

Building ontologies and working with PROTÉGÉ

Aim: To build, edit and run inferences on ontologies using Protégé.

Theory -

- **Ontology** - An ontology is a set of concepts and categories in a subject area or domain that shows their properties and the relations between them.
- **PROTÉGÉ**- Protege is a free, open-source ontology editor and framework for building intelligent systems.

Process:

Step 1 : Defining our ontology

In the realm of knowledge representation, an ontology is a structured and formal representation of a domain's concepts, categories, properties, and the relationships that exist between them. It serves as a means to capture and encode knowledge in a way that can be understood by both humans and machines. In this project, we embark on the task of constructing an ontology to represent the intricate world of **Programming Languages**.

Domain Description

The domain we have chosen for our ontology is "**Programming Languages**." This domain encompasses a vast and dynamic landscape, including a multitude of programming languages, each with its unique characteristics, paradigms, and usage. Programming languages are the foundational tools of computer science, enabling the development of software applications, system architectures, and much more.

Components of our Programming Languages Ontology

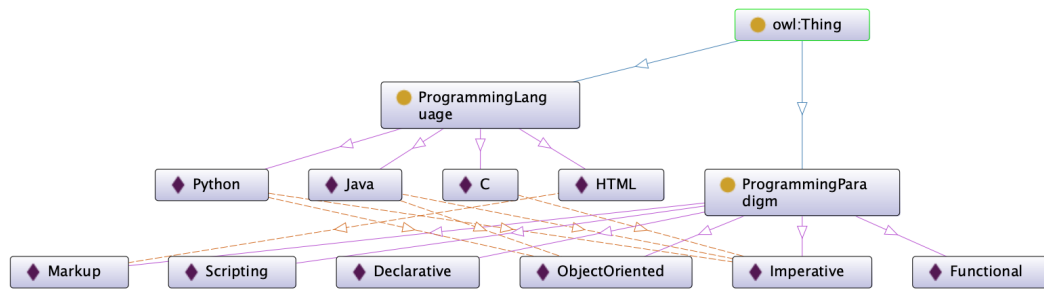
Our ontology for programming languages consists of the following essential components:

1. **Concepts and Classes:** In our ontology, we define the central concept of a "Programming Language" as a class. Each specific programming language (e.g., Python, Java, C++) is represented as an individual member of this class. We have also introduced the concept of "Programming Paradigms" as another class to describe the various programming approaches, such as object-oriented, imperative, declarative, functional, and more.

- 2. Properties:** We have incorporated a range of data properties and object properties to describe the characteristics of programming languages. These properties include "isObjectOriented," "isDeclarative," "isImperative," "isFunctional," "isScripting," "isCompiled," "isInterpreted," "isStaticTyping," "isDynamicTyping," and many more. These properties help define the nature and attributes of each programming language.
- 3. Relationships:** The "hasParadigm" object property establishes relationships between programming languages and the programming paradigms they support. For instance, a programming language like Python is associated with the "Object-Oriented" and "Imperative" paradigms. This relationship helps in categorizing and understanding the languages.
- 4. Documentation and Metadata:** We have included properties like "hasOfficialDocumentation," "hasReleaseDate," "hasLatestVersion," and "hasCommunityForums" to provide additional metadata and documentation-related information for each programming language. This data helps users access crucial resources and stay up-to-date with language developments.
- 5. Usage and Application:** To shed light on the practical applications of programming languages, we've included properties such as "isWebDevelopment" and "isMobileDevelopment." These properties describe the domains or industries where a language is predominantly employed.
- 6. Syntax and Structure:** The "hasSyntax" property offers insights into the syntax or grammar of each language, enabling users to understand the language's fundamental structure.
- 7. Performance and Efficiency:** Properties like "isHighPerformance" and "hasMemoryManagement" provide details regarding performance characteristics and memory management approaches used by the language.
- 8. Ecosystem and Community:** To gauge the language's popularity and the resources available, our ontology includes properties such as "isPopular" and "hasCommunityForums."
- 9. License and Legal Aspects:** The "hasLicense" property specifies the software license under which a programming language is distributed, addressing legal and compliance considerations.

With these components, our ontology for programming languages aims to comprehensively represent the diverse and multifaceted world of programming, facilitating knowledge management and aiding in various computational tasks, including reasoning and inference.

