

RDFlib

ECSDI

Curso 2021/2022

CS-FIB-UPC 



- ⊙ La librería `rdflib` permite crear, manipular, consultar y almacenar grafos RDF (y OWL)
- ⊙ Documentación completa en <https://rdflib.readthedocs.org/en/latest/>

Elementos básicos de RDFlib

- ⊙ La estructura básica es el objeto Graph
- ⊙ Permite almacenar las definiciones y hechos que representan un dominio
- ⊙ Un grafo estará compuesto por tripletas (sujeto/predicado/objeto)
- ⊙ Estos elementos podrán ser nodos (con un URI) o literales
- ⊙ Podemos crear un nodo:
 - con una URI específica usando la clase `URIRef`
 - con nodos vacíos mediante la clase `BNode`
 - con literales mediante la clase `Literal`

```
1 from rdflib import URIRef, BNode, Literal
2
3 pedro = URIRef('http://mundo.mundial.org/persona/pedro')
4 maria = BNode()
5
6 nombre = Literal('Pedro')
7 edad = Literal(22)
```

Podemos definir un espacio de nombres y crear nodos dentro de este espacio mediante la clase `Namespace`

```
1 from rdflib import Namespace
2
3 myns = Namespace('http://my.namespace.org/personas')
4
5 a = myns.tom
6 #rdflib.term.URIRef(u'http://my.namespace.org/personas/tom')
```

Hay definidas clases que representan espacios de nombre usuales como `RDF` y `FOAF`.

Podemos añadir tripletas a un grafo mediante el método `add`

```
1 from rdflib import URIRef, BNode, Literal, Namespace, RDF, FOAF
2
3 g = Graph()
4 mm = Namespace('http://mundo.mundial.org/persona/')
5
6 pedro = mm.pedro
7 maria = mm.maria
8
9 g.add((pedro, RDF.type, FOAF.persona))
10 g.add((maria, RDF.type, FOAF.persona))
11 g.add((pedro, FOAF.name, Literal('Pedro')))
12 g.add((maria, FOAF.name, Literal('Maria')))
13 g.add((pedro, FOAF.knows, maria))
```

Podemos modificar el valor de una propiedad funcional (cardinalidad 1) mediante el método set:

```
1 g.add((pedro, FOAF.age, Literal(22)))
```

```
2  
3 g.set((pedro, FOAF.age, Literal(23)))
```


Se eliminan tripletas con `remove`, indicando la tripleta específica o usando `None` para dejar sin especificar algún elemento.

```
1 g.add((pedro, RDF.type, FOAF.persona))
2 g.add((maria, RDF.type, FOAF.persona))
3
4 g.add((pedro, FOAF.age, Literal(22)))
5 g.add((maria, FOAF.age, Literal(23)))
6
7 # Eliminar la edad de pedro
8 g.remove((pedro, FOAF.age, Literal(22)))
9
10 # Eliminar todas las tripletas que se refieren a maria
11 g.remove((maria, None, None))
```

Para cargar un fichero deberemos saber el formato (xml, turtle, n3, ...) y utilizar el método parse del objeto `Graph`

```
1 g= Graph()  
2 g.parse('http://my.ontology.org/ontologia.owl',  
3         format='xml')
```

Se puede cargar un fichero local, o usar una URL para cargar ficheros remotos

Para grabar los datos que tenemos en un grafo RDF podemos usar el método `serialize`

```
1 g = Graph()
2 n = Namespace('http://ejemplo.org/')
3
4 p1 = n.persona1
5 v = Literal(22)
6 g.add((p1, FOAF.age, v))
7 g.serialize('a.rdf')
```

Métodos de consulta

Podemos iterar sobre un grafo obteniendo todas sus tripletas

```
1 g = Graph()
2 n = Namespace('http://ejemplo.org/')
3
4 p1 = n.persona1
5 v = Literal(22)
6 g.add((p1, FOAF.age, v))
7
8 for s, p, o in g
9     print s, p, o
```

retornaría los valores de la tripleta que hemos almacenado

También podemos seleccionar tripletas del grafo a partir de un sujeto específico

```
1 for p, o in g[p1]
2     print p, o
```

retornaría los predicados y objetos que están relacionados con el sujeto p1

El operador `in` esta sobrecargado para los grafos, asi que podemos hacer consultas sencillas de existencia con esta sintaxis:

```
1 if (mm.pedro, RDF.type, FOAF.persons) in g:  
2     print "Pedro es una persona"
```

Podemos usar también el método `triples` para hacer una consulta simple, poniendo `None` en la parte de la tripleta que queremos que sea variable

```
g.triples((None, FOAF.age, Literal(22)))
```


También tenemos las siguientes funciones para hacer consultas:

- ⊙ `objects`, retorna los objetos relacionados con el sujeto y predicado que se pasa como parámetro.
- ⊙ `predicates`, retorna los predicados relacionados con el sujeto y objeto que se pasa como parámetro.
- ⊙ `subjects`, retorna los sujetos relacionados con el predicado y objeto que se pasa como parámetro.

También tenemos las siguientes funciones para hacer consultas:

- ⊙ `predicate_objects`, retorna los predicados y objetos relacionados con el sujeto que se pasa como parámetro.
- ⊙ `subject_objects`, retorna los sujetos y objetos relacionados con el predicado se pasa como parámetro.
- ⊙ `subject_predicates`, retorna los sujetos y predicados relacionados con el objeto que se pasa como parámetro.

Consulta mediante SPARQL

Podemos hacer consultas complejas y modificaciones en el grafo RDF con SPARQL mediante query y update

“Personas que tienen más de 18 años, con sus nombres”

```
1 res = g.query("""
2     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3     SELECT DISTINCT ?a ?name
4     WHERE {
5         ?a foaf:age ?edad .
6         ?a foaf:name ?name .
7         FILTER (?edad > 18)
8     }
9 """)
```

Con el parámetro `initNs` se le puede pasar un diccionario con espacios de nombre para no tener que añadir la definición de los prefijos usados

```
1 res = g.query("""
2     SELECT DISTINCT ?a ?name
3     WHERE {
4         ?a foaf:age ?edad .
5         ?a foaf:name ?name .
6         FILTER (?edad > 18)
7     }
8     """, initNs = {'foaf': FOAF})
```

Si queremos hacer modificaciones en el grafo podemos hacer

```
1 g.update("""
2     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3     INSERT DATA
4     {
5         juan a foaf:person;
6             foaf:name "Juan".
7     }
8 """)
```