TEAM 4

Team members: Zhiqing YUAN #26258840

Nian Liu #40044346

Yaohua Zhang #40073090

Jia Ming Wei #40078192

Abstract

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

comp474 project

Assignment #1

**Table of Contents**

Contents

1.0 Introduction 2

1.1 The Goal 2

1.2 The Team 2

3.0 Result and Discussion 5

3.1 Result 5

3.2 Discussion 8

4.0 Rasa 9

4.1 Design 9

4.1.1 nlu.yml 9

4.1.2 stories.yml 9

4.1.3 domain.yml 10

4.2 Implementation 10

4.3 Result 12

# **1.0 Introduction**

## **1.1 The Goal**

The project 1 is to create a knowledge base using the own vocabulary and populate the knowledge base with information about Concordia’s courses from the open datasets.

## **1.2 The Team**

The project has been implemented by four team members via Github and google driver.

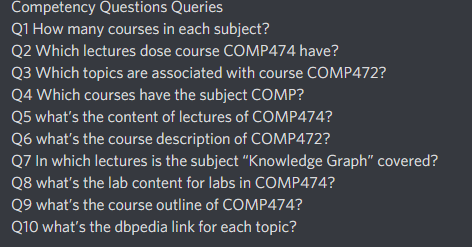
The team meeting is 1-2 times per week and to follow up the process of the project.

**2.0 Implementation**

**2.1 Design**

In order to construct the knowledge base, we have to model the graph to represent the information which could answer/match the queries.

The first process is to clear the competency questions which the intelligent agent could answer. For example, how many courses in each subject? Which lectures dose course COMP474 have? Which topics are associated with course COMP472? etc.



The next step is to model the schema for the knowledge base, including the reused vocabulary and developed/defined vocabulary. Based on the schema, the dataset which is obtained via the open data source of Concordia could be added to server. In this case, our team chooses server Apache Jena Fuseki with localhost:3030. Finally, the questions are transformed to SPARAL queries and outputs/answers will be provided after executing these queries.

**2.2 Implementation**

**2.2.1 Vocabulary**

In order to model the schema for the knowledge base, the implementation includes the vocabularies reused and vocabulary extension developed.

For the vocabularies reused, besides the common vocabularies, such as rdf, rdfs xsd and dbr, we also choose aiiso, teach and vivo. For example, aiiso is the Academic Institution Internal Structure Ontology and it provides classes and properties to describe the academic institution (eg, aiiso:code for course number, aiiso:name for course name). Meanwhile, it can also work/participate with FOAF and roles to describe the people play with an institution.

For the vocabulary extension developed, we defined focu for the schema, including Course Outline, Lab, Reading Slide, Subject, Topic, Tutorial and Worksheet. The reason is because these classes are specific and served for Concordia. Therefore, we need to define these classes/subclasses(hierarchy) and relationship.

The benefits to use the vocabularies reused include save the time(you don’t need to define some classes and properties by yourself), keep standard and work well with other data link resources. For example, aiiso participates with FOAF, role etc. However, we still need to define some classes and properties by ourselves while dealing with some areas which the reused vocabularies don’t cover or maybe not very fit our requirement.

**2.2.2 Knowledge Base Construction**

After I implement the crawler process based on the project requirements/constraints.

First, I set the start url is Concordia; second, I set the crawler has to obey the standard for robot exclusion with setting “ROBOTSTXT BOEY: True”; Third, I set the upper bound on the total number of files to be downloaded with “docbound=100” (this amount can be revised if we need to retrieve more documents); Forth, I set the duration for time out control.

**2.2.3 Knowledge**

I obtain the new id in this python file, there are two parts to implement.

The first part is to read documents which are retrieved from the web crawler. Then, the function final\_index(path,k) will process the documents to final index. The parameter path is the location to store the documents and k is block size while processing the terms temporarily. In the body of this function, it called tokenize(doc), add\_dictionary(token, own\_dict), posting\_list(document\_list, k), merge\_tf and merge\_df. These methods are used to tokenize the text, encode tf with the docID in the posting list and df with vocabulary terms.

