COMP474 PROJECT

Assignment #1

Abstract

To build Study_bot, an intelligent agent that can answer university course-related questions using a knowledge graph and natural language processing

TEAM: FL_U_04

Zhiqing YUAN #26258840

Nian Liu #40044346

Yaohua Zhang #40073090

Jia Ming Wei #40078192

Table of Contents

Table of Contents

1. Introduction

1.1 The Goal

Assignment 1 is to create an intelligent agent using existed and created vocabularies based on the information about courses of Concordia University.

Assignment 2 is to create a chatbot which is possible to answer the questions about a course.

1.2 The Team

The project has been implemented by Python language. Our four team members shared our works via Github and Google drive. The team meets one or two times per week to follow up on the process of the project.

2. Design and Implementation

To create a Studybot, firstly, we design ten different competency questions that we would like our bot to answer. Based on these questions, we create a schema with reused and self-defined vocabularies for further development. And finally, we generate the knowledge base using Concordia open database. We also add the details for two courses COMP474 and COMP472.

2.1 Competency Questions

The first process is to design the competency questions which the intelligent agent could answer. The competency questions are as follows:

1. How many courses in each subject?
2. Which lectures does course COMP474 have?
3. Which topics are associated with course COMP472?
4. Which courses have the subject COMP?
5. What's the content of the lectures of COMP474?
6. What's the course description of COMP472?
7. In which lectures is the subject "Knowledge Graph" covered?
8. What's the lab content for labs in COMP474?
9. What's the course outline of COMP474?
10. What's the DBpedia link for each topic?

The first three are generalized questions as required. We also create a query for each question to validate our design after the knowledge base is constructed.

2.2 Vocabulary

To model the schema for the knowledge base, the implementation includes choosing the reused vocabularies and developing our vocabularies.

2.2.1 Reused Vocabulary

For the vocabularies reused, besides the common vocabularies, such as rdf, rdfs, and xsd, we also use dbr, aiiso, teach, and vivo.

“dbr” is used to link the relevant page on DBpedia.

“aiiso” is Academic Institution Internal Structure Ontology and it provides classes and properties to describe the academic institution.

1. aiiso:code a property for a course, lecture, lab, and tutorial number.
2. aiiso:name a property for the name of courses, universities, lectures, labs, tutorials, subjects, and topics.

“teach” is Teaching Core Vocabulary which provides terms relate to a course that a teacher teaches.

1. teach:Lecture a class for a lecture.
2. teach:Course a class for a course.
3. teach:courseDescription a property for course description.
4. teach:courseTitle a property for a course title.

“vivo” is an ontology of the academic and research domain.

1. vivo:University a class of a University

2. vivo:courseCredits a property to present the credits for a course
3. vivo:Video a class for a video

There are the following benefits for reused vocabularies. Firstly, time-saving, we don't need to define classes and properties by ourselves. Secondly, it is easy to understand. For example, we commonly use rdf and rdfs to define a class and property. We don't need extra time and effort to figure the meaning of vocabulary. Thirdly, it is easy to link to external graphs, such as DBpedia vocabulary to link to DBpedia.

2.2.2 Developed Vocabulary

Although we use many reused vocabularies, we still need to define some classes and properties to deal with the areas that are not covered by the reused vocabularies.

To develop vocabulary extensions, we use focu to define a class and a property and focudata to create data. For classes, we have:

```
focu:CourseOutline
  a rdfs:Class ;
  rdfs:label "Course Outline"@en ;
  rdfs:comment "Course Outline"@en .

focu:Lab
  a rdfs:Class ;
  rdfs:label "Lab"@en ;
  rdfs:comment "Lab Class"@en ;
  rdfs:subClassOf teach:Lecture .

focu:Material
  a rdfs:Class ;
  rdfs:label "Course Material"@en ;
  rdfs:comment "Course Material"@en .

focu:Reading
  a rdfs:Class ;
  rdfs:label "Reading"@en ;
  rdfs:comment "Reading"@en ;
```

```
  rdfs:subClassOf focu:Material .

focu:Slide
  a rdfs:Class ;
  rdfs:label "Slide"@en ;
  rdfs:comment "Slide"@en ;
  rdfs:subClassOf focu:Material .

focu:Subject
  a rdfs:Class ;
  rdfs:label "Subject"@en ;
  rdfs:comment "Subject Class"@en .

focu:Topic
  a rdfs:Class ;
  rdfs:label "Topic"@en ;
  rdfs:comment "Topic Class"@en .

focu:Tutorial
  a rdfs:Class ;
  rdfs:label "Tutorial"@en ;
  rdfs:comment "Tutorial Class"@en ;
  rdfs:subClassOf teach:Lecture .

focu:Worksheet
  a rdfs:Class ;
  rdfs:label "Worksheet"@en ;
  rdfs:comment "Worksheet"@en ;
  rdfs:subClassOf focu:Material .
```

For properties, we have:

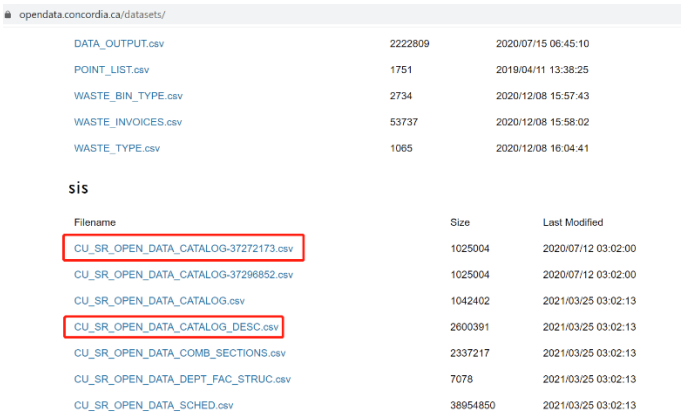
```
focu:content
  a rdf:Property ;
  rdfs:label "content"@en ;
  rdfs:comment "Lab content or tutorial content."@en ;
  rdfs:domain teach:Lecture ;
  rdfs:range focu:Material, vivo:Video .

focu:outline
  a rdf:Property ;
  rdfs:label "outline"@en ;
  rdfs:comment "Course outline."@en ;
  rdfs:domain teach:Course ;
  rdfs:range focu:CourseOutline .

focu:subject
  a rdf:Property ;
  rdfs:label "subject"@en ;
  rdfs:comment "Course subject."@en ;
  rdfs:domain teach:Course ;
```

```
  rdfs:range focu:Subject .

focu:labAssociateWith
  a rdf:Property ;
  rdfs:label "lab associated with a specific lecture"@en ;
  rdfs:comment "lab associated with a specific lecture"@en ;
  rdfs:domain focu:Lab ;
  rdfs:range teach:Lecture .

focu:tutorialAssociateWith
  a rdf:Property ;
  rdfs:label "tutorial associated with a specific lecture"@en ;
  rdfs:comment "tutorial associated with a specific lecture"@en ;
  rdfs:domain focu:Tutorial ;
  rdfs:range teach:Lecture .

focu:offeredAt
  a rdf:Property ;
  rdfs:label "offered in"@en ;
  rdfs:comment "a course is offered at a univeristy."@en ;
  rdfs:domain teach:Course ;
  rdfs:range vivo:University .

focu:offeredIn
  a rdf:Property ;
  rdfs:label "lecture in"@en ;
  rdfs:comment "a lecture is in a course."@en ;
  rdfs:domain teach:Lecture ;
  rdfs:range teach:Course .

focu:topicAssociateWith
  a rdf:Property ;
  rdfs:label "topics"@en ;
  rdfs:comment "topics that are covered in a course or a lecture in a course."@en ;
  rdfs:domain focu:Topic ;
  rdfs:range teach:Lecture, teach:Course, focu:Material .
```

2.3 Knowledge Base Construction

2.3.1 Dataset

The dataset we used is on the website https://opendata.concordia.ca/datasets/. We use the kb_generator.py Python file to extract data from these two CSV files.



opendata.concordia.ca/datasets/

DATA_OUTPUT.csv	2222809	2020/07/15 06:45:10
POINT_LIST.csv	1751	2019/04/11 13:38:25
WASTE_BIN_TYPE.csv	2734	2020/12/08 15:57:43
WASTE_INVOICES.csv	53737	2020/12/08 15:58:02
WASTE_TYPE.csv	1065	2020/12/08 16:04:41

sis

Filename	Size	Last Modified
CU_SR_OPEN_DATA_CATALOG-37272173.csv	1025004	2020/07/12 03:02:00
CU_SR_OPEN_DATA_CATALOG-37296852.csv	1025004	2020/07/12 03:02:00
CU_SR_OPEN_DATA_CATALOG.csv	1042402	2021/03/25 03:02:13
CU_SR_OPEN_DATA_CATALOG_DESC.csv	2600391	2021/03/25 03:02:13
CU_SR_OPEN_DATA_COMB_SECTIONS.csv	2337217	2021/03/25 03:02:13
CU_SR_OPEN_DATA_DEPT_FAC_STRUC.csv	7078	2021/03/25 03:02:13
CU_SR_OPEN_DATA_SCHED.csv	38954850	2021/03/25 03:02:13

2.3.2 Knowledge Base Construction

We use kb_generator.py to automatically construct the knowledge base from the dataset.

Firstly, we import all libraries we used.

```
from rdflib import URIRef, Literal, Namespace, Graph
from rdflib.namespace import XSD, RDF, RDFS
from pathlib import Path
from os import getcwd
import pandas as pd
```

Secondly, we create a new graph and setup all namespaces. And then, bind the namespace to the graph.

```
g = Graph()

FC = Namespace('http://focu.io/schema#')
FCD = Namespace('http://focu.io/data#')
DBR = Namespace('http://dbpedia.org/resource/')
VIVO = Namespace('http://vivoweb.org/ontology/core#')
AIISO = Namespace('http://purl.org/vocab/aiiso/schema#')
TEACH = Namespace('http://linkedscience.org/teach/ns#')

g.bind('focu', FC)
g.bind('focudata', FCD)
g.bind('dbr', DBR)
g.bind('vivo', VIVO)
g.bind('aiiso', AIISO)
g.bind('teach', TEACH)
```

Thirdly, we create all classes and properties. The followings are two examples:

```
lab = FC['Lab']
g.add((lab, RDF.type, RDFS.Class))
g.add((lab, RDFS['subClassOf'], TEACH['Lecture']))
g.add((lab, RDFS.label, Literal('Lab', lang='en')))
g.add((lab, RDFS.comment, Literal('Lab Class', lang='en')))
```

```
content = FC['content']
g.add((content, RDF.type, RDF.Property))
g.add((content, RDFS.label, Literal('content', lang='en')))
g.add((content, RDFS.comment, Literal('Lab content or tutorial content.', lang='en')))
g.add((content, RDFS.domain, FC['Lab']))
g.add((content, RDFS.domain, FC['Tutorial']))
g.add((content, RDFS.range, FC['Slide']))
g.add((content, RDFS.range, FC['Worksheet']))
g.add((content, RDFS.range, FC['Reading']))
g.add((content, RDFS.range, VIVO['Video']))
```

