Assignment

In this assignment you will apply the principles you learned during the theory of Chapters 14 - 15. The lab contains three subsequent parts, which should be performed in chronological order. Importantly, while time will be provided on Tuesday the 16th of May to work on the lab and ask questions, **the solutions themselves should be submitted to Ufora on the 21th of May (23:59 Brussels time)** ***at the latest***, so that you can have the feedback in time for the exam.

We would also like to remind you that plagiarism will not be tolerated. While you are allowed to discuss the lab with other students, copying answers off one another is not permitted and, if detected, will be reflected in the scoring.

Part 1: Modeling symptoms

The assignment for Lab 6 on Semantics will require you to go through a **data integration** pipeline. This pipeline (cfr., Image 1 below) is composed of three logically distinct steps, which are represented by the three parts in the assignment. The purpose of the lab as a whole is to highlight the utility of having a data representation expressed in terms of formal semantics when dealing with heterogeneous data sources. To this end, part 1 will require you to perform parts of what is called **conceptual modeling** or **ontology engineering**, which is the process whereby the concepts governing a certain application domain are formally defined. Part 2 will then ask you to define **mapping rules** that are used to convert data, such as csv or json files, to the ontology model defined in part 1. This will allow you to integrate data from different sources and make it interoperable. Lastly, having acquired an integrated dataset, in part 3 you will look at how queries can be performed on this dataset to gain insights that would not be possible without an integrated view on the data.

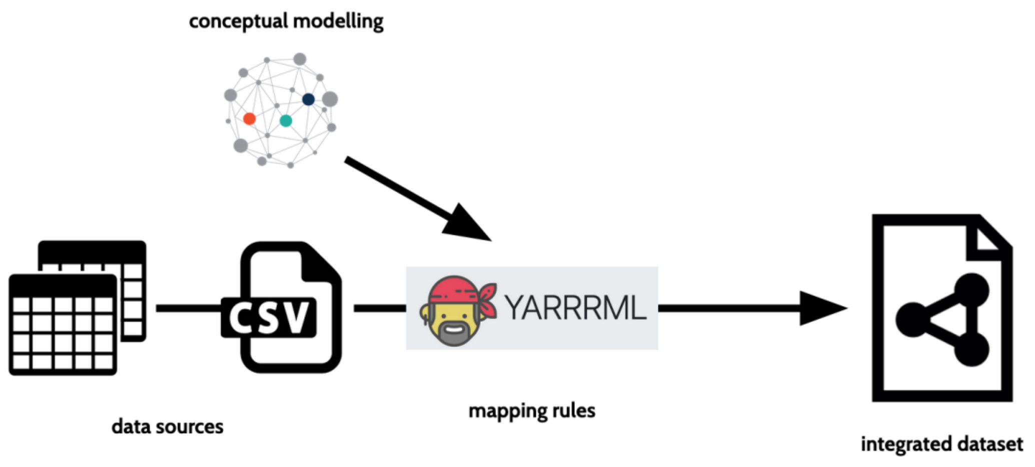


Image 1: data integration pipeline

More specifically, the purpose of part 1 is to make use of the Protégé editor to extend an existing ontology model by adding a class hierarchy of symptoms. Throughout you will learn about the differences between classes and individuals, how to link different concepts to one another via properties, and how to use the reasoner to perform classification and instantiation. You will also encounter some of the pitfalls common to OWL ontology modeling.

To model symptoms related to diseases we will rely on a version of the human disease ontology, which is part of the Open Biological and Biomedical Ontology (OBO) Foundry's family of related ontologies (cfr. http://www.obofoundry.org/). The full human disease ontology can be found at:

https://raw.githubusercontent.com/DiseaseOntology/HumanDiseaseOntology/master/src/ontology/doid.owl.

As its name suggests, this ontology provides an elaborate class structure for human diseases; when considering biological and biomedical application domains, knowing the formal characteristics of different diseases is often highly valuable. Proper ontology development would require us to import the most recent version of the ontology directly from its on-line source. By importing through an IRI (e.g. an URL), one is always sure that the most recent version is adopted and one can check if the updates remain compatible. However, the ontology is quite extensive and would be too convoluted for this lab session. Therefore, we provided an older and much smaller version of the ontology in the lab assignment.

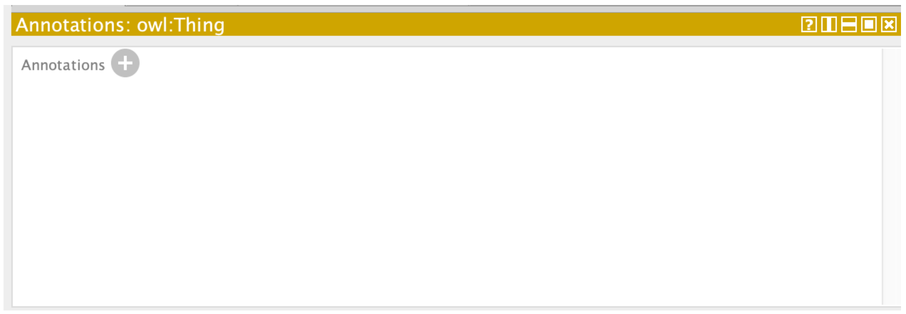
The following steps should be executed chronologically. At a certain point, you will encounter a question enquiring after the rationale behind your results. You should answer these in the space provided beneath each question.

During this part of the assignment, you will be asked to model various ontological concepts. For the connection with part 2 and part 3, it is important that you **adhere to the same class and property names we define in the document**. This also includes case sensitivity. Some versions of Protégé tend to automatically generate class names, **make sure this option is turned off!** When creating a new class you can click the "New entity options…" button and validate that the "User supplied name" is checked for the "Ends with" option (instead of the Auto-generated ID). Another option is to go to Preferences/Settings, search for the "new entities" tab, and for Entity IRI, select End with: "User supplied name".

1. Open Protégé and open the ontology **disease.owl** in Protégé. (Or you can just open the ontology directly with the correct version of Protégé.) For the purposes of this lab session, we will be using **Protégé version 5.2.0**, which can be downloaded at the bottom of the release page here. *Make sure to select the binary archive suited for your operating system.* More detailed instructions for installing Protégé on Windows, Linux, or Mac OS can be found here.