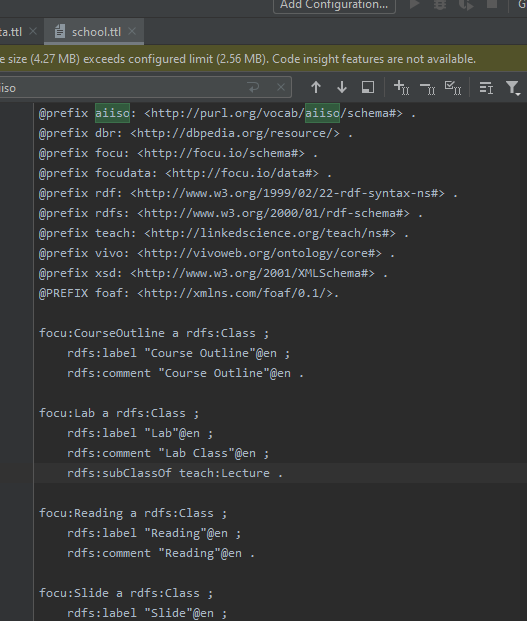
The second part is to create the ranking methods, including BM25 and simple tf/idf weighting. The function bm25(merge\_list, documents\_list, query\_list, query\_result, k,b) is to implement the bm25 formular and the rank\_doc\_tfidf(merge\_list, documents\_list, query\_list,query\_return\_list). The query types methods are also used to let user choose different query type AND/OR. These functions will implement the intersection and union operation based on the user’inputs.

# **3.0 Result and Discussion**

## **3.1 Result**

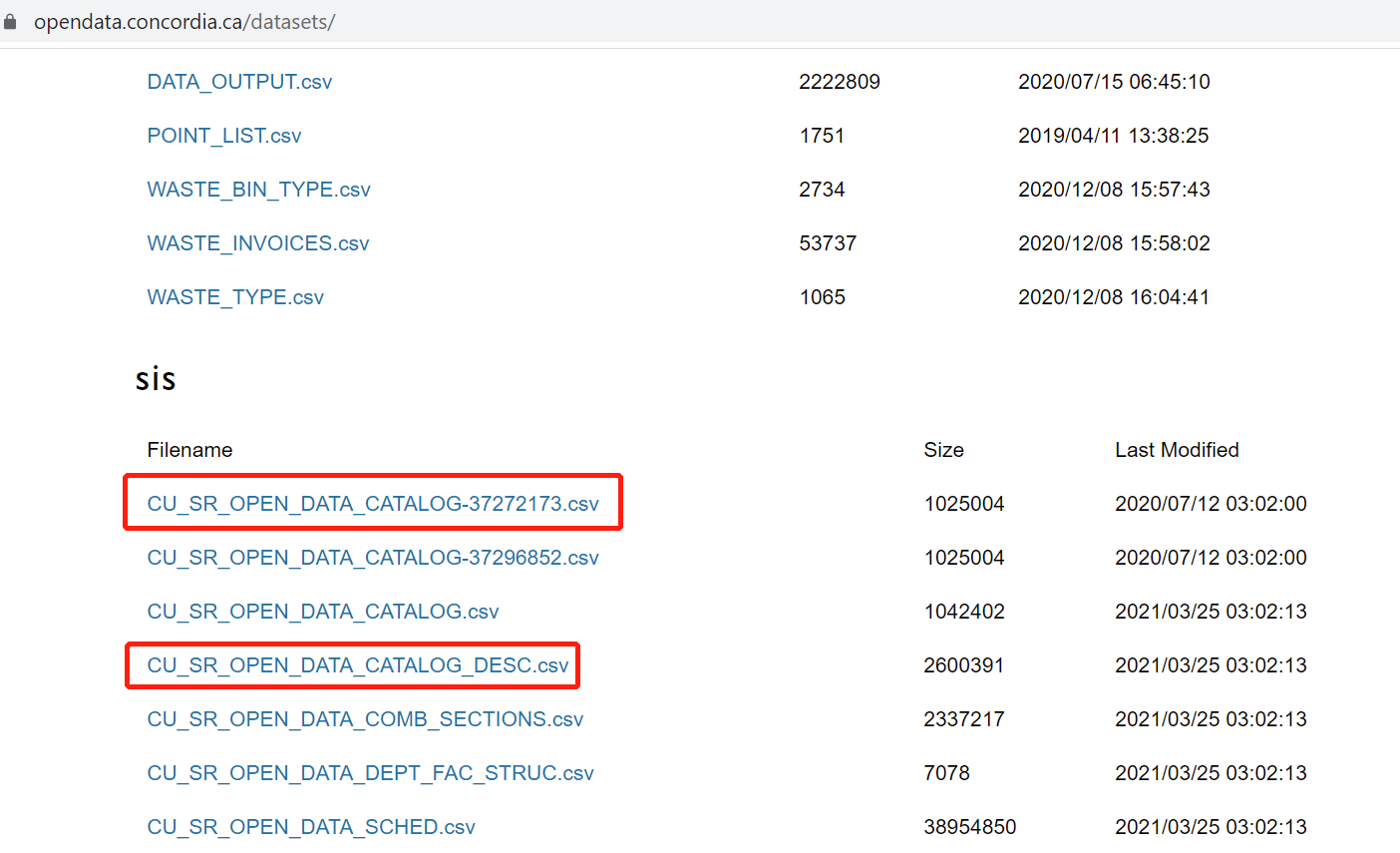
After the design and implement, the schema, dataset and server are set up for the queries.