Fourthly, we create data, Concordia University, and all details for course COMP474 and COMP472, such as labs, lectures, and topics. For these specific examples, we also use rdfs:seeAlso to link entities on DBpedia.

```
concordia = FCD['Concordia_University']
g.add((concordia, RDF.type, VIVO['University']))
g.add((concordia, AIISO['name'], Literal('Concordia University')))
g.add((concordia, RDFS.seeAlso, DBR['Concordia_University']))
```

Finally, we merge the two CSV files and extract all information needed to create triples.

```
table1 = pd.read_csv("CU_SR_OPEN_DATA_CATALOG_DESC.csv", header=0)
table2 = pd.read_csv("CU_SR_OPEN_DATA_CATALOG-37272173.csv", header=0)

table_merged = pd.merge(table1, table2, on='Course ID', how='inner')

subjects = []
course_ids = []

for index, row in table_merged.iterrows():
    if row['Subject'] not in subjects:
        subjects.append(row['Subject'])
    if row['Course ID'] not in course_ids:
        course_ids.append(row['Course ID'])
        course_generator(row['Subject'], row['Catalog'], row['Long Title'], row['Class Units'], row['Descr'])

for item in subjects:
    subject_generator(item)

g.serialize(format='nt', destination="school.nt")
```

2.4 Topics

First, we use pdfplumber library to extract text from pdf files. Then, we send the text to DBpedia spotlight API to get topics. For each topic, we add it to specific course event and resources.

total number of triples	number of distinct topics	number of topic instances
93571	6047	18796

```
def add_topics_file(file_name, course_name, lec, slide):
    print(file_name)
    pdf = pdfplumber.open(file_name)
    for i in range(len(pdf.pages)):
        page = pdf.pages[i]
        text = page.extract_text()
        add_topics(text, course_name, lec, slide)
    pdf.close()
```

```
def add_topics(text, course_name, lec, slide):
    # print(text)
    spot_light_url = f'https://api.dbpedia-spotlight.org/en/annotate?text={text}'
    headers = {'accept': 'application/json'}
    if requests.get(url=spot_light_url, headers=headers).status_code == 200:
        response = requests.get(url=spot_light_url, headers=headers).json()
        response = response.get('Resources')
        if response is not None:
            for item in response:
                dbr_link = item['@URI']
                dbr_name = dbr_link[28:]
                # print(dbr_name)

                topic = FCD[dbr_name]
                g.add((topic, RDF.type, FC['Topic']))
                g.add((topic, RDFS.label, Literal(dbr_name, lang='en')))
                g.add((topic, AIISO['name'], Literal(dbr_name)))
                g.add((topic, FC['topicAssociateWith'], FCD[course_name]))
                if lec is not None:
                    g.add((topic, FC['topicAssociateWith'], lec))
                if slide is not None:
                    g.add((topic, FC['topicAssociateWith'], slide))
                g.add((topic, RDFS.seeAlso, DBR[dbr_name]))
```

3. Result and Analysis

3.1 RDF Schema

Our RDF schema is in the school_template.ttl file. Here is a part of the file.

```
@prefix aiiso: <http://purl.org/vocab/aiiso/schema#> .
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix focu: <http://focu.io/schema#> .
@prefix focudata: <http://focu.io/data#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix teach: <http://linkedscience.org/teach/ns#> .
@prefix vivo: <http://vivoweb.org/ontology/core#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

focu:CourseOutline
  a rdfs:Class ;
  rdfs:label "Course Outline"@en ;
  rdfs:comment "Course Outline"@en .

focu:content
  a rdf:Property ;
  rdfs:label "content"@en ;
  rdfs:comment "Lab content or tutorial content."@en ;
  rdfs:domain focu:Lab, focu:Tutorial ;
  rdfs:range focu:Reading, focu:Slide, focu:Worksheet, vivo:Video .
```

3.2 Knowledge Base

Our constructed knowledge base in N-Triples format is in the school.nt file. Here is a part of the file.

```
<http://focu.io/data#INST250> <http://focu.io/schema#subject> <http://focu.io/data#INST> .
<http://focu.io/data#CLAS490> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://linkedscience.org/teach/ns#Course> .
<http://focu.io/data#COMP6411> <http://focu.io/schema#subject> <http://focu.io/data#COMP> .
```

