person
http://dbpedia.org/ontology/Actor
http://dbpedia.org/ontology/Place
http://dbpedia.org/ontology/Publisher
[...]
```

Para fazer a consulta a um endpoint SPARQL a partir de um código Python, o pacote SPARQLWrapper [7] pode ser utilizado, tanto diretamente como por meio de um plugin (*sparl store*) de RDFLib. Por exemplo, para fazer uma consulta à DBpedia para obter os títulos (em português) dos filmes estrelados pelo ator Sean Connnery, é possivel utilizar o seguinte código:

```
from rdflib import Graph, Namespace
dbo = Namespace('http://dbpedia.org/ontology/')
graph = Graph('SPARQLStore', identifier="http://dbpedia.org")
graph.open("http://dbpedia.org/sparql")
q = """
    PREFIX dbr: <http://dbpedia.org/resource/>
    PREFIX dbo: <http://dbpedia.org/ontology/>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    SELECT ?movietitle
       WHERE { ?movie dbo:starring dbr:Sean_Connery .
          ?movie rdfs:label ?movietitle .
       FILTER (lang(?movietitle) = 'pt')}
"""
res = graph.query(q)
for r in res:
    print(r.movietitle)
```

O início do resultado da execução desse código é similar a:

```
007 - Os Diamantes São Eternos
Sol Nascente (filme)
Os Vingadores (filme de 1998)
O Nome da Rosa (filme)
Darby O'Gill and the Little People
Dragonheart
Zardoz
[...]
```

4.2 Dados abertos conectados

A conexão entre conjuntos de dados ocorre a partir do momento em que um conjunto de dados tem, entre os nós de seu grafo, nós que têm como valor um identificador referente a outro conjunto de dados.

A título de exemplo, considere que ao grafo de disciplinas de um curso da Unicamp foi acrescido, ao recurso que representa a disciplina, uma propriedade para definir o seu assunto, usando o termo dc:subject. Considere ainda que, quando possível, o valor desse assunto toma um valor do recurso correspondente na DBpedia. Por exemplo, para a disciplina Estrutura de dados, uma possível representação (em turtle) seria:

```
:SI201 dc:identifier SI201;
  rdfs:label Estrutura de dados;
  dc:subject http://dbpedia.org/resource/Data_structure ;
[...]
```

A grande vantagem de realizar essa conexão é que, a partir desse elo, é possível ter acesso a todas as informações já registradas no outro conjunto de dados. Por exemplo, nesse recurso do exemplo há a definição de uma propriedade 'dbo:abstract', onde dbo: é o prefixo usual para o *namespace* `http://dbpedia.org/ontology/`, que poderia ser alcançada a partir dessa conexão.

O site *The Linked Open Data Cloud* (`https://lod-cloud.net/`) apresenta um amplo (no momento, cerca de 1400) conjunto de dados conectados e, em

princípio, qualquer desses conjuntos de dados poderiam ser referenciados em um novo conjunto de dados. Na prática, é prudente avaliar se o conjunto selecionado permanece com seu *endpoint* SPARQL ativo.

Na versão atual de SPARQL (1.1), a conexão entre o grafo local e o grafo remoto pode ser feita por meio de uma cláusula SERVICE, em algo similar a:

```
SELECT ?cod ?tit ?res
  WHERE {
     ?d dc:identifier ?cod .
     ?d dc:subject ?dbp .
     ?d rdfs:label ?tit .
  SERVICE <http://dbpedia.org/sparql> {
     ?dbp dbo:abstract ?res}
  }
```

Nesse exemplo, as variáveis enumeradas em SELECT virão do grafo local (?cod e ?tit) e dos dados em DBpedia (?res). A conexão entre os dois grafos é realizada, nesse exemplo, pela variável ?dbp, que aparece no padrão WHERE do grafo local e, via SERVICE, do grafo remoto. O resultado seria, para todas as disciplinas que tivessem esses dados definidos, a sua sigla, seu nome e o resumo do assunto (obtido da DBpedia).

No entanto, o recurso de SERVICE em SPARQL nem sempre está implementado. Esse também é o caso da versão atual (4.2.2) de RDFLib. Para contornar essa limitação, o que se faz nessa situação é obter, via o *endpoint* SPARQL, os nós de interesse do grafo remoto e incorporá-los ao grafo local. Após essa incorporação, a consulta ao grafo local poderá fazer uso dos nós que originalmente estavam no grafo remoto.

A título de exemplo, o seguinte fragmento de código incorpora, no grafo `g` de disciplinas de um curso da Unicamp, uma propriedade dc:subject à disciplina de Estrutura de dados (:SI201) com valor igual ao recurso correspondente a esse assunto na DBpedia, `http://dbpedia.org/resource/Data_structure` (importações inicias omitidas):

```
dbpedia = Namespace('http://dbpedia.org/resource/')
cod = Literal('SI201')
disc = g.value(None, DC.identifier, cod)
dbres = URIRef(dbpedia.Data_structure)
g.add((disc, DC.subject, dbres))
```

Nesse trecho de código, a variável `disc` tem o URI da disciplina, obtido a partir do valor da propriedade dc:identifier, que representa a sigla da disciplina. A esse URI é acrescentado, na última linha desse trecho, a propriedade dc:subject com o URI do recurso na DBpedia.

No fragmento de código a seguir, o grafo da DBpedia para esse recurso é obtido e todas as propriedades associadas são agregadas ao grafo local:

```
remg = Graph('SPARQLStore', identifier="http://dbpedia.org")
remg.open("http://dbpedia.org/sparql")
nodes = remg.predicate_objects(dbres)
for p, o in nodes:
    g.add((dbres, p, o))
```

Nesse fragmento, usamos o método `predicate_objects` para obter todos os predicados e seus valores para o recurso indicado (nesse exemplo, dbpedia:Data_structure) e agregamos a esse recurso (com o método `add`) cada par predicado-objeto retornado.

Com o grafo assim expandido, é possivel fazer a consulta ao grafo local envolvendo os predicados e objetos oriundos do grafo remoto. Por exemplo:

```
result = g.query("""
        PREFIX dbr: <http://dbpedia.org/resource/>
        PREFIX dbo: <http://dbpedia.org/ontology/>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        PREFIX dc: <http://purl.org/dc/elements/1.1/>
        SELECT ?cod ?tit ?res
        WHERE {
        ?d dc:identifier ?cod .
```

```
        ?d dc:subject ?dbp .
        ?d rdfs:label ?tit .
        ?dbp dbo:abstract ?res .
        FILTER (lang(?res) = 'pt')
        }""")

for r in result:
        print(r.cod, r.tit, r.res)
```

A execução dessa consulta no grafo local expandido como acima resulta em:

```
SI201 Estrutura de Dados I Na Ciência da computação, uma
estrutura de dados é um modo particular de armazenamento
e organização de dados em um computador [...] padrões.
```

Se visitarmos a página da DBpedia para o recurso Estrutura de Dados[1], veremos que o texto após o nome da disciplina veio do valor da propriedade dbo:abstract, em português (Figura 4.2).

[1] http://dbpedia.org/page/Data_structure

Figura 4.2: Página da DBpedia para Estruturas de Dados

Capítulo 5

Comentários finais

A concepção da Web Semântica tem aproximadamente vinte anos [4], porém seu potencial ainda está por ser completamente explorado. Um dos objetivos desta disciplina (e deste texto) foi auxiliar na divulgação dessas tecnologias envolvidas na Web Semântica para que possam ser incorporadas no desenvolvimento de novas aplicações, preparadas para serem integradas a essa Web de Dados.

Tim Berners-Lee definiu quatro princípios para a criação de dados conectados abertos:

1. Usar URIs como nomes de coisas. Quando for criar um recurso em um conjunto de dados, esse recurso deve ser internamente identificado por um URI.

2. Usar HTTP para as URIs. Preferencialmente, a base da URI deve estar associada a um endereço URL, de modo que as pessoas possam buscar e ver algo associado a esse nome.

3. Coloque alguma informação útil no endereço associado à URI, preferencialmente usando os vocabulários padronizados (rdf, rdfs, dc...).

4. Incluir no conjunto de dados links para outros URIs, para permitir novas descobertas a partir dos dados originais.

Por fim, os mesmos princípios explorados nos exemplos em Python devem ser facilmente transportados a outras linguagens. Por exemplo, a biblioteca Apache Jena [19] oferece as mesmas funcionalidades para a programação com a Web Semântica e dados ligados em Java.

Referências Bibliográficas

[1] Aki Ariga. Tabula-py. https://github.com/chezou/tabula-py, 2019.

[2] Association for Information Science and Technology. Dublin core metadata initiative. https://www.dublincore.org/, 1995–2019.

[3] Tim Berners-Lee. 5 estrelas dos dados abertos. https://5stardata.info/pt-BR/, 2012.

[4] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[5] Dan Brickley and Libby Miller. Foaf vocabulary specification 0.99. http://xmlns.com/foaf/spec/, 2014.

[6] ECMA International. The JSON data interchange format. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, 2017.

[7] Sergio Fernández, Carlos Tejo, Ivan Herman, and Alexey Zakhlestin. Sparql endpoint interface to python. https://rdflib.github.io/sparqlwrapper/, 2013.

[8] HTTP Working Group. Hypertext Transfer Protocol – HTTP/1.1. https://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-02.html, 1996.

[9] JetBrains. PyCharm. https://www.jetbrains.com/pycharm/, 2000-2019.

[10] NumFOCUS. Python data analysis library. https://pandas.pydata.org/, 2019.

[11] OWL Working Group. Owl 2 web ontology language. https://www.w3.org/TR/owl2-syntax/, 2012.

[12] PhaseIt, Inc. Pypdf2. http://mstamy2.github.io/PyPDF2/, 2016.

[13] Python, v.3.7. https://www.python.org/, 2001-2019.

[14] RDF Schema Working Group. Rdf vocabulary description language. https://www.w3.org/2001/sw/RDFCore/Schema/200203/, 2002.

[15] RDFLib Team. Rdflib documentation. https://rdflib.readthedocs.io/en/stable/, 2009–2013.

[16] Kenneth Reitz. Requests: HTTP para humanos. https://requests.kennethreitz.org/pt_BR/latest/, 2013.

[17] Leonard Richardson. Beautiful soup documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc/, 2004–2015.

[18] SPARQL Working Group. Sparql query language for rdf. https://www.w3.org/TR/rdf-sparql-query/, 2013.

[19] The Apache Software Foundation. Apache jena. https://jena.apache.org/, 2011–2019.

[20] W3C. Skos simple knowledge organization system. https://www.w3.org/2004/02/skos/, 2012.

[21] W3C Working Group. Xml path language (xpath). https://www.w3.org/TR/1999/REC-xpath-19991116/, 1999.

[22] W3C Working Group. URL. https://www.w3.org/TR/url/, 2016.

[23] W3C Working Group. W3C HTML. https://www.w3.org/html/, 2019.

[24] Edd Wilder-James. Doap: Description of a project. https://github.com/ewilderj/doap/wiki, 2019.