2. First, look for a subclass of **owl:Thing** called **Symptom** in the "Classes" tab (Entities -> Classes). (Double clicking a class will reveal its subclasses.) Then, look for a subclass of

owl:Thing called **disease**. Notice how the class **disease** is actually called http://purl.obolibrary.org/obo/DOID_4, or **obo:DOID_4**, because it is part of the OBO ontology. How does Protégé make sure that the class is actually displayed as "disease" and not as "obo:DOID_4"? Hint: look in the annotations panel of the disease class. The annotations panel for owl:Thing is shown below:



3. Look for the class **PhysicalSymptom**, modeled as a subclass of **Symptom**. Five different kinds of Physical Symptoms have already been defined (cfr., Image 2 for a schematic overview of the class hierarchy): **Hyperventilation**, **Bleeding**, **JointPain**, **Chills**, **WeightLoss.**

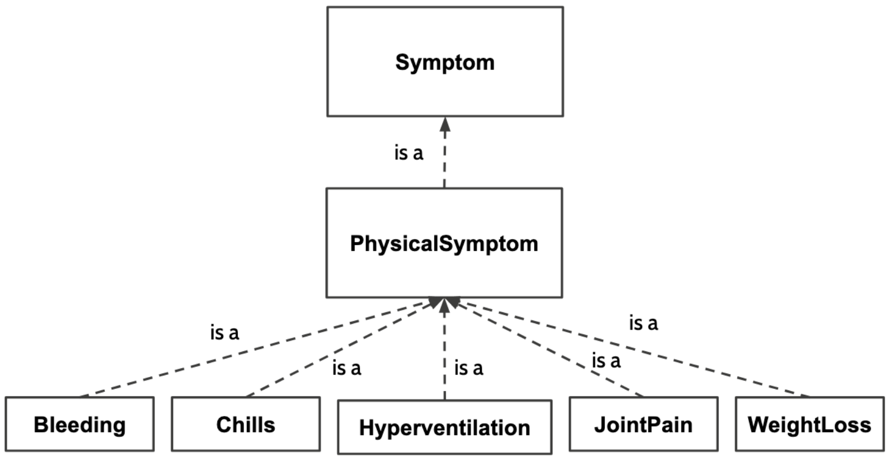
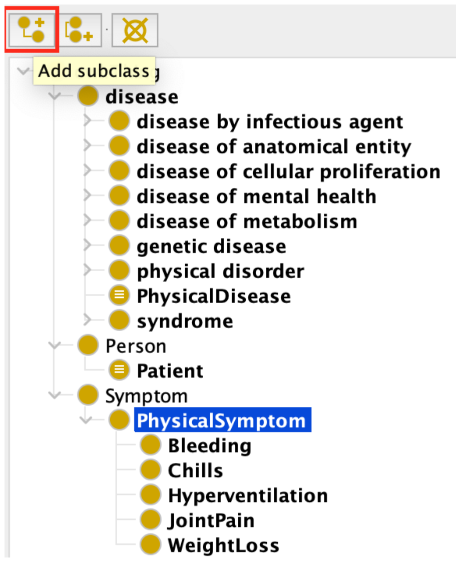


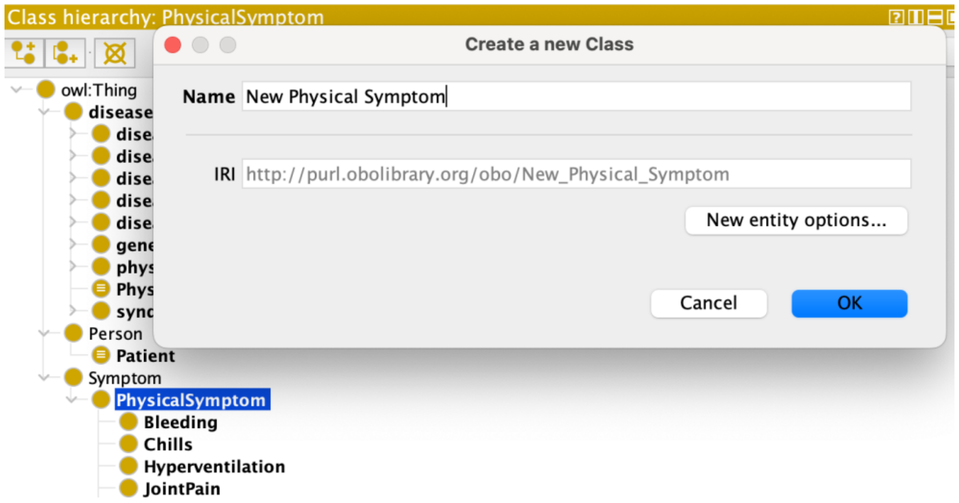
Image 2: The Symptom class hierarchy

Adding a new Symptom goes as follows:

a. Select the category class we want to extend (in our case e.g., PhysicalSymptom) and use the “Add subclass” functionality provided by Protégé.



b. A creation panel will pop up that allows you to fill in the name of the new symptom (here, **as an example**, *New Physical Symptom*) you are creating. The corresponding URI (in the obo namespace) is automatically generated.

