Project Part 1

Team FS_U_03

Daniel Grimut	40211401
Alessio Cipriano-Kardous	40210549
Nicholas Pop	40210550
Chance Sztyk	40213384

Comp 474

Instructor: Dr. Rene Witte
22nd March 2024

Specialization:

Name	**Specialization**
Daniel Grimut	RDFS vocabulary, Student and Grade parsing, Automated Knowledge Base, Report
Nihoclas Pop	Query Construction, Automated Knowledge Base Construction, Report
Chance	Query Construction, Code Review, Proofreading, Report
Alessio Cipriano-Kardous	Automated Knowledge Base Construction, Python SPARQL script, web scraping, Report

Github:

https://github.com/D-grimut/Knowledge_AI

"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality",

Signatures:

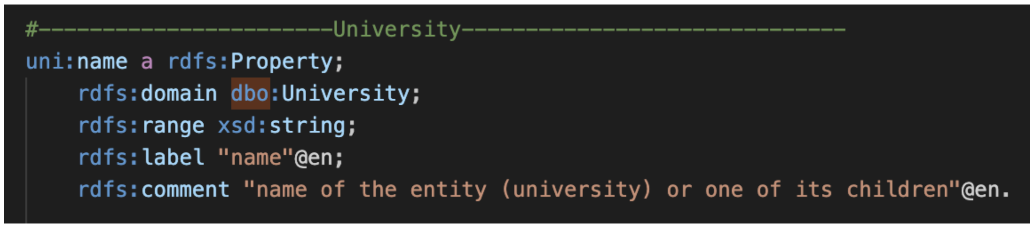
Daniel Grimut (40211401), Chance Sztyk (40213384), Alessio Cipriano-Kardous (40210549), Nicholas Pop (40210550)

Vocabulary

In this section, we will outline in detail the ways we modeled our RDFS vocabulary for this project. This section is split into 5 sections: University, Course, Lectures, Topics, and Student. Each category corresponds to its RDFS counterpart in the vocab.ttl file. Each section will present a table with the vocabulary we reused, the vocabulary extensions we made, and a short description of the resource itself and its interaction with other relevant resources we created. Finally, this section will conclude with a general overview of how all resources within our RDFS vocabulary interact with each other.

University

The university resource models the university entity within our knowledge graph. The entity encapsulates information about the university with the help of the uni:name property we defined, and all other information by being an instance of the dbo:University, an existing University resource within DBpedia.



```
#----------------------University-----------------------------
uni:name a rdfs:Property;
    rdfs:domain dbo:University;
    rdfs:range xsd:string;
    rdfs:label "name"@en;
    rdfs:comment "name of the entity (university) or one of its children"@en.
```

Vocabularies Reused:

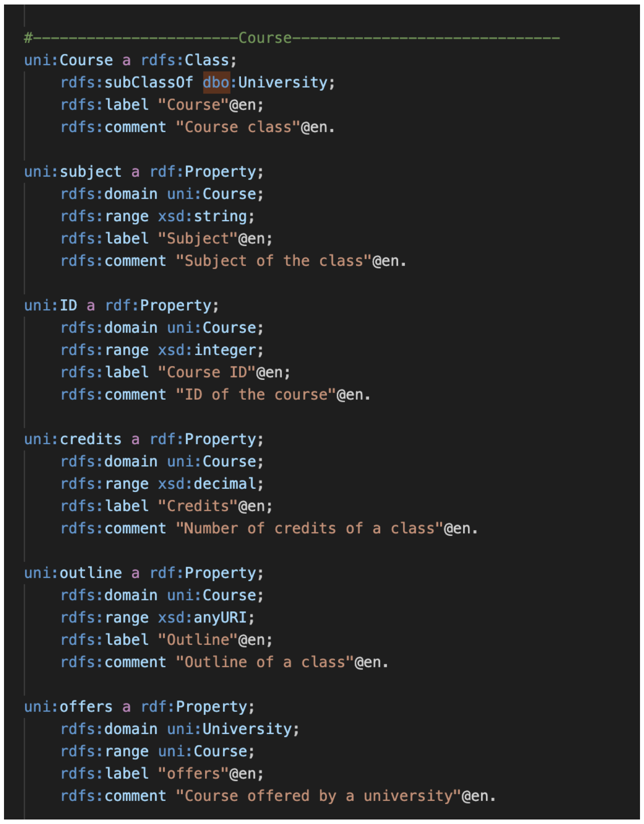
Vocab Name	**Description**
dbo:University	We re-used the dbo:University vocabulary from DBPedia's existing ontologies. This allowed us to model a University object without having to define properties about it and simply reuse existing ones.
owl:sameAs	To link the University instances in our knowledge base, with their counterpart URIs in DBPedia and Wikidta, we decide the reuse the owl:sameAs property.

New Concepts Defined/Vocabularies Extended:

Vocab Name	**Type**	**Description**
uni:name	rdfs:Property;	The uni:name is a RDFS property that we defined to store the name, a human-readable string, of all university instances and children

<u>Course</u>

The course resource models the courses offered in a university. To keep consistency within our resources, we have decided to make the course a sub-class of dbo:University, as that allows the course to be treated as an instance of dbo:University. This is beneficial as it allows us to recycle the uni:name property between both university and course instances. Additionally, this creates a logical hierarchy within our code, where courses are subclasses of universities.



```
#----------------------Course-----------------------------
uni:Course a rdfs:Class;
    rdfs:subClassOf dbo:University;
    rdfs:label "Course"@en;
    rdfs:comment "Course class"@en.

uni:subject a rdf:Property;
    rdfs:domain uni:Course;
    rdfs:range xsd:string;
    rdfs:label "Subject"@en;
    rdfs:comment "Subject of the class"@en.

uni:ID a rdf:Property;
    rdfs:domain uni:Course;
    rdfs:range xsd:integer;
    rdfs:label "Course ID"@en;
    rdfs:comment "ID of the course"@en.

uni:credits a rdf:Property;
    rdfs:domain uni:Course;
    rdfs:range xsd:decimal;
    rdfs:label "Credits"@en;
    rdfs:comment "Number of credits of a class"@en.

uni:outline a rdf:Property;
    rdfs:domain uni:Course;
    rdfs:range xsd:anyURI;
    rdfs:label "Outline"@en;
    rdfs:comment "Outline of a class"@en.

uni:offers a rdf:Property;
    rdfs:domain uni:University;
    rdfs:range uni:Course;
    rdfs:label "offers"@en;
    rdfs:comment "Course offered by a university"@en.
```