**3.1.1 RDF Schema**  The RDF schema includes the vocabularies reused and vocabulary extension developed.



**3.1.2 Dataset**

The open dataset we used is https://opendata.concordia.ca/datasets/. The data format is csv which we need to convert it via generate.py.



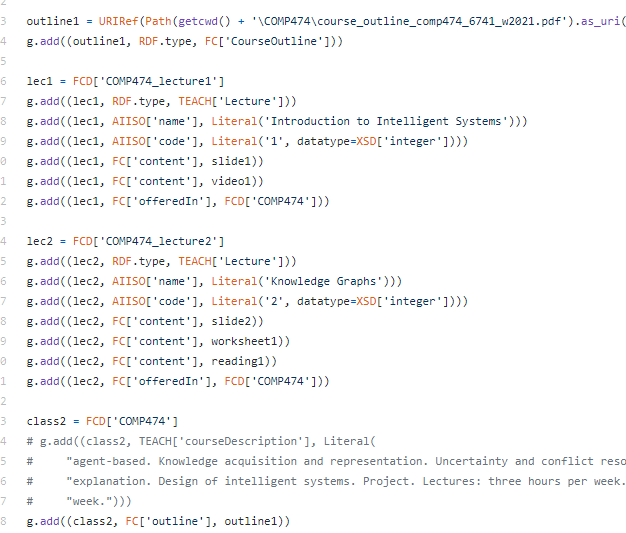
**3.1.3 KB Construction**

We use ttl\_generator.py to automatically construct the knowledge base from the dataset.

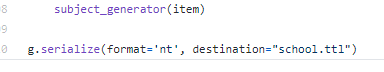
First of all, using the rdf library to bind namespace and build class and property.



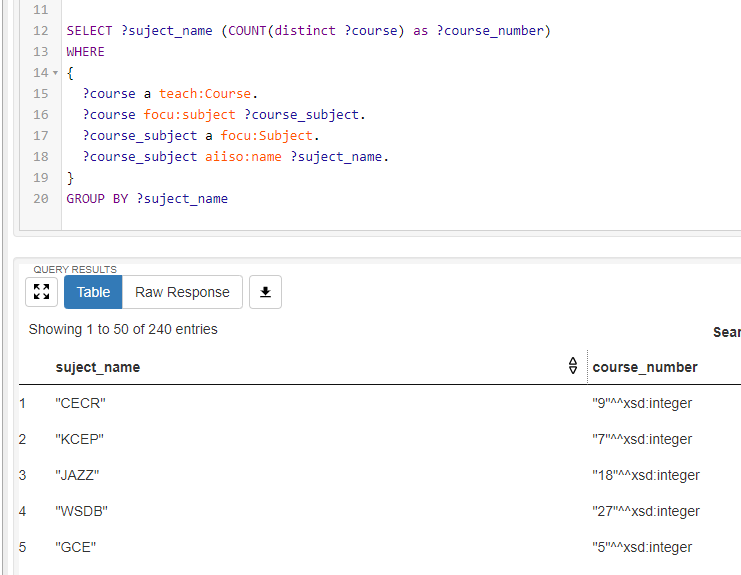
Second, the course COMP 474 and COMP 472 are chosen to our knowledge base. They have more information which are added to the knowledge base.



Finally, we merge the data and generate the ttl file which will be used for queries via Fuseki server.



**3.1.4 Queries and Result** Set up the Fuseki server and run queries. One example is as follows. The output of query is to display the subject names and how many courses are in each subject.