Here's how our ontology will look like -



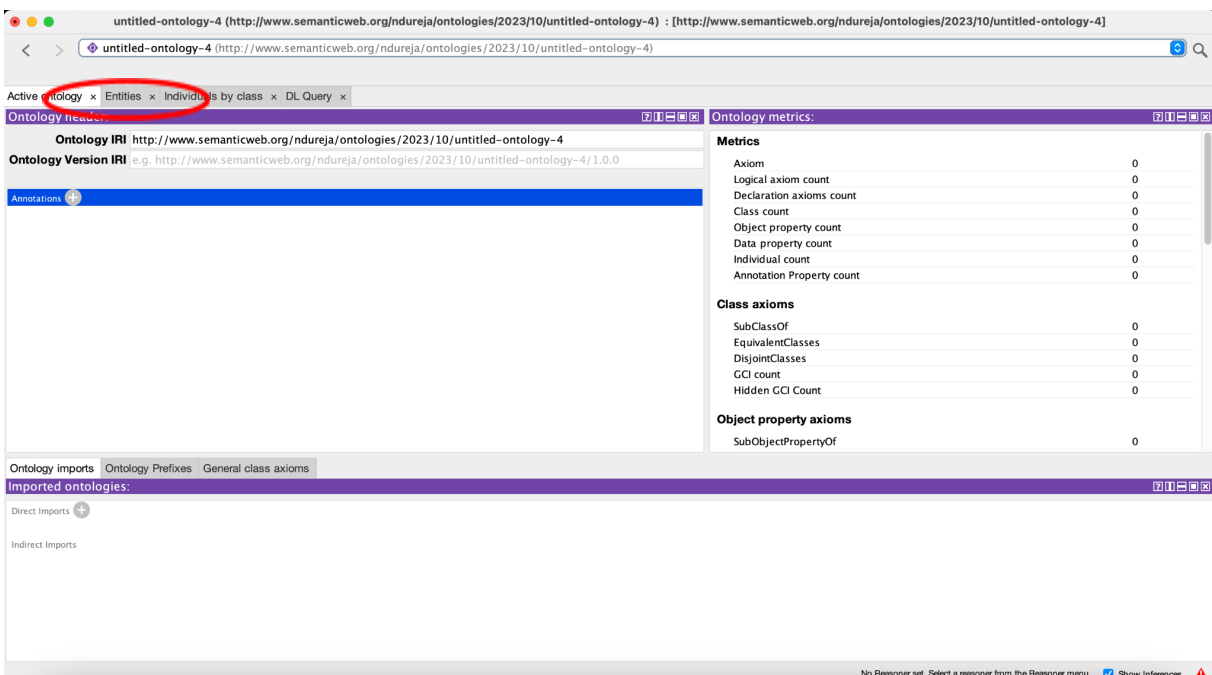
Step 2 : Installing and setting up Protégé

1. Protégé is an open-source tool and can be downloaded from their official website: <https://protege.stanford.edu/software.php>. Protégé also requires a Java Runtime Environment to be installed on your system. If you don't have a JRE installed, then you can install it using the Protégé installer itself.
2. After downloading, launch the installer and follow the steps on the screen. Let all the settings be in their default mode for the sake of this tutorial.
3. After successful installation, launch Protégé.