<u>Vocabularies Reused:</u>

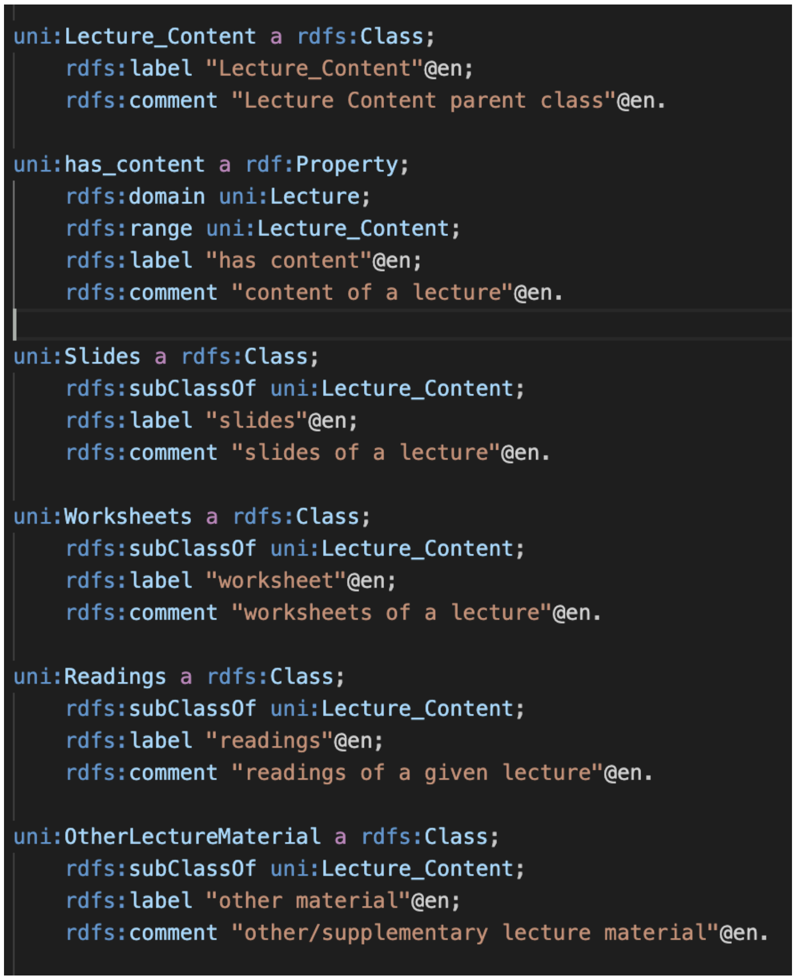
<u>Vocab Name</u>	**<u>Description</u>**
dbo:University	As mentioned, reusing dbo:University allows us to treat course entities as University entries with in turn allows us to recycle uni:name used to associate names with courses and universities.

New Concepts Defined/Vocabularies Extended:

Vocab Name	**Type**	**Description**
uni:Course	rdfs:Class	As we could not find an existing ontology for course, we defined our own course class in our RDFS vocabulary to model all entities.
uni:subject	rdf:Property	Associates a course instance with the subject of the course. The subject is a human-readable string.
uni:ID	rdf:Property	Associates a course instance with an ID (course number). The course is associated with an Integer.
uni:credits	rdf:Property	Associates a course instance with the then number of credits for that course. The number of credits is a decimal.
uni:outline	rdf:Property	Associates a course instance its outline. The outline is represented by a URI, on the user's computer or somewhere on the Web.
uni:offers	rdf:Property	Associates a university instance with a given course. It indicates which courses are offered by a university

Lectures

The lecture resource was defined to model the lectures that are given by various courses. Each lecture is its own entity that is connected to a specific course. In other words, course entities in our RDF graph are connected to the lecture that they give. Additionally, each lecture has an array of educational material associated with them. Such material includes slides, worksheets, readings, and other lecture materials. All of the aforementioned lecture content entities were made into their classes; all sub-classes of one parent resource called Lecture_Content. This hierarchy enabled us to regroup educational material that is related to some lecture logically. This sub-hierarchy can be observed in the following screenshot from our RDFS:



```
uni:Lecture_Content a rdfs:Class;
    rdfs:label "Lecture_Content"@en;
    rdfs:comment "Lecture Content parent class"@en.

uni:has_content a rdf:Property;
    rdfs:domain uni:Lecture;
    rdfs:range uni:Lecture_Content;
    rdfs:label "has content"@en;
    rdfs:comment "content of a lecture"@en.

uni:Slides a rdfs:Class;
    rdfs:subClassOf uni:Lecture_Content;
    rdfs:label "slides"@en;
    rdfs:comment "slides of a lecture"@en.

uni:Worksheets a rdfs:Class;
    rdfs:subClassOf uni:Lecture_Content;
    rdfs:label "worksheet"@en;
    rdfs:comment "worksheets of a lecture"@en.

uni:Readings a rdfs:Class;
    rdfs:subClassOf uni:Lecture_Content;
    rdfs:label "readings"@en;
    rdfs:comment "readings of a given lecture"@en.

uni:OtherLectureMaterial a rdfs:Class;
    rdfs:subClassOf uni:Lecture_Content;
    rdfs:label "other material"@en;
    rdfs:comment "other/supplementary lecture material"@en.
```

In summary, for all Lecture related resources (i.e. Lectures, Lecture_Content, Slides, Worksheets, Readings, and OtherLectureMaterial) we created new RDFS resources from scratch, without extending existing vocabularies. We saw it most appropriate to model these entities as we could not find them in existing ontologies, and as that it would make these resources tailored for our purpose.