****

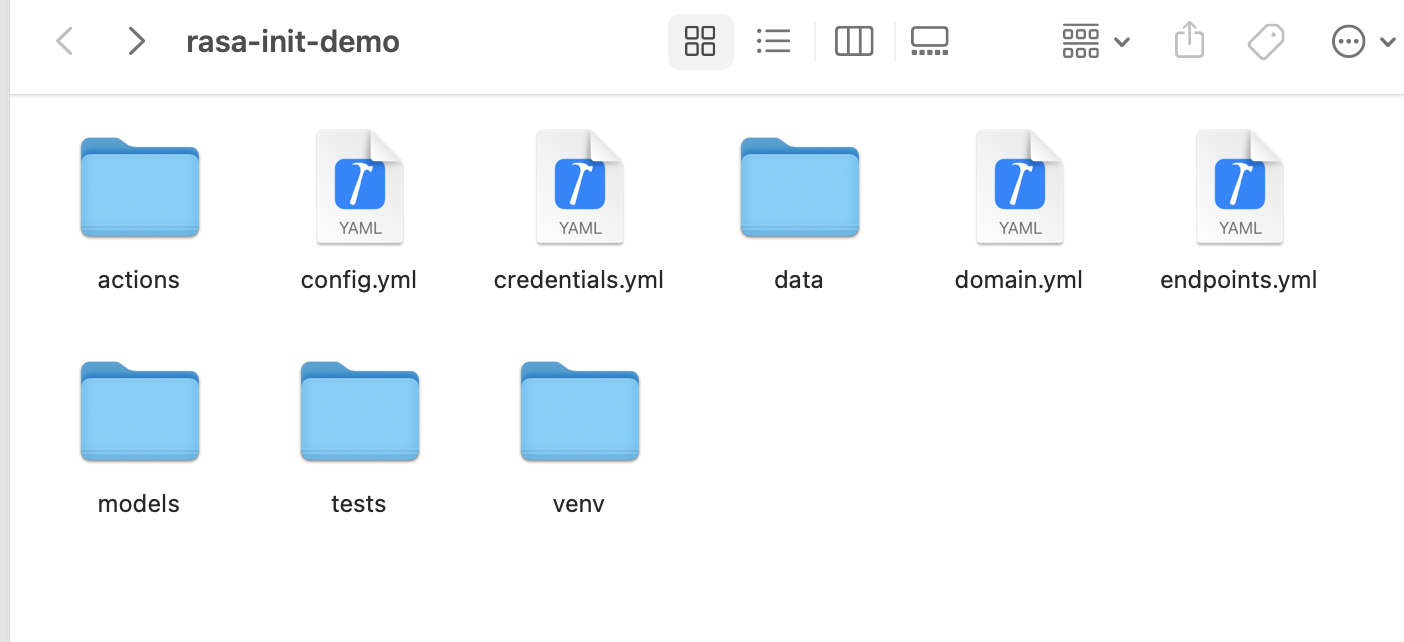
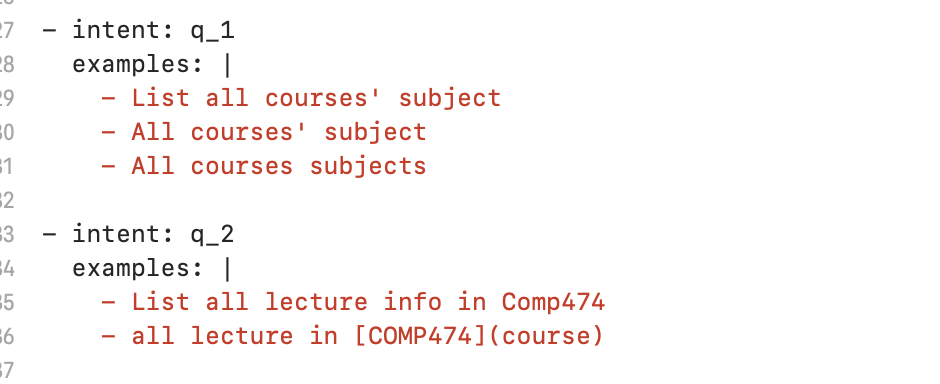
## **3.2 Discussion**

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 information/answer.