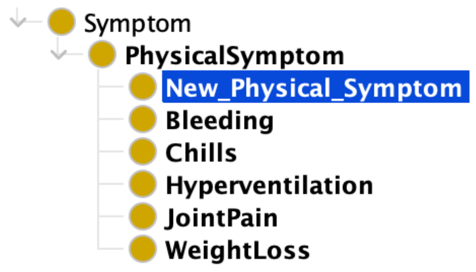


Create a new Class

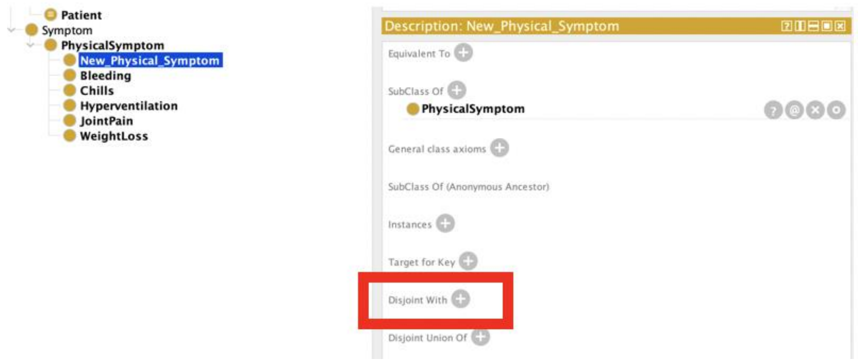
Name: New Physical Symptom

IRI: http://purl.obolibrary.org/obo/New_Physical_Symptom

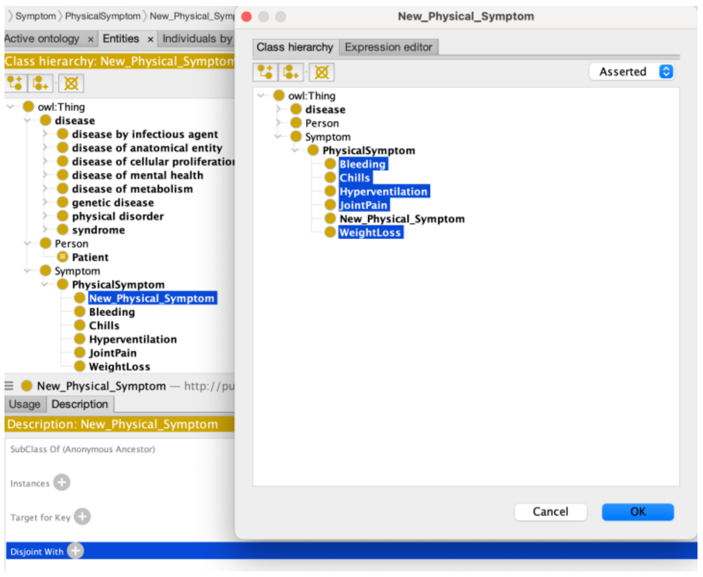
c. The new physical symptom is now part of the existing class hierarchy.



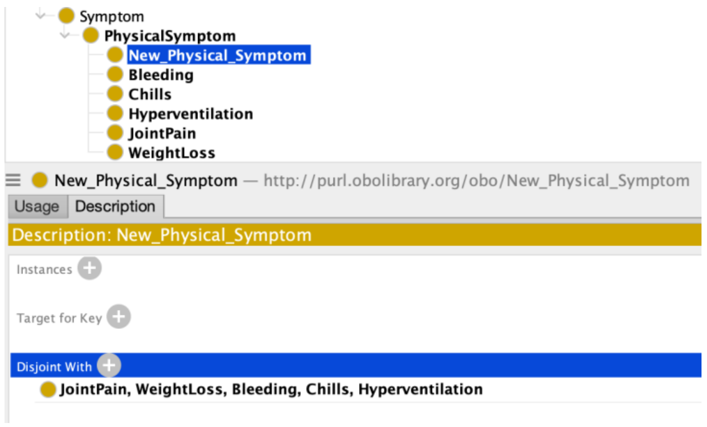
d. To make the example class *New Physical Symptom* disjoint from the other physical symptoms, you can do the following: Go to the 'Disjoint With' field in the "Description" panel and select '+'.



A window will appear where you can select the classes you want the selected class to be disjoint with. In our example, we want *New Physical Symptom* to be disjoint with every other subclass of PhysicalSymptom.



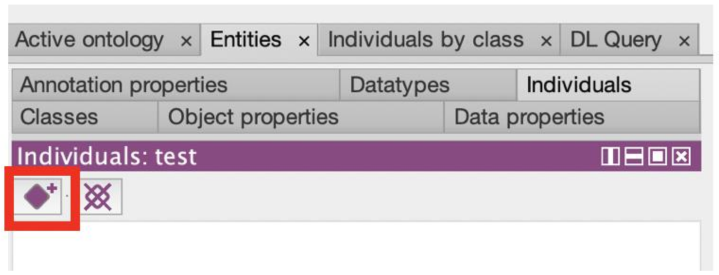
The result is as follows:

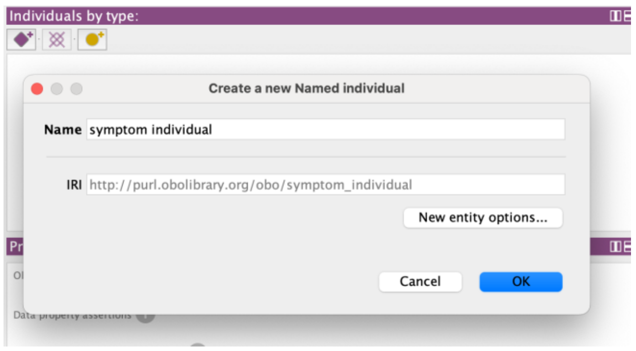

Symptom
PhysicalSymptom
New_Physical_Symptom
Bleeding
Chills
Hyperventilation
JointPain
WeightLoss
New_Physical_Symptom — http://purl.obolibrary.org/obo/New_Physical_Symptom
Usage
Description
Description: New_Physical_Symptom
Instances
Target for Key
Disjoint With
JointPain, WeightLoss, Bleeding, Chills, Hyperventilation

Now model the rest of the symptoms hierarchy. Your final hierarchy should be three levels deep (including the Symptom root class):

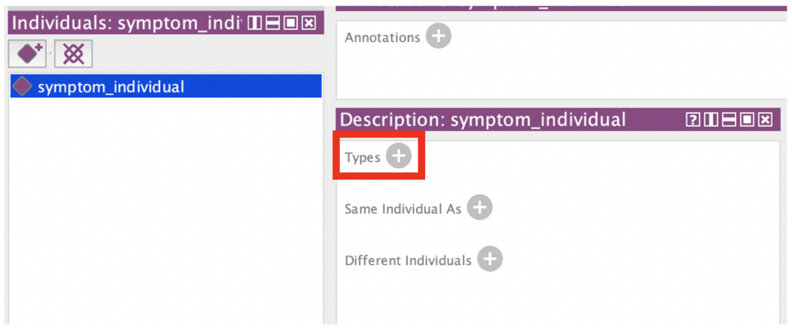
e. Create a **PsychologicalSymptom** class that is a subclass of Symptom. Create two specific kinds of psychological symptoms: **Anxiety**, **Hallucination.**
f. Ensure that all the defined symptoms are disjoint. Make sure to use the smallest number of disjoint definitions to define the disjointness between all symptoms.
g. (Note that *New Physical Symptom* (see above) should **not** be part of your class hierarchy. It was just an example.)

4. If you want to verify that the symptoms have been modeled correctly, you can do the following:
   a. Create a symptom individual with two different types: e.g., Bleeding and Anxiety. This can be done in the "Individuals" tab (Entities -> Individuals).

