Lab 9 – RDF in code

Today we are going to manipulate RDF in code mostly using RDFlib in python.

T1: Write python code to load the camera ontology built previously (`https://mdaquin.github.io/o/sw-lab7-camera`) into an RDFlib graph. Test that it loaded properly by printing on the screen the list of all subclasses of the class CameraModel (`https://mdaquin.github.io/o/sw-lab7-camera#CameraModel`).

T2: The construct query below was a possible answer to T5 of the last lab session: It builds a small graph of the known camera models in DBpedia with their manufacturers. Apply it on the DBpedia SPARQL endpoint (`https://dbpedia.org/sparql`), save the result in a file, and load it in the same graph as the ontology in RDFlib. Test that it worked by printing a list of all instances of the class Organisation (`https://mdaquin.github.io/o/sw-lab7-camera#Organisation`).

```
prefix ico: <https://mdaquin.github.io/o/sw-lab7-camera#>
construct {
   ?cam a ico:CameraModel ;
             rdfs:label ?caml .
   ?maker a ico:Organisation ;
                ico:manufacturerOf ?cam ;
                rdfs:label ?makerl .
} where {
   ?cam a <http://dbpedia.org/class/yago/Camera102942699> .
   ?cam <http://dbpedia.org/property/maker> ?maker .
   ?cam rdfs:label ?caml .
   ?maker rdfs:label ?makerl .
    filter (lang(?caml) = "en" && lang(?makerl) = "en") .
}
```

T3: Would it be possible to load the result of this query without first saving it into a file?

T4: The select query below gets the information about the lenses compatible with the camera models in DBpedia. Use an HTTP request to execute this query from Python, and for each result pair (camera, lens), add a triple in the RDLlib graph using the compatibleWith property as predicate (`https://mdaquin.github.io/o/sw-lab7-camera#compatibleWith`). Add some code that counts the number of cameras compatible with each type of lens in the graph.

```
select distinct ?cam ?l where {
   ?cam a <http://dbpedia.org/class/yago/Camera102942699> .
   ?cam <http://dbpedia.org/property/lens> ?l .
   filter (isURI(?l))
}
```

T5: Write some code that will load the RDF description of each camera manufacturer from DBpedia into the RDF graph in RDFlib. What is the size of the graph before and after this operation?

T6: The turtle code below adds the definition of manufacturer as the inverse of manufacturerOf and of Japanese camera as a camera which manufacturer is located in Japan. Write the python code that makes the corresponding inferences for all the cameras in the graph (i.e. make them instances of JapaneseCamera when their manufacturer is located in Japan.

```
@prefix : <https://mdaquin.github.io/o/sw-lab7-camera#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

:manufacturer owl:inverseOf :manufacturerOf .

:JapaneseCamera rdf:type owl:Class ;
     owl:equivalentClass [
         owl:intersectionOf (
            :CameraModel
            [ rdf:type owl:Restriction ;
              owl:onProperty :manufacturer ;
              owl:someValuesFrom [
                    rdf:type owl:Restriction ;
                    owl:onProperty <http://dbpedia.org/ontology/location> ;
                    owl:hasValue <http://dbpedia.org/resource/Japan>]])] .
```

T7: Look at the owlrl module/plugin for RDFlib (`https://owl-rl.readthedocs.io/en/latest/stubs/owlrl.html#module-owlrl`), which is a simplified OWL reasoner. Could it have made those inferences?