3.3 Queries and Results

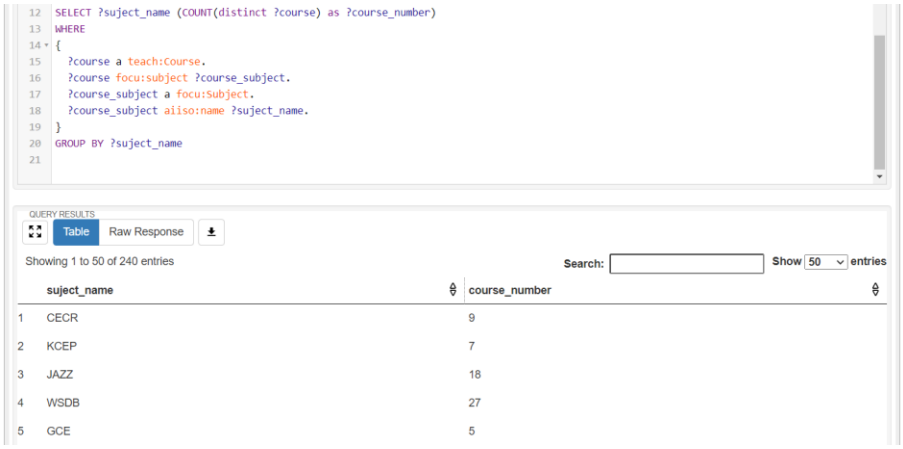
There are two types of queries, competency question queries and knowledge base queries. We set up the Fuseki server to run the queries. We use the same prefix for all queries.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
Prefix aiiso: <http://purl.org/vocab/aiiso/schema#>
Prefix dbr: <http://dbpedia.org/resource/>
Prefix focu: <http://focu.io/schema#>
Prefix focudata: <http://focu.io/data#>
Prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix teach: <http://linkedscience.org/teach/ns#>
Prefix vivo: <http://vivoweb.org/ontology/core#>
Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
```

3.3.1 Competency Question Queries

There are 10 competency question queries and outputs in the folder Competency Question Queries.

1. How many courses in each subject?



```
SELECT ?suject_name (COUNT(distinct ?course) as ?course_number)
WHERE
{
  ?course a teach:Course.
  ?course focu:subject ?course_subject.
  ?course_subject a focu:Subject.
  ?course_subject aiiso:name ?suject_name.
}
GROUP BY ?suject_name
```

QUERY RESULTS

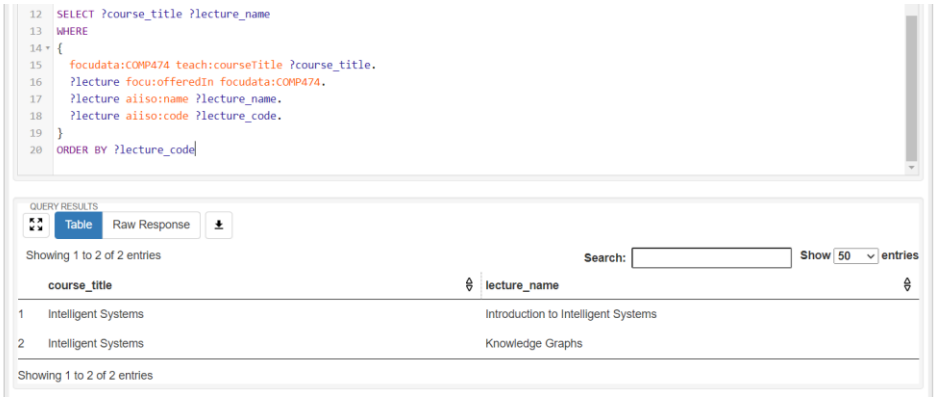
Table | Raw Response

Showing 1 to 50 of 240 entries

Search: Show 50 entries

	suject_name	course_number
1	CECR	9
2	KCEP	7
3	JAZZ	18
4	WSDB	27
5	GCE	5

2. Which lectures does course COMP474 have?



```
SELECT ?course_title ?lecture_name
WHERE
{
  focudata:COMP474 teach:courseTitle ?course_title.
  ?lecture focu:offeredIn focudata:COMP474.
  ?lecture aiiso:name ?lecture_name.
  ?lecture aiiso:code ?lecture_code.
}
ORDER BY ?lecture_code
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 2 of 2 entries

Search: Show 50 entries

	course_title	lecture_name
1	Intelligent Systems	Introduction to Intelligent Systems
2	Intelligent Systems	Knowledge Graphs

Showing 1 to 2 of 2 entries

3. Which topics are associated with course COMP472?

```
SELECT ?topics
WHERE
{
  ?topics focu:topicAssociateWith focudata:COMP472.
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 2 of 2 entries

Search: Show 50 entries

	topics
1	focudata:Breadth-first_search
2	focudata:Depth-first_search

Showing 1 to 2 of 2 entries

4. Which courses have the subject COMP?

```
SELECT ?course
WHERE
{
  ?course a teach:Course.
  ?course focu:subject focudata:COMP .
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 50 of 105 entries

Search: Show 50 entries

	course
1	focudata:COMP208
2	focudata:COMP333
3	focudata:COMP691
4	focudata:COMP6231
5	focudata:COMP739

5. What's the content of the lectures of COMP474?

```
SELECT ?lecture ?name ?content
WHERE
{
  ?lecture aiiso:name ?name.
  ?lecture focu:offeredIn focudata:COMP474 .
  ?lecture focu:content ?content.
  ?lecture aiiso:code ?lecture_code.
}
ORDER BY ?lecture_code
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 5 of 5 entries

Search: Show 50 entries

	lecture	name	content
1	focudata:COMP474_lecture1	Introduction to Intelligent Systems	file:///C:/Users/YZ/Desktop/COMP474/project/COMP474/Lectures/slides01.pdf
2	focudata:COMP474_lecture1	Introduction to Intelligent Systems	<https://www.youtube.com/watch?v=P18EdAKuC1U>
3	focudata:COMP474_lecture2	Knowledge Graphs	file:///C:/Users/YZ/Desktop/COMP474/project/COMP474/Lectures/py_tut.pdf
4	focudata:COMP474_lecture2	Knowledge Graphs	file:///C:/Users/YZ/Desktop/COMP474/project/COMP474/Lectures/slides02.pdf
5	focudata:COMP474_lecture2	Knowledge Graphs	file:///C:/Users/YZ/Desktop/COMP474/project/COMP474/Lectures/worksheet01.pdf

Showing 1 to 5 of 5 entries

6. What's the course description of COMP472?