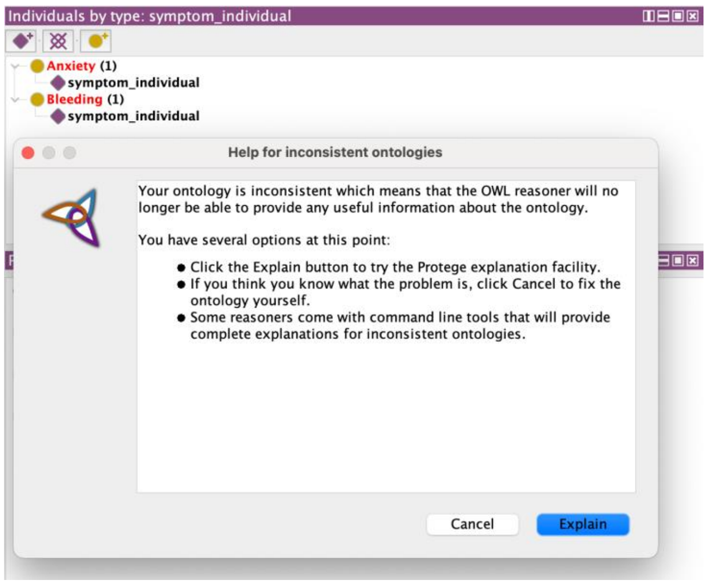



Individuals by type:
Create a new Named individual
Name
symptom individual
IRI
http://purl.obolibrary.org/obo/symptom_individual
New entity options...
Cancel
OK

Created individuals should be assigned to a type. In this case, e.g., Anxiety and Bleeding.



b. Execute the (HermiT) reasoner in Protégé. (To do this, navigate to "Reasoner", select the Hermit reasoner, and run "Start reasoner".) You should get the following message:



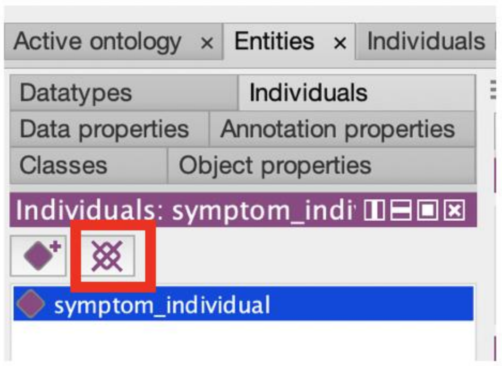
Your ontology is inconsistent which means that the OWL reasoner will no longer be able to provide any useful information about the ontology.

You have several options at this point:

- Click the Explain button to try the Protege explanation facility.
- If you think you know what the problem is, click Cancel to fix the ontology yourself.
- Some reasoners come with command line tools that will provide complete explanations for inconsistent ontologies.

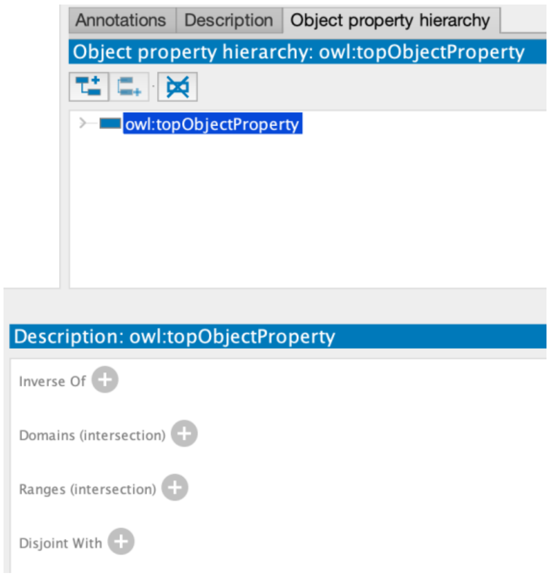
c. By selecting the "Explain" option, we get an explanation for the inconsistency that was created, and this explanation should indicate to us that it is not possible to create an individual with two types that are mutually exclusive.

d. Remove the individual and stop the reasoner. (To do this, navigate to "Reasoner" and select "Stop reasoner".)



5. Now you will associate symptoms with various diseases.

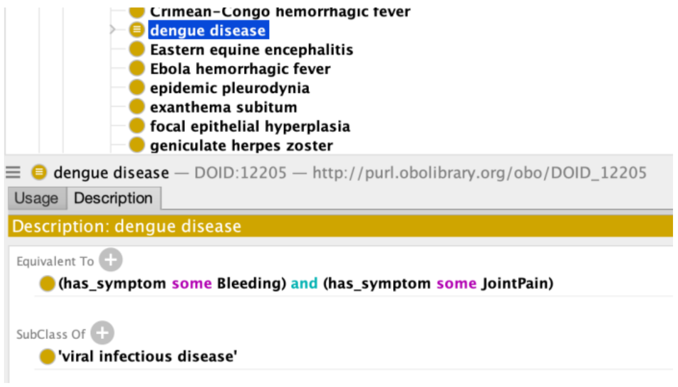
   a. Look for the **has_symptom** property in the “Object properties” tab. Make sure that this property relates diseases (domain) to symptoms (range). Look for Protégé’s “Object property hierarchy” in the Entities->Object properties tab. Then, in the description pane of the selected property, you can specify its domain and range.



   b. Create an inverse relation **is_symptom_of** and make sure to specify that it is inversely related to **has_symptom**. To do this, you can use ‘Inverse Of’ in the property description panel and press the ‘+’ to get a selection window. New object properties can be created in a similar fashion as classes (see 3.a).

   c. Look for the definitions of the following classes and investigate their descriptions in the “Classes” tab. (Use Protégé’s search functionality, to be found in Edit/Find…) These classes are defined in terms of the **existence** of certain symptoms, which are defined in the ‘Equivalent To’ panel. Each of these diseases is a disease with **at least** the described symptoms.