Vocabularies Reused:

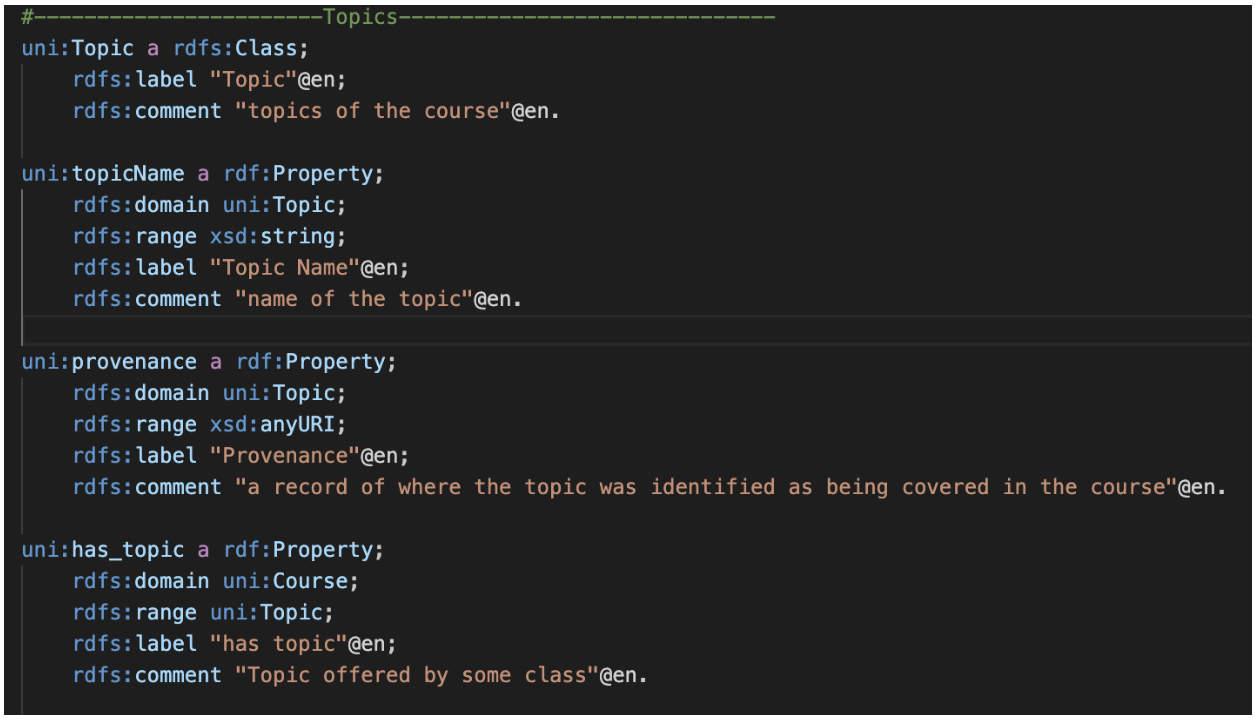
Vocab Name	**Description**
n/a	n/a

New Concepts Defined/Vocabularies Extended:

Vocab Name	**Type**	**Description**
uni:Lecture	rdfs:Class	The Lecture resource that was created to model Lectures given by Courses.
uni:lecture_number	rdf:Property	The number of lectures given by some course, is a property used to associate a unique identifier of a lecture of a course.
uni:has_lecture	rdf:Property	This property associates lectures to course entities. That said, it gives a possibility to query the lectures offered by a course.
uni:Lecture_Content	rdfs:Class	This RDFS class was created as the parent class to logically regroup all lecture materials.
uni:has_content	rdf:Property	This property associates lecture content to a lecture. That is, it allows us to query all material given by some lecture.
uni:Slides	rdfs:Class	This RDFS class was made to represent all entities that are URIs associated with the Slides of a lecture.
uni:Worksheet	rdfs:Class;	This RDFS class was made to represent all entities that are URIs associated with the Worksheets of a lecture.
uni:Readings	rdfs:Class;	This RDFS class was made to represent all entities that are URIs associated with the Readings of a lecture.
uni:OtherLectureMaterial	a rdfs:Class;	This RDFS class was made to represent all entities that are URIs associated to all other material of a lecture (Websites, extra readings, etc.)

Topics

The topic resource was created to model all topics associated with courses. That is, the topic resource represents the topics that are offered in different courses. It is worth mentioning that only one node is ever created for a specific Topic. That is, if two courses cover the same topic (ex: COMP 474 and COMP 472 both cover the topic AI), the two distinct course entities will point to the same topic entity (same topic URI), within the knowledge graph. This modeling allows us to save memory and improve efficiency by avoiding the creation of redundant nodes that represent the same things. Additionally, this allows for easier querying when determining which topics are covered in which courses, as well as finding courses that offer the same topic.



```
#-----------------------Topics-----------------------------
uni:Topic a rdfs:Class;
    rdfs:label "Topic"@en;
    rdfs:comment "topics of the course"@en.

uni:topicName a rdf:Property;
    rdfs:domain uni:Topic;
    rdfs:range xsd:string;
    rdfs:label "Topic Name"@en;
    rdfs:comment "name of the topic"@en.

uni:provenance a rdf:Property;
    rdfs:domain uni:Topic;
    rdfs:range xsd:anyURI;
    rdfs:label "Provenance"@en;
    rdfs:comment "a record of where the topic was identified as being covered in the course"@en.

uni:has_topic a rdf:Property;
    rdfs:domain uni:Course;
    rdfs:range uni:Topic;
    rdfs:label "has topic"@en;
    rdfs:comment "Topic offered by some class"@en.
```

Vocabularies Reused:

Vocab Name	Justification
n/a	n/a

New Concepts Defined/Vocabularies Extended:

Vocab Name	Type	Description
uni:Topic	rdfs:Class	The Topic RDFS that models all topic entities within the knowledge graph.
uni:topicName	df:Property	The RDFS property to identify the name of a Topic. This is a simple human-readable string.
uni:provenance	rdf:Property	This RDFS property was created to keep track of the provenance of the Topic. That is, it helps keep track of Topics and where in the course (which lectures and which lecture materials) were these topics identified in. It follows that the provenance link points to a Lecture_Content URI where the Topic was identified.
uni:has_topic	rdf:Property	This RDFS property was created to associate courses with topics. In other words, it connects courses to topics that said courses offer.

