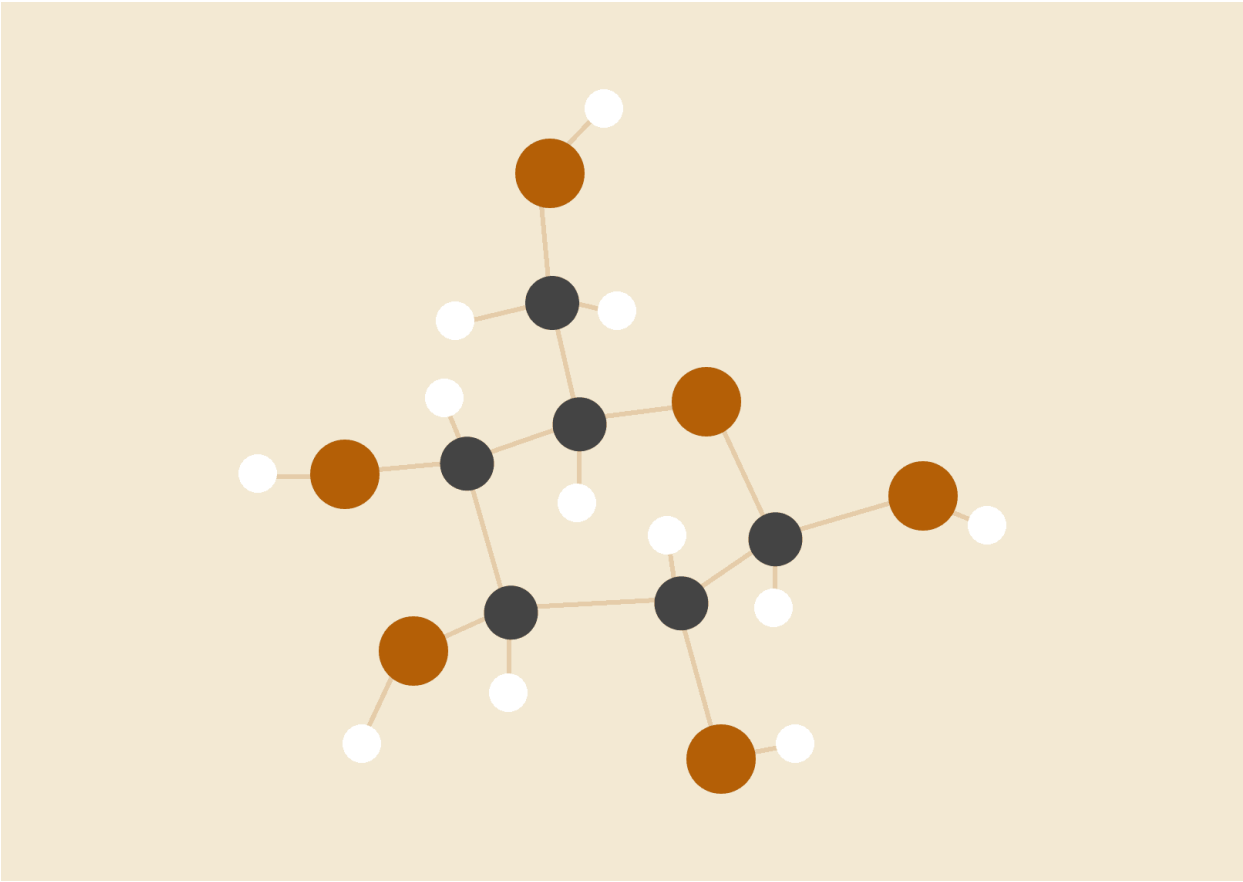
PROJECT REPORT

Comic Characters Ontology



INTRODUCTION

This report presents the development of an ontology for comic book characters, their details, the comics they appear in, and the publishers involved. By consolidating multiple datasets and utilizing Python's rdflib library, we created a comprehensive knowledge graph that captures the relationships and properties within the comic book domain. The ontology facilitates efficient organization and querying through SPARQL, allowing users to explore and retrieve valuable information about comic book entities.

