Mastering maplib

Veronika Heimsbakk

Mastering maplib

Veronika Heimsbakk

Knowledge Graph Specialist | Data Treehouse

DATA TREEHOUSE

veronika@data-treehouse.com

veleda
vheimsbakk
veronahe.no

Agenda

DATA TREEHOUSE
MAPLIB
DATA
ONTOLOGIES
MAPPING
DATALOG
SPARQL
SHACL

DataTreehouse/maplib-masterclass

DATA
TREEHOUSE

DATA TREEHOUSE

Data Treehouse is a Norwegian knowledge graph solutions software company. Our technology is based upon Magnus Bakken's PhD work.

Magnus Bakken
magnusbakken

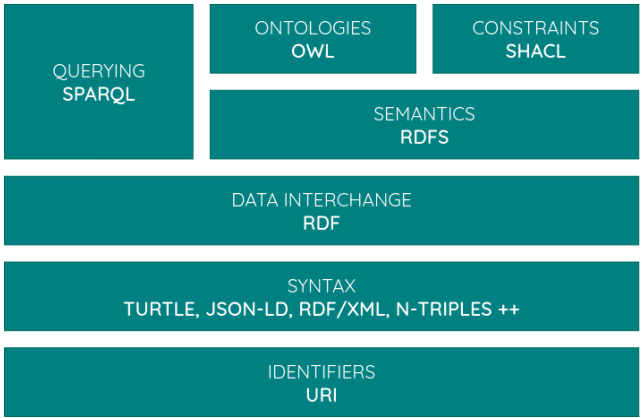
Øivind Rui
øivind-rui-a720492

Veronika Heimsbakk
vheimsbakk

We develop and maintain a series of open source[!] frameworks for Python.

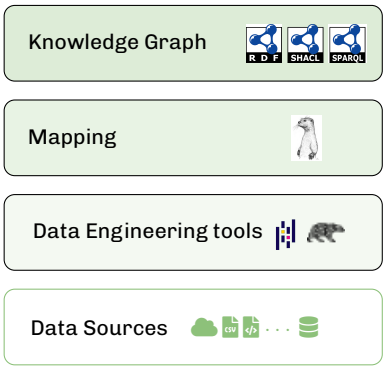
- **maplib** build and enrich harmonized knowledge graphs from any sources.
- **chrontext**: semantic integration technology that protect your existing platforms and infrastructure while leveraging the value of data.
- **querymesh**: context-enabled queries over analytical data sets provides the first level of querymesh, the Operational Data Mesh, where data products are created, maintained and managed.

[!] Some functionalities are under licensing.


QUERYING
SPARQL
ONTOLOGIES
OWL
CONSTRAINTS
SHACL
SEMANTICS
RDFS
DATA INTERCHANGE
RDF
SYNTAX
TURTLE, JSON-LD, RDF/XML, N-TRIPLES ++
IDENTIFIERS
URI

maplib at a glance

maplib is developed in Rust on *Polars* and *Apache Arrow*, and available as Python frameworks.



Knowledge Graph Support for most knowledge graph operations, as RDF Model, SPARQL, SHACL, and Datalog reasoning.

Mapping Template based mapping using Reasonable Ontology Templates (OTTR). Serialise data frames to RDF in no time!

Data Engineering tools maplib supports instance data handling with *data frames* in Python by core.

Data Sources data bist data.

Check out: `https://datatreehouse.github.io/maplib`

DATA

	A	B	C	D	E	F	G
1	planet	name	gm	radius	density	magnitude	albedo
2	Earth	Moon	4902.801±0.001	1737.5±0.1	3.344±0.005	-12.74	0.12
3	Mars	Phobos	0.0007112±0.000001	11.1±0.15	1.872±0.076	11.4±0.2	0.071±0.012
4	Mars	Deimos	0.0000985±0.000002	6.2±0.18	1.471±0.166	12.45±0.05	0.068±0.007
5	Jupiter	Io	5959.916±0.012	1821.6±0.5	3.528±0.006	5.02±0.03	0.63±0.02
6	Jupiter	Europa	3202.739±0.009	1560.8±0.5	3.013±0.005	5.29±0.02	0.67±0.03
7	Jupiter	Ganymede	9887.834±0.017	2631.2±1.7	1.942±0.005	4.61±0.03	0.43±0.02
8	Jupiter	Callisto	7179.289±0.013	2410.3±1.5	1.834±0.004	5.65±0.10	0.17±0.02
9	Jupiter	Amalthea	0.138±0.030	83.45±2.4	0.849±0.199	14.1±0.2	0.090±0.005
10	Jupiter	Himalia	0.45	85	2.6	14.2R	0.04
11	Jupiter	Elara	0.058	43	2.6	16.0R	0.04
12	Jupiter	Pasiphae	0.020	30	2.6	16.8R	0.04
13	Jupiter	Sinope	0.0050	19	2.6	18.2R	0.04

