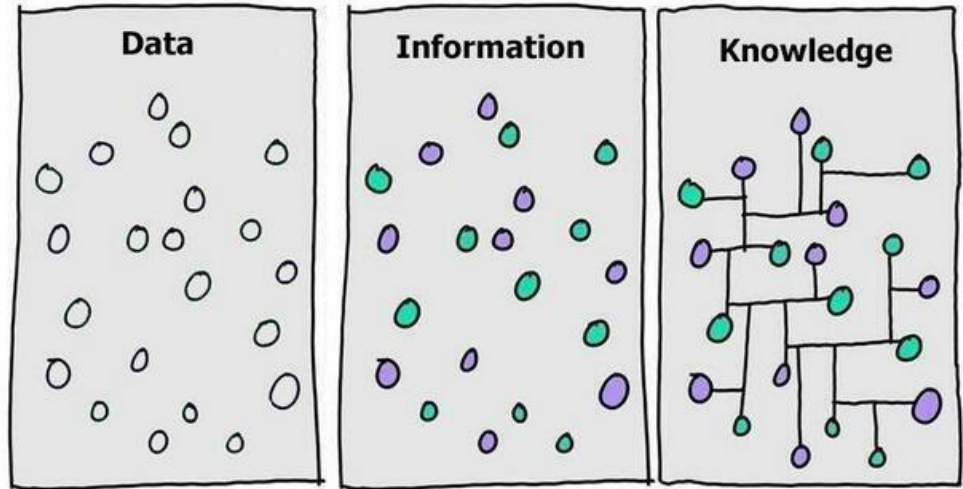


Linked Open Data

Knowledge

Knowledge: how to represent, organize and digitize it.



knowledge noun

knowl·edge (ˈnä-lij )

[Synonyms of *knowledge* >](#)

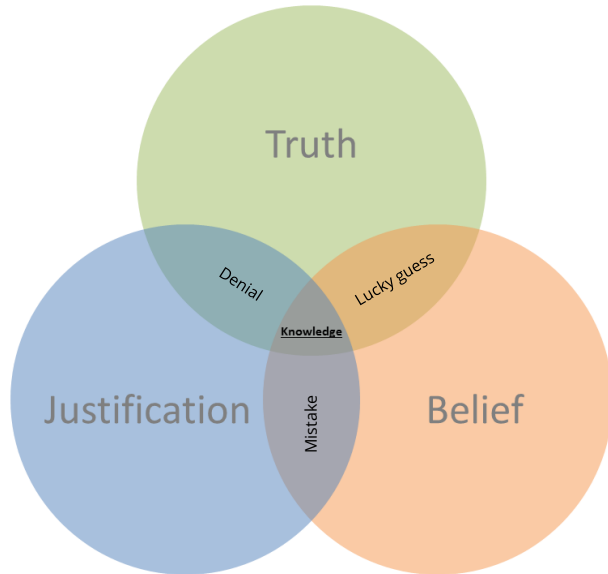
- 1 a (1)** : the fact or condition of knowing something with familiarity gained through experience or [association](#)

<https://www.merriam-webster.com/dictionary/knowledge>

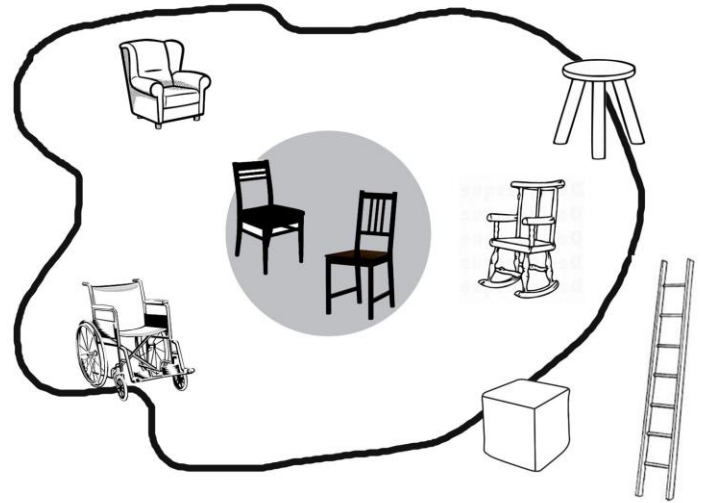
Knowledge: a journey through different disciplines

Philosophy:

Justified true belief (Plato)

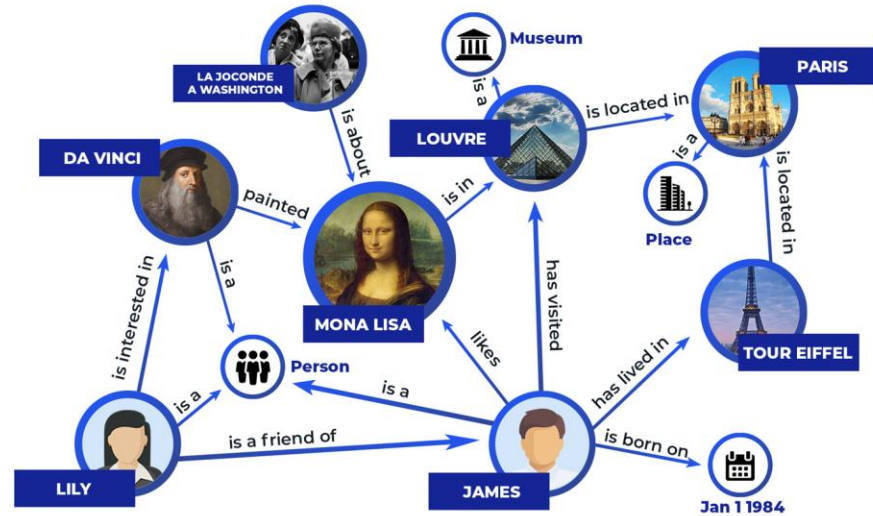


Family resemblance



Knowledge: a journey through different disciplines

Computer Science: Network of entities (objects, events, situations or abstract concepts) organized and connected according to a certain semantics.

