

**COMP 6741**

**Intelligent Systems**

Project Report

**Studybot**



Submitted To: Prof. Dr. René Witte

Date: 23 April, 2021

Team **FL\_G\_02**

Indrayan Ittoo (40044893)

Raj Mistry (40119206)

Table of Contents

[1. Abstract 2](#_Toc70110073)

[2. Knowledge base 3](#_Toc70110074)

[2.1 Vocabulary 3](#_Toc70110075)

[2.2 Schema 3](#_Toc70110076)

[2.2.1 Prefixes 4](#_Toc70110077)

[2.2.2 Specifications 4](#_Toc70110078)

[2.2.3 Class Modeling 4](#_Toc70110079)

[2.2.4 Property Modeling 6](#_Toc70110080)

[3. Data extraction 6](#_Toc70110081)

[3.1 Topic Generation 7](#_Toc70110082)

[4. Knowledge base 8](#_Toc70110083)

[5. Queries 9](#_Toc70110084)

[6. Chatbot using Rasa 13](#_Toc70110085)

[6.1 Intents 13](#_Toc70110086)

[6.2 Entities 13](#_Toc70110087)

[6.3 Stories 14](#_Toc70110088)

[6.4 Actions 14](#_Toc70110089)

[6.4.1 Program structure 14](#_Toc70110090)

[6.4.2 Examples – Sample output for each questions 14](#_Toc70110091)

[7. Statistics 17](#_Toc70110092)

[8. Running program 17](#_Toc70110093)

[9. References 17](#_Toc70110094)

# Abstract

Students often have a lot of questions about courses, course topics, course materials and so on. To answer these questions, they often have to navigate through several webpages, or try to ask friends who have completed the same course. This project proposed a new “friend” referred as Studybot, which is an intelligent agent that can answer university-related questions using a knowledge graph and natural language processing.

In the first part of the project, we are going to focus on building the knowledge graph. Therefore, we are going to explore the approach to build the knowledge graph, the vocabularies used or extended.

Then we are going to write a series of competency questions which will ascertain that the knowledge graph can support these queries.

# Knowledge base

## 2.1 Vocabulary

For this project, we explored some of the public vocabularies, including:

* RDF
* FOAF
* OWL
* Dublin Core
* VIVO
* BIBO
* EVENT
* SKOS

Most of the vocabularies and ontologies does not completely fit the requirements and they have to be extended or should use other vocabularies.

The VIVO vocabulary [5][6] provides a good and complete option, but for the sake of exploring, we decided to go with Dublin Core [3].

*There are fifteen terms of the Dublin Core™ Metadata Element Set (also known as "the Dublin Core") plus several dozen properties, classes, datatypes, and vocabulary encoding schemes. The "Dublin Core" plus these extension vocabularies are collectively referred to as "DCMI metadata terms" ("Dublin Core terms" for short). These terms are intended to be used in combination with metadata terms from other, compatible vocabularies in the context of application profiles.*

*DCMI metadata terms are expressed in RDF vocabularies for use in Linked Data. Creators of non-RDF metadata can use the terms in contexts such as XML, JSON, UML, or relational databases by disregarding both the global identifier and the formal implications of the RDF-specific aspects of term definitions. Such users can take domain, range, sub property, and subclass relations as usage suggestions and focus on the natural-language text of definitions, usage notes, and examples.*

The Dublin core provides a good basis for building out own vocabulary.

## 2.2 Schema

The diagram gives and overview of the main classes and properties extended from Dublin Core.



## 2.2.1 Prefixes

|  |  |  |
| --- | --- | --- |
| **Prefix** | **URI** | **Comments** |
| rdfs | <http://www.w3.org/2000/01/rdf-schema#> |  |
| rdf | <http://www.w3.org/1999/02/22-rdf-syntax-ns#> |  |
| xsd | <http://www.w3.org/2001/XMLSchema#> |  |
| dbo | <http://www.dbpedia.org/ontology/> |  |
| dbr | <http://www.dbpedia.org/resource/> |  |
| dcterms | <http://purl.org/dc/terms/> |  |
| dcmitype | <http://purl.org/dc/dcmitype/> |  |
| dc | <http://purl.org/dc/elements/1.1/> |  |
| uni | <http://uni.io/schema#> |  |
| unidata | <http://uni.io/data#> |  |

## 2.2.2 Specifications

The following table provide the vocabulary that has been used in the schema

|  |  |  |
| --- | --- | --- |
| **Vocabulary** | **Type of Term** | **Comments** |
| rdf | rdf:type | Defines a type, used “a” |
| rdf:Property | Defines a property |
| rdfs | rdfs:Class | Identifies a class |
| rdfs:subClassOf | Identifies a sub class and the class from which it is sub classed |
| rdfs:domain | Identifies the domain |
| rdfs:range | Identifies the range |
| rdfs:seeAlso | Provides a link to a related resource such as website |
| rdfs:label | Provide a label |
| rdfs:comment | Provides a description |
| dcterms | dcterms:Agent | A resource that acts, or has the power to act, in that case a University or Course |
| dcterms:subject | A topic of the resource. Can use a URI or literal |
| dcterms:relation | A related resource. Recommended to identify resource by a URI, but can use a string conforming formal identification |
| dcterms:BibliographicResource | A bibliographic resource, for example a book, article or other documentary resource |
| dcmitype | dcmitype:Event | A class that is non-persistent, time based occurrence |