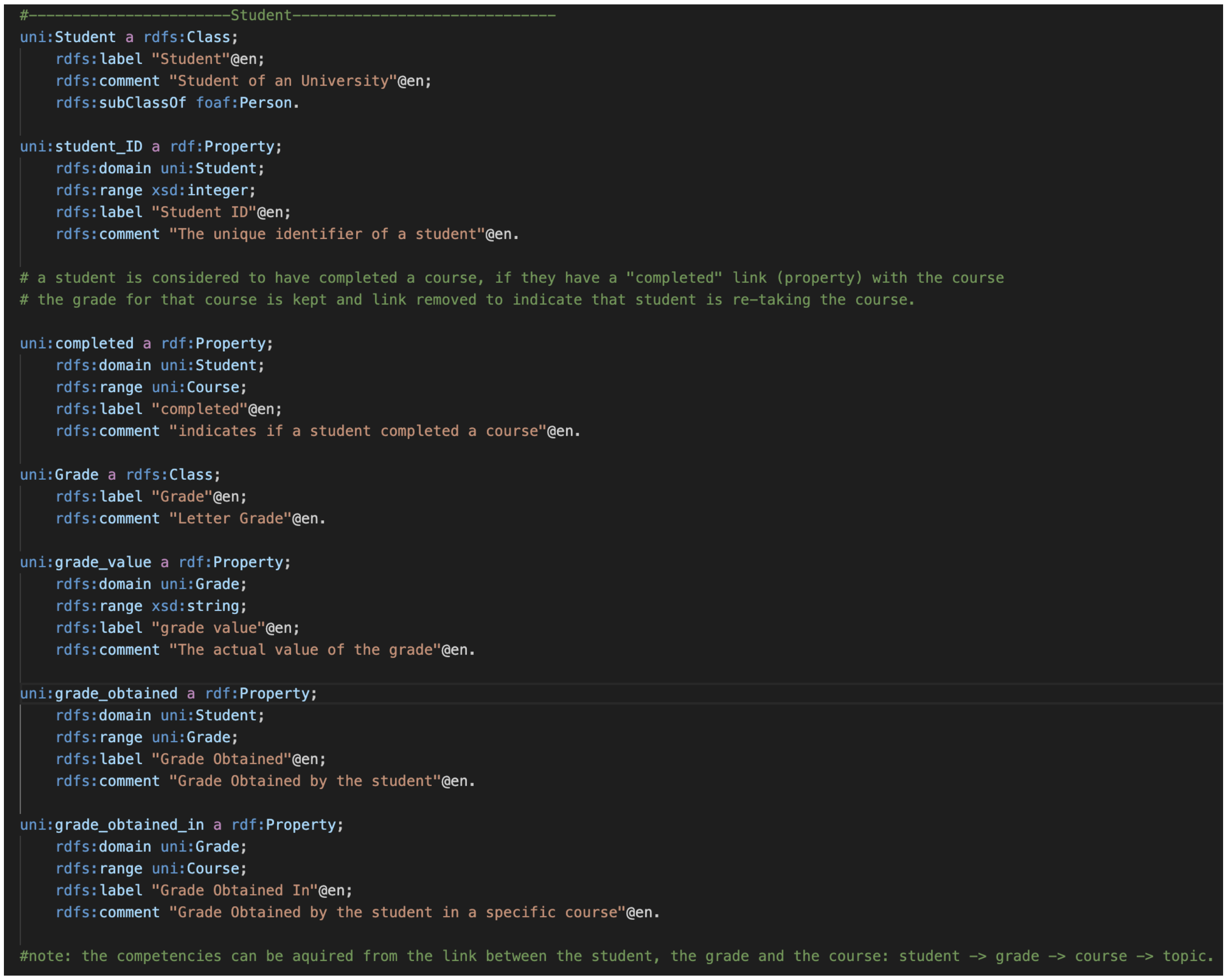
<u>Student</u>

The student RDFS resource was made to model students who are enrolled in a specific course. The student class is a sub-class of FOAF: Person, which allows to reuse of the first name, last name, and email properties from the FOAF vocabulary, to model the student entity. In addition to FOAF, we added our properties such as student ID to associate unique student IDs to each student.

We modeled a grade resource in RDFS to represent the grade obtained by a student in a specific class. It is worth noting that each grade for a student is identified as being the letter grade and the student ID concatenated. Thus, each grade obtained in a course by a student is only created once for that student. For example, if a student with ID 42, gets an A in COMP 474 and 472, then only one grade node for that student is created; the grade being identified as “A_42” (in the URI). This identification mechanism allows us to create grades for students that are shared amongst courses but refer back to a unique student. As such, we do not create a new grade entity when a student obtains a grade for a course they had already obtained for another course (refer to the example previously mentioned). In consequence, we avoid creating redundant nodes and facilitate the querying of grades for a student.

Additionally, we can determine if a student completed a course by verifying if they have a link called “completed” (an RDFS property we created), between them and a given course. As such, a student can have a grade obtained for a specific course, but if it is a failing grade (or the student re-takes the class), the completed property will be missing between the student and the course node, signaling that the student has not yet completed the course or is in the process of re-doing it.