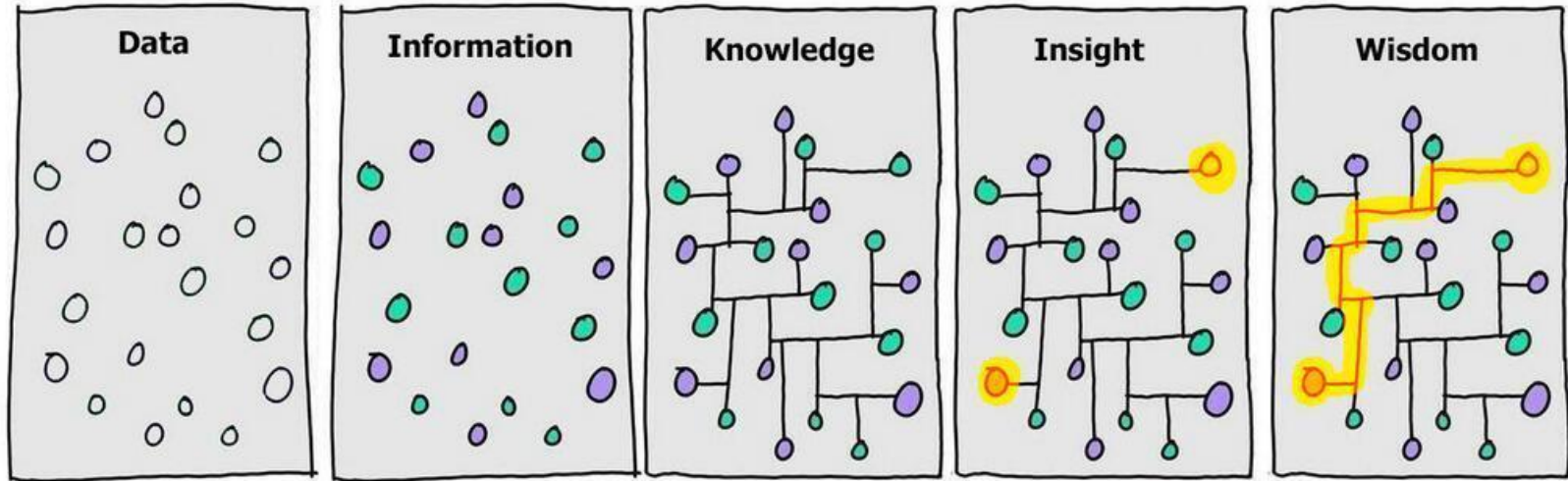


Knowledge: a journey through different disciplines

For the journey through knowledge, **no previous knowledge** of philosophy, logic or computer science is required...but of course it is welcome!

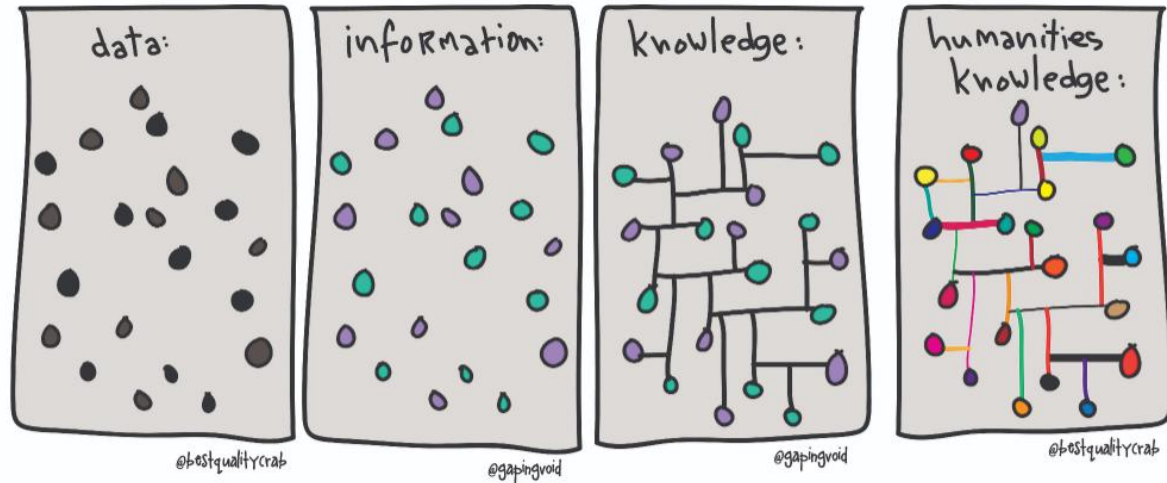
Why digital Knowledge?

A digital, **machine-readable** knowledge allows a broader overview. A machine works on quantity while a human being works on quality (distant reading theory)



Even more interdisciplinary

We will apply the methodology and techniques of knowledge organization and representation in museums domain.



What is Data?

data noun

da·ta ('dā-tə ◀▶) ('da- ◀▶) also ('dä- ◀▶)

plural in form but singular or plural in construction

often attributive

Synonyms of data >

- 1** : factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation
- 2** : information in digital form that can be transmitted or processed

<https://www.merriam-webster.com/dictionary/data>

Unstructured Data

Information in digital form that can be transmitted and processed **BUT does not follow a data model.**

- Images
- Texts
- Videos

Unstructured Data

What we see:

Yaoundé is the capital of
Cameroon and has 2.7 million
inhabitants

What a computer sees:

```
01011001 01100001 01101111 01110101 01101110 01100100
11000011 10101001 00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000 01100011 01100001
01110000 01101001 01110100 01100001 01101100 00100000
01101111 01100110 00100000 01000011 01100001 01101101
01100101 01110010 01101111 01101111 01101110 00100000
01100001 01101110 01100100 00100000 01101000 01100001
01110011 00100000 00110010 00101110 00110111 00100000
01101101 01101001 01101100 01101100 01101001 01101111
01101110 00100000 01101001 01101110 01101000 01100001
01100010 01101001 01110100 01100001 01101110 01110100
01110011
```