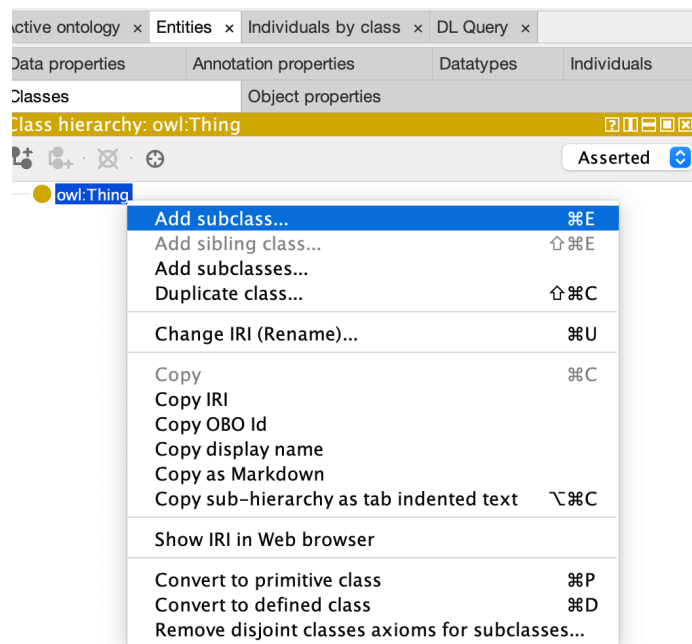
Step 3 : Building the ontology

To start building the ontology, first launch Protégé, then follow these steps -

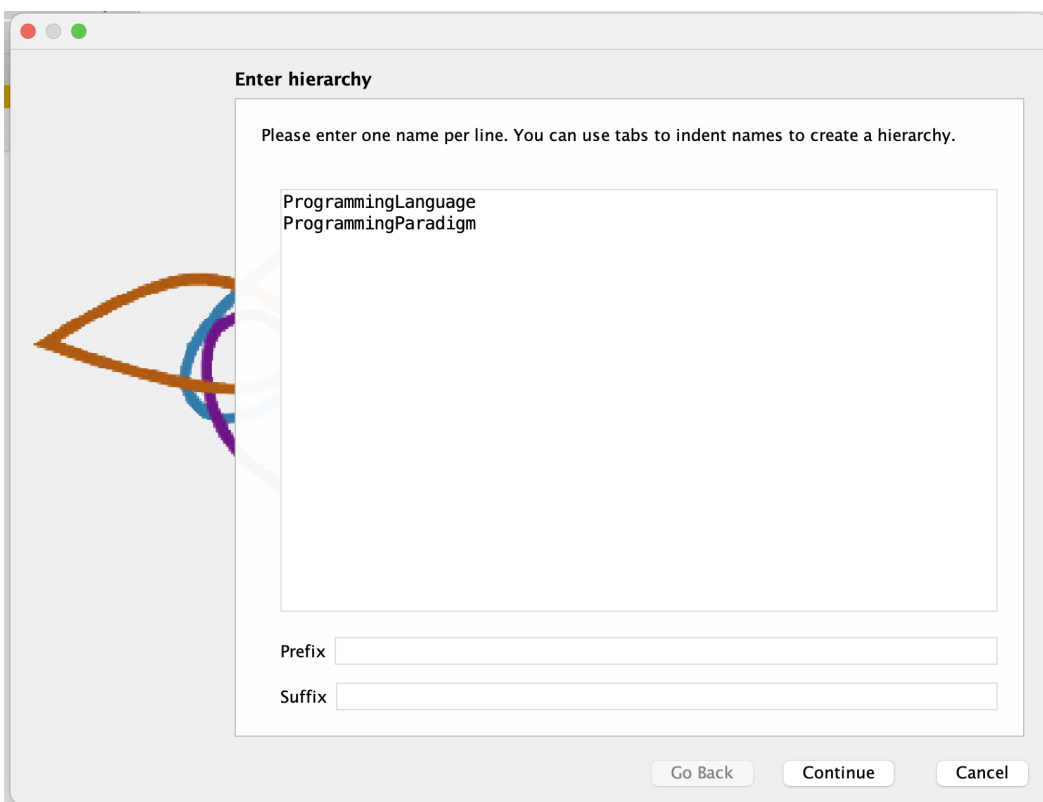
1. This is what your startup screen will look like. Click on the “**Entities**” tab. This is where we will define our classes and properties.



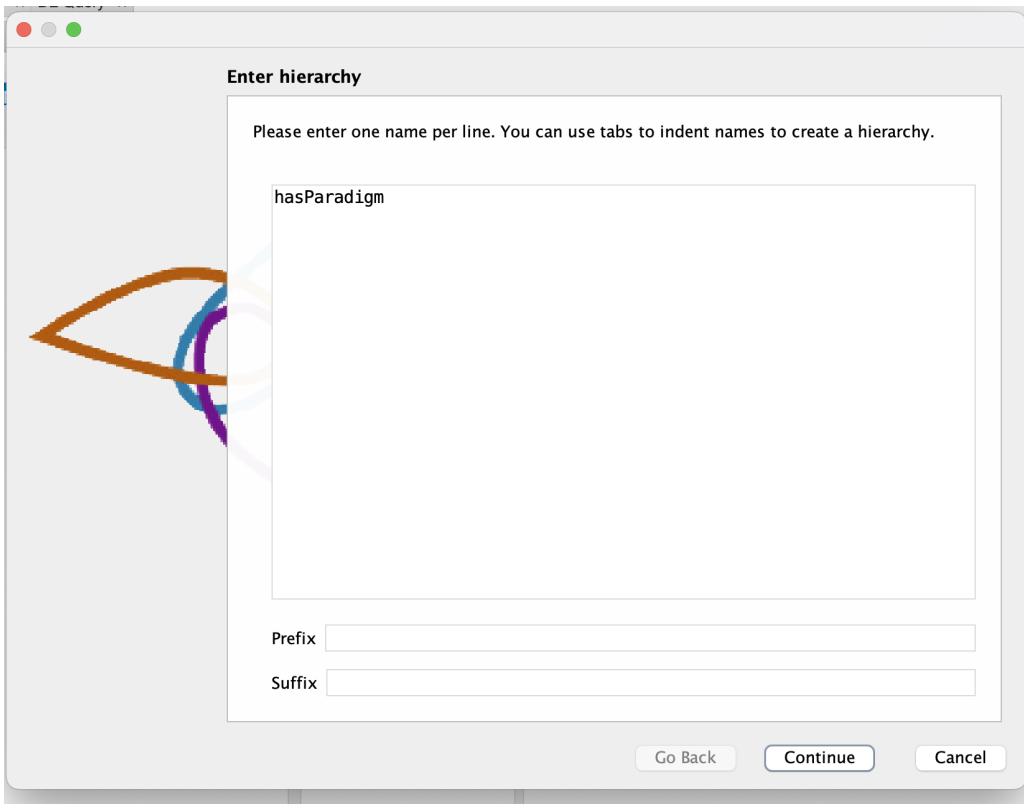
2. Click on the sub-tab “**Classes**”. To add a class, we right-click on “**owl:thing**” which is always our parent class for all classes that we add in Protégé. Then click on the “**Add subclasses...**” option.