```
SELECT ?course_title ?description
WHERE
{
  focudata:COMP472 teach:courseTitle ?course_title.
  focudata:COMP472 teach:courseDescription ?description.

}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

	course_title	description
1	Artificial Intelligence	Scope of AI. First-order logic. Automated reasoning. Search and heuristic search. Game-playing. Planning. Knowledge representation. Probabilistic reasoning. Introduction to machine learning. Introduction to natural language processing. Project. Lectures: three hours per week. Laboratory: two hours per week. Prerequisite: COMP 352 or COEN 352.

Showing 1 to 1 of 1 entries

7. In which lectures is the subject "Knowledge Graph" covered?

```
SELECT ?subject_name ?lecture ?lecture_name
WHERE
{
  focudata:Knowledge_Graph aiiso:name ?subject_name.
  focudata:Knowledge_Graph focu:topicAssociateWith ?lecture.
  ?lecture aiiso:name ?lecture_name.
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

	subject_name	lecture	lecture_name
1	Knowledge Graph	focudata:COMP474_lecture2	Knowledge Graphs

Showing 1 to 1 of 1 entries

8. What's the lab content for labs in COMP474?

```
SELECT ?lab ?name ?content
WHERE
{
  ?lecture focu:offeredIn focudata:COMP474 .
  ?lab focu:labAssociatedWith ?lecture.
  ?lab aiiso:name ?name.
  ?lab aiiso:code ?lab_code.
  ?lab focu:content ?content.
}
ORDER BY ?lab_code
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 2 of 2 entries

Search: Show 50 entries

	lab	name	content
1	focudata:COMP474_lab1	COMP474_Lab_1	<https://moodle.concordia.ca/moodle/mod/page/view.php?id=2608092>
2	focudata:COMP474_lab2	COMP474_Lab_2	<https://moodle.concordia.ca/moodle/mod/page/view.php?id=2575768>

Showing 1 to 2 of 2 entries

9. What's the course outline of COMP474?

```
SELECT ?course_title ?course_outline
WHERE
{
  focudata:COMP474 teach:courseTitle ?course_title.
  focudata:COMP474 focu:outline ?course_outline.

}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

	course_title	course_outline
1	Intelligent Systems	file:///C:/Users/YZ/Desktop/COMP474/project/COMP474/course_outline_comp474_6741_w2021.pdf

Showing 1 to 1 of 1 entries

10. What's the DBpedia link for each topic?

```
SELECT ?topic ?topic_name ?dbpedia_link
WHERE
{
  ?topic a focu:Topic.
  ?topic aiiso:name ?topic_name.
  ?topic rdfs:seeAlso ?dbpedia_link.
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 4 of 4 entries

Search: Show 50 entries

	topic	topic_name	dbpedia_link
1	focudata:Breadth-first_search	Breadth-first_search	dbr:Breadth-first_search
2	focudata:Depth-first_search	Depth-first_search	dbr:Depth-first_search
3	focudata:Knowledge_Graph	Knowledge Graph	dbr:Knowledge_Graph
4	focudata:Expert_system	Expert System	dbr:Expert_system

Showing 1 to 4 of 4 entries

3.3.2 Knowledge Base Queries

There are 7 Knowledge Base queries and outputs in the folder Knowledge Base Queries.

1. How many courses in the database?

```
SELECT (COUNT(distinct ?course) as ?count)
WHERE
{
  ?course a teach:Course.
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

	count
1	7103

Showing 1 to 1 of 1 entries

2. How many universities in the database?

```
SELECT (COUNT(distinct ?university) as ?count)
WHERE
{
  ?university a vivo:University.
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

	count
1	1

Showing 1 to 1 of 1 entries

3. How many subjects in the database?

```
SELECT (COUNT(distinct ?subject) as ?count)
WHERE
{
  ?subject a focu:Subject.
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

	count
1	251

Showing 1 to 1 of 1 entries

4. How many lectures in the database?

```
SELECT (COUNT(distinct ?lecture) as ?count)
WHERE
{
  ?lecture a teach:Lecture.
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

	count
1	4

Showing 1 to 1 of 1 entries

5. How many labs in the database?

```
SELECT (COUNT(distinct ?lab) as ?count)
WHERE
{
  ?lab a focu:Lab.
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

	count
1	4

Showing 1 to 1 of 1 entries

6. How many topics in the database?

```
SELECT (COUNT(distinct ?topic) as ?count)
WHERE
{
  ?topic a focu:Topic.
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