1. Dataset

An extensive code was written in Python for mapping the provided dataset to the base ontology. First the necessary libraries were imported.

Importing Libraries

```
!pip install rdflib
!pip install pandas
!pip install numpy
```

```
from rdflib import Graph
from rdflib import URIRef, BNode, Literal
from rdflib import Namespace
from rdflib.namespace import OWL, RDF, RDFS, FOAF, XSD

import pandas as pd
import numpy as np
```

Multiple datasets namely “characters.csv”, “charactersToComics.csv”, “comics.csv” and “marvel_characters_info.csv” were combined into a single dataset.

Loading Dataset

```
1  df1 = pd.read_csv("/content/characters.csv")
2  df1
```

	characterID	name
0	1009220	Captain America
1	1010740	Winter Soldier
2	1009471	Nick Fury
3	1009552	S.H.I.E.L.D.
4	1009228	Sharon Carter
...	...	...
1165	1011395	Talon (Fraternity of Raptors)
1166	1011196	Captain Flint
1167	1009397	Lava-Man
1168	1011113	Blue Blade
1169	1011094	Xavin

1170 rows × 2 columns

```
1  df2 = pd.read_csv("/content/charactersToComics.csv")
2  df2
```

	comicID	characterID
0	16232	1009220
1	16232	1010740
2	16248	1009220
3	16248	1009471
4	16248	1009552
...	...	...
75252	45951	1009337
75253	45951	1011428
75254	45951	1011086
75255	45951	1009546
75256	45951	1009724

75257 rows × 2 columns

After the datasets were joined into one dataset. Then some data cleaning was performed where NULL values were replaced and data types were made consistent.

```
[ ]  1  df
```

	Name	Alignment	Gender	EyeColor	Race	HairColor	Publisher	SkinColor	Height	Weight	title	description
0	A-Bomb	good	Male	yellow	Human	No Hair	Marvel Comics	-	203	441	None	None
1	Abe Sapien	good	Male	blue	Icthyo Sapien	No Hair	Dark Horse Comics	blue	191	65	None	None
2	Abin Sur	good	Male	blue	Ungaran	No Hair	DC Comics	red	185	90	None	None
3	Abomination	bad	Male	green	Human / Radiation	No Hair	Marvel Comics	-	203	441	None	None
4	Abraxas	bad	Male	blue	Cosmic Entity	Black	Marvel Comics	-	-99	-99	None	None
...	...	...	...	...	...	...	...	...	...	...	...	...
729	Yellowjacket II	good	Female	blue	Human	Strawberry Blond	Marvel Comics	-	165	52	None	None
730	Ymir	good	Male	white	Frost Giant	No Hair	Marvel Comics	white	304	-99	None	None
731	Yoda	good	Male	brown	Yoda's species	White	George Lucas	green	66	17	None	None
732	Zatanna	good	Female	blue	Human	Black	DC Comics	-	170	57	None	None
733	Zoom	bad	Male	red	-	Brown	DC Comics	-	185	81	None	None

55797 rows × 12 columns

2. Ontology Creation

1. The code starts by importing the required libraries and creating a new RDF graph using the `Graph()` function.

2. Namespaces are defined using the `Namespace()` function. Namespaces are used to define the prefixes for URIs in the ontology. In this code, the namespaces `onto`, `rdf`, `rdfs`, and `owl` are defined.

3. Class definitions are created using the `onto.` prefix. Three classes are defined: `Character`, `Publisher`, and `Comic`.

4. Object properties are defined using the `onto.` prefix. Two object properties are defined: `hasPublisher` and `appearsIn`. These properties define the relationships between entities in the ontology.

5. Data properties are defined using the `onto.` prefix. Several data properties are defined, such as `has_alignment_property`, `has_eye_color_property`, `has_gender_property`, and so on. These properties are used to define attributes or characteristics of entities in the ontology.