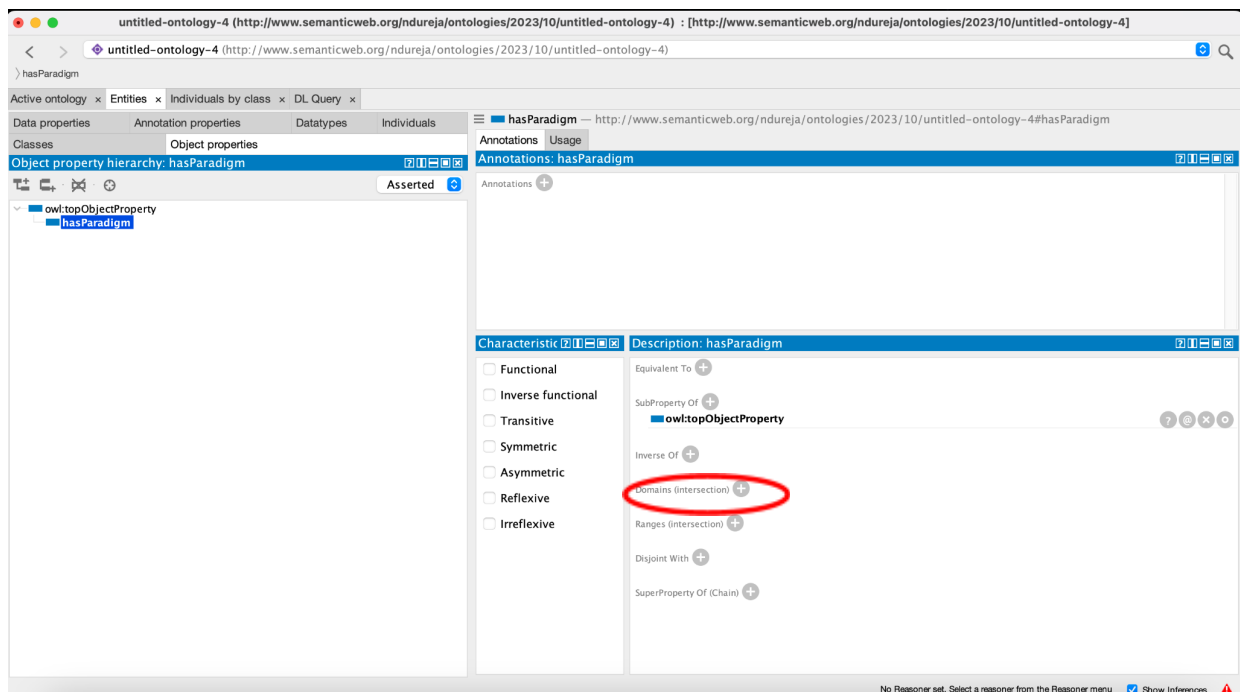
3. In the “**Add subclasses...**” popup, we will add our two classes “**ProgrammingLanguage**” and “**ProgrammingParadigm**”.



4. Next, we head on to the “**Object Properties**” sub-tab to define our object properties. Right-click on the “**owl:topObjectProperty**” and select “**Add sub-properties...**”. Enter the sub-property “**hasParadigm**” as shown in the image and click on “**Continue**”.