## 2.2.3 Class Modeling

This section provides a description of the different terms used in the schema and for building the data components

|  |  |
| --- | --- |
| **University: Class**  University is defined as a Class, and extends the Agent class from Dublin Core. The University is the primary agent that has the power to act | |
| Schema | Data example |
| uni:University  a rdfs:Class ;  rdfs:subClassOf dcterms:Agent ;  rdfs:label "University"@en ;  rdfs:comment "University information" . | unidata:Concordia\_University  a uni:University ;  dcterms:title "Concordia University"@en ;  rdfs:seeAlso dbr:Concordia\_University . |
|  | |

|  |  |
| --- | --- |
| **Course: Class**  Course is defined as a Class, and extends the Agent class from Dublin Core. The class is an agent that has the power to act. The following terms are also used: | |
| Schema | Data example |
| uni:Course  a rdfs:Class ;  rdfs:subClassOf dcterms:Agent ;  rdfs:label "Course"@en ;  rdfs:comment "Courses offered at University"@en . | unidata:COMP6741  a uni:Course ;  dcterms:title "Intelligent Systems"@en ;  dcmitype:subject "COMP" ;  dcmitype:identifier "6741";  dcterms:description "Knowledge representation"@en ;  rdfs:seeAlso <https://moodle.concordia.ca> ;  uni:topic dbr:Intelligent\_Systems ;  dcterms:isPartOf unidata:Concordia\_University ;  uni:hasContent unidata:Doc1 . |
| **Terms associated with Course**: | |
| dcterms:title | Identifies the name of the course |
| dcmitype:subject | Identifies subject of course, e.g. COMP, SOEN |
| dcmitype:identifier | Identifies the number of the course, e.g 6741 |
| dcterms:description | Provides a description of the course |
| rdfs:seeAlso | Link to webpage with the course information |
| uni:topic | Identifies topic of course, linked to DBpedia |
| dcterms:isPartOf | Identifies the University which course belongs to |
| uni:hasContent | Content, mainly the course outline |

|  |  |
| --- | --- |
| **Lecture: Class**  A lecture is an event that belongs to a course. Lecture extens the event class from Dublin Core as it is a non-persistent time base occurence | |
| Schema | Data example |
| uni:Lecture  a rdfs:Class ;  rdfs:subClassOf dcmitype:Event ;  rdfs:label "Lecture"@en ;  rdfs:comment "Information about lecture"@en . | unidata:COMP6741L01  a uni:Lecture ;  dcmitype:identifier "1";  dcterms:title "Introduction to Intelligent S."@en ;  rdfs:seeAlso <https://moodle.concordia.ca> ;  uni:topic dbr:Intelligent\_Systems ;  uni:hasContent unidata:Doc02 ;  dcterms:isPartOf unidata:COMP6741 . |
| **Terms associated with Lecture**: | |
| dcmitype:identifier | Identifies lectures number |
| dcterms:title | Identifies the lecture name |
| rdfs:seeAlso | Link to webpage with the lecture information |
| uni:topic | Identifies topic of a lecture, linked to DBpedia |
| dcterms:isPartOf | Identifies the course to which lecture belongs to |
| uni:hasContent | Content, any documents that is associated with lecture such as slides, worksheets, etc |

|  |  |
| --- | --- |
| **Event: Class**  An event extends a lecture as it is associated with the lecture and is also time based. It can be any event associated with a lecture, such as a tutorial or lab | |
| Schema | Data example |
| uni:Event  a rdfs:Class ;  rdfs:subClassOf uni:Lecture ;  rdfs:label "Lecture Event"@en ;  rdfs:comment "Events associated with a Lecture such as Lab ror Tutorial"@en . | unidata:L674101  a uni:Event ;  dcterms:type "LAB" ;  uni:isPartOf unidata:COMP6741L01 ;  uni:topic dbr:Python . |
| **Terms associated with Event**: | |
| dcterms:type | Identifies type of event, can be lab, tutorial or some other event |
| dcterms:isPartOf | Identifies the lecture to which event belongs to |
| uni:hasContent | Identifies content specific to that lab |
| uni:topic | Identifies topic of lab, linked to DBpedia |

## 2.2.4 Property Modeling

|  |  |
| --- | --- |
| **topic: Property**  A topic is a sub property of subject, which identifies a link to DBpedia | |
| Schema | Data example |
| uni:topic  a rdf:property ;  rdfs:subPropertyOf dcterms:subject ;  rdfs:label "Topic"@en ;  rdfs:comment "URI"@en . | unidata:L674101  a uni:Event ;  dcterms:type "LAB" ;  uni:isPartOf unidata:COMP6741L01 ;  uni:topic dbr:Python . |
|  | |

|  |  |
| --- | --- |
| **hasContent: Property**  Identifies the contents that are associated with a course, lecture or event | |
| Schema | Data example |
| uni:hasContent  a rdf:property ;  rdfs:subClassOf dcterms:relation ;  rdfs:label "Content"@en ;  rdfs:range dcterms:BibliographicResource  rdfs:comment "Content associated with a course, lecture"@en . | unidata:COMP6741L01  a uni:Lecture ;  dcmitype:identifier "1";  dcterms:title "Introduction to Intelligent Systems"@en ;  rdfs:seeAlso <https://moodle.concordia.ca> ;  uni:topic dbr:Intelligent\_Systems ;  uni:hasContent unidata:Doc02 ;  dcterms:isPartOf unidata:COMP6741 . |
| The content itself is not described as part of the schema. The content is a BibliographicResource defined as follows:  unidata:Doc06  a dcterms:BibliographicResource ;  dcterms:type "LECTURE" ;  dcterms:source /University/COMP6741/slides03.pdf> . | |
| dcterms:type | Identifies the type of content, e.g. OTHER, SLIDES, WORKSHEET |
| dcterms:source | Locates the resource |