	count
1	4

Showing 1 to 1 of 1 entries

7. How many triples are there in the database?

```
SELECT (COUNT(?s) as ?sCount)
WHERE
{
  ?s ?p ?o .
}
```

QUERY RESULTS

Table | Raw Response

Showing 1 to 1 of 1 entries

Search: Show 50 entries

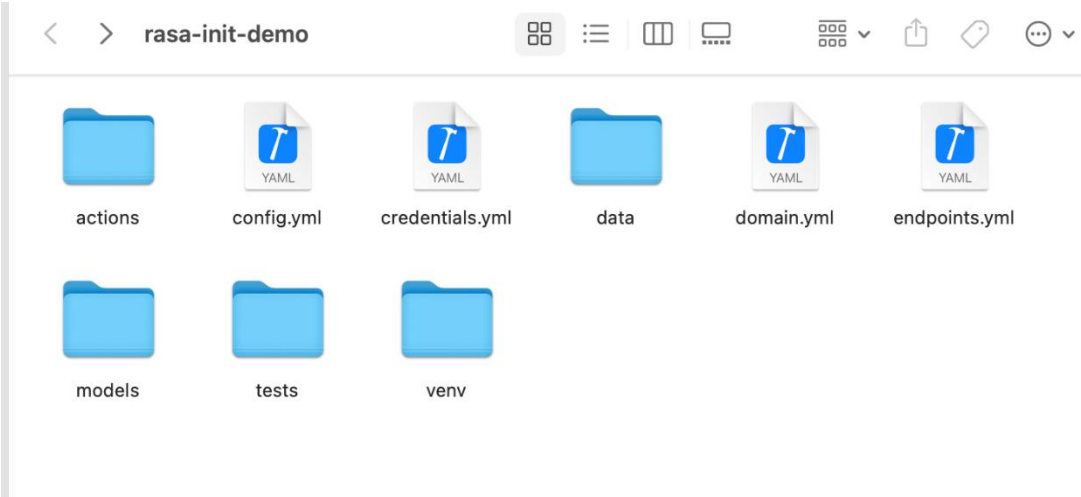
	sCount
1	50399

Showing 1 to 1 of 1 entries

4. Rasa

4.1 Design

4.1.1 nlu.yml



After installation Rasa, we can init rasa project. The project looks like the picture below.

Go to data file, find nlu.yml file.

Create some intents with example, then when you use chatbot it will find out your questions' intention.

```
- intent: q_1
  examples: |
    - How many courses in each subject?
    - List each subject with courses
    - All subjects with courses
    - All subjects courses

- intent: q_2
  examples: |
    - Which lectures does course COMP474 have?
    - List all lecture in Comp474
    - all lecture in [COMP474](course)
```

4.1.2 stories.yml

After we create intents in nlu.yml, we need to create the stories for those intents.

```yaml
- story: answer query 1
  steps:
    - intent: q_1
    - action: query_1
```

The story will create the format that how chatbot will answer those intents with actions.

4.1.3 domain.yml

Then you need to register intents and actions in domain.yml file.

domain.yml 〉 No Selection

```yaml
version: "2.0"

intents:
  - greet
  - goodbye
  - affirm
  - deny
  - mood_great
  - mood_unhappy
  - bot_challenge
  - about_person
  - about_course
  - action_course_topic
  - action_course_event
  - q_1
  - q_2
  - q_3
  - q_4
  - q_5
  - q_6
  - q_7
  - q_8
  - q_9
  - q_10

entities:
  - person
  - course
  - topic
  - event

slots:
  person:
    type: any
    initial_value: "initial"

  course:
    type: any
    initial_value: "initial"

  topic:
    type: any
    initial_value: "initial"

  event:
    type: any
    initial_value: "initial"

actions:
  - action_person_info
  - action_course_info
  - action_course_topic
  - action_course_event
  - query_1
  - query_2
  - query_3
  - query_4
  - query_5
  - query_6
  - query_7
  - query_8
  - query_9
  - query_10
```

If in the intents' example questions, you need to get some keywords to facilitate the processing of SPARQL query later, you can create some entities to record the key world as slot value.

4.2 Rasa Implementation

Now we have basic logic about question and answer for chatbot, but if we want to chatbot really can answer the question with demand content. We need to implement the actions in actions.py .

The code should like this below.

actions.py nlu.yml

```
# 1. How many courses in each subject?
class Query1(Action):
    def name(self) -> Text:
        return "query_1"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker, domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        query_var = """
                PREFIX dbo: <http://dbpedia.org/ontology/>
                Prefix aiiso: <http://purl.org/vocab/aiiso/schema#>
                Prefix dbr: <http://dbpedia.org/resource/>
                Prefix focu: <http://focu.io/schema#>
                Prefix focudata: <http://focu.io/data#>
                Prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                Prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
                Prefix teach: <http://linkedscience.org/teach/ns#>
                Prefix vivo: <http://vivoweb.org/ontology/core#>
                Prefix xsd: <http://www.w3.org/2001/XMLSchema#>
                SELECT ?suject_name (COUNT(distinct ?course) as ?course_number)
                WHERE
                {
                  ?course a teach:Course.
                  ?course focu:subject ?course_subject.
                  ?course_subject a focu:Subject.
                  ?course_subject aiiso:name ?suject_name.
                }
                GROUP BY ?suject_name
            """
        response = requests.post('http://localhost:3030/focu/sparql', data={'query': query_var})
        res = response.json()
        ans = []
        for row in res['results']['bindings']:
            ans.append([row['suject_name']['value'], row['course_number']['value']])

        dispatcher.utter_message(
            text=f"The number of courses in each subject are listed as following: {ans}, "
                 f"HaHa, there are so many but it is just what you asked (^-^)")

        return []
```

Send the sparql query to the Fuseki server and get result data in JSON format.

And dispatch the answer to the action response as a string.