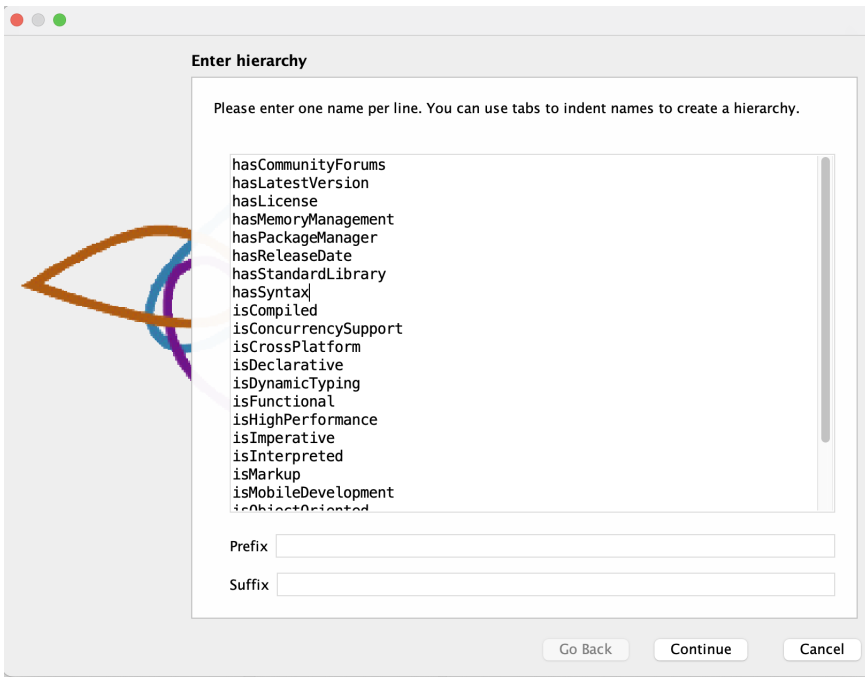
5. Now that we have added the objectProperty, we will define its range and domain. In the bottom-right window, click the “+” icon beside the “Domains” label.



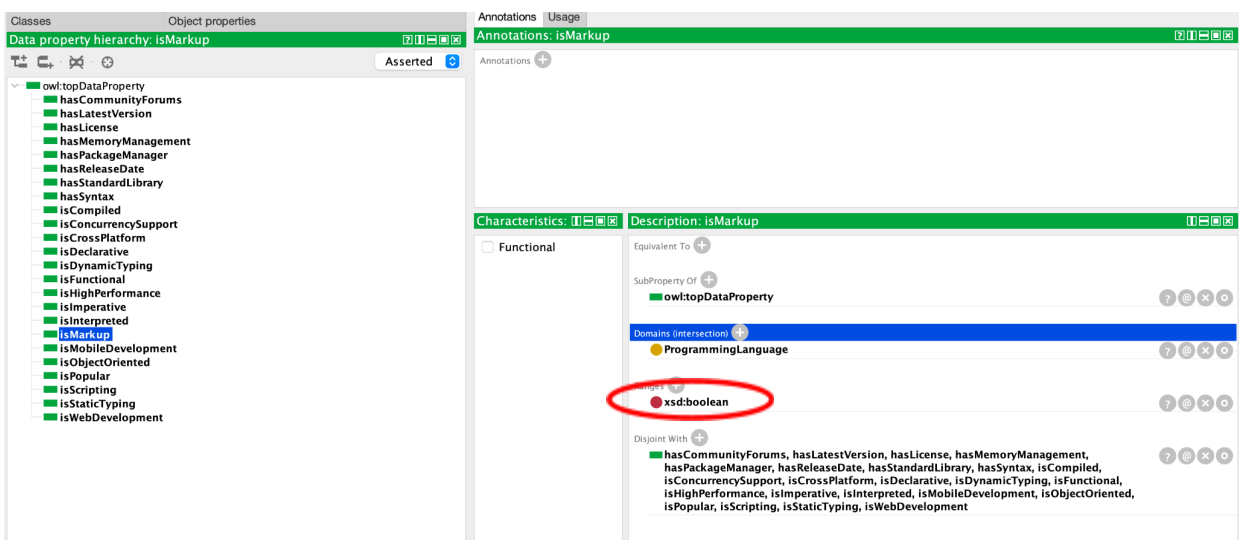
The screenshot shows the Protégé interface with the 'hasParadigm' property selected. The left pane displays the 'Object property hierarchy' with 'owl:topObjectProperty' expanded and 'hasParadigm' selected. The right pane shows the 'Description: hasParadigm' tab. On the left side of the right pane, there are checkboxes for various characteristics: Functional, Inverse functional, Transitive, Symmetric, Asymmetric, Reflexive, and Irreflexive. On the right side, there are several relationship types with plus icons: Equivalent To, SubProperty Of, Inverse Of, Domains (intersection), Ranges (intersection), Disjoint With, and SuperProperty Of (Chain). The 'Domains (intersection)' relationship is highlighted with a red circle.

Once the popup appears, select “**ProgrammingLanguage**” as your domain. Similarly, click on the “+” icon besides “**Range**” and choose “**ProgrammingParadigm**” as the range.