As a final note, we have not explicitly defined the competencies of a student. Since we have established a link between the student, the completed course, and the topic (student -> course -> topic), we can query all the competencies of a student through SPARQL. This design allows us to keep the creation of nodes to a minimum, to preserve memory, and to improve efficiency, by relying more on querying to retrieve the desired information from the knowledge graph.



```
#----------------------Student------------------------------
uni:Student a rdfs:Class;
    rdfs:label "Student"@en;
    rdfs:comment "Student of an University"@en;
    rdfs:subClassOf foaf:Person.

uni:student_ID a rdf:Property;
    rdfs:domain uni:Student;
    rdfs:range xsd:integer;
    rdfs:label "Student ID"@en;
    rdfs:comment "The unique identifier of a student"@en.

# a student is considered to have completed a course, if they have a "completed" link (property) with the course
# the grade for that course is kept and link removed to indicate that student is re-taking the course.

uni:completed a rdf:Property;
    rdfs:domain uni:Student;
    rdfs:range uni:Course;
    rdfs:label "completed"@en;
    rdfs:comment "indicates if a student completed a course"@en.

uni:Grade a rdfs:Class;
    rdfs:label "Grade"@en;
    rdfs:comment "Letter Grade"@en.

uni:grade_value a rdf:Property;
    rdfs:domain uni:Grade;
    rdfs:range xsd:string;
    rdfs:label "grade value"@en;
    rdfs:comment "The actual value of the grade"@en.

uni:grade_obtained a rdf:Property;
    rdfs:domain uni:Student;
    rdfs:range uni:Grade;
    rdfs:label "Grade Obtained"@en;
    rdfs:comment "Grade Obtained by the student"@en.

uni:grade_obtained_in a rdf:Property;
    rdfs:domain uni:Grade;
    rdfs:range uni:Course;
    rdfs:label "Grade Obtained In"@en;
    rdfs:comment "Grade Obtained by the student in a specific course"@en.

#note: the competencies can be aquired from the link between the student, the grade and the course: student -> grade -> course -> topic.
```

Vocabularies Reused:

Vocab Name	Justification
foaf:Person	The student RDFS class is a subclass of person. As mentioned previously, this allows us to re-use properties such as name, last name, and email, that identify a student, without explicitly defining these entities in our graph.

New Concepts Defined/Vocabularies Extended:

Vocab Name	Type	Justification
uni:student_ID	rdfs:Property	RDFS property to define the student ID. The ID is a simple integer.
uni:completed	rdfs:Property	RDFS property to indicate if a student completed a course. If the student is retaking the course or did not complete it, this property does not exist between the course and the student.
uni:Grade	rdfs:Class;	RDFS class that models all grade entities.
uni:grade_value	rdfs:Property	RDFS property that we created to identify the value of a grade (A, B, C, D, etc.)
uni:grade_obtained	rdfs:Property	RDFS property that links the student to their grade. In other words, a student node is connected to a grade node using this property.
uni:grade_obtained_in	rdfs:Property	RDFS property connects a specific grade obtained by a student, to the course in which the grade was obtained. This is an important property as it allows the regrouping of all courses where the student got a specific grade.

Summary of the RDFS Vocabulary

In summary, the University RDFS resource extends the DBpedia ontology to model the university. University instances serve as logical regroupments for courses offered in said universities; that is courses are subclasses of dbo:University. Courses are modeled to have subjects, IDs, credits, and outlines (as URIs) attached to them. Courses are associated with specific university nodes through the uni:offers property we defined. Following, lectures are explicitly modeled and are associated with courses through uni:has_lecture. Each lecture also possesses Lecture Content that is modeled in a sub-hierarchy, where each lecture content type is its own class. Connected to specific Lectures and Lecture materials are topics, which model specific topics covered in the courses. Finally, Students are modeled in our RDFS as sub-classes of foaf:Person, with additional properties defined for informational completeness about students. Each student is connected to their respective grades, where each grade is defined as its own node, with a unique identifier that includes the ID of the student to whom this node pertains; which is connected to the course where the student obtained this grade. For a deeper understanding of our RDFS vocabulary, we have included an image file within our submission folder - it is too big to be added to the report - generated using RDF Grapher[1].

[1] https://www.ldf.fi/service/rdf-grapher

Knowledge base construction

We used pre-existing data we found from the Concordia catalogs, course lectures, and some data we created ourselves. The Concordia Catalogs (CATALOG.csv and CU_SR_OPEN_DATA_CATALOG.csv) hold information about every course given at Concordia. The most notable and important ones are the course key, course number, course code, course title, course units, and course description. As for the data that we manually created, we created two datasets (students.csv and grades.csv). The student.csv was a list of information on 10 fake students. This includes their student ID, first name, last name, and email. Our grades.csv links the students by their student ID to a grade they got in both COMP 354 and COMP 474. Lastly, we retrieved each lecture given by both classes respectively and stored them. To do this, we accessed Moodle to download every file, categorize them into different folders, and upload them into our program. We needed all this information because of how the vocabulary was set up. The use of methods that we created to gather the information and format it to respect our vocabulary was crucial to ensure that our program worked as intended.

To create the knowledge base, we created different methods to access the information and formulate the knowledge structure. The first method is get_course_info() which opens both Concordia Catalogs CSV files, as mentioned before, and gathers the information necessary for each course. This method takes all of the information mentioned before and stores it into a dictionary, for future ease to get the information. The next method is get_files(), and this gathers all of the lectures of the courses available in the program and formats a name. This formatted name gets sent to create_URI(), which creates the URI for each file. The last method to gather information is student_info_extract(). This method gathers the information from the manually created data (grades and students), and adds them into a dictionary for future use.

Once all of the information was gathered, we created the graph using the create_course_graph() method. We defined all of the namespaces and added triples to the graph using the dictionaries we created prior. The only triples we added manually were the Topics triples, as it was mentioned to do so in the assignment. Once all of the triples were added, we serialized the graph in both Turtle and n-triples format.

Graph Queries

Translating the questions provided in the assignment into SPARQL queries was one of the easier challenges when creating this project. All of the queries could be independently tested using the Fuseki server's web interface prior to their incorporation in our code.