# **4.0 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.

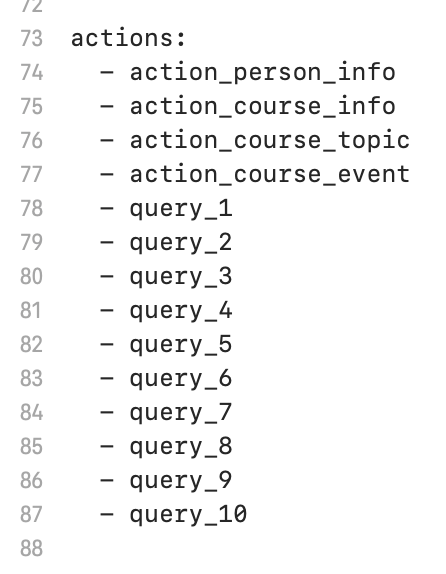
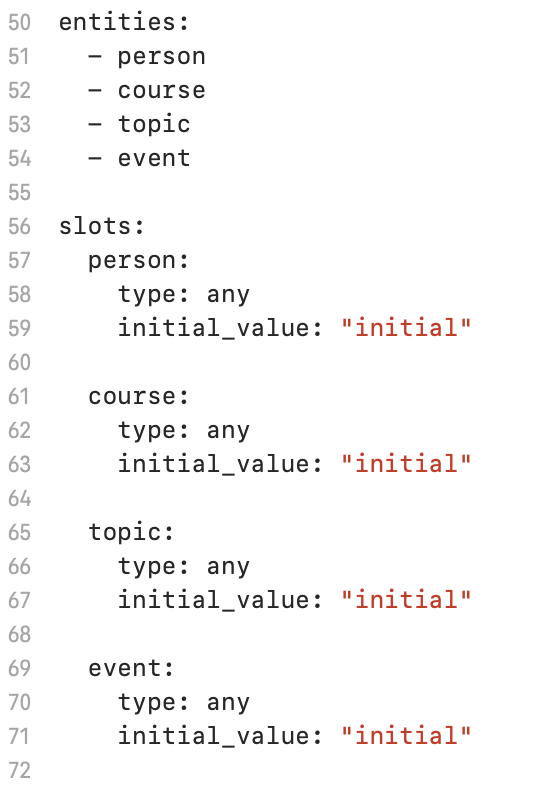
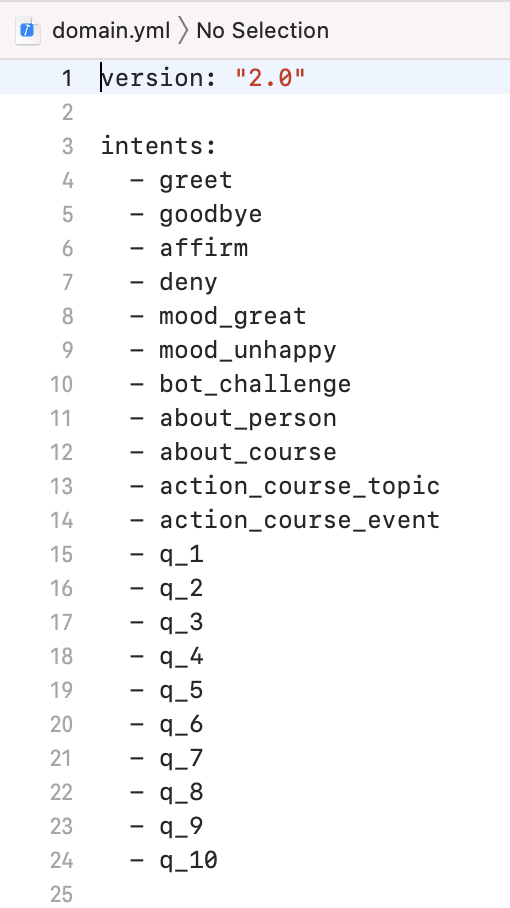
Create some intents with example, then when you use chatbot it will find out your questions’ intention.