# Data extraction

The data was extracted from the <https://opendata.concordia.ca/datasets/>. The following files were used to create the .csv files we require for the project:

* CU\_SR\_OPEN\_DATA\_CATALOG-37296852.csv (To get Course ID, Subject, Number, Title)
* CU\_SR\_OPEN\_DATA\_CATALOG\_DESC.csv (To get Course Description)
* CATALOG.csv and Experiential Learning.csv (To get Course Website/seeAlso)

The required files for project were generated by doing some work manually like extracting required columns from, removing duplicates, adding seeAlso data, data for two courses and other work like merging 2 .csv files based on common column value is done using small python script: data.py.

|  |
| --- |
| import pandas as pd  data1 = pd.read\_csv("Data.csv",encoding= 'latin1')  data2 = pd.read\_csv("DES.csv",encoding= 'latin1')  #merge function by setting how='left'  output = pd.merge(data1, data2, on='Course ID', how='left')  output.to\_csv("output.csv")  data3 = pd.read\_csv("output.csv",encoding= 'latin1')  data4 = pd.read\_csv("WEBSITE.csv",encoding= 'latin1')  output1 = pd.merge(data3, data4, on='Course ID', how='left')  output1.to\_csv("course.csv") |

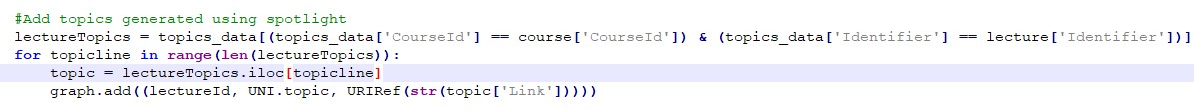
\*data.py\*

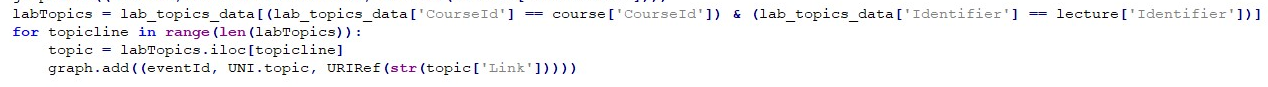
The following files are generated and used for creating Knowledge Graph:

1. course\_data.csv: contains all the courses and descriptions extracted from opendata.concordia.
2. lecture\_data.csv: contains the lectures information for 2 courses that were manually updated.
3. event\_data.csv: contains the lab events information for 2 courses that were manually updated.
4. content\_data.csv: contains the content information for 2 courses that were manually updated.
5. lab\_content\_data.csv: contains the lab content for 2 courses that were manually added.
6. topics.csv : contains topics for 2 courses for lectures generated from dbpedia spotlight sunig lecture materials(slides and Worksheets)
7. lab\_topics.csv : contains topics for 2 courses for labs generated from dbpedia spotlight sunig lab material

## 3.1 Topic Generation

* In order to generate topics for the two courses we followed the following steps:
* First we collected all the lecture materials for the 2 courses i.e. all the lecture slides, worksheets and lab pdfs (for COMP 6741 we saved content from moodle into pdf)
* After that we converted these pdfs into separate text file using **‘pdftotxt.py’** file. While creating this text file we removed duplicate lines in text file for particular material (slide01 , worksheet02,etc) in order to minimize unnecessary duplicate requests on dbpedia spotlight.
* Than we used this text files in **‘topics\_generation.py’** file which is reading this files line by line and making request on local dbpedia spotlight server to get topics from it. During this for lecture we removed the duplicate topics for particular lecture and lab.
* This topics are saved in topics.csv and lab\_topics.csv file.
* After this we updated **‘kbuilder.py’** which is used to generate knowledge graph.
* We read the 2 topics csv files and iterate through its line to add topics for 2 courses.





# Knowledge base

The building of the knowledge base uses python programming language and the following librairies:

1. pandas – for reading the .csv data files
2. uuid – for generating unique identifies for the resources
3. rdflib –
   1. Graph – for creating knowledge graph
   2. Namespace – to manage the namespace use and creation
   3. RDFS, RDF DC, DCTERMS – provided namespace in rdflib
   4. URIRef – for creating URI
   5. Literal – for adding literal vales, not URI

The following diagram give a general flow of the code structure.



* The university is added through the code to the graph

|  |
| --- |
| #Add Univeristy to graph  graph.add( (URIRef(UNIDATA.Concordia\_University), RDF.type , UNI.University))  graph.add( (URIRef(UNIDATA.Concordia\_University), DCTERMS.title , Literal("Concordia University", lang="en")))  graph.add( (URIRef(UNIDATA.Concordia\_University), RDFS.seeAlso, DBR.Concordia\_University)) |

* For each line in the course file, the information for the course is read, and inserted into the graph.