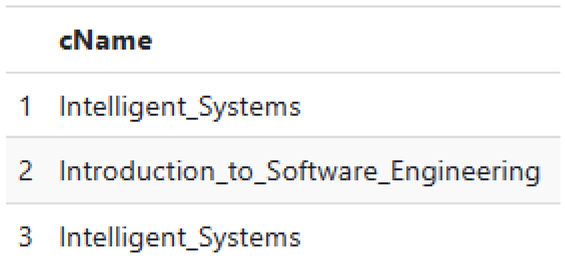
Here are some example outputs for each query. Note that every query has this set of definitions for namespaces for it to work properly:

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix dbo: <http://dbpedia.org/ontology/>
prefix uni: <http://uni.com/schema#>
prefix owl: <http://www.w3.org/2002/07/owl#>
```

Query 1: Provides all courses offered by a university

```
SELECT ?cName
WHERE{
?uni rdf:type dbo:University.
?uni uni:name ?uniName.
?uni  uni:offers ?course.
?course uni:subject ?cName.
}
LIMIT 100
```

Output:



	cName
1	Intelligent_Systems
2	Introduction_to_Software_Engineering
3	Intelligent_Systems

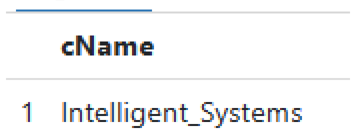
Query 2: Provides which courses discuss a specific topic

```
SELECT ?cName
WHERE{
?course uni:has_topic ?topic.

?topic uni:topicName ?tName.
?course uni:subject ?cName.

FILTER(REGEX(STR(?tName), 'Deep Learning', "i"))
}
LIMIT 100
```

Output:

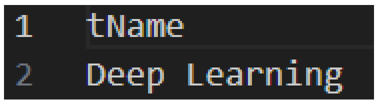


	cName
1	Intelligent_Systems

Query 3: Provides topics covered in a course during a specific lecture

```
SELECT ?tName
WHERE{
?course uni:subject ?cName.
?course uni:has_lecture ?lecture.
?lecture uni:lecture_number ?lecNum.
?topic uni:provenance ?courseMat.
?course uni:has_lecture ?courseMat.
?topic uni:topicName ?tName.
FILTER(REGEX(STR(?cName), 'Intelligent_Systems', "i")).
FILTER(?lecNum=5).
}
LIMIT 100
```

Output:

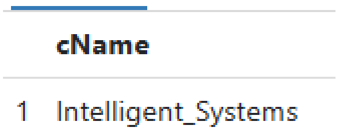


```
1   tName
2   Deep Learning
```

Query 4: Provides courses offered by a university within a subject

```
SELECT ?cName
WHERE{
?uni rdf:type dbo:University.
?uni uni:name ?uniName.
?uni uni:offers ?course.
?course rdf:type uni:Course.
?course uni:subject ?cName.
?course uni:has_topic ?topic.
?topic uni:topicName ?tName.
FILTER(REGEX(STR(?uniName), 'Concordia', "i")).
FILTER(REGEX(STR(?tName), 'Deep Learning', "i")).
}
LIMIT 100
```

Output:

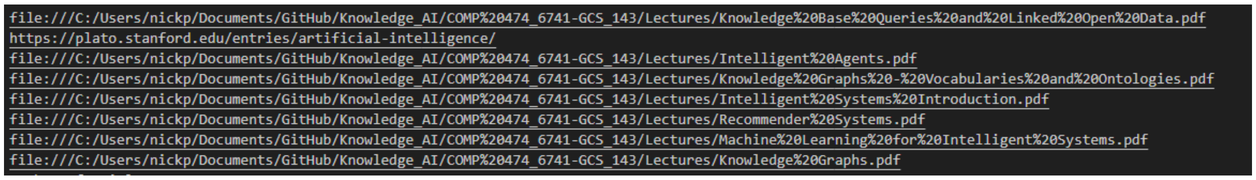


	cName
1	Intelligent_Systems

<u>Query 5: Provides recommended materials for a topic in a course with it's number</u>

```
 SELECT ?content
WHERE{
?course uni:ID ?cnumber.
?course uni:has_topic ?topic.
?course uni:has_lecture ?lecture.
?lecture uni:has_content ?content.
FILTER(?cnumber=474).
}
LIMIT 100
```

Output:

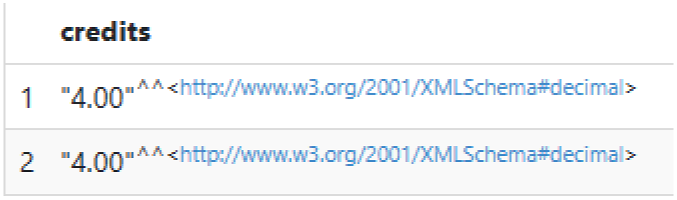


```
file:///C:/Users/nickp/Documents/GitHub/Knowledge_AI/COMP%20474_6741-GCS_143/Lectures/Knowledge%20Base%20Queries%20and%20Linked%20Open%20Data.pdf
https://plato.stanford.edu/entries/artificial-intelligence/
file:///C:/Users/nickp/Documents/GitHub/Knowledge_AI/COMP%20474_6741-GCS_143/Lectures/Intelligent%20Agents.pdf
file:///C:/Users/nickp/Documents/GitHub/Knowledge_AI/COMP%20474_6741-GCS_143/Lectures/Knowledge%20Graphs%20-%20Vocabularies%20and%20Ontologies.pdf
file:///C:/Users/nickp/Documents/GitHub/Knowledge_AI/COMP%20474_6741-GCS_143/Lectures/Intelligent%20Systems%20Introduction.pdf
file:///C:/Users/nickp/Documents/GitHub/Knowledge_AI/COMP%20474_6741-GCS_143/Lectures/Recommender%20Systems.pdf
file:///C:/Users/nickp/Documents/GitHub/Knowledge_AI/COMP%20474_6741-GCS_143/Lectures/Machine%20Learning%20for%20Intelligent%20Systems.pdf
file:///C:/Users/nickp/Documents/GitHub/Knowledge_AI/COMP%20474_6741-GCS_143/Lectures/Knowledge%20Graphs.pdf
```

<u>Query 6: Provides the number of credits that each course number is worth</u>