Unstructured Data

What we see:

Yaoundé is the capital of
Cameroon and has 2.7 million
inhabitants

What a computer sees:

```
01011001 01100001 01101111 01110101 01101110 01100100
11000011 10101001 00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000 01100011 01100001
01110000 01101001 01110100 01100001 01101100 00100000
01101111 01100110 00100000 01000011 01100001 01101101
01100101 01110010 01101111 01101111 01101110 00100000
01100001 01101110 01100100 00100000 01101000 01100001
01110011 00100000 00110010 00101110 00110111 00100000
01101101 01101001 01101100 01101100 01101001 01101111
01101110 00100000 01101001 01101110 01101000 01100001
01100010 01101001 01110100 01100001 01101110 01110100
01110011
```

Can be transmitted

Unstructured Data

What we see:

Yaoundé is the capital of
Cameroon and has 2.7 million
inhabitants

What a computer sees:

```
01011001 01100001 01101111 01110101 01101110 01100100
11000011 10101001 00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000 01100011 01100001
01110000 01101001 01110100 01100001 01101100 00100000
01101111 01100110 00100000 01000011 01100001 01101101
01100101 01110000 01101110 01101100 01101110 00100000
01100001 01101110 01100100 00100000 01101000 01100001
01110011 00100000 00110010 00101110 00110111 00100000
01101101 01101001 01101100 01101100 01101001 01101111
01101110 00100000 01101001 01101110 01101000 01100001
01100010 01101001 01110100 01100001 01101110 01110100
01110011
```

Unstructured Data

What we see:

Yaoundé is the capital of
Cameroon and has 2.7 million
inhabitants

What a computer sees:

```
01011001 01100001 01101111 01110101 01101110 01100100
11000011 10101001 00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000 01100011 01100001
01110000 01101001 01110100 01100001 01101100 00100000
01101111 01100110 00100000 01000011 01100001 01101101
01100101 01110000 01101110 01101110 01101110 00100000
01100001 01101110 01100100 00100000 01101000 01100001
01110011 00100000 00110010 00101110 00110111 00100000
01101101 01101001 01101001 01101001 01101001 01101111
01101110 00100000 01101001 01101110 01101000 01100001
01100010 01101001 01110100 01100001 01101110 01110100
01110011
```

Can be transmitted

Can be processed

No Data Model

Unstructured Data

What we see:

Yaoundé is the capital of
Cameroon and has 2.7 million
inhabitants



We process information, we
acquire knowledge! 🧠

What a computer sees:

```
01011001 01100001 01101111 01110101 01101110 01100100
11000011 10101001 00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000 01100011 01100001
01110000 01101001 01110100 01100001 01101100 00100000
01101111 01100110 00100000 01000011 01100001 01101101
01100101 01110000 01101110 01101110 01101110 00100000
01100001 01101110 01100100 00100000 01101000 01100001
01110011 00100000 00110010 00101110 00110111 00100000
01101101 01101001 01101001 01101001 01101001 01101111
01101110 00100000 01101001 01101110 01101000 01100001
01100010 01101001 01110100 01100001 01101110 01110100
01110011
```


Unstructured Data

What we see:

Yaoundé is the capital of
Cameroon and has 2.7 million
inhabitants



We process information, we
acquire knowledge! 🧠

What a computer sees:

```
01011001 01100001 01101111 01110101 01101110 01100100
11000011 10101001 00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000 01100011 01100001
01110000 01101001 01110100 01100001 01101100 00100000
01101111 01100110 00100000 01000011 01100001 01101101
01100101 01110000 01101110 01101110 01101110 00100000
01100001 01101110 01100100 00100000 01101000 01100001
01110011 00100000 00110010 00101110 00110111 00100000
01101101 01101001 01101001 01101001 01101001 01101111
01101110 00100000 01101001 01101110 01101000 01100001
01100010 01101001 01110100 01100001 01101110 01110100
01110011
```

Can be transmitted

Can be processed

No Data Model



??? 🤖

Structured Data

Information in digital form that can be transmitted and processed **structured according to a data model**.

Data model: A model that organizes the data and defines the connection between them.

Structured Data

What we see:

city	capital_of	#_inhabitants
Yaoundé	Cameroon	2.700.000

What a computer sees:

city	capital_of	#_inhabitants
Yaoundé	Cameroon	2.700.000

Structured Data

What we see:

city	capital_of	#_inhabitants
Yaoundé	Cameroon	2.700.000



We process information, we
acquire knowledge! 🧠

What a computer sees:

city	capital_of	#_inhabitants
Yaoundé	Cameroon	2.700.000



The computer processes information,
it acquires knowledge! 🤖

An iceberg floating in the ocean. The tip of the iceberg is above the water line, and the much larger base is submerged. The sky is blue with some clouds. The water is a deep blue. The iceberg is white and jagged. The text '20%' is in a blue hexagon above the tip. The text '80%' is in a blue hexagon below the water line. The text 'Structured Data' is to the right of the tip. The text 'Unstructured Data' is to the right of the submerged part. A list of file types is to the right of the submerged part.

20%

Structured Data

80%

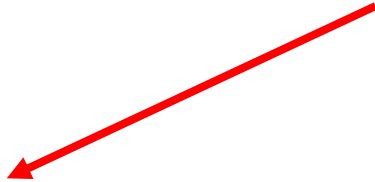
Unstructured Data

PDFs
Emails
Word docs
Spreadsheets
Images
Presentations
Web pages
Social media
and more...