i. Dengue disease: has symptoms *Bleeding **and** JointPain*



Crimean–Congo hemorrhagic fever
dengue disease
Eastern equine encephalitis
Ebola hemorrhagic fever
epidemic pleurodynia
exanthema subitum
focal epithelial hyperplasia
geniculate herpes zoster

dengue disease — DOID:12205 — http://purl.obolibrary.org/obo/DOID_12205

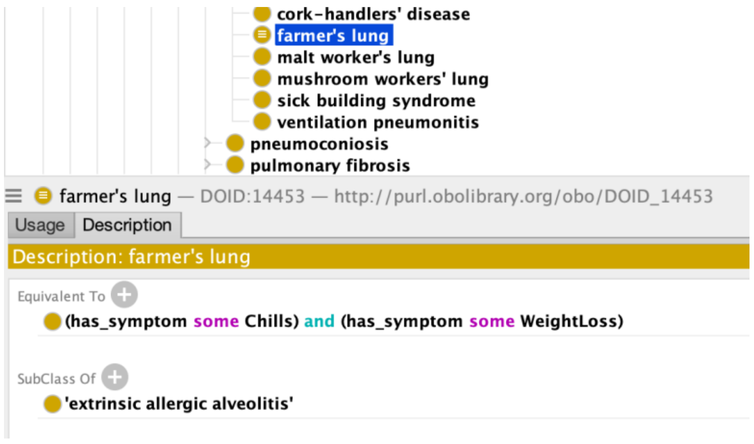
Usage | Description

Description: dengue disease

Equivalent To
(has_symptom some Bleeding) and (has_symptom some JointPain)

SubClass Of
'viral infectious disease'

ii. Farmer’s lung: has symptoms *Chills **and** WeightLoss*



cork–handlers' disease
farmer's lung
malt worker's lung
mushroom workers' lung
sick building syndrome
ventilation pneumonitis
pneumoconiosis
pulmonary fibrosis

farmer's lung — DOID:14453 — http://purl.obolibrary.org/obo/DOID_14453

Usage | Description

Description: farmer's lung

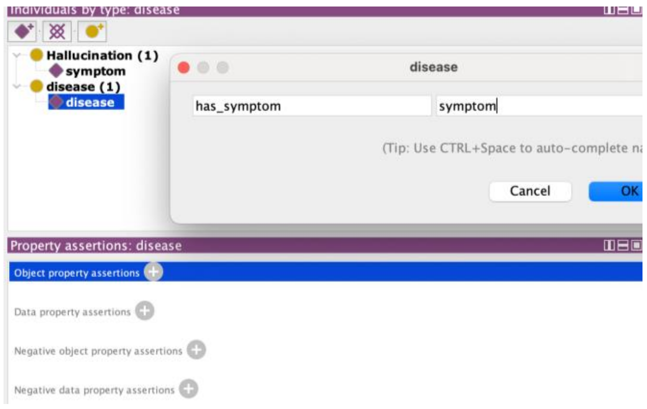
Equivalent To
(has_symptom some Chills) and (has_symptom some WeightLoss)

SubClass Of
'extrinsic allergic alveolitis'

Add definitions to the following classes, **such that** they can be defined through the existence of certain symptoms. This means that each of the following diseases **should be** a disease with **at least** the described symptoms.

i. Intrinsic asthma: has symptoms *Anxiety **and** Hyperventilation*
ii. Delusional disorder: has symptom *Hallucination*

6. Add an individual of the type **disease** and an individual of the type **Hallucination,** and connect them through a **has_symptom** relation. (The “has_symptom” relation can be added in the “property assertions panel” by clicking on the plus-sign (‘+’) next to Object property assertions. Refer to the screenshot for more details.)


Individuals by type: disease
Hallucination (1)
symptom
disease (1)
disease
disease
Cancel
OK
Property assertions: disease
Object property assertions
Data property assertions
Negative object property assertions
Negative data property assertions

has_symptom symptom

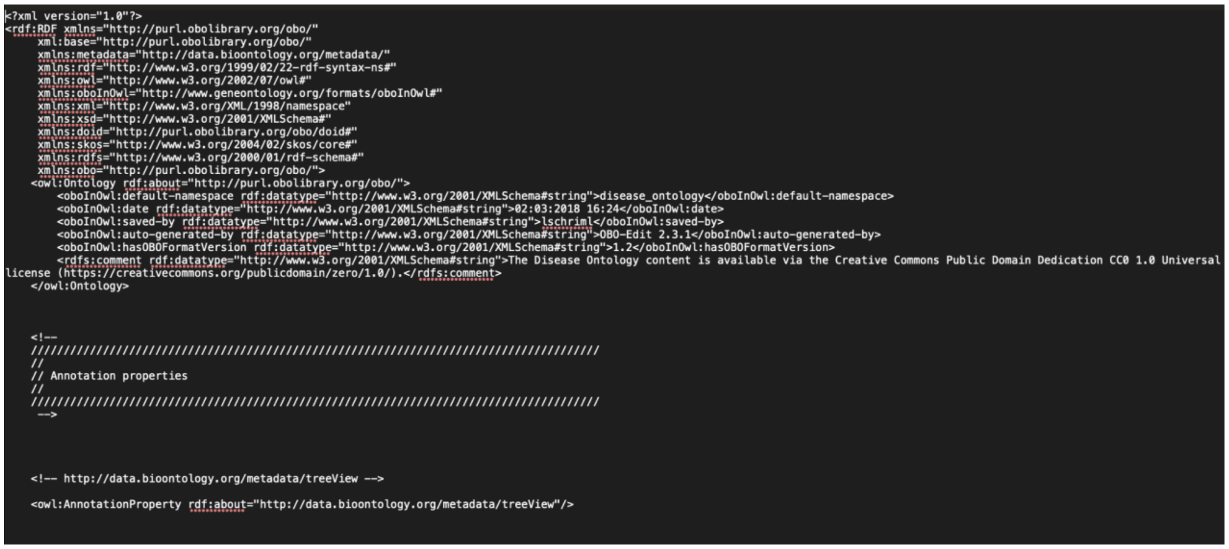
(Tip: Use CTRL+Space to auto-complete na

   a. Execute the reasoner.
   b. Question: What happens and why does this happen?

   c. Remove the created individuals and stop the reasoner. (See step 4.d)

7. Finally, in preparation of part 2, notice that the following classes and properties have also been defined:
   a. The class **Person**, modeled as a subclass of **owl:Thing**.
   b. The object property **has_disease** relating people to diseases.
   c. The class **Patient,** modeled as a **Person** that has at least one **disease.**
   d. The data properties **citizenOf**, **firstName**, and **lastName** relating people to strings.
   e. The data property **hasAge**, relating people to integers.