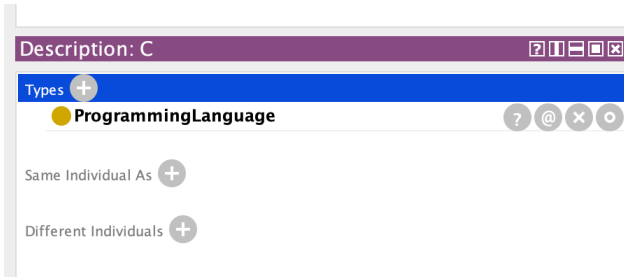
- Now, we head on to the Data properties sub-tab to define some data properties each of our languages will have. Right-click on the “**owl:topDataProperty**” and select the “**Add subproperties...**” option. Once the popup appears, enter the data properties as shown in the image below -



- Once the properties have been added, select each property one by one and define their domain and range. Domain for each property here would be “**ProgrammingLanguage**” and range will be the appropriate data type. For example, the property isMarkup will have a range as “**xsd:boolean**”.



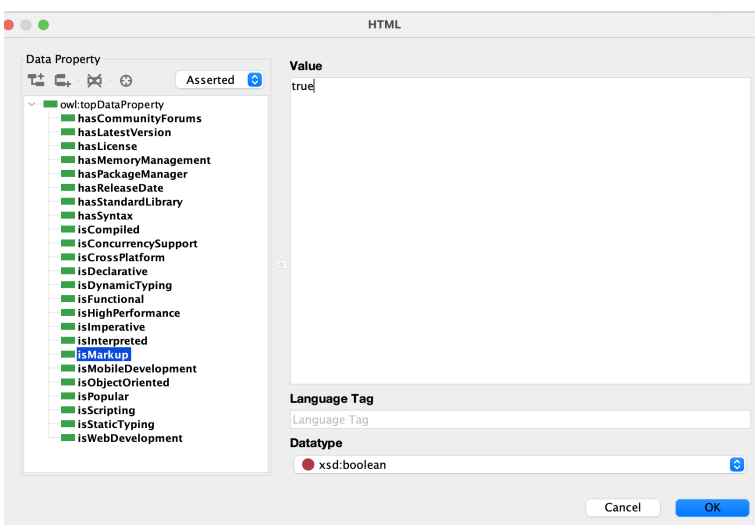
8. Next, we go to the **"Individuals"** sub-tab to add instances to our classes. Click on the **Add Individual** icon. Let's begin by adding programming languages first. In the popup that opens, type **"C"** and press OK. Similarly add other languages like **Python, Java, and HTML**.
9. Next, add individuals for paradigms, namely **Declarative, Imperative, Functional, Markup, Object Oriented, and Scripting**.
10. In the bottom right window, we begin defining these individuals and adding the properties. First, click on the **"+"** icon beside **"Types"** and select the type of the individual. For example, for individual **"C"** ->

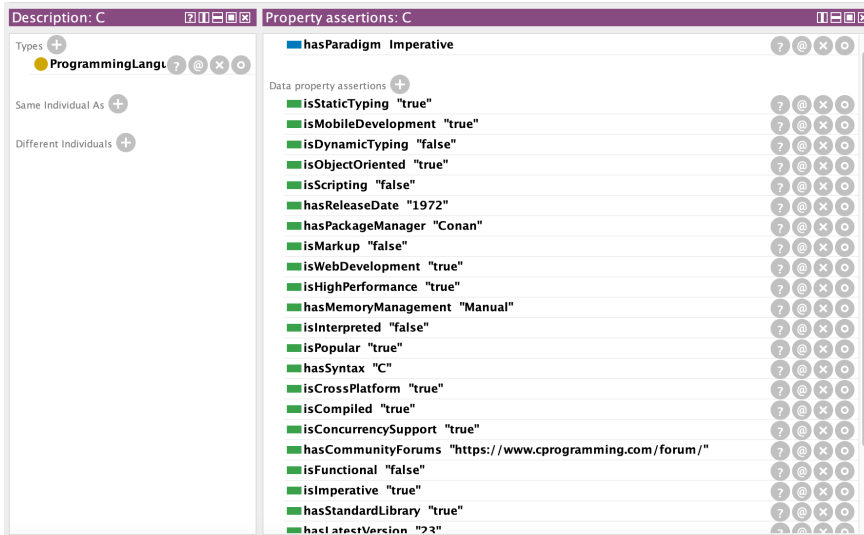


11. Next, click on the **"+"** icon beside **"Object property assertions"**, type object property **hasParadigm** and enter individual name. For example, for individual **"C"** -> the hasParadigm value will be **Imperative** and for **HTML**, the paradigm would be **Declarative** and **Markup**.



12. Now we add data properties to each individual. Click on the **"+"** icon beside **"Data property assertions"**. Now, in the popup which opened, select the property on the left side and enter its value on the right side.

