This lab sheet and all other materials can be found on GitHub here: https://github.com/QMUL-ECS735P/lab-week-4/

Introduction to SPARQL

> Session objectives: * Write and execute SPARQL queries * Merge RDF data from multiple sources

1. Introduction

As with the last lab, we'll be using Python to create SPARQL database queries and constructing RDF graphs from the result of those queries.

If you're new to Python it would be wise to keep the basic tutorial handy for reference:

- https://docs.python.org/3.7/tutorial/

Python scripts are run by calling the python command, followed by the name of the python script to execute, for example:

```
python hello_world.py
```

It is also possible to run python in an interactive mode known as a REPL, to do this simply call python without any arguments:

```
python
```

2. Writing SPARQL Queries

In this lab we'll be using an external API endpoint hosted by DBpedia. DBpedia is a community-run resource that extracts linked data from Wikipedia.

To write SPARQL queries as Python strings, we'll be making good use of multi-line strings

```
'''
this is
a multi-line
string
'''
```

Begin by importing the SRARQLWrapper library:

```
!pip install sparqlwrapper # This is only necessary for the online notebook
```

```
from SPARQLWrapper import SPARQLWrapper, JSON
```

DBpedia has a SPARQL endpoint located at `http://dbpedia.org/sparql`. We're going to wrap that endpoint using the SPARQLWrapper, which means we can use a lot of handy methods to construct, execute, and receive queries.

```
endpoint = SPARQLWrapper('http://dbpedia.org/sparql')
```

Now we're going to write a simple SPARQL query to get some information from the database:

```
endpoint.setQuery('''
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpr: <http://dbpedia.org/resource/>
SELECT ?label
WHERE { dbpr:Asturias rdfs:label ?label }
''')
```

This doesn't execute the query. Before we execute the query, look at how it is constructed. Does it look familiar?

The endpoint can respond in a variety of different serialisation forats, for today we'll be using JSON.

```
endpoint.setReturnFormat(JSON)
```

Now to execute the query and print the results:

```
results = endpoint.query().convert()
for result in results['results']['bindings']:
  print(result['label']['value'])
```

Let's take a moment to unpack what's happening here. First we execute the query with `endpoint.query()` and then tell the SPARQLWrapper library to convert the response into Python nested dictionaries with `.convert()`.

We then iterate over the array contained in `results['results']['bindings']` and print each value contained in `result['label']['value']`.

To get a better idea of how the results dictionary is structured we can use the pprint library to pretty print the data.

```
!pip install prettyprinter # This is only necessary for the online notebook
```

```
from pprint import PrettyPrinter
pretty = PrettyPrinter(indent = 2)
pretty.pprint(results)
```

Take a moment to familiarse with this data structure, all our responses from DBpedia will be in a similar format.

Now we'll use the FILTER constraint to filter out all responses other than those in English and Spanish.

```
endpoint.setQuery('''
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpr: <http://dbpedia.org/resource/>
SELECT ?label
WHERE {
  dbpr:Asturias rdfs:label ?label .
  FILTER( LANG(?label)="es" || LANG(?label)="en") .
}
''')
```

As before, we'll execute the query and print the results:

```
results = endpoint.query().convert()
```

```
for result in results['results']['bindings']:
  print(result['label']['value'])
```

We didn't need to specify the endpoint url or return format because we already did that previously. You can read more about the FILTER constraint here:

- https://jena.apache.org/tutorials/sparql_filters.html.

Try changing the query to select responses in some languages that aren't English or Spanish. (Hint, run the query with no `FILTER` and look at the `xml:lang` predicate).

3. Using SPARQLQrapper and RDFlib

Now we'll use our knowledge from last week to create an RDF graph and populate it with the result from some SPARQL queries.

For clarity, let's import everything we need first (even though some of these things are already imported). You'll likely need these imports for the coursework, so its handy to lay them out here.

```
import rdflib
from rdflib import plugin
from rdflib.graph import Graph
from rdflib.namespace import Namespace
from SPARQLWrapper import SPARQLWrapper, JSON, XML
endpoint = SPARQLWrapper('http://dbpedia.org/sparql')
```

We'll be using a CONSTRUCT query which tells SPARQL to construct new rdf triples based on the result of the query. For simplicity we'll simply create an exact copy of the graph.

```
query = '''
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
CONSTRUCT {
    <%(uri)s> a ?type .
    <%(uri)s> ?property ?value .
    <%(uri)s> rdfs:label ?label .
    ?value rdfs:label ?vlabel .
    ?property rdfs:label ?plabel .
}
WHERE {
    <%(uri)s> a ?type.
    <%(uri)s> ?property ?value .
    <%(uri)s> rdfs:label ?label .
    ?value rdfs:label ?vlabel .
    ?property rdfs:label ?plabel .
    FILTER( LANG(?label)="es" || LANG(?label)="en" ) .
}
'''
```

The above query string is using a Python feature called named placeholders. We use the % operator to replace the named placeholders with another string, it saves us some typing!

```
endpoint.setQuery(query % { 'uri': 'http://dbpedia.org/resource/Asturias' })
```

This time we'll set the return format to XML:

```
endpoint.setReturnFormat(XML)
```

And then execute the query (note: this may take a little time, don't worry!)

```
graph = endpoint.query().convert()
```

The resulting graph is quite large, to print it in a notebook would take quite a while!

If you're cleverer than me you might want to look at the `itertools` package and see if you can print just a subset of the graph. Maybe something like:

```
import itertools
for s, p, o in itertools.islice(graph, 100):
  print("subject: " + str(s))
  print("predicate: " + str(p))
  print("object: " + str(o) + "\n")
```

The graph variable is an rdflib graph like the ones we were creating and manipulating last week. Today we'll use a new method, graph.query to write SPARQL queries for our local rdf graph.

```
query = '''
PREFIX dbpp: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?location ?party ?leader
WHERE {
    <%(uri)s> rdfs:label ?location .
    <%(uri)s> dbpp:leaderName ?leaderResource .
    ?leaderResource rdfs:label ?leader .
    <%(uri)s> dbpp:leaderParty ?partyResource .
    ?partyResource rdfs:label ?party .
    FILTER( LANG(?location)="%(lang)s" && LANG(?party)="%(lang)s" && LANG(?lea
}
'''
replacements = {
  'uri': 'http://dbpedia.org/resource/Asturias',
  'lang': 'es'
}
```

Now we can run the query just like we did for our SPARQL endpoint:

```
results = graph.query(query % replacements)
for row in results:
  print('The leading party of %s is %s. Their leader is %s.' % row)
```

Our initial construct query has both English and Spanish triples. Try re-running the query above to select English triples instead of Spanish ones.

If you poked around using `itertools` to inspect the construct query response, you might want to play around here and construct your own queries.

4. The Interactive DBpedia Endpoint

DBpedia provides a web-based tool for writing and testing SPARQL queries. You can access it here: http://dbpedia.org/sparql/

Type or copy the following query into the text area:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpp: <http://dbpedia.org/property/>
```

```
SELECT DISTINCT ?location ?leader ?party
FROM <http://dbpedia.org>
WHERE {
  dbpedia:Asturias rdfs:label ?location ;
  dbpp:leaderName ?leaderResource .
  ?leaderResource rdfs:label ?leader .
  dbpedia:Asturias dbpp:leaderParty ?partyResource .
  ?partyResource rdfs:label ?party .
}
```

You can then click the Execute Query to run the query and explore the results. This will generate a visualisation of the query that can be handy to see what it is you're selecting.