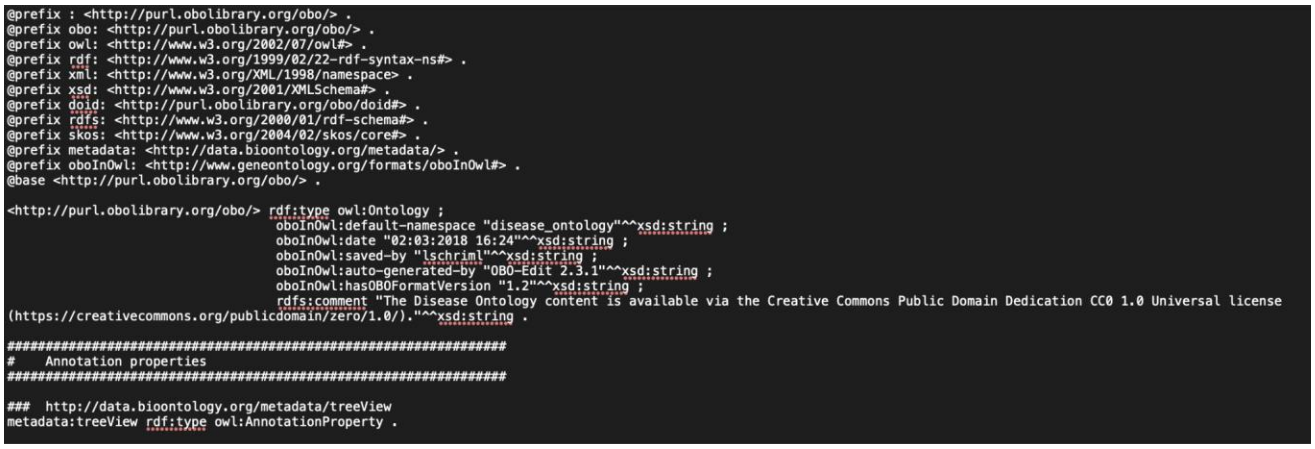
8. **You will need to hand in the answers to the questions as well as the extended ontology file.** Make sure to save the ontology file from time to time so that your changes are recorded. Note that the ontology file you will be handing in has a concrete syntax; by default, usually RDF/XML, which is commonly used to express RDF ontologies. Other syntaxes include Turtle and JSON-LD, both of which are supported by Protégé. Indeed, Protégé enables format conversions between these different syntaxes through its 'Save as…' functionality. You can recognize RDF/XML in particular by its XML-style tags:



```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://purl.obolibrary.org/obo/"
     xml:base="http://purl.obolibrary.org/obo/"
     xmlns:metadata="http://data.bioontology.org/metadata/"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:oboInOwl="http://www.geneontology.org/formats/oboInOwl#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:doid="http://purl.obolibrary.org/obo/doid#"
     xmlns:skos="http://www.w3.org/2004/02/skos/core#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:obo="http://purl.obolibrary.org/obo/">
    <owl:Ontology rdf:about="http://purl.obolibrary.org/obo/">
        <oboInOwl:default-namespace rdf:datatype="http://www.w3.org/2001/XMLSchema#string">disease_ontology</oboInOwl:default-namespace>
        <oboInOwl:date rdf:datatype="http://www.w3.org/2001/XMLSchema#string">02:03:2018 16:24</oboInOwl:date>
        <oboInOwl:saved-by rdf:datatype="http://www.w3.org/2001/XMLSchema#string">lschriml</oboInOwl:saved-by>
        <oboInOwl:auto-generated-by rdf:datatype="http://www.w3.org/2001/XMLSchema#string">OBO-Edit 2.3.1</oboInOwl:auto-generated-by>
        <oboInOwl:hasOBOFormatVersion rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1.2</oboInOwl:hasOBOFormatVersion>
        <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">The Disease Ontology content is available via the Creative Commons Public Domain Dedication CC0 1.0 Universal
license (https://creativecommons.org/publicdomain/zero/1.0/).</rdfs:comment>
    </owl:Ontology>



    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Annotation properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->



    <!-- http://data.bioontology.org/metadata/treeView -->

    <owl:AnnotationProperty rdf:about="http://data.bioontology.org/metadata/treeView"/>
```

Image 3: RDF/XML syntax snippet

The same ontology rendered according to the Turtle syntax, would look as follows:



```
@prefix : <http://purl.obolibrary.org/obo/> .
@prefix obo: <http://purl.obolibrary.org/obo/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix doid: <http://purl.obolibrary.org/obo/doid#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix metadata: <http://data.bioontology.org/metadata/> .
@prefix oboInOwl: <http://www.geneontology.org/formats/oboInOwl#> .
@base <http://purl.obolibrary.org/obo/> .

<http://purl.obolibrary.org/obo/> rdf:type owl:Ontology ;
                                   oboInOwl:default-namespace "disease_ontology"^^xsd:string ;
                                   oboInOwl:date "02:03:2018 16:24"^^xsd:string ;
                                   oboInOwl:saved-by "lschriml"^^xsd:string ;
                                   oboInOwl:auto-generated-by "OBO-Edit 2.3.1"^^xsd:string ;
                                   oboInOwl:hasOBOFormatVersion "1.2"^^xsd:string ;
                                   rdfs:comment "The Disease Ontology content is available via the Creative Commons Public Domain Dedication CC0 1.0 Universal license
(https://creativecommons.org/publicdomain/zero/1.0/)."^^xsd:string .

#################################################################
#    Annotation properties
#################################################################

###  http://data.bioontology.org/metadata/treeView
metadata:treeView rdf:type owl:AnnotationProperty .
```

Image 4: Turtle syntax snippet

Part 2: Mapping patients

The purpose of part 2 is to learn how to convert raw data (such as csv or json files) to semantic data, such that it can be integrated. Through the use of a mapping file, raw data can be automatically annotated with the semantics expressed through the ontology designed in Part 1.