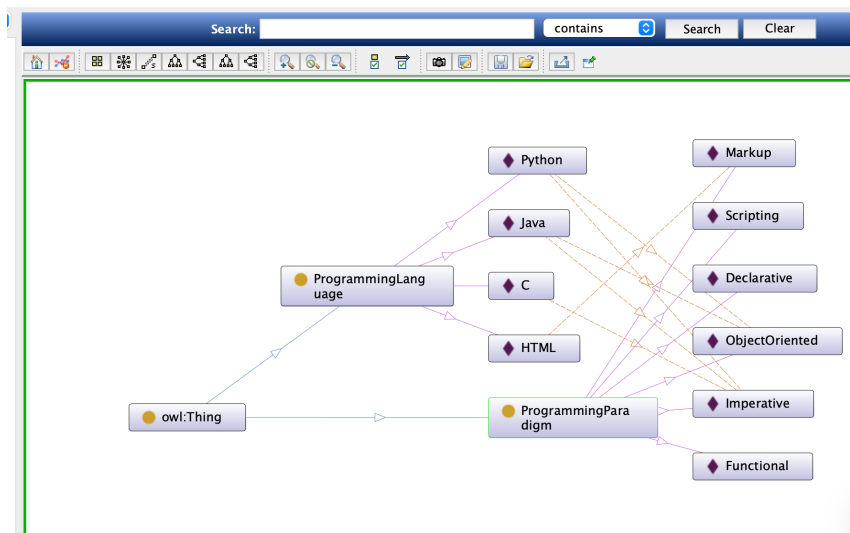
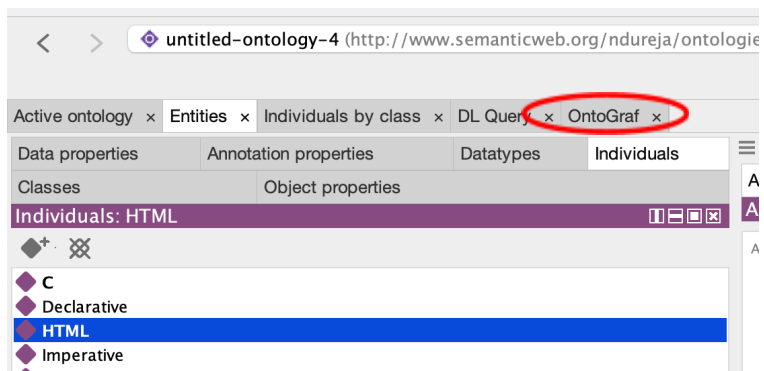




13. Once we have all the properties setup. Now is the time for visualizing and inferencing our ontology.

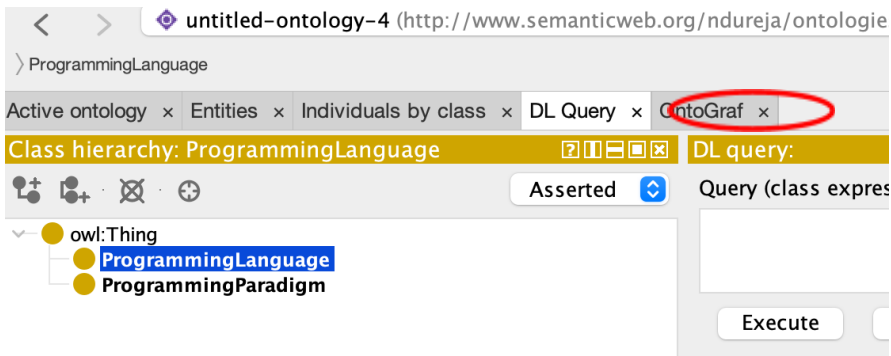
Step 4 : Visualizing the ontology

To visualize the ontology, head on to the **“OntoGraf”** tab in Protégé. If it is not visible, click on the **“Window”** option in the top bar and enable the tab.