4.3 Dynamic Keyword Acquisition

By importing spacy, the questions asked by users are obtained through tracker and then put into spacy's NLP for processing. The different processing can be solved by writing different patterns. Here is the code for query3 to get the keywords dynamically

```
class Query3(Action):
    def name(self) -> Text:
        return "query_3"

    def run(self,dispatcher:CollectingDispatcher,
            tracker: Tracker, domain:Dict[Text,Any])->List[Dict[Text,Any]]:

        #query body
        #entity com472
        query_var = """
                Prefix aiiso: <http://purl.org/vocab/aiiso/schema#>
                Prefix dbr: <http://dbpedia.org/resource/>
                Prefix focu: <http://focu.io/schema#>
                Prefix focudata: <http://focu.io/data#>
                Prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                Prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
                Prefix teach: <http://linkedscience.org/teach/ns#>
                Prefix vivo: <http://vivoweb.org/ontology/core#>
                Prefix xsd: <http://www.w3.org/2001/XMLSchema#>

                SELECT ?topics
                WHERE
                {
                  ?topics focu:topicAssociateWith focudata:COMP472.
                }
            """
#
#       response = requests.post('http://localhost:3030/focu/query', data={'query': query_var})
#       res = response.json()
#       s=tracker.slots['course']
#       print(s)

#-----------------------------------------
# Dynamic keyword acquistion
#-----------------------------------------
        intent=tracker.latest_message['intent']
        print(intent)
        ents=tracker.latest_message['entities']
        print(ents)
        ssstt=tracker.latest_message['text']
        print(ssstt)
        ssx=ssstt[9:16]
        print(ssx)
        s=ssx
```

```
# pattern
        nlp=spacy.load("en_core_web_sm")
        matcher = Matcher(nlp.vocab)
        pattern=[[{"POS":"NOUN"}]]
        matcher.add("CLASS_PATTERN",pattern)
#        doc=nlp(ssstt)
#        print("below is pattern from spacy")
#        print(doc)
#        doc=nlp("Upcoming iPhone X release date leaked")
        doc=nlp(ssstt)
        matches=matcher(doc)
        k=""
        print(matches)
        for match_id, start, end in matches:
            matched_span=doc[start:end]
            if "COMP" in matched_span.text:
                k=matched_span.text
            print(matched_span.text)
        print(k)
#        sst=tracker.latest_message['entities'][0]['value']
#        print(sst)
        dispatcher.utter_message(text="this is from query 3 "+s)

        return[]
```

```
Rasa is passing value -> event is ->  lecture2
Rasa is passing value -> course is ->  COMP472
The final event name => COMP472_lecture2
json return ==>  [{'topic': {'type': 'uri', 'value': 'ht
ype': 'uri', 'value': 'http://focu.io/data#Breadth-first
ocu.io/data#Depth-first_search'}}, {'topic': {'type': 'u
': {'type': 'uri', 'value': 'http://focu.io/data#3x3_bas
```