The ID of the course is obtained from the course\_data.csv file

|  |
| --- |
| for courseline in range(len(course\_data)):  course = course\_data.iloc[courseline]  courseId = UNIDATA[str(course['CourseId'])]  graph.add((courseId, RDF.type, UNI.Course))  …. |

* For each course, the lecture for the course and information of lectures are inserted in graph

The ID of the lecture is a unique Id generated. Given temporal existence of lecture.

|  |
| --- |
| #For each course, add lectures  courseLectures = lecture\_data[lecture\_data['CourseId'] == course['CourseId']]  for lectureline in range(len(courseLectures)):  lecture = courseLectures.iloc[lectureline]  lectureId = UNIDATA["l" + str(uuid4())]  graph.add((lectureId, RDF.type, UNI.Lecture))  … |

* For each lecture, the events (lab or tutorial) and the contents of the lecture are inserted in the graph

|  |
| --- |
| #Content associated withe a lecture  lectureContents = content\_data[(content\_data['CourseId'] == course['CourseId']) & (content\_data['Identifier'] == lecture['Identifier'])]  for contentline in range(len(lectureContents)):  content = lectureContents.iloc[contentline]  ….  #Events (labs/tutorials) associated with a lecture  lectureEvents = event\_data[(event\_data['CourseId'] == course['CourseId']) & (event\_data['Identifier'] == lecture['Identifier'])]  for eventline in range(len(lectureEvents)):  event = lectureEvents.iloc[eventline]  eventId = UNIDATA["e" + str(uuid4())] |

* Content creation. A unique ID is created for each content

Local content should be place in the fuseki server in the /webapp folder. The namespace LOCAL was used in the code

|  |
| --- |
| def add\_documents(location, type):  contenId = UNIDATA["c" + str(uuid4())]  graph.add((contenId, RDF.type, DCTERMS.BibliographicResource))  graph.add((contenId, DCTERMS.type, Literal(type)))  graph.add((contenId, DCTERMS.source, URIRef(location)))  return contenId |

# Queries

* The following queries have been used to test the graph. The queries can be found in the accompanying queries.sparql file. In the report, we will limit to 10 queries, but more are defined in the queries.sparql file

# Question 1

# What is the course [title] and [description] of [subject] [number]?

# What is the course title and description of COMP 6741?

SELECT ?title ?descr

WHERE {

?course dcmitype:subject "COMP" .

?course dcmitype:identifier "6741" .

?course dcterms:title ?title .

?course dcterms:description ?descr

}

# Question 2

# Which topics are covered in [subject] [number] lectures?

# Which topics are covered in [COMP] [6741] lectures?

SELECT ?course ?lecture ?topic

WHERE {

?course1 dcmitype:subject "COMP" .

?course1 dcmitype:identifier "6741" .

?course1 dcterms:title ?course .

?lecture1 dcterms:isPartOf ?course1 .

?lecture1 dcmitype:identifier ?lecture .

?lecture1 uni:topic ?topic

} ORDER BY ?lecture

# Question 3

# Which lecture of [subject][number] covers [topic]?

# Which lecture of [COMP][6741] covers [Knowledge\_grap]?

SELECT ?number

WHERE {

?course dcmitype:subject "COMP" .

?course dcmitype:identifier "6741" .

?lecture dcterms:isPartOf ?course .

?lecture uni:topic dbr:Knowledge\_Graph .

?lecture dcmitype:identifier ?number

}

# Question 4

# How many courses are offered at Concordia University?

SELECT (count(?courseId) as ?CourseCount)

WHERE{

?courseId rdf:type uni:Course

}

# Question 5

# What are the recommended reading materials for [subject][number] by lecture?

# What are the recommended reading materials for [COMP][6741] by lecture?

SELECT ?number ?source

WHERE {

?course dcmitype:subject "COMP" .

?course dcmitype:identifier "6741" .

?lecture uni:hasContent ?content .

?lecture dcterms:isPartOf ?course .

?lecture dcmitype:identifier ?number .

?content dcterms:type "READING" .

?content dcterms:source ?source .

} order by ?number

# Question 6

# What are the contents in [subject][number] for each lecture?

# What are all the contents for COMP 6741 for each lecture?

SELECT ?number ?content\_type ?source

WHERE {

?course dcmitype:subject "COMP" .

?course dcmitype:identifier "6741" .

?lecture uni:hasContent ?content .

?lecture dcterms:isPartOf ?course .

?lecture dcmitype:identifier ?number .

?content dcterms:type ?content\_type .

?content dcterms:source ?source .

} order by ?number

# Question 7

# List the courses for subject [Subject]?

# List the courses for subject COMP?

SELECT (concat(?subject, " ", ?number) AS ?courseName) ?title

WHERE {

?course dcmitype:subject "COMP" .

?course dcmitype:subject ?subject .

?course dcmitype:identifier ?number .

?course dcterms:title ?title .

} ORDER BY ?number

#Question 8

# What is the outline for [Subject] [number] ?

# What is the outline for COMP 6741 ?

SELECT ?outline

WHERE {

?course rdf:type uni:Course .

?course dcmitype:subject "COMP" .