Traditional Structured Data Limitations

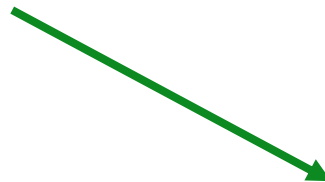
- Structured data values are still textual

{"city": "Yaoundé"}



What a computer sees:

01011001 01100001 01101111
01110101 01101110 01100100
11000011 10101001



What's behind that name:

An inhabited place also known as
Yaounde or Ongola that has
coordinates 3°51'28"N, 11°31'5"E

Traditional Structured Data Limitations

- Data structured according to different data models and formats cannot interoperate.

city	capital_of	#_inhabitants
Yaoundé	Cameroon	2.700.000

```
{  
  "place_name": "Dodoma",  
  "country": "Tanzania",  
  "citizens": 2.200.000  
}
```

Traditional Structured Data Limitations

- Data structured according to different data models and formats cannot interoperate.

city	capital_of	#_inhabitants
Yaoundé	Cameroon	2.700.000



```
{  
  "place_name": "Dodoma",  
  "country": "Tanzania",  
  "citizens": 2.200.000  
}
```


Linked Open Data

AKA

Semantic Web

AKA

Web 3.0



Linked Open Data: 3 Rules

1. We Use URIs as names for things.

URI (Uniform Resource Identifier)

1. Important information are linked to things

When someone looks up a URI, provide useful information, using the standards (RDF)

1. Knowledge is relationship

Include links to other URIs so that they can discover more things.

Uniform Resource Identifier (URI)

URI (Uniform Resource Identifier)

http: Hypertext Transfer Protocol (protocol to access the resource)

We can identify anything we put online

<https://www.wikidata.org/wiki/Q3808> (Yaoundé)

<https://www.wikidata.org/wiki/Q1009> (Cameroon)

URI Best Practices

1. Unique → It must be unique
2. Human-friendly → Keep it short and to the point
3. Consistent → Once you pick your URI structure, be consistent and follow it
4. Structured → Structure must be adaptable
5. Explicit → The URI should include keywords that are important to the Entity

URI Best Practices

<http://example.com/> is a free domain for examples.

URI → <domain>/[key information]/[name]

<http://example.com/country/cameroon>

<http://example.com/event/war/ww2>

- Spaces should be replaced consistently with hyphens (-) or underscore (_)
- No special characters

FAIR Principles

Linked Open Data are:

- **F**indable: data are findable on the web.
- **A**ccessible: data are freely accessible thanks to http protocol.
- **I**nteroperable: data are structured in a formal, accessible, shared, and broadly applicable language.
- **R**eusable: data can be replicated and/or combined in different settings.

FAIR Principles: <https://www.go-fair.org/fair-principles/>

Open Data

“Open data and content can be freely **used, modified, and shared** by **anyone** for any **purpose**”

(<https://opendefinition.org/>)

Open Data: science, government and non-profit organizations→
Museums

Semantic Web

Standards

1. RDF

Resource Description Framework (RDF)

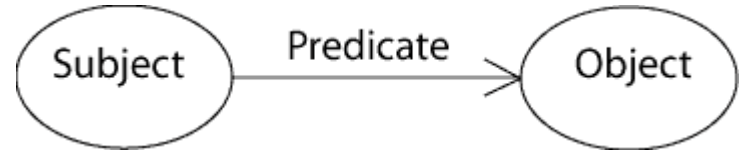
The standard framework for representing information in the Web (i.e. LOD!).

In RDF, each concept is expressed in the form of a **triple**.

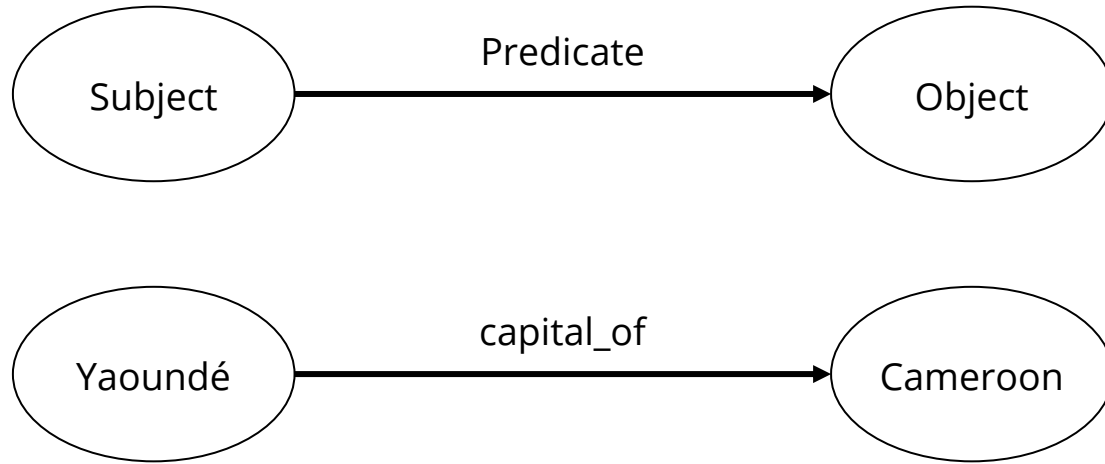
A triple represents a statement of a **relationship** between things.

A triple is composed by:

- 1 **Subject**
- 1 **Predicate** (aka Property)
- 1 **Objects**



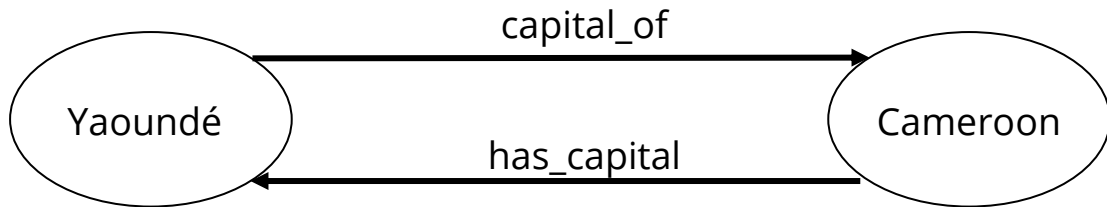
Graph Data Model



Knowledge is Relationship

Graph Data Model

Relationships can be multiple and two-sided

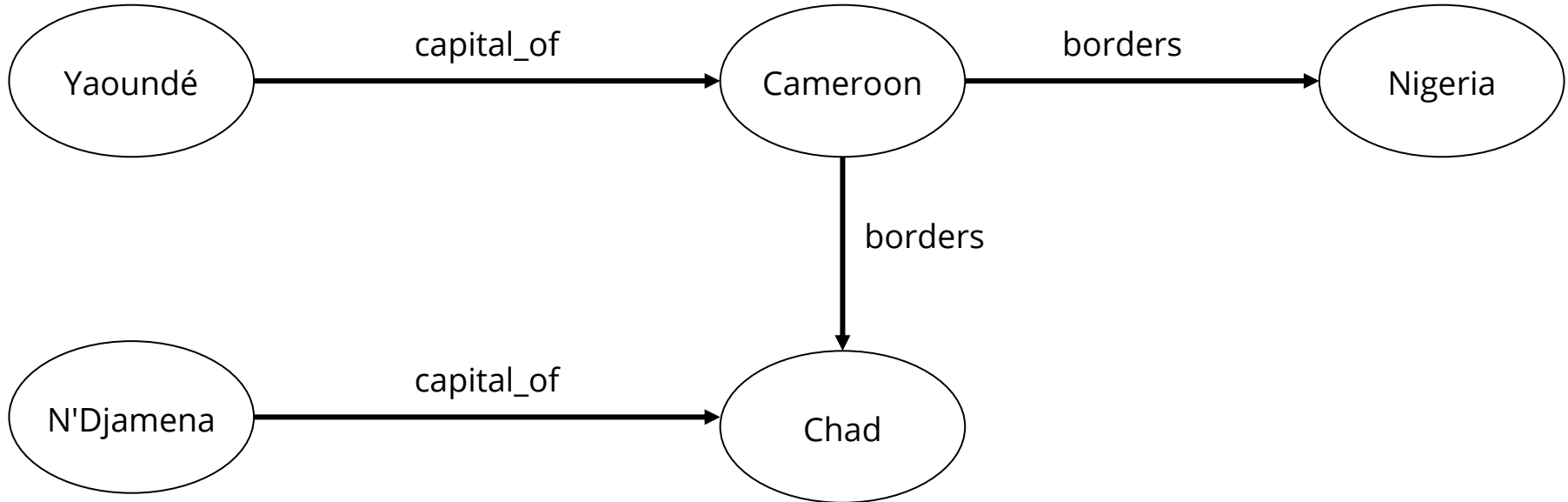


In the example we have 2 triples:

1. Yaoundé —capital_of→ Cameroon
2. Cameroon —has_capital→ Yaoundé

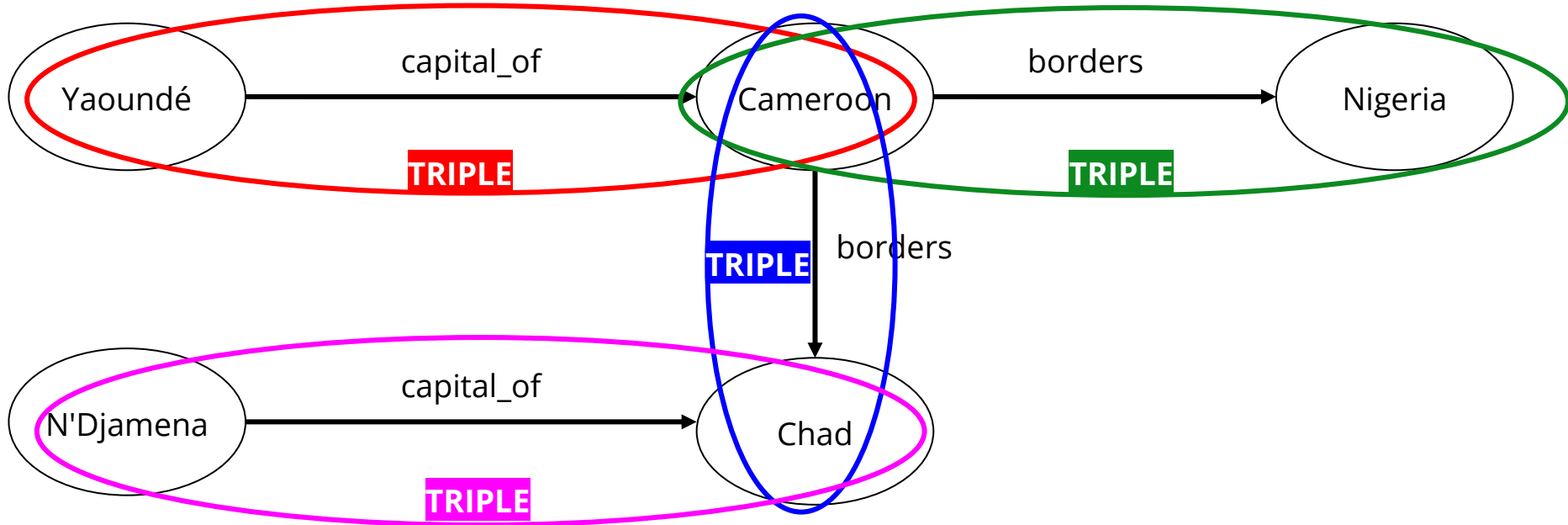
Graph Data Model

One or more triples form a **graph**.



Graph Data Model

One or more triples form a **graph**.



URI-Based Vocabulary

- The **Subject** of an Triple is a Thing, identified with an **URI**
- The **Predicate** of an Triple is a Relationship, identified with an **URI**
- The **Object** of an Triple can be:
 - a Thing, identified with an **URI**
 - a Value, expressed as a **Literal**

Literals

Literals are used to identify values by means of a lexical representation.