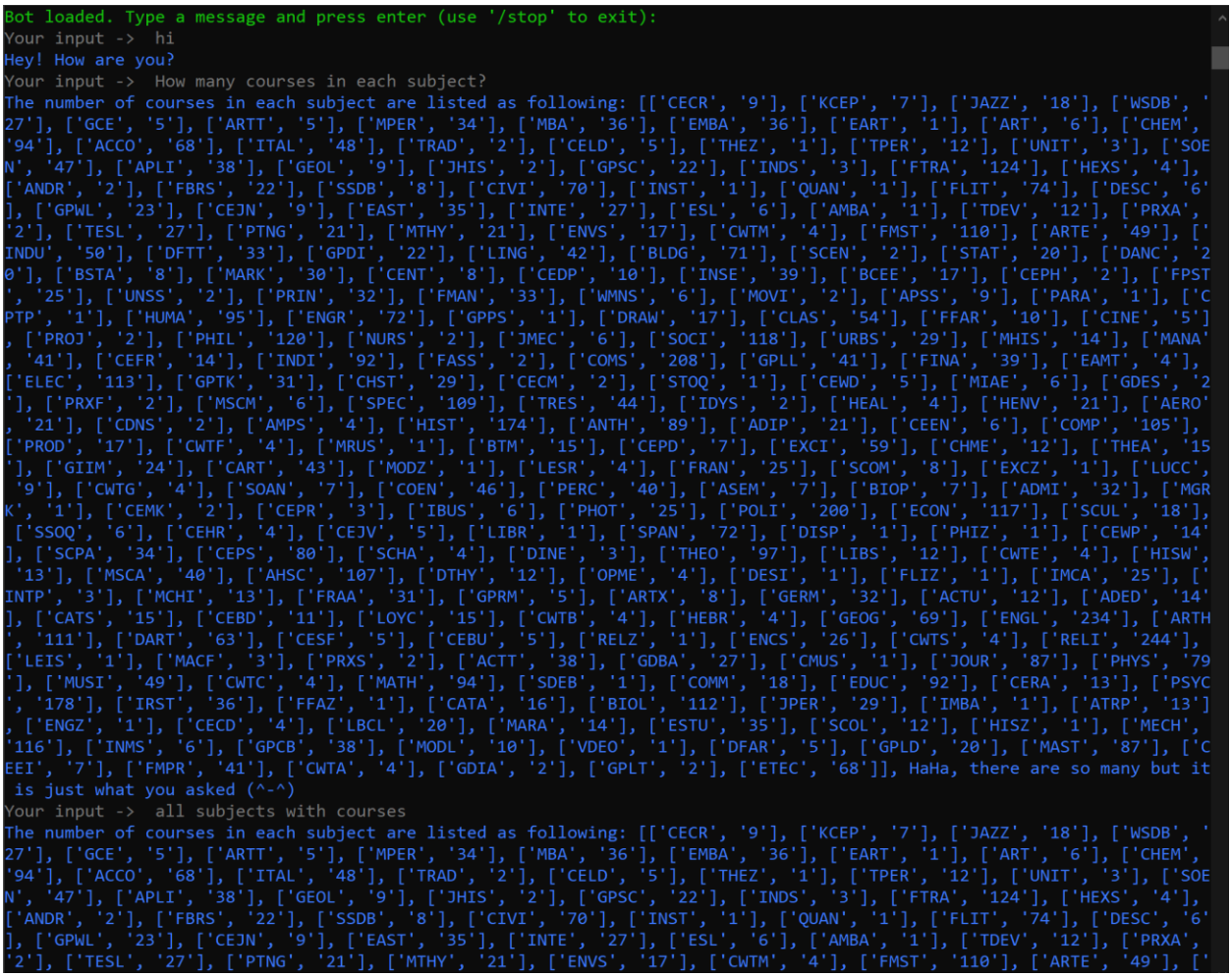
Here are the results to show implement automatically.

```
    chosen_type = tracker.slots['event']
    chosen_course = tracker.slots['course']
    print('Rasa is passing value -> event is -> ', tracker.slots['event'])
    print('Rasa is passing value -> course is -> ', tracker.slots['course'])
    # print(chosen_course)
    chosen_event = chosen_course+'_'+chosen_type
    # print('the event name is',chosen_event)
    query_var = """
        Prefix aiiso: <http://purl.org/vocab/aiiso/schema#>
        Prefix dbr: <http://dbpedia.org/resource/>
        Prefix focu: <http://focu.io/schema#>
        Prefix focudata: <http://focu.io/data#>
        Prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        Prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
        Prefix teach: <http://linkedscience.org/teach/ns#>
        Prefix vivo: <http://vivoweb.org/ontology/core#>
        Prefix xsd: <http://www.w3.org/2001/XMLSchema#>

        SELECT ?topic
        WHERE
        {{
          ?topic a focu:Topic.
          ?topic focu:topicAssociateWith focudata:{t}.

        }}

    """.format(t=chosen_event)
```

```
Your input ->  which courses cover Insurance
If you are asking about Insurance, the courses are as follows: ,['http://focu.io/data#A
cu.io/data#ECON393', 'http://focu.io/data#COMP474', 'http://focu.io/data#FMPR336', 'htt
LL20', 'http://focu.io/data#GPWL943', 'http://focu.io/data#COMP474_lecture9']
Your input ->  which course cover Inter_Core_2
If you are asking about Insurance, the courses are as follows: ,['http://focu.io/data#A
cu.io/data#ECON393', 'http://focu.io/data#COMP474', 'http://focu.io/data#FMPR336', 'htt
LL20', 'http://focu.io/data#GPWL943', 'http://focu.io/data#COMP474_lecture9']
Your input ->  which topics are covered in lecture1 of COMP445
If you are asking about COMP445_lecture1, the event includes all topics, such as[]
Your input ->  what is description of COMP335
If you are asking about COMP335, the results are as follows:['Finite state automata and
 Push-down automata and context-free languages. Pumping lemmas. Applications to parsing
Unde\xadcidability and decidability. Lectures: three hours per week. Tutorial: one hour
quisite: COMP 232 or COEN 231; COMP 249 or COEN 244.\r\n\r\n\r\n\r\n\r\n\r\n', 'Finite stat
lar languages. Push-down automata and context-free languages. Pumping lemmas. Applicati
ing machines. Unde\xadcidability and decidability. Lectures: three hours per week. Tuto
week.\nPrerequisite: COMP 232 or COEN 231; COMP 249 or COEN 244.\n\n\n\n\n']
Your input ->  What is description of MAST218
If you are asking about MAST218, the results are as follows:['Vector geometry; lines an
 Rn; vector functions; vector differential calculus; extrema and Lagrange multipliers.
tiple integrals and coordinate transformations. Problem solving with MAPLE.\r\nPrerequi
tics 105 or 201-NYC, 203 or 201-NYB or equivalent.\r\nNOTE: Students who have received
 may not take this course for credit.\r\nNOTE: Only three credits will be awarded from
\r\n\r\n\r\n\r\n', 'Vector geometry; lines and planes; curves in Rn; vector functions; vect
culus; extrema and Lagrange multipliers. Introduction to multiple integrals and coordin
. Problem solving with MAPLE.\nPrerequisite: Cegep Mathematics 105 or 201-NYC, 203 or 2
t.\nNOTE: Students who have received credit for MATH 262 may not take this course for c
three credits will be awarded from MAST 218; MATH 264.\n\n\n']
Your input ->  what is description of ENGR213
If you are asking about ENGR213, the results are as follows:['This course introduces En
to the theory and application of ordinary differential equations. Definition and termin
e problems, separable differential equations, linear equations, exact equations, soluti
, linear models, orthogonal trajectories, complex numbers, form of complex numbers; pow
```

4.4 Result of Query from Assignment#1

When you implement the python code, you need to restart the rasa server and use rasa shell to enter the chatbot interface in command line.



5. Conclusion

Based on the competency questions, the team works together to design the intelligent agent. The integration of different parts is the key to implement and establish the project, including retrieving the data from open sources and convert it to the knowledge base by using reused vocabularies and own designed vocabularies, and setting up the endpoint server to execute the queries to obtain the answers.

Through project 1, the knowledge base is constructed well and the endpoint server is also set up for executing queries. The fundamental infrastructure has been implemented which will serve the next part, the natural language processing.

Through project 2, the topics are extracted using DBpedia spotlight. And the chatbot can answer the questions about a course using Rasa library and our knowledge base.