?course dcmitype:identifier "6741" .

?course uni:hasContent ?content .

?content dcterms:type "OUTLINE" .

?content dcterms:source ?outline .

}

# Question 9

# Which courses cover [topic]?

# Which courses cover [Machine\_learning]?

SELECT (concat(?subject, " ", ?number) AS ?courseName)

WHERE {

?course dcmitype:subject ?subject .

?course dcmitype:identifier ?number .

?lecture dcterms:isPartOf ?course .

?lecture uni:topic dbr:Machine\_learning .

}

# Question 10

# Does [subject][number] has [event]?

# Does [COMP][6841] has [LAB]?

ASK{

?course dcmitype:subject "COMP" .

?course dcmitype:identifier "6741" .

?lecture dcterms:isPartOf ?course .

?event dcterms:isPartOf ?lecture .

?event dcterms:type "LAB".

}

* Following are the 3 queries asked in Project 2:

# Query 1

# For a course c, list all covered topics t, printing out their English labels and their DBpedia URI,

together with the course event URI (e.g., ’lab3’) and resource URI (e.g., ’slides10’) where they

appeared.

* For this Query we have two separate queries Q1\_1 and Q1\_2 for lecture and lab topics.
* Q1\_1

|  |
| --- |
| SELECT ?topic ?lectureID ?resource  WHERE {  ?course dcmitype:subject "COMP".  ?course dcmitype:identifier "6741".  ?lecture dcterms:isPartOf ?course .  ?lecture uni:topic ?topic.  ?lecture dcmitype:identifier ?lectureID.  ?lecture uni:hasContent ?content .  ?content dcterms:type "SLIDES" .  ?content dcterms:source ?resource .  } ORDER BY ?lectureID |

* Q1\_2

|  |
| --- |
| SELECT ?topic ?labID ?resource  WHERE {  ?course dcmitype:subject "COMP".  ?course dcmitype:identifier "6741".  ?lecture dcterms:isPartOf ?course .  ?event dcterms:isPartOf ?lecture .  ?event dcterms:type "LAB".  ?event uni:topic ?topic .  ?event dcmitype:identifier ?labID.  ?event uni:hasContent ?content.  ?content dcterms:source ?resource.  } ORDER BY ?labID |

# Query 2

# For a given topic t (DBpedia URI), list all courses where they appear, together with a count, sorted by

frequencyQ2

* Q2

|  |
| --- |
| SELECT ?courseName (count (?courseName) as ?count)  WHERE {  ?course rdf:type uni:Course.  ?lecture dcterms:isPartOf ?course .  ?event dcterms:isPartOf ?lecture .  ?event dcterms:type "LAB".  {?lecture uni:topic dbr:Machine\_learning} UNION {?event uni:topic dbr:Machine\_learning }.  ?course dcterms:title ?courseName.  } GROUP BY ?courseName ORDER BY DESC(?count) |

# Query 3

# For a given topic t , list the precise course URI, course event URI and corresponding resource URI

where the topic is covered (e.g., “NLP” is covered in COMP474 → Lecture 10→ Lab 10 → Lab

Notes)

* For this Query we have two separate queries Q3\_1 and Q3\_2 for lecture and lab topics.
* Q3\_1

|  |
| --- |
| SELECT ?courseName ?lectureID ?resource  WHERE {  ?course rdf:type uni:Course.  ?lecture dcterms:isPartOf ?course .  ?course dcterms:title ?courseName.  ?lecture uni:topic dbr:Machine\_learning.  ?lecture dcmitype:identifier ?lectureID.  ?lecture uni:hasContent ?content.  ?content dcterms:type "SLIDES".  ?content dcterms:source ?resource.  } ORDER BY ?lectureID |

* Q3\_2

|  |
| --- |
| SELECT ?courseName ?labID ?resource  WHERE {  ?course rdf:type uni:Course.  ?lecture dcterms:isPartOf ?course .  ?course dcterms:title ?courseName.  ?event dcterms:isPartOf ?lecture .  ?event dcterms:type "LAB".  ?event uni:topic dbr:Machine\_learning.  ?event dcmitype:identifier ?labID.  ?event uni:hasContent ?content.  ?content dcterms:source ?resource.  } ORDER BY ?labID |

# Chatbot using Rasa

Rasa is an open source machine learning framework for automated text and voice-based conversations. Understand messages, hold conversations, and connect to messaging channels and APIs.

For this project, the chatbot will be implemented with Rasa. New intents will be created, and each intent will be linked to a query. New entities will also be added to identify the values the chatbot can query on.

# Intents

The nlu.yml file has to be modified to include the intents. For each intents, a question will be asked in with different ways of asking the same questions, so that the model can be trained to identify the question intent.