Ontology Creation

```
[ ]
# Create a new RDF graph
g = Graph()

# Define the namespaces
onto = Namespace("http://comicCharacters.com/")
rdf = Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#")
rdfs = Namespace("http://www.w3.org/2000/01/rdf-schema#")
owl = Namespace("http://www.w3.org/2002/07/owl#")

# Define the classes
character_class = onto.Character
publisher_class = onto.Publisher
comic_class = onto.Comic

# Define the object property
has_publisher_property = onto.hasPublisher
appearsIn_property = onto.appearsIn
has_characters_property = onto.hasCharacters

# Define the data properties
has_alignment_property = onto.hasAlignment
has_eye_color_property = onto.hasEyeColor
has_gender_property = onto.hasGender
has_hair_color_property = onto.hasHairColor
has_height_property = onto.hasHeight
has_name_property = onto.hasName
has_skin_color_property = onto.hasSkinColor
has_weight_property = onto.hasWeight
has_title_property = onto.hasTitle
hasRace = onto.hasRace
hasDescription = onto.hasDescription
```

6. Class and property definitions are added to the graph using the `g.add()` function. Each definition is added as a triple with the subject, predicate, and object.

```
# Add class and property definitions to the graph
g.add((character_class, RDF.type, owl.Class))
g.add((publisher_class, RDF.type, owl.Class))
g.add((comic_class, RDF.type, owl.Class))

g.add((has_publisher_property, RDF.type, owl.ObjectProperty))
g.add((has_publisher_property, RDFS.domain, comic_class))
g.add((has_publisher_property, RDFS.range, publisher_class))

g.add((appearsIn_property, RDF.type, owl.ObjectProperty))
g.add((appearsIn_property, RDFS.domain, character_class))
g.add((appearsIn_property, RDFS.range, comic_class))

g.add((has_characters_property, RDF.type, owl.ObjectProperty))
g.add((has_characters_property, OWL.inverseOf, appearsIn_property))

g.add((has_alignment_property, RDF.type, owl.DatatypeProperty))
g.add((has_eye_color_property, RDF.type, owl.DatatypeProperty))
g.add((has_gender_property, RDF.type, owl.DatatypeProperty))
g.add((has_hair_color_property, RDF.type, owl.DatatypeProperty))
g.add((has_height_property, RDF.type, owl.DatatypeProperty))
g.add((has_name_property, RDF.type, owl.DatatypeProperty))
g.add((has_skin_color_property, RDF.type, owl.DatatypeProperty))
g.add((has_weight_property, RDF.type, owl.DatatypeProperty))
g.add((has_title_property, RDF.type, owl.DatatypeProperty))
g.add((hasRace, RDF.type, owl.DatatypeProperty))
g.add((hasDescription, RDF.type, owl.DatatypeProperty))
```

7. Domain and range restrictions are set for the properties using the `RDFS.domain` and `RDFS.range` predicates. These restrictions define which classes the properties can be applied to and what types of values they can have.

8. Finally, the graph is serialized and saved to disk as a Turtle file format (.ttl) with the name "comicChar.owl".

```
# Set domain for the datatype properties
g.add((has_alignment_property, RDFS.domain, character_class))
g.add((has_eye_color_property, RDFS.domain, character_class))
g.add((has_gender_property, RDFS.domain, character_class))
g.add((has_hair_color_property, RDFS.domain, character_class))
g.add((has_height_property, RDFS.domain, character_class))
g.add((has_name_property, RDFS.domain, character_class))
g.add((has_name_property, RDFS.domain, publisher_class))
g.add((has_skin_color_property, RDFS.domain, character_class))
g.add((has_weight_property, RDFS.domain, character_class))
g.add((has_title_property, RDFS.domain, comic_class))
g.add((hasRace, RDFS.domain, character_class))
g.add((hasDescription, RDFS.domain, comic_class))

# Set range for the datatype properties
g.add((has_alignment_property, RDFS.range, XSD.string))
g.add((has_eye_color_property, RDFS.range, XSD.string))
g.add((has_gender_property, RDFS.range, XSD.string))
g.add((has_hair_color_property, RDFS.range, XSD.string))
g.add((has_height_property, RDFS.range, XSD.int))
g.add((has_name_property, RDFS.range, XSD.string))
g.add((has_name_property, RDFS.range, XSD.string))
g.add((has_skin_color_property, RDFS.range, XSD.string))
g.add((has_weight_property, RDFS.range, XSD.int))
g.add((has_title_property, RDFS.range, XSD.string))
g.add((hasRace, RDFS.range, XSD.string))
g.add((hasDescription, RDFS.range, XSD.string))

# Save the graph to disk
g.serialize("comicChar.owl", format="ttl")
```