A **Literal** can be:

- A **String**: “Yaoundé” (written between double quotes!)
- An **Integer**: 2700000 (a whole number: positive, negative, or 0)
- A **Float**: 3.14 (a number with decimal values)

PUBLIC SERVICE ANNOUNCEMENT:

OUR DIFFERENT WAYS OF WRITING DATES AS NUMBERS CAN LEAD TO ONLINE CONFUSION. THAT'S WHY IN 1988 ISO SET A GLOBAL STANDARD NUMERIC DATE FORMAT.

THIS IS *THE* CORRECT WAY TO WRITE NUMERIC DATES:

2013-02-27


THE FOLLOWING FORMATS ARE THEREFORE DISCOURAGED:

02/27/2013 02/27/13 27/02/2013 27/02/13

20130227 2013.02.27 27.02.13 27-02-13

27.2.13 2013.II.27. $27\frac{1}{2}$ -13 2013.158904109

MMXIII-II-XXVII MMXIII $\frac{\text{LVII}}{\text{CCCLXV}}$ 1330300800

$((3+3) \times (111+1) - 1) \times 3 / 3 - 1 / 3^3$ ~~2013~~  Hissss

10/11011/1101 02/27/20/13 $\begin{matrix} 2 & 3 & 1 & 4 \\ 0 & 1 & 2 & 3 & 7 \\ 5 & 6 & 7 & 8 \end{matrix}$

Literals: Dates

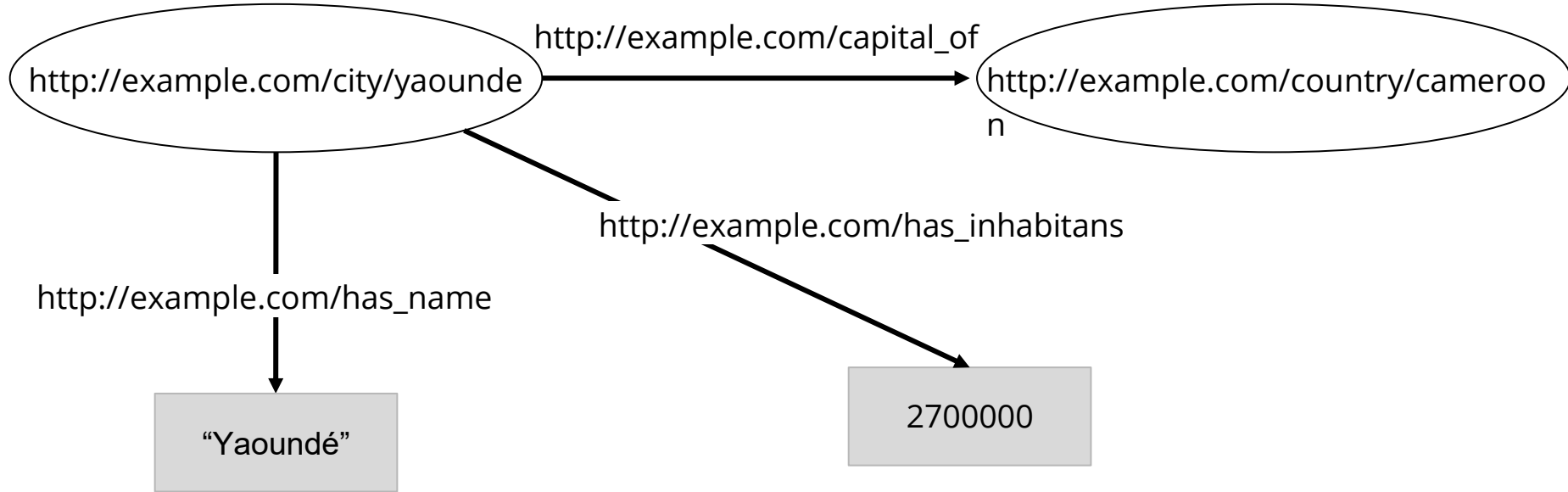
Dates are recorded as **String** following the **ISO 8601 Standard**:

“yyyy”: year (“1961”)

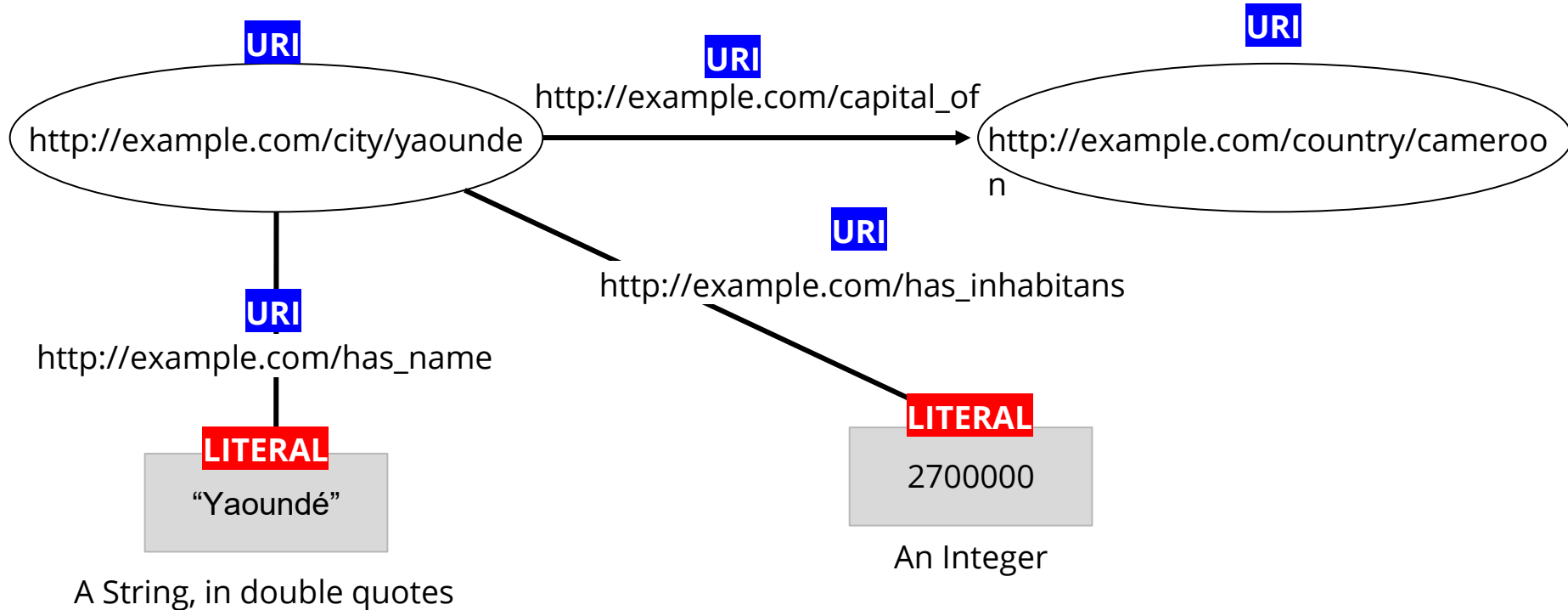
“yyyy-mm”: year and month (“1961-10”)