The domain.yml file has to be updated to identify the intents

- intent: about\_course

- intent: about\_outline

- intent: about\_course\_number

- intent: about\_course\_reading

- intent: about\_subject

- intent: about\_course\_content

- intent: about\_lecture\_topics

- intent: about\_event

- intent: about\_lecture\_by\_topic

- intent: about\_course\_by\_topic

# Entities

The domain.yml file has to be modified to identify the different entities to be extracted from the questions, namely (course) which is the course, (event) which the event type e.g. LAB, and (topic)

entities:

- course

- event

- topic

The slots section is also modify to identify the type and default value for the entities

slots:

course:

type: text

initial\_value: "NA"

event:

type: text

initial\_value: "NA"

topic:

type: text

initial\_value: "NA"

# Stories

A single story will be used to basically ask for the questions. The storie.yml file has to be modified to process the different possible intents

- story: get course info

steps:

- intent: greet

- action: utter\_iamunibot

- or:

- intent: about\_course

- intent: about\_outline

- intent: about\_course\_number

- intent: about\_course\_reading

- intent: about\_subject

- intent: about\_course\_content

- intent: about\_lecture\_topics

- intent: about\_event

- intent: about\_lecture\_by\_topic

- intent: about\_course\_by\_topic

- action: action\_course\_query

# Actions

The domain.yml has to be modified to include the action. Finally, the actions.py file has to be modified to process the intent.

# Program structure

1. Create a new class *ActionCourseQuery* that is attached to the action *action\_course\_query*
2. Get the information for the *course, topic, event and intent* from the tracker slots
3. Call the *buildquery()* function by passing *course, topic, event and intent* as parameters
4. In *buildquery()* verify the intent and call the appropriate sparql query through a request. Process the response and return the output message
   1. The topic is identify by annotating the topic through dbpedia spotlight

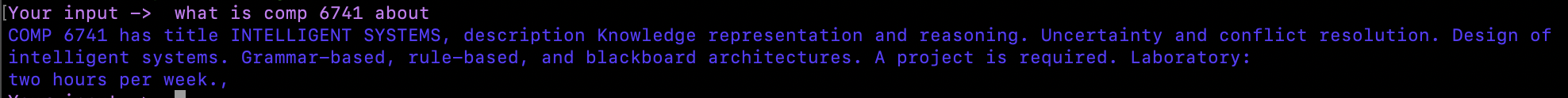
spotligh\_url = 'http://api.dbpedia-spotlight.org/en/annotate?text={text}'

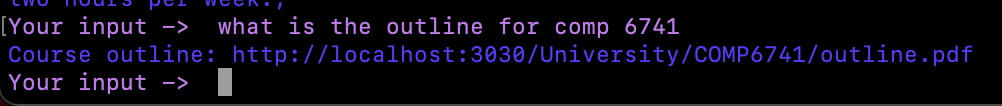
1. Send the message back to console through dispatcher

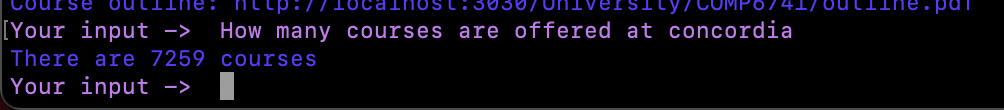
The queries have been defined at the beginning of the file.

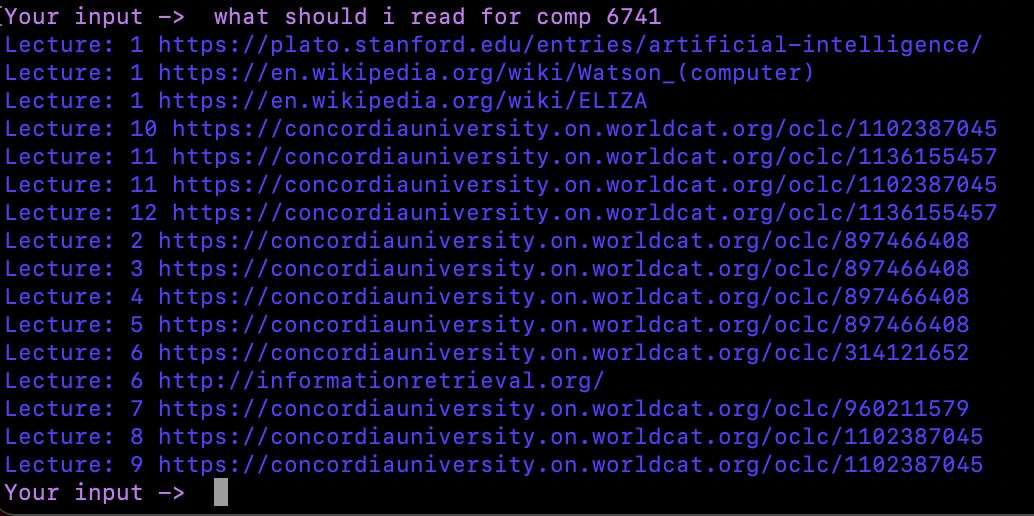
The fuseki server to query is defined on localhost:3030