Step 5 : Running Queries/Inferences

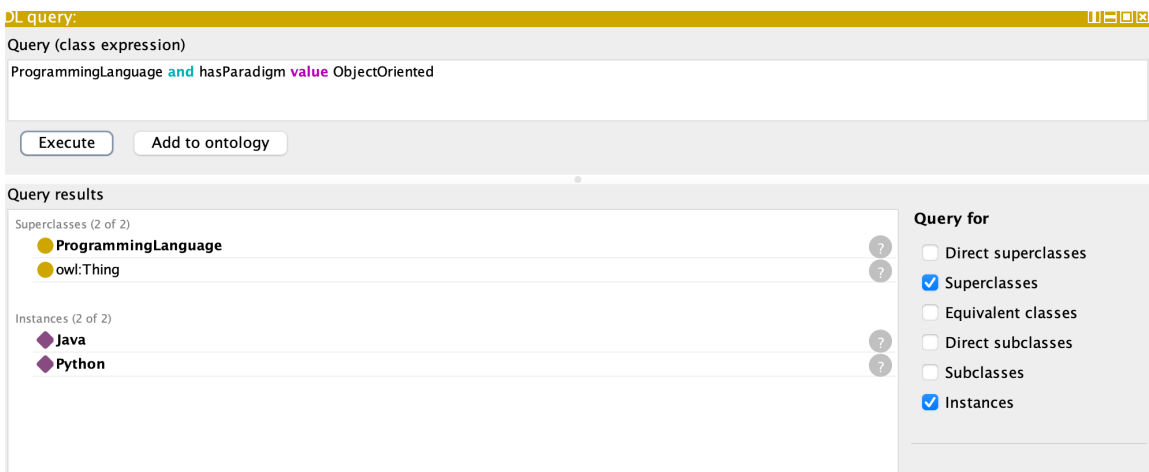
1. Click on the “DL Query” tab.



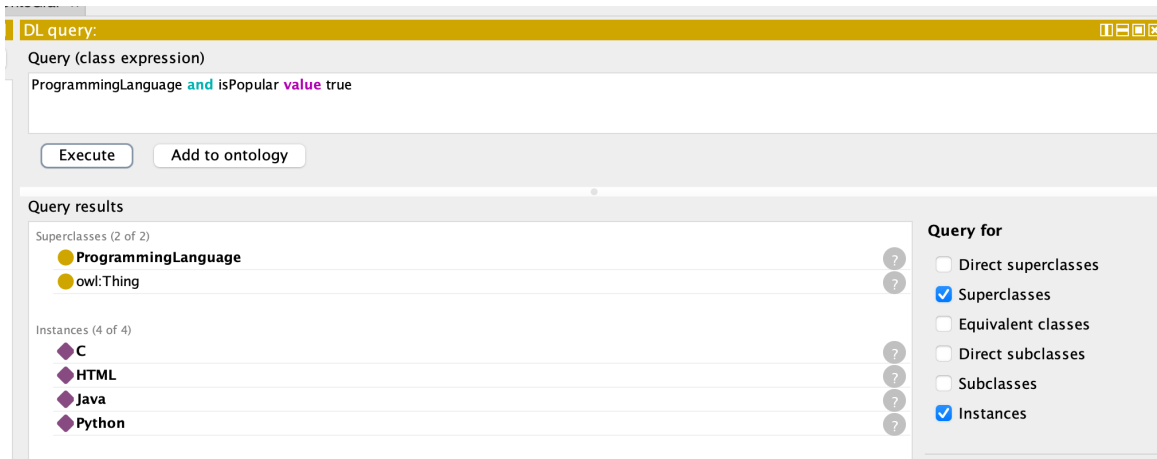
2. Select the “Reasoner” option present in the top bar on the screen. Select the available reasoner and click on “Start Reasoner”.

Now in the “Query” section, we can type our query. We will run the following queries -

1. Find Object-Oriented Languages:



2. Discover Popular Languages:



3. Find languages which are compiled and have high performance

DL query: ⌵ ⌶ ⌵ ⌵

Query (class expression)

ProgrammingLanguage and isHighPerformance value true and isCompiled value true

Execute Add to ontology

Query results

Subclasses (1 of 1)

- owl:Nothing

Instances (1 of 1)

- C

Query for

- ☐ Direct superclasses
- ☐ Superclasses
- ☐ Equivalent classes
- ☐ Direct subclasses
- ☒ Subclasses
- ☒ Instances

4. Find interpreted languages with community forums:

DL query: ⌵ ⌶ ⌵ ⌵

Query (class expression)

ProgrammingLanguage and isInterpreted value true and hasCommunityForums some xsd:string

Execute Add to ontology

Query results

Subclasses (1 of 1)

- owl:Nothing

Instances (1 of 1)

- Python

Query for

- ☐ Direct superclasses
- ☐ Superclasses
- ☐ Equivalent classes
- ☐ Direct subclasses
- ☒ Subclasses
- ☒ Instances

Result filters

Name contains

- ☒ Display owl:Thing (in superclass results)
- ☒ Display owl:Nothing (in subclass results)

5. Find languages released before 1980:

DL Query x

DL query: ⌵ ⌶ ⌵ ⌵

Query (class expression)

(ProgrammingLanguage) and hasReleaseDate max 1980

Execute Add to ontology

Query results

Subclasses (1 of 1)

- owl:Nothing

Instances (1 of 1)

- C

Query for

- ☐ Direct superclasses
- ☐ Superclasses
- ☐ Equivalent classes
- ☐ Direct subclasses
- ☒ Subclasses
- ☒ Instances