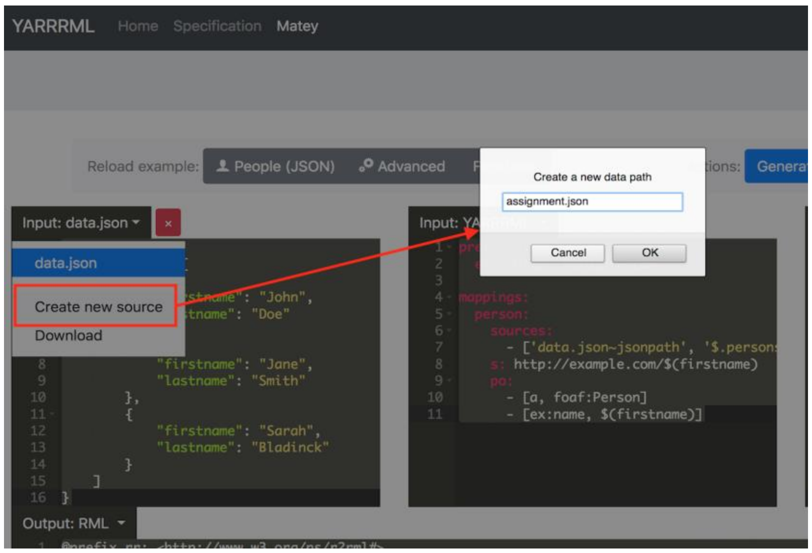
We will map patient information contained in csv and json files to concepts modeled in part 1, so that this information can be accessed in a more unified way. The information in question can be found in the files **patients1.csv, patients2.csv** and **symptoms.json**, which were included in the lab assignment. Make sure to open these files with a text editor when you copy the content (see guidelines below) and refrain from using excel to open the files.

We will use YARRRML to map the patient information to RDF triples. You are encouraged to use the online Matey editor to do this. First, however, we need to go over some of the basics in relational mapping.

For context, there is something called the RDF mapping language (RML), which allows users to define how they can generate RDF-based Knowledge Graphs from external sources in different formats (e.g. CSV or JSON). As such, it is used to facilitate what is called **data integration,** which involves combining data from multiple, potentially heterogeneous sources into a single, unified view**.** The specific language we will be using throughout this exercise is called YARRRML and it relies on a YAML-based syntax to provide a more human-readable approach to the construction of RML mappings. (Please refer to the YARRRML tutorial for extended reference material.)

A few guidelines to help you start the assignment:

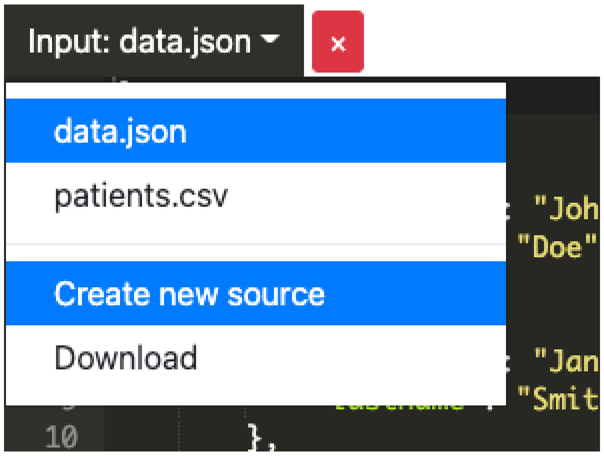
1. Create a new source in the input-pane for each data source (i.e., two csv files and one json file). Make sure to adhere to the original names of the files. See ‘Remarks’ further down below for more info.



```
stname": "John",
tname": "Doe"

        "firstname": "Jane",
        "lastname": "Smith"
    },
    {
        "firstname": "Sarah",
        "lastname": "Bladinck"
    }
  ]
}
```

```
mappings:
  person:
    sources:
      - ['data.json~jsonpath', '$.person
    s: http://example.com/$(firstname)
    po:
      - [a, foaf:Person]
      - [ex:name, $(firstname)]
```

2. To check whether a new source has been created properly, make sure to select the name of that source in the leftmost Input dropdown menu. Selecting the source also allows you to populate it with data (by copy-pasting the right contents into the box).



3. Each row from **patients1.csv** and **patients2.csv** will have to be mapped to:
    a. An individual person of type **Person** with a first and last name, an age, and a citizenship (you can use the created data properties for this).
    b. A disease that is linked to the person.
    c. Two symptom IDs that are linked to the created disease. Note that the symptoms themselves are defined in the file **symptoms.json**. Each symptom ID needs to be mapped to its corresponding symptom type.

Please note that patients2.csv and patients2.csv have been deliberately given different file structures to emphasize the heterogeneity (in terms of format, syntax, etc.) often involved in data integration. The input heterogeneity should not prohibit you from mapping these files to identically structured outputs.

4. As a guideline, the final result should resemble the following:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix sd: <http://www.w3.org/ns/sparql-service-description#> .
@prefix obo: <http://purl.obolibrary.org/obo/> .

obo:symptom_0 rdf:type obo:Bleeding .
obo:symptom_1 rdf:type obo:JointPain .
obo:symptom_2 rdf:type obo:Chills .
obo:symptom_3 rdf:type obo:WeightLoss .
obo:symptom_4 rdf:type obo:Anxiety .
obo:symptom_5 rdf:type obo:Hyperventilation .

obo:person_0 rdf:type obo:Person ;
   obo:firstName "John" ;
   obo:lastName "Doe" ;
   obo:hasAge "25"^^xsd:integer ;
   obo:citizenOf "UK" ;
   obo:has_disease obo:disease_0 .

obo:person_1 rdf:type obo:Person ;
   obo:firstName "Jane" ;
   obo:lastName "Smith" ;
   obo:hasAge "32"^^xsd:integer ;
   obo:citizenOf "US" ;
   obo:has_disease obo:disease_1 .

obo:person_2 rdf:type obo:Person ;
   obo:firstName "Sarah" ;
   obo:lastName "Bladinck" ;
   obo:hasAge "47"^^xsd:integer ;
   obo:citizenOf "UK" ;
   obo:has_disease obo:disease_2 .

obo:disease_0 rdf:type obo:DOID_4 ;
   obo:has_symptom obo:symptom_0, obo:symptom_1 .
obo:disease_1 rdf:type obo:DOID_4 ;
   obo:has_symptom obo:symptom_2, obo:symptom_3 .