```
SELECT ?credits
WHERE{
?course rdf:type uni:Course.
?course uni:subject ?cName.
?course uni:credits ?credits.
FILTER(REGEX(STR(?cName), 'Intelligent_Systems', "i")).
}
LIMIT 100
```

Output:

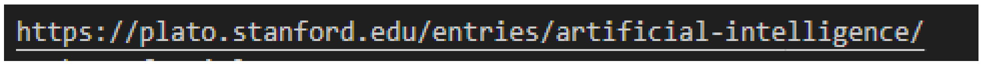


	credits
1	"4.00"^^<http://www.w3.org/2001/XMLSchema#decimal>
2	"4.00"^^<http://www.w3.org/2001/XMLSchema#decimal>

<u>Query 7: Provides additional resources for some course number</u>

```
SELECT ?lectureContent
WHERE{
?course uni:ID ?courseNumber.
?course uni:has_lecture ?lecture.
?lecture uni:has_content ?lectureContent.
?lectureContent rdf:type uni:OtherLectureMaterial.
FILTER(?courseNumber=474).
}
LIMIT 100
```

Output:

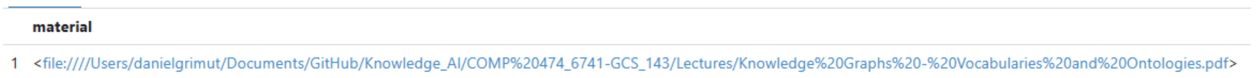


```
https://plato.stanford.edu/entries/artificial-intelligence/
```

<u>Query 8: Provides the content available for a lecture in a course number</u>

```
SELECT ?material
WHERE{
?course rdf:type uni:Course.
?course uni:subject ?cName.
?course uni:has_lecture ?lecture.
?lecture uni:lecture_number ?lecNum.
?lecture uni:has_content ?material.
FILTER(REGEX(STR(?cName), 'Intelligent_Systems', "i")).
FILTER(?lecNum=5).
}
LIMIT 100
```

Output:

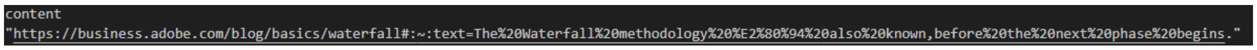


	material
1	<file:////Users/danielgrimut/Documents/GitHub/Knowledge_AI/COMP%20474_6741-GCS_143/Lectures/Knowledge%20Graphs%20-%20Vocabularies%20and%20Ontologies.pdf>

Query 9: Provides recommended reading materials for a topic in a course

```
SELECT ?content
WHERE{
?coures uni:subject ?cName.
?course uni:has_topic ?tName.
?course uni:has_lecture ?lecture.
?lecture uni:has_content ?content.
?content rdf:type uni:OtherLectureMaterial.
FILTER(REGEX(STR(?tName), 'Engineering_Practices', "i")).
FILTER(REGEX(STR(?cName), 'Introduction_to_Software_Engineering', "i")).
}
LIMIT 100
```

Output:

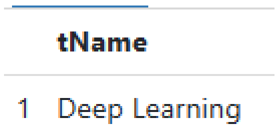


```
content
"https://business.adobe.com/blog/basics/waterfall#:~:text=The%20Waterfall%20methodology%20%E2%80%94%20also%20known,before%20the%20next%20phase%20begins."
```

Query 10: Provides competencies (topics) that a student gains when completing a course number

```
SELECT ?tName
WHERE{
?course rdf:type uni:Course.
?course uni:subject ?cName.
?course uni:has_topic ?topic.
?topic rdf:type uni:Topic.
?topic uni:topicName ?tName.
FILTER(REGEX(STR(?cName), 'Intelligent_Systems', "i")).
}
LIMIT 100
```

Output:

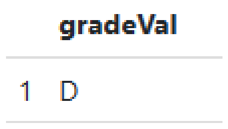


```
  tName
1 Deep Learning
```

Query 11: Provides the grade that a student gained in a course number

```
SELECT ?gradeVal
WHERE{
?student rdf:type uni:Student.
?student uni:student_ID ?ID.
?student uni:grade_obtained ?grade.
?grade rdf:type uni:Grade.
?grade uni:grade_obtained_in ?course.
?course uni:subject ?cName.
?grade uni:grade_value ?gradeVal.
FILTER(?ID=1).
FILTER(REGEX(STR(?cName), 'Intelligent_Systems', "i")).
}
LIMIT 100
```

Output:

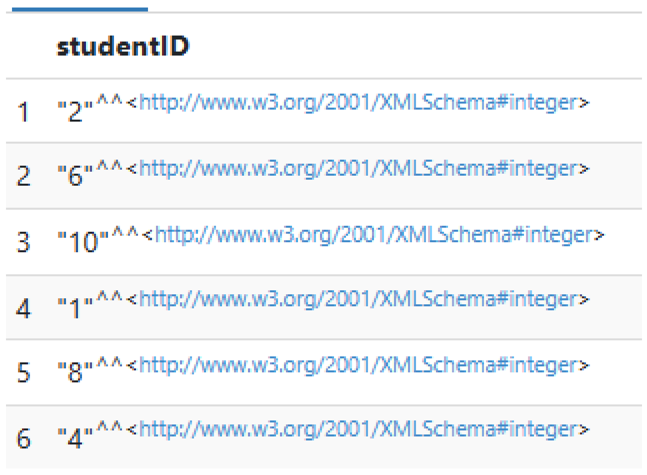


	gradeVal
1	D

Query 12: Provides the students that have completed a course number

```
SELECT ?studentID
WHERE{
?student rdf:type uni:Student.
?student uni:student_ID ?studentID.
?student uni:completed ?course.
?course uni:subject ?cName.
FILTER(REGEX(STR(?cName), 'Intelligent_Systems', "i")).
}
LIMIT 100
```

Output:



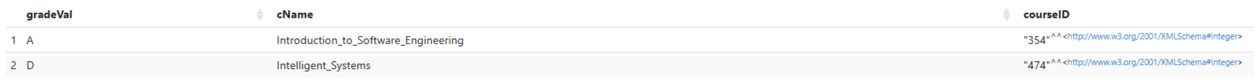
	studentID
1	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
2	"6"^^<http://www.w3.org/2001/XMLSchema#integer>
3	"10"^^<http://www.w3.org/2001/XMLSchema#integer>
4	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
5	"8"^^<http://www.w3.org/2001/XMLSchema#integer>
6	"4"^^<http://www.w3.org/2001/XMLSchema#integer>

<u>Query 13: Provides a student transcript listing all courses taken and their grades</u>

```
SELECT ?gradeVal ?cName ?courseID
WHERE{
?student rdf:type uni:Student.
?student uni:grade_obtained ?grade.
?grade uni:grade_value ?gradeVal.
?grade uni:grade_obtained_in ?course.
?course uni:subject ?cName.
?course uni:ID ?courseID.
?student uni:student_ID ?ID.
FILTER(?ID=1).

