

COMP 6721 Applied Artificial Intelligence (Fall 2023)

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 format¹ 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 in the next lecture & lab; for now, just use a URI like <http://example.org/studiesAt>.
- Remember to use the correct form for Wikidata URIs using the `entity` path, e.g., <http://www.wikidata.org/entity/Q326342> for Concordia. The `wiki/` 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.
- For now, use the URIs `user:joe` and `user:jane` instead of the `user:joe#me` and `user:jane#me` we had on the worksheet.²

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 three triples should look like Figure 1.

There are a number of other RDF-related tools available online; for example, try out the RDF converter at <https://issemantic.net/rdf-converter> 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 in Lecture #9 (“*Knowledge Graphs, Part II*”).

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

²Unlike we did it on the worksheet, the fragment identifier is typically used as part of a namespace, as we will see in the next lecture.

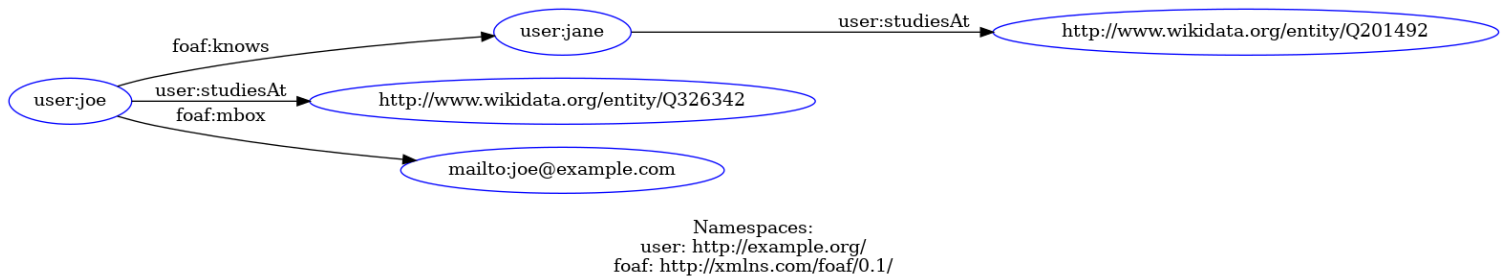


Figure 1: Visualizing the RDF triples encoding the knowledge “Joe’s email is joe@example.com”, “Joe knows Jane”, “Joe studies at Concordia University”, and “Jane studies at McGill University”

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.³ Here, we will use the RDFLib for Python.⁴

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*”.⁵

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 in Turtle format
g.parse(<RDFFile_Path>, format='turtle')
```

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*”,⁶ 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.

³See <http://jena.apache.org/>

⁴See <https://rdflib.dev/> and <https://github.com/RDFLib/rdflib>

⁵See <https://rdflib.readthedocs.io/en/stable/gettingstarted.html>

⁶See 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*”.⁷

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.⁸

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.

Remember that triples are a way to represent graphs, so you can navigate a knowledge graph by querying its triples. 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*”⁹ 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 in above with the knowledge about Concordia from Wikidata.

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`:

⁷See https://rdflib.readthedocs.io/en/stable/intro_to_creating_rdf.html

⁸See https://rdflib.readthedocs.io/en/stable/namespaces_and_bindings.html for details

⁹See https://rdflib.readthedocs.io/en/stable/intro_to_graphs.html

```
import rdflib
```

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

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

```
# Parse the Wikidata graph about Concordia (in Turtle format)
g1.parse("http://www.wikidata.org/entity/Q326342.ttl")
```

To check the number of triples in our Wikidata graph about Concordia University, use the command below. The output should show a large number of triples, more than 3500:¹⁰

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

Similarly, load the graph you 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?"*. Try running your code from Question 4 on this merged graph to see all the information you now have available. Then, write some triple patterns to print out the city where Joe is studying. This exercise serves as an introduction to the expansive capabilities of knowledge graphs in AI systems. By further exploring linked data, such as loading additional details about Montreal (Q340) from Wikidata, an autonomous agent can enhance its understanding of related concepts like the country in which the city is located. This approach exemplifies the power of knowledge graphs in providing comprehensive answers to complex queries. It underscores the importance of data provenance, ensuring that all responses generated by the AI are backed by traceable and verifiable sources. The integration of diverse knowledge sources like this can be applied in numerous real-world applications, ranging from intelligent personal assistants to advanced data analysis in research, offering a glimpse into the potential of knowledge graphs in building informed, reliable AI solutions.

¹⁰The exact number of triples changes over time as knowledge about Concordia is updated on Wikidata.