	A	B	C	D	E	F	G	H	I	J
1	planet	mass	diameter	density	gravity	escape_velocity	rotation_period	length_of_day	distance_from_sun	perihelic
2	Mercury	0.330	4879	5427	3.7	4.3	1407.6	4222.6	57.9	46.0
3	Venus	4.87	12104	5243	8.9	10.4	-5832.5	2802.0	108.2	107.5
4	Earth	5.97	12756	5514	9.8	11.2	23.9	24.0	149.6	147.1
5	Mars	0.642	6792	3933	3.7	5.0	24.6	24.7	227.9	206.6
6	Jupiter	1898	142984	1326	23.1	59.5	9.9	9.9	778.6	740.5
7	Saturn	568	120536	687	9.0	35.5	10.7	10.7	1433.5	1352.6
8	Uranus	86.8	51118	1271	8.7	21.3	-17.2	17.2	2872.5	2741.3
9	Neptune	102	49528	1638	11.0	23.5	16.1	16.1	4495.1	4444.5
10	Pluto	0.0146	2370	2095	0.7	1.3	-153.3	153.3	5906.4	4436.8

devstronomy/nasa-data-scraper

DATA ENGINEERS ❤ DATAFRAMES — AND SO DOES MAPLIB 

Data frames

https://pola.rs/

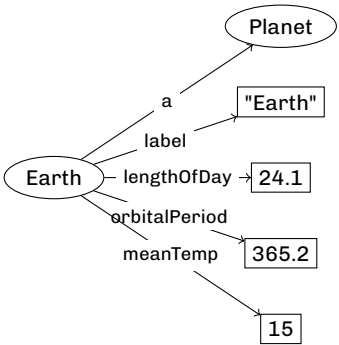
```
df_planets = pl.read_csv("data/planets.csv")

# Create URI for subject
df_planets = df_planets.with_columns(
  (ns + pl.col("planet")).alias("planet_uri")
)
```

Data frames

```
df_planets = df_planets.select(
  ["planet",
  "planet_uri", # the new column we just made :-)
  "mean_temperature",
  "length_of_day",
  "orbital_period"]
)
```

Planets


Planet
a
"Earth"
label
Earth
lengthOfDay
24.1
orbitalPeriod
meanTemp
365.2
15

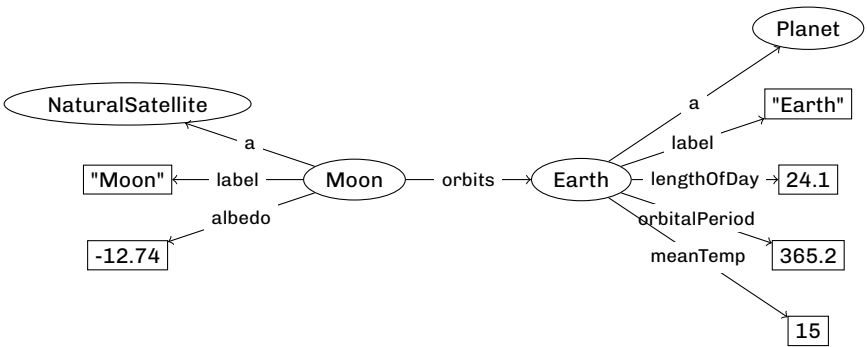
Data frames—satellites

```
df_satellites = pl.read_csv("data/satellites.csv")

df_satellites = df_satellites.with_columns(
  (ns + pl.col("planet")).alias("planet_uri")
)

df_satellites = df_satellites.with_columns(
  (ns + pl.col("name")
  .str.replace_all("/", "-"))
  .str.replace_all(" ", "-")
  .alias("satellite_uri")
)
```

...with their satellites



MAPPING

Reasonable Ontology Templates

maplib currently support mapping using OTTR[1]

Reasonable Ontology Templates (OTTR)
Aims to provide lightweight syntax for defining templates for RDF.
Serialising chunks of data in a defined pattern. `https://ottr.xyz/`

We're diving into stOTTR (Terse Syntax for Reasonable Ontology Template):

- terms
- types
- instances
- template example

[1] *On the roadmap:* RML and SPARQL Anything.

stOTTR terms

A term is, syntactically, either a variable, a constant or a list of terms.

```
?planet_uri
?PLANET

<http://data-treehouse.com/example>
ex:mean_temperature

[]
_:b

"Jupiter"
"4879"^^xsd:integer
57.9
5427
true

("Phobos", "Deimos")
(("Phobos", "Deimos"), ex:Mars, (0.642))
```

stOTTR types

A type, like the type of a term, is either a basic type, or a list type.

```
xsd:double