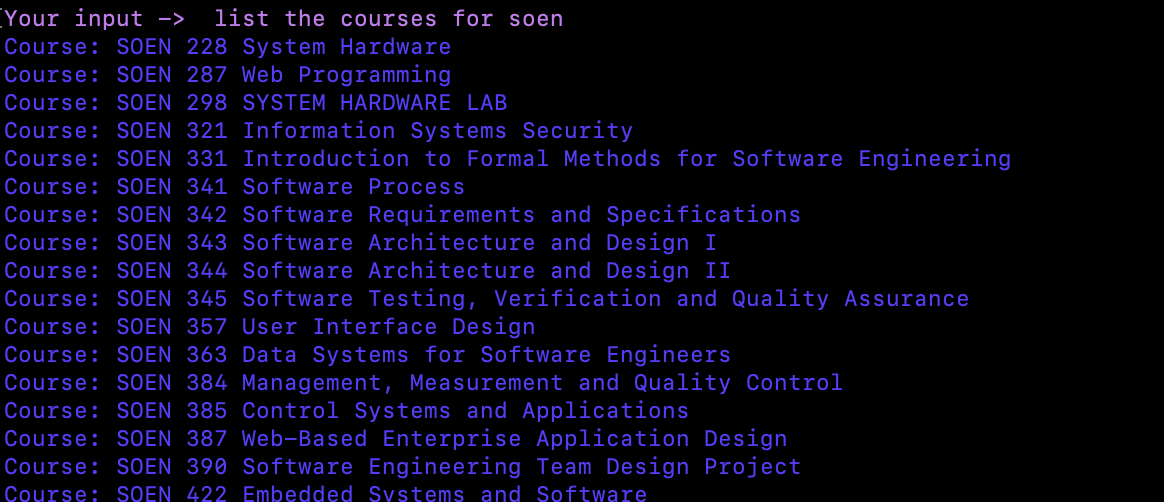
# Examples – Sample output for each questions