3. Mapping Dataset to Ontology

The provided code is a function called `createTriples()` that performs ontology mapping. Let's go through the code step by step:

1. It starts by creating a new instance of the `Graph` class from the `rdflib` library. This graph will be used to represent the ontology and store the triples.

2. It parses an input ontology file named "comicChar.owl" in Turtle format and adds its content to the graph. The `g.parse()` function is used for parsing and loading the ontology into the graph.

3. Next, it defines several namespaces using the `Namespace` class from `rdflib`. These namespaces are used to create compact URIs for the ontology concepts. Three namespaces are defined:

- `dbo`: Represents the "https://dbpedia.org/ontology/" namespace.

- `dbr`: Represents the "https://dbpedia.org/resource/" namespace.

- `comic`: Represents the "http://comicCharacters.com/" namespace.

4. It binds the defined namespaces to their respective prefixes using the `g.bind()` function. This allows using the prefixes instead of the full URIs in the triple statements.

5. A loop is started to iterate over a DataFrame (`df`) to extract information about comic book characters.

6. Inside the loop, various attributes of the comic book characters, such as Name, Alignment, Gender, EyeColor, Race, HairColor, Publisher, SkinColor, Height, Weight, Title, and Description, are extracted from the DataFrame.

```
def createTriples():

    g = Graph()
    g.parse("/content/comicChar.owl", format="ttl")

#     Special namspaces to create
    dbo = Namespace("https://dbpedia.org/ontology/")
    dbr = Namespace("https://dbpedia.org/resource/")
    comic = Namespace("http://comicCharacters.com/")

#     Prefixes
    g.bind("dbo", dbo) #dbo is a newly created namespace
    g.bind("dbr", dbr)
    g.bind("comic", comic)

    for i in range(len(df)):
    # for i in range(50):

        Name = df.iloc[i,0]
        Alignment = df.iloc[i, 1]
        Gender = df.iloc[i,2]
        EyeColor = df.iloc[i,3]
        Race = df.iloc[i,4]
        HairColor = df.iloc[i,5]
        Publisher = df.iloc[i,6]
        SkinColor = df.iloc[i,7]
        Height = df.iloc[i,8]
        Weight = df.iloc[i,9]
        Title = df.iloc[i,10]
        Description = df.iloc[i,11]
```

7. Literal values are created for each attribute using the `Literal` class from

`rdflib`. These literals are used to represent attribute values in the ontology.

8. Some preprocessing is done on the Name, Publisher, and Title attributes to replace spaces with underscores and create valid URIs.

9. URIs are created using the `URIRef` class from `rdflib` for the Character, Publisher, and Title.

```
################ Making Literals #############

        Name_l = Literal(Name, datatype=XSD.string)
        Alignment_l = Literal(Alignment, datatype=XSD.string)
        Gender_l = Literal(Gender, datatype=XSD.string)
        EyeColor_l = Literal(EyeColor, datatype=XSD.string)
        Race = Literal(Race, datatype=XSD.string)
        SkinColor_l = Literal(SkinColor, datatype=XSD.string)
        HairColor_l = Literal(HairColor, datatype=XSD.string)
        Height_l = Literal(Height,datatype=XSD.int)
        Weight_l = Literal(Weight, datatype=XSD.int)
        Publisher_l = Literal(Publisher, datatype=XSD.string)
        Title_l = Literal(Title, datatype=XSD.string)
        Desc_l = Literal(Description, datatype=XSD.string)

############## Making URIs ##############

        Name = Name.replace(' ', '_')
        str1 = 'http://comicCharacters.com/' + Name
        Character = URIRef(str1)

        Publisher_str = Publisher.replace(' ', '_')
        str1 = 'http://comicCharacters.com/' + Publisher_str
        Publisher = URIRef(str1)
        str2 = 'https://dbpedia.org/resource/' + Publisher_str
        dbr = URIRef(str2)
        g.add((Publisher, OWL.sameAs, dbr))

        Title_str = Title.replace(' ', '_')
        str1 = 'http://comicCharacters.com/' + Title_str
        Title = URIRef(str1)
```

