COMP 6721 Applied Artificial Intelligence (Winter 2022)

Lab Exercise #08: Knowledge Graphs, Part I

Question 1 Your first task is to translate the knowledge graph you developed on Worksheet #7 into a real RDF graph. Write down the triples using the Turtle format[1] discussed in the lecture. Some notes:

- We have not yet covered the details how predicates like "studies at" are encoded in URIs. The details will come next week; for now, just use a URI like `http://example.org/studiesAt`.
- Remember to use the correct form for DBpedia URIs using the `resource` path, e.g., `http://dbpedia.org/resource/Concordia_University`. The `page` path is a redirect for displaying a human-readable HTML page in a browser and cannot be used for working with RDF triples in a program.

Now, validate your graph:

- Using a browser, go to `http://www.ldf.fi/service/rdf-grapher/` and paste your RDF triples into the text field.
- Make sure "Turtle" is selected as the input format.
- Click the "Visualize" button.
- Examine the results of parsing the input. Correct any mistakes that you might have made accordingly.
- If no mistakes are found in the input, you should see the output in form of a graph; e.g., the first two triples should look like Figure 1.

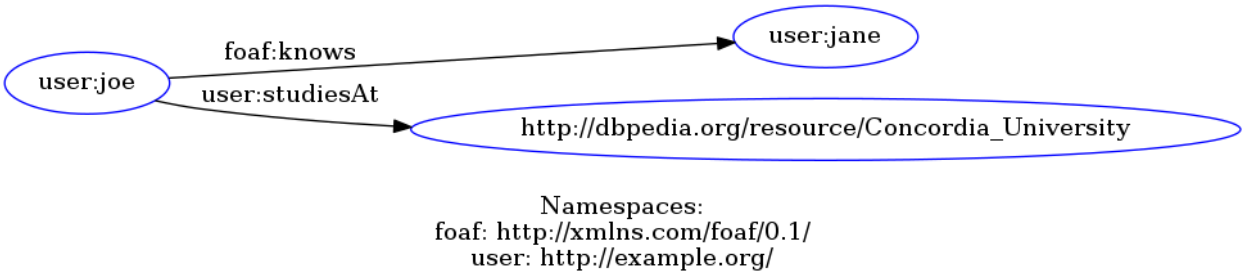

Figure 1: Visualizing the RDF triples encoding the knowledge "Joe knows Jane" and "Joe studies at Concordia University"

[1] Use `.ttl` for the file extension, like in `mytriples.ttl`

There are a number of other RDF-related tools available online; for example, try out the RDF converter at http://rdfvalidator.mybluemix.net/ and convert your Turtle file (`.ttl`) into JSON-LD and RDF/XML to get an idea how these formats look like. The validator at https://www.w3.org/RDF/Validator/ only accepts RDF/XML, but it can also draw you a graph corresponding to your triples.

Note: you will probably encounter references to `rdfs:` (RDF Schema) in examples you find online; we will cover RDFS this week in Lecture #9 *("Knowledge Graphs, Part II")*.

Question 2 For working with RDF and related standards, there are a multitude of libraries available. For example, a popular open source framework for Java is Apache Jena.[2] Here, we will use the RDFLib for Python.[3]

First, we need to install RDFLib. Run the command below in your Conda environment:

```
pip install rdflib
```

For the details, please refer to the RDFLib documentation section, *"Getting started with RDFLib"*.[4]

Now, we want to load the graph you prepared in Question #1. The code you need from RDFLib is `<graph>.parse`, as shown below:

```
import rdflib

# Create a Graph
g = rdflib.Graph()

# Parse an RDF file
g.parse(<RDFFile_Path>) # Provide the RDF file path here
```

You can now print out your whole graph `g` using the code below:

```
# Loop through each triple in the graph (subj, pred, obj)
for s,p,o in g:
   # Print the subject, predicate and the object
   print s,p,o
```

Now, go through the RDFLib documentation section, *"Loading and saving RDF"*, [5] to see how to read and write RDF graphs in different formats. Add code to write the triples in a different format, e.g., RDF/XML and N-Triples.

[2] See http://jena.apache.org/
[3] See https://rdflib.dev/ and https://github.com/RDFLib/rdflib
[4] See https://rdflib.readthedocs.io/en/stable/gettingstarted.html
[5] See https://rdflib.readthedocs.io/en/stable/intro_to_parsing.html

Question 3 We can also create triples directly with RDFLib. To begin, read the documentation section, *"Creating RDF triples"*.[6]

Now, start your program by importing the following libraries:

```
from rdflib import Graph, Literal, RDF, Namespace
from rdflib.namespace import FOAF, RDFS
```

Then, create a new graph instance with:

```
g = Graph()
```

Now you can use the `g.add()` function to add triples to the graph. Write code that adds triples representing:

- `<Joe> <is a> <foaf:Person>`
- `<Joe> <rdfs:label> "Joe"`
- `<Joe> <foaf:knows> <Jane>`

For printing the generated graph, you can use the `g.serialize()` function.

Note: RDFLib has pre-defined name spaces that can be used for graph creation. In order to add another prefix to the knowledge graph, like `foaf`, use the `g.bind()` function.[7]

Question 4 The next step is to learn how to *navigate* graphs, in order to retrieve the knowledge we need, for example, to answer a question.

RDFLib supports basic *triple pattern matching* through the `triples()` function. You can match specific parts of a graph with the methods `objects()`, `subjects()`, `predicates()`, etc. Look at the documentation section *"Navigating Graphs"*[8] for the details.

Now, write a Python program that prints out all triples in your graph for the subject URI corresponding to `Joe`.

Question 5 A powerful feature of knowledge graphs is that we can *merge* different graphs together, in order to connect knowledge from different sources. In this question, we will merge the graph you created above with the knowledge about Concordia from DBpedia.

RDFLib supports various graph operations, like union (addition), intersection, difference (subtraction) and XOR. Like before, we first import RDFLib and create a graph instance, which we'll call `g1`:

```
import rdflib

# Create graph g1
g1 = rdflib.Graph()
```

[6] See https://rdflib.readthedocs.io/en/stable/intro_to_creating_rdf.html

[7] See https://rdflib.readthedocs.io/en/stable/namespaces_and_bindings.html for details

[8] See https://rdflib.readthedocs.io/en/stable/intro_to_graphs.html

We can now parse the DBpedia graph, by directly loading it from the online knowledge base URI:

```
# Parse the DBpedia graph about Concordia
g1.parse("https://dbpedia.org/resource/Concordia_University")
```

To check the number of triples in our DBpedia graph about Concordia University, use the command below. The output should show a count of 2332:

```
# Print the number of triples in the graph
print(len(g1))
```

Similarly, load the graph your create above in Question #1 using the `g.parse()` function. To merge the two graphs (union), use the addition (+) operator.

Print out the merged graph using the `g.serialize()` function and verify that they were indeed merged using Concordia's URI:

Note that you can now answer questions from this merged graph that were not possible to answer from each of them alone, for example, *"In which city is the university located that Joe is studying at?"*