obo:disease_2 rdf:type obo:DOID_4 ;
   obo:has_symptom obo:symptom_4, obo:symptom_5 .
```

You can start from the mapping file presented below, where the following has already been defined for you:

- The loading of the correct sources.
- The ‘s:’ (subject) definition of the symptoms, people and disease individuals, based on the IDs defined in the csv and json files. Note that from patients*.csv, we model patients and diseases separately. Also note that two different rules (people1 and people2) have been defined to enable patient mapping, because (as stated earlier) patients1.csv and patients2.csv have different formats and so, require different mappings.
- For people2, the ‘po:’ (predicate-object) definition has been partially provided.
- The linking of the diseases to their symptoms.

Please use the provided mapping file ‘mapping.yaml’ and do *not* copy the text box below into the Matey editor, as this may cause formatting problems.

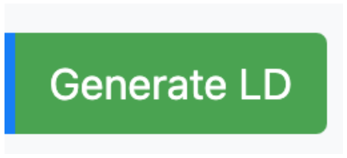
```
prefixes:
  obo: http://purl.obolibrary.org/obo/

mappings:
  symptoms:
          sources:
          - ['symptoms.json~jsonpath', "$.symptoms[*]"]
          s: obo:symptom_$(id)

  people1:
          sources:
          - ['patients1.csv~csv']
          s: obo:person_$(id)

  people2:
          sources:
          - ['patients2.csv~csv']
          s: obo:person_$(id)
          po:
          - [obo:hasAge, $(age), xsd:integer]
          - [obo:citizenOf, $(country)]

  disease:
          sources:
          - ['patients1.csv~csv']
          - ['patients2.csv~csv']
          s: obo:disease_$(id)
          po:
          - [obo:has_symptom, obo:symptom_$(symptomID1)~iri]
          - [obo:has_symptom, obo:symptom_$(symptomID2)~iri]
```

5. Concretely, **complete** the above mapping as follows:
    1. Add the type of each symptom, as described in the symptoms.json file.
    2. Assign the obo:Person type to each patient defined in the patients*.csv files.
    3. Add the first and last name to each patient.
    4. Link each patient to their respective age (make sure it has the correct datatype).
    5. Link each patient to their respective country.
    6. Link each patient to their diseases, using the “has_disease” property.
    7. Add the generic disease type (obo:DOID_4) to each disease. Note that for the sake of simplicity, the patient and disease share the same ID.

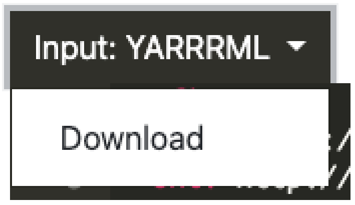
6. **Answer** the following question: In this exercise you have mapped various input sources using a single domain ontology. This is fine for the purposes of the lab, but what would be a better, scalable strategy?

Remarks:

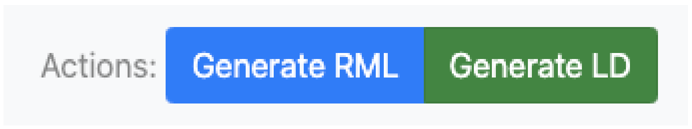
- You can convert to Linked Data by clicking the “Generate LD” button on the right:



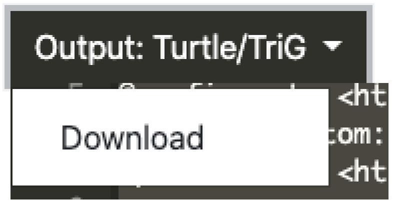
- You can use the ‘~iri’ suffix to convert a variable to an IRI, e.g. test:$(id)~iri will create the iri <http://test.owl/0>. Without this iri suffix, the string “http://test.owl/0” will be generated instead. Note that ‘test’ in test:$(id) is an abbreviation of the prefix http://test.owl/.
- For this part, you will need to **hand in the mapping file**, which you can download by clicking YARRRML input button:



- You will need to download the resulting mapped triples, as it is needed for the next part:
    - Click the generate LD button:



    - Click 'download' in the output pane:



    - Rename the file to triples.ttl and put it in the same folder as the notebook of part3, together with the ontology (disease.owl) created in part 1.

Part 3: Querying

In part 3, you will make use of the results from the previous two parts to gain insights into the afflictions plaguing various patients. To get you started we have prepared a notebook (also included in the lab assignment) in which the triples created by the RML mapping from the previous part will be loaded into the ontology created in part 1. In the following questions you will be asked to analyze certain queries that retrieve particular patient information.
A reference to the SPARQL query language can be found here.

1. Start from the provided notebook. To start the notebook, please follow the instructions provided during the preparations.

2. In the notebook, load your integrated dataset (triples file) generated as a result of the mapping step.

3. The notebook provides three queries, it is up to you to decide what data each query retrieves. You can execute the queries in the notebook to give you some hints. Select one answer for each query, by putting the correct answer in **bold text**.

3.1. Query 1:

```
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX schema: <http://schema.org/>

SELECT ?firstName ?lastName
WHERE {
   ?p a obo:Person.
   ?p schema:givenName ?firstName.
   ?p schema:lastName ?lastName.
   ?p obo:has_disease [obo:has_symptom [a obo:WeightLoss]]
}
```

What data does Query 1 retrieve?

a) Retrieves the patient's first and last name who do not have weight loss as one of their disease symptoms.

b) Retrieves the patient's first and last name who have weight loss as one of their disease symptoms.

c) Retrieves the patient IDs who have weight loss as one of their disease symptoms.

d) Retrieves the patient IDs who have a disease with weight loss as the only disease symptom.

3.2. Query 2:

```
PREFIX obo: <http://purl.obolibrary.org/obo/>
SELECT (Count(?p) as ?num_patients) ?country
WHERE {
        ?p a obo:Person; obo:citizenOf ?country; obo:has_disease ?disease.
}
GROUP BY ?country
ORDER BY DESC(?num_patients)
```

What data does Query 2 retrieve?

a) Retrieves for each country, the number of patients that have at least one disease, ordered according to the number of patients in a descending fashion.

b) Counts the patients that have a country definition, ordered in function of the country.
c) Retrieves for each disease, the number of patients that have that disease.

d) Counts the number of diseases each person has and orders them by country.

3.3. Query 3

```
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?p ?age
 WHERE {
        ?p a obo:Person;
           obo:has_disease ?d;
           obo:hasAge ?age

        FILTER(xsd:integer(?age) > "30"^^xsd:integer)
 }
```

What data does Query 3 retrieve?

a) Retrieves the age of each patient that has a disease.

b) Counts the patients that are older than 30 and have at least one disease.

c) Retrieves the patients that are older than 30 and have at least one disease.

d) Counts the number of diseases each person has.

4. Question: What is the advantage of using SPARQL over SQL within this exercise?

Remarks:

- You can interact with the notebook and play around with the queries to give you some hints as to what data each query retrieves.