10. Various triple statements are added to the graph (`g.add()`) to represent the relationships and attributes of the comic book characters, the publisher, and the title.

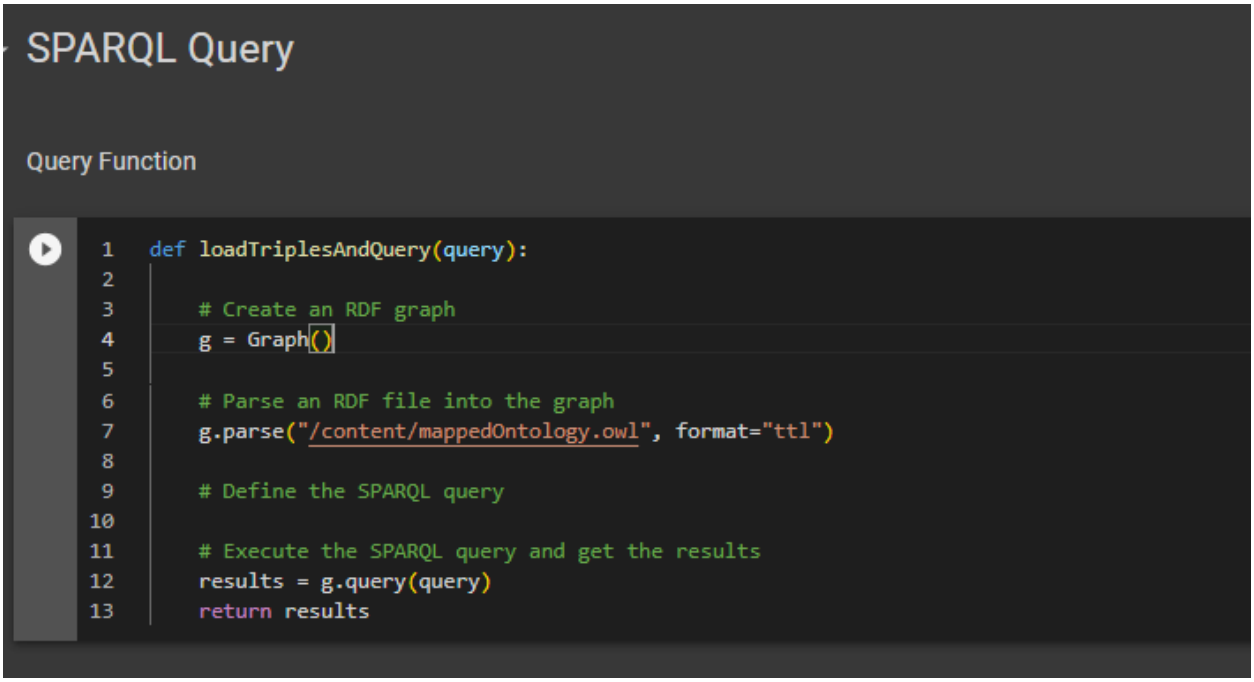
11. After the loop finishes, the graph is serialized to a file named "mappedOntology.owl" in Turtle format using the `g.serialize()` function.