# **4.1.2 stories.yml**

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

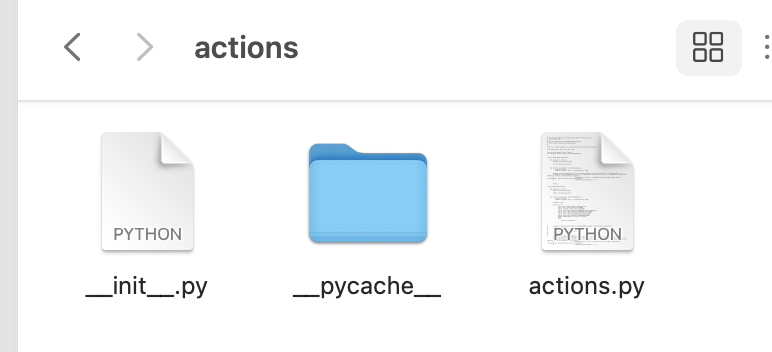
The story will create the format that how chatbot will answer those intents with actions.****

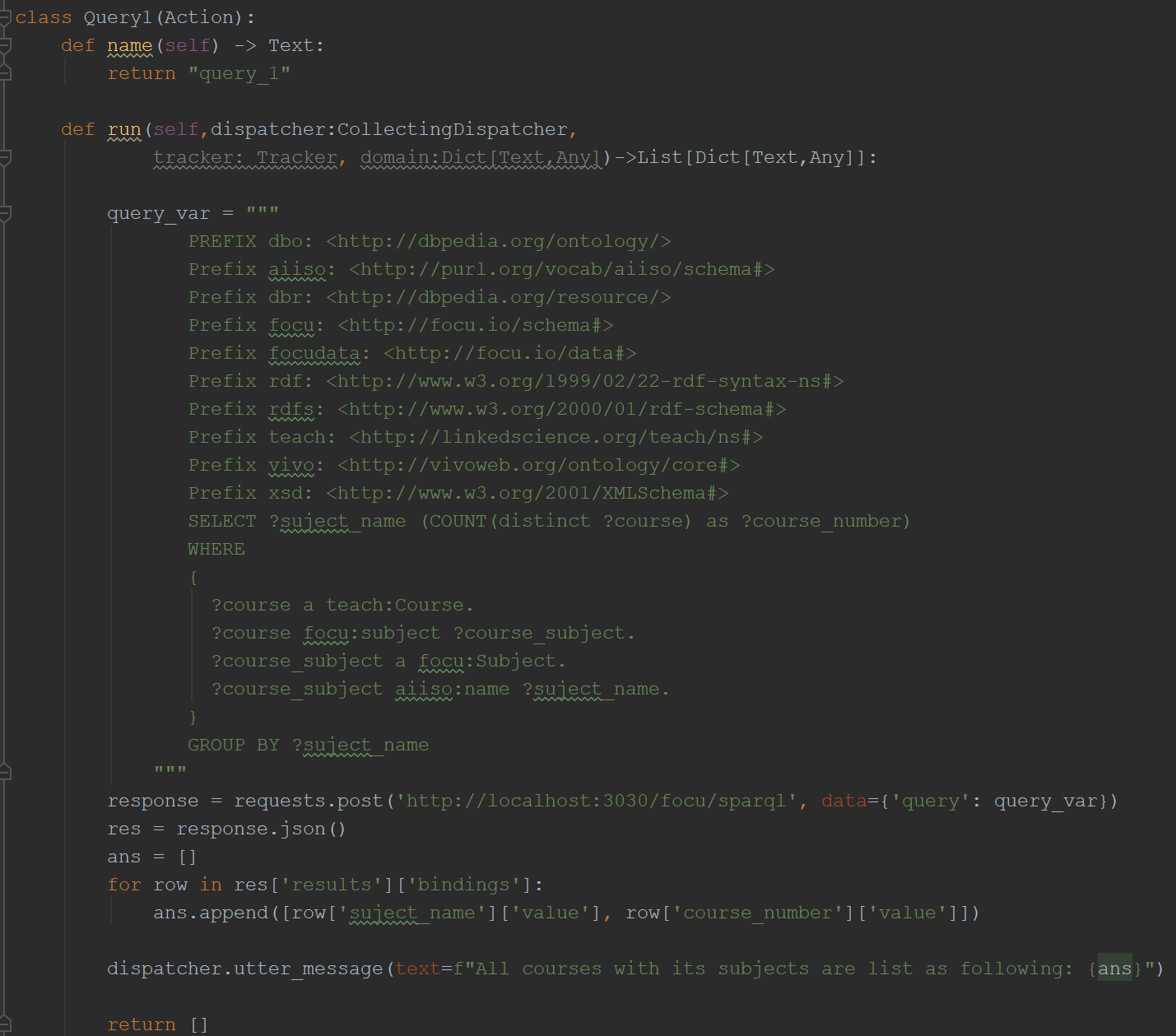
# **4.1.3 domain.yml**

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

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 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.**

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 Result**

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.****