“yyyy-mm-dd”: year, month, and day (“1961-10-01”)

RDF Graph



RDF Graph



Semantic Web

Standards

2. Turtle

Turtle - Terse RDF Triple Language

Turtle (.ttl) is a format for writing RDF graphs.

It is an **open format** → it does not require specific software to be opened and compiled. Any generic text editor can open a .ttl file (i.e., TextEdit for Mac, Wordpad for Windows)

We have prepared an online .ttl file editor for our course:

<https://ttleditor.onrender.com/>

Turtle Syntax - URIs

URIs are written between angle brackets.

`<http://example.com/city/yaounde>`

`<http://example.com/country/cameroon>`

`<http://example.com/capital_of>`

Turtle Syntax - Literals

A **String**: "Yaoundé" (written between double quotes!)

An **Integer**: 2700000 (a whole number: positive, negative, or 0)

A **Float**: 3.14 (a number with decimal values)

A **Date**: "1961-10-01" (format **yyyy-mm-dd**)

Turtle Syntax - Triple

Triples are a sequence of subject, predicate, object terms, separated by whitespace and terminated by period (.)

`<http://example.com/city/yaounde> <http://example.com/capital_of> <http://example.com/country/cameroon> .`

`<http://example.com/city/yaounde> <http://example.com/has_name> "Yaoundé" .`

`<http://example.com/city/yaounde> <http://example.com/has_inhabitans> 2700000 .`

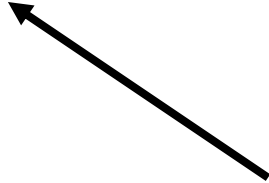
Turtle Syntax - Triple

Triples are a sequence of subject, predicate, object terms, separated by whitespace and terminated by period (.)

`<<http://example.com/city/yaounde> <http://example.com/capital_of> <http://example.com/country/cameroon> .`

`<http://example.com/city/yaounde> <http://example.com/has_name> "Yaoundé" .`

`<http://example.com/city/yaounde> <http://example.com/has_inhabitans> 2700000 .`



Mind the period!

Turtle Syntax - Multiple Statements

Should we have multiple statements (triples) concerning the same subject, we can avoid repeating the subject by dividing each triple with a semi period (;), except the last one, ending with a period.

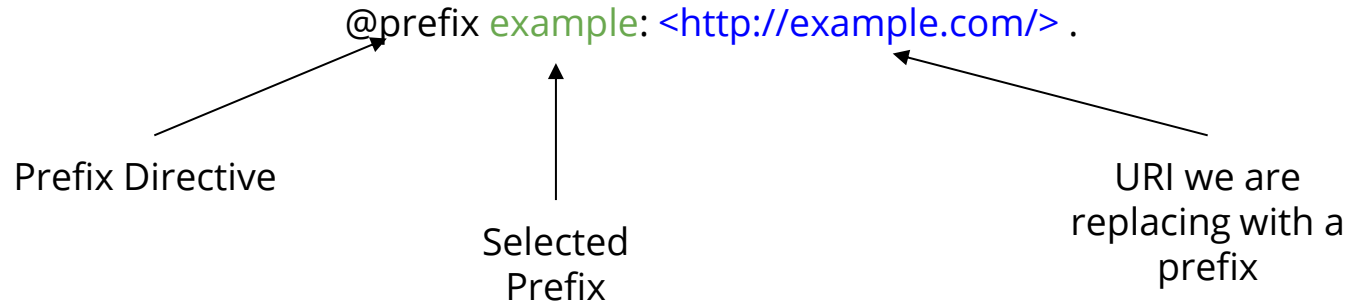
```
<http://example.com/city/yaounde> <http://example.com/capital_of> <http://example.com/country/cameroon> ;  
    <http://example.com/has_name> "Yaoundé" ;  
    <http://example.com/has_inhabitans> 2700000 .
```



Indenting the lines (leaving extra space) following the first triple is not mandatory, but it improves readability.

Turtle Syntax - URI abbreviation

URIs can be abbreviated by using @prefix directive that allows declaring a short prefix name for a long prefix of repeated URIs.



Turtle Syntax - URI abbreviation

URIs can be abbreviated by using @prefix directive that allows declaring a short prefix name for a long prefix of repeated URIs.

```
@prefix example: <http://example.com/> .
```

```
<http://example.com/city/yaounde> example:capital_of <http://example.com/country/cameroon> ;
```

```
example:has_name "Yaoundé" ;
```

```
example:has_inhabitans 2700000 .
```

It is good practice to state all prefixes at the beginning of the document.

Semantic Web

Standards

3. Ontologies

Ontologies for Classes and Properties

An ontology is a set of URIs for classes and properties, semantically and logically defined.

To create an ontology, we use the properties and classes of the RDF and RDFS (RDF Schema) ontologies.