12. Finally, a message is printed to indicate that the graph has been saved.

```
############# Making Connnections ###########

        g.add((Character, RDF.type, comic.Character))
        g.add((Character, comic.hasName, Name_l))
        g.add((Character, RDFS.label, Name_l))
        g.add((Character, comic.hasAlignment, Alignment_l))
        g.add((Character, comic.hasGender, Gender_l))
        g.add((Character, comic.hasSkinColor, SkinColor_l))
        g.add((Character, comic.hasEyeColor, EyeColor_l))
        g.add((Character, comic.hasHairColor, HairColor_l))
        g.add((Character, comic.hasHeight, Height_l))
        g.add((Character, comic.hasWeight, Weight_l))
        g.add((Character, comic.hasRace, Race))
        g.add((Character, comic.hasPublisher, Publisher))
        g.add((Character, comic.appearsIn, Title))

################################

        g.add((Publisher, RDF.type, comic.Publisher))
        g.add((Publisher, comic.hasName, Publisher_l))
        g.add((Publisher, RDFS.label, Publisher_l))

################################

        g.add((Title, RDF.type, comic.Comic))
        g.add((Title, comic.hasName, Title_l))
        g.add((Title, RDFS.label, Title_l))
        g.add((Title, comic.hasPublisher, Publisher))
        g.add((Title, comic.hasDescription, Desc_l))

################################

    print("\nSaving graph to 'mappedOntology.ttl':\n\n")
    g.serialize(destination="mappedOntology.owl", format="ttl")
```

4. SPARQL Queries

For querying we load triples and run the query function.



Here I have demonstrated a sample query where the names of all characters and their respective publishers are retrieved

1. Retrieve the names of all characters and their respective publishers:

```
#Load triples and query local graph
results = loadTriplesAndQuery( query="""
        PREFIX comic: <http://comicCharacters.com/>
        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

        SELECT ?characterName ?publisherName
        WHERE {
            ?character a comic:Character ;
                       comic:hasName ?characterName ;
                       comic:hasPublisher ?publisher .
            ?publisher comic:hasName ?publisherName .
        }

    """)

# Process and print the query results
for row in results:
    characterName = row['characterName']
    publisherName = row['publisherName']
    print(f"Character Name: {characterName}, Publisher Name: {publisherName}")
```

```
Character Name: A-Bomb, Publisher Name: Marvel Comics
Character Name: Abe Sapien, Publisher Name: Dark Horse Comics
Character Name: Abin Sur, Publisher Name: DC Comics
Character Name: Abomination, Publisher Name: Marvel Comics
Character Name: Abraxas, Publisher Name: Marvel Comics
Character Name: Absorbing Man, Publisher Name: Marvel Comics
Character Name: Adam Monroe, Publisher Name: NBC - Heroes
Character Name: Adam Strange, Publisher Name: DC Comics
Character Name: Agent 13, Publisher Name: Marvel Comics
Character Name: Agent Bob, Publisher Name: Marvel Comics
Character Name: Agent Zero, Publisher Name: Marvel Comics
Character Name: Air-Walker, Publisher Name: Marvel Comics
Character Name: Ajax, Publisher Name: Marvel Comics
Character Name: Alan Scott, Publisher Name: DC Comics
Character Name: Alex Mercer, Publisher Name: Wildstorm
Character Name: Alex Woolsly, Publisher Name: NBC - Heroes
Character Name: Alfred Pennyworth, Publisher Name: DC Comics
Character Name: Alien, Publisher Name: Dark Horse Comics
Character Name: Allan Quatermain, Publisher Name: Wildstorm
Character Name: Amazo, Publisher Name: DC Comics
Character Name: Ammo, Publisher Name: Marvel Comics
```

CONCLUSION

In conclusion, the creation of an ontology for comic book characters, comics, and publishers, coupled with the utilization of rdflib and SPARQL, offers a valuable resource for researchers, enthusiasts, and industry professionals. The ontology enhances the organization and accessibility of comic book information, providing a structured framework for efficient data retrieval and analysis. Further expansion and refinement of the ontology will continue to enrich the representation of the comic book domain, enabling deeper exploration and understanding of this vibrant universe.