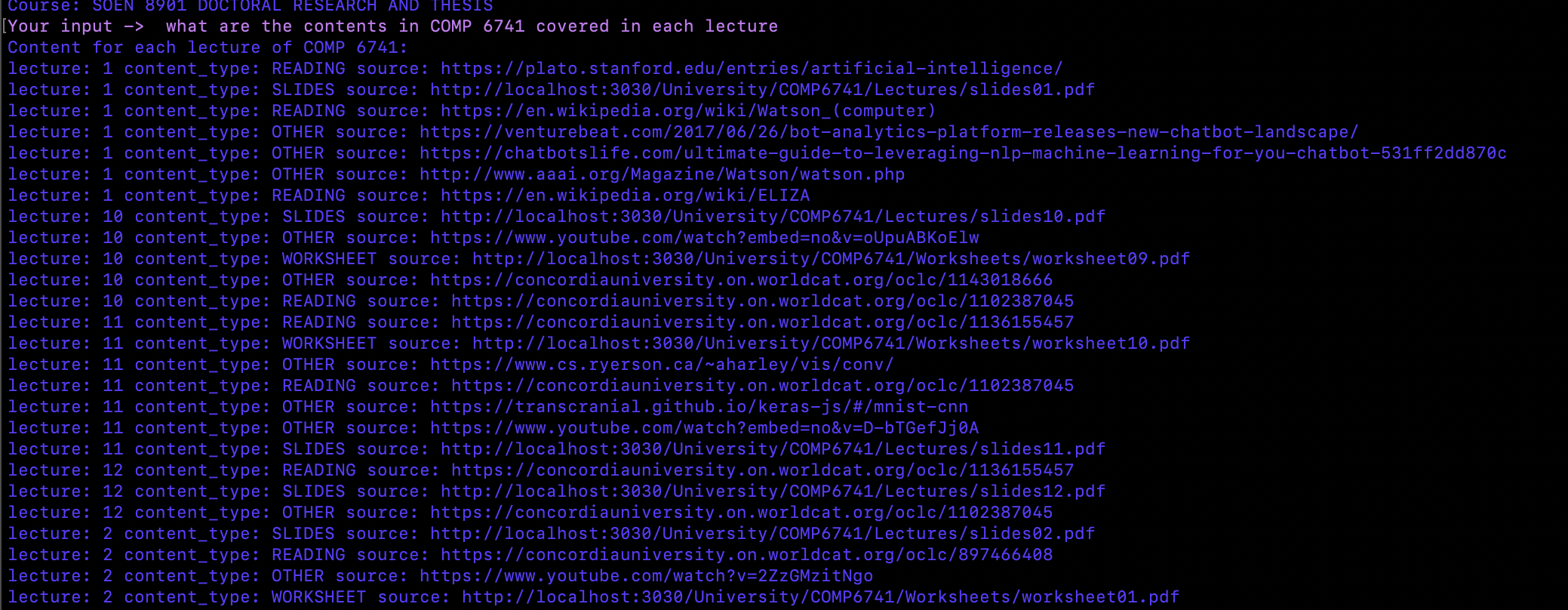




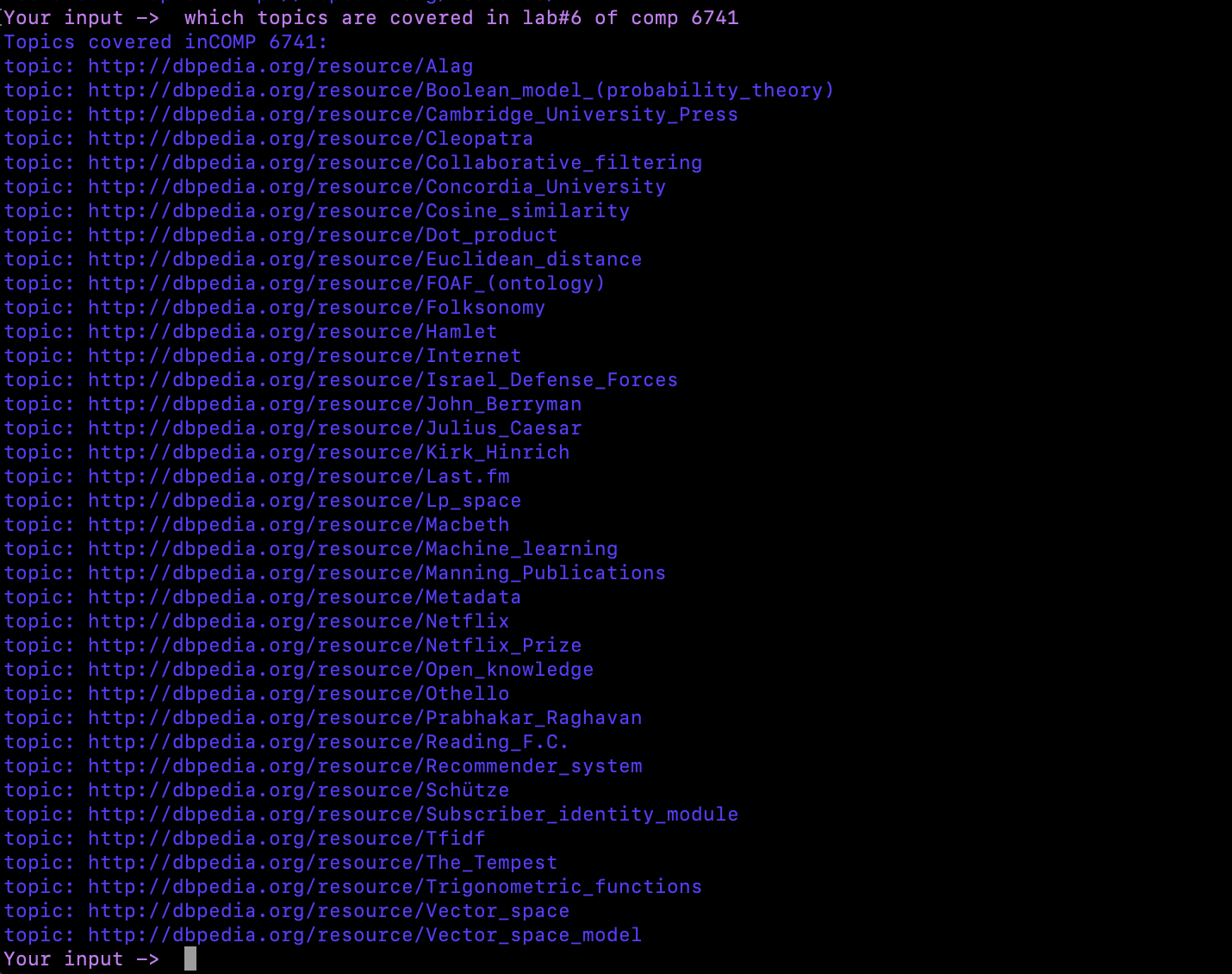


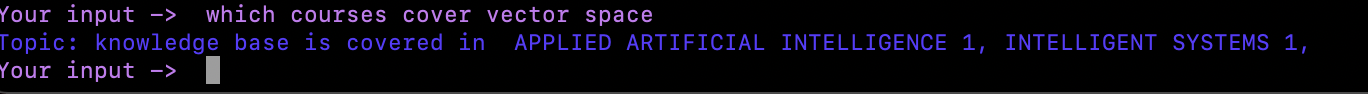












# Statistics

|  |  |
| --- | --- |
| **Statistics** | **Count** |
| Number of triples | 53519 |
| Number of courses | 7259 |
| Number of lectures | 24 |
| Number of events | 24 |
| Number of contents | 173 |
| Number of unique topics (COMP 6741 combining lab and lecture) | 803 |
| Number of unique topics (COMP 6721 combining lab and lecture) | 722 |
| Number of topic instance for COMP 6741 | 865 |
| Number of topic instance for COMP 6721 | 799 |

# Running program

1. Ensure rdflib, and pandas are installed in python environment
2. Place the university folder on the Fuseki server /webapp folder
3. Run the kbuilder.py
4. Run the Fuseki server and create a new dataset uni and upload Knowdlegde\_base.nt
5. From the Queries.sparql file, copy with PREFIX section, copy the query that should be executed

# References

1. RDF Schema 1.1

<https://www.w3.org/TR/rdf-schema/>

1. FOAF Vocabulary

<http://xmlns.com/foaf/spec/>

1. Dublin Core Metadata Initiative

<https://www.dublincore.org/specifications/dublin-core/dcmi-terms/#http://purl.org/dc/dcmitype/Event>

1. Vivo Core Ontology

<https://lov.linkeddata.es/dataset/lov/vocabs/vivo>

1. Vivo Tutorial by Shanshan Chen, Yuyin Sun, Ying Ding

<https://info.sice.indiana.edu/~dingying/Teaching/S604/VIVO-tutorial-v1.pdf>

1. Merge 2 CSV files.

<https://www.geeksforgeeks.org/how-to-merge-two-csv-files-by-specific-column-using-pandas-in-python/>

1. Write SPARQL query in Fuseki-Server

<https://www.youtube.com/watch?v=5-UfFV5XmTI>

1. Remove blanks from text file

https://www.kite.com/python/answers/how-to-remove-empty-lines-from-a-string-in-python