RDF Ontology

RDF provides basic elements for the description of RDF data:

We can use a prefix → @prefix **rdf:** <<http://www.w3.org/1999/02/22-rdf-syntax-ns>> .

rdf:Property → the class that defines that a URI is a property

rdf:type → the property for assigning a class to an URI

Since **rdf:type** is the most common property, in Turtle we have a beautiful shortcut → a

<<http://example.com/city/yaounde>> a <<http://example.com/city>> .

RDFS: RDF Schema

RDFS provides elements for the description of ontologies:

We can use a prefix → @prefix **rdfs:** <<https://www.w3.org/2000/01/rdf-schema#>> .

rdfs:Class → the class that defines that a URI is a class

rdfs:Literal → the class representing a literal

rdfs:label → a property for labelling an URI with a string

rdfs:comment → a property for describing an URI with a string

Defining a Class

@prefix **rdfs**: <<https://www.w3.org/2000/01/rdf-schema#>> .

<<http://example.com/City>> a **rdfs**:Class ;

rdfs:label "City" ;

rdfs:comment "A permanent settled place with administratively defined boundaries." .

Defining a Class

@prefix **rdfs**: <<https://www.w3.org/2000/01/rdf-schema#>> .

<<http://example.com/City>> a **rdfs**:Class ;

rdfs:label "City" ;

rdfs:comment "A permanent settled place with administratively defined boundaries." .

! It is good practice to write the class name in URIs with the first letter capitalized → <<http://example.com/City>>

Ontologies: Classes

Classes are URIs used to classify things.

In describing one class, an ontology defines its relationship to others.

One class can be a **subclass** of another.

For example, an ontology may define that, the class City is a subclass of the class Place. Both Yaoundé and Cameroon are a Place, but only Yaoundé is a City.

RDFS: RDF Schema

RDFS provides elements for the description of ontologies:

`rdfs:subClassOf` → the property that defines that a class is subclass of another class

Defining a Class

@prefix **rdfs**: <<https://www.w3.org/2000/01/rdf-schema#>> .

<<http://example.com/City>> a **rdfs**:Class ;

rdfs:label "City" ;

rdfs:comment "A permanent settled place with administratively defined boundaries." ;

rdfs:subClassOf <<http://example.com/Place>> .

Ontology: Properties

As we have seen, properties are used to construct triples. We can imagine them as attributes (or even verbs sometimes).

A property can be **subproperty** of another.

An ontology can define that "capital of" is, for example, a subproperty of "city of." Both Yaoundé and Douala are cities of Cameroon. But only Yaoundé is the capital of Cameroon.

RDFS: RDF Schema

RDFS provides elements for the description of ontologies:

`rdfs:subPropertyOf` → the property that defines that a property is subproperty of another property

Defining a Property

@prefix **rdfs**: <<https://www.w3.org/2000/01/rdf-schema#>> .

@prefix **rdf**: <<http://www.w3.org/1999/02/22-rdf-syntax-ns>> .

<http://example.com/capital_of> a **rdf**:Property ;

rdfs:label "is capital of" ;

rdfs:comment "The relationship between a city and a country as its capital city." ;

rdfs:subPropertyOf <http://example.com/city_of> .

Defining a Property

@prefix **rdfs**: <<https://www.w3.org/2000/01/rdf-schema#>> .

@prefix **rdf**: <<http://www.w3.org/1999/02/22-rdf-syntax-ns>> .

<http://example.com/capital_of> a **rdf**:Property ;

rdfs:label "is capital of" ;

rdfs:comment "The relationship between a city and a country as its capital city." ;

rdfs:subPropertyOf <http://example.com/city_of> .

! It is good practice to write the property name in URIs in lower case → <http://example.com/capital_of>

Ontology: Properties

Finally, an ontology defines the **domains** and **ranges** of properties.

The **domain** indicates which class the **subject** should belong in the triple that uses a property. For example, in an ontology, the domain of a property such as "capital of" is the class "City." Only a city can be capital of a country.

Instead, the **range** defines which class the **object** should belong in the triple that uses a property. For example, in an ontology, the range of a property such as "capital of" is the class "Country." A thing can only be capital of a country.

RDFS: RDF Schema

RDFS provides elements for the description of ontologies:

`rdfs:domain` → the property that defines the domain of a property

`rdfs:range` → the property that defines the range of a property

Defining a Property

@prefix **rdfs**: <https://www.w3.org/2000/01/rdf-schema#> .

@prefix **rdf**: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .

<http://example.com/capital_of> a **rdf**:Property ;

rdfs:label "is capital of" ;

rdfs:comment "The relationship between a city and a country as its capital city." ;

rdfs:subPropertyOf <http://example.com/city_of> ;

rdfs:domain <http://example.com/City> ;

rdfs:range <http://example.com/Country> .

Ontologies for Classes and Properties

Whether a class is domain or range of a property. All of its subclasses inherit this characteristic.

If "has coordinates" has as domain the class Place, then all subclasses of Place, such as City or Country, are domains of the property.

Ranges

If a property has a **Class** as its range, it is called an Object Property.

If a property has a **Literal** as its range, it is called a Data Property.

Defining a Data Property

@prefix **rdfs**: <<https://www.w3.org/2000/01/rdf-schema#>> .

@prefix **rdf**: <<http://www.w3.org/1999/02/22-rdf-syntax-ns>> .

<http://example.com/has_name> a **rdf**:Property ;

rdfs:label "has name" ;

rdfs:comment "The property defining the name of a thing." ;

rdfs:domain <<http://example.com/Thing>> ;

rdfs:range **rdfs**:Literal .

Defining a Data Property

@prefix **rdfs**: <https://www.w3.org/2000/01/rdf-schema#> .

@prefix **rdf**: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .

<http://example.com/has_name> a **rdf**:Property ;

rdfs:label "has name" ;

rdfs:comment "The property defining the name of a thing." ;

rdfs:domain <http://example.com/Thing> ;

rdfs:range **rdfs**:Literal .

← The object of a triple using this property
can be only a Literal (NO URI)