}
LIMIT 100
```

Output:



	gradeVal	cName	courseID
1	A	Introduction_to_Software_Engineering	"354"^^<http://www.w3.org/2001/XMLSchema#integer>
2	D	Intelligent_Systems	"474"^^<http://www.w3.org/2001/XMLSchema#integer>

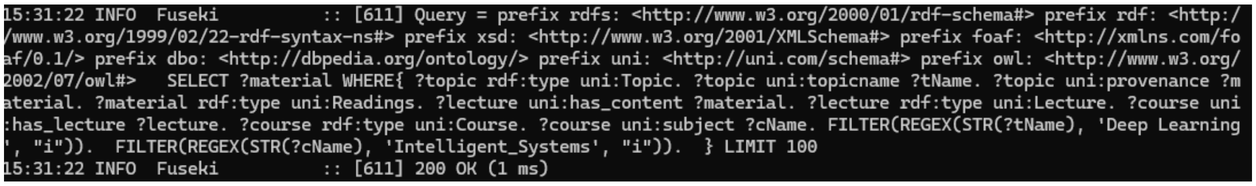
Triplestore and SPARQL Endpoint Setup

The team faced different challenges while trying to set up the SPARQL Endpoint. There were 3 phases to these challenges:

- Getting Fuseki Server to run on each member's PC
- Connecting Fuseki Server with SPARQL to the team's program
- Understanding the JSON data that was being returned and formatting it to CSV

For the first part, understanding the documentation and adding the data that we created with our program was a challenge. There was little documentation for the setup, and we had to make sure that the data we provided to the server was the right data.

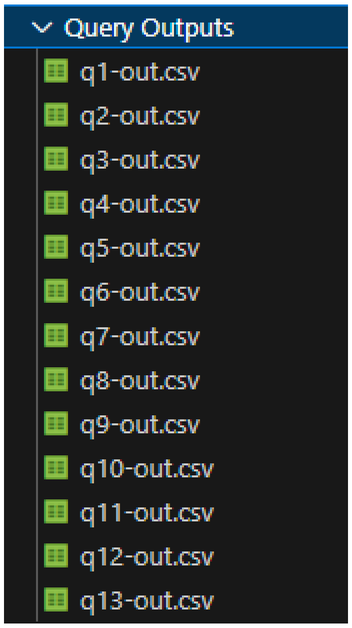
The second and toughest part was connecting the server we just created to a program. Here, the hardest part was the setup, as there was almost no documentation provided with the publicly available libraries. The team agreed to implement the code for the endpoint using Python rather than Java. Although Java would have provided a better interface to connect to any web applications in the future, python provided better information regarding the libraries we used as well as being more forgiving when handling our data. We first had to write all of the queries into different .txt files and be able to read each one of them. We then had to see how to set up the server, connect our program to the server and provide queries to it, from the previously mentioned .txt files. After connecting to the Fuseki Server, we were able to run a query and see what the output was. We confirmed this output by looking at the command prompt where the server was running (getting a 200 OK request back), as well as comparing the response with that of the GUI server in the browser.



```
15:31:22 INFO  Fuseki          :: [611] Query = prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> prefix rdf: <http:/
/www.w3.org/1999/02/22-rdf-syntax-ns#> prefix xsd: <http://www.w3.org/2001/XMLSchema#> prefix foaf: <http://xmlns.com/fo
af/0.1/> prefix dbo: <http://dbpedia.org/ontology/> prefix uni: <http://uni.com/schema#> prefix owl: <http://www.w3.org/
2002/07/owl#>   SELECT ?material WHERE{ ?topic rdf:type uni:Topic. ?topic uni:topicname ?tName. ?topic uni:provenance ?m
aterial. ?material rdf:type uni:Readings. ?lecture uni:has_content ?material. ?lecture rdf:type uni:Lecture. ?course uni
:has_lecture ?lecture. ?course rdf:type uni:Course. ?course uni:subject ?cName. FILTER(REGEX(STR(?tName), 'Deep Learning
', "i")).  FILTER(REGEX(STR(?cName), 'Intelligent_Systems', "i")).  } LIMIT 100
15:31:22 INFO  Fuseki          :: [611] 200 OK (1 ms)
```

Picture 1: Server response to the query search

The last part of the challenge was to understand what the server would return and format that data to a more human-readable format. On the web app, it would return the response in 1 of 2 ways: either in a CSV-type format or in a JSON format. Looking at the Python program, the result being returned was of the JSON format. To format the data, we had to manipulate it into a CSV-type format and export it into each queries respectable CSV files.



```
Query Outputs
    q1-out.csv
    q2-out.csv
    q3-out.csv
    q4-out.csv
    q5-out.csv
    q6-out.csv
    q7-out.csv
    q8-out.csv
    q9-out.csv
    q10-out.csv
    q11-out.csv
    q12-out.csv
    q13-out.csv
```

Picture 2: Output CSV Files

Conclusion

What would be ideal for the next installment of this assignment would be to implement software that would visualize the outputs of the queries rather than simply having them saved locally. This would possibly extend to an interface that accepts additional queries or possibly constructs new ones. Doing so would likely involve using a different programming language as well as a completely different set of libraries, such as Java for the backend and JavaScript for the front end. The most challenging parts of this project were the generation of the knowledge base and vocabulary as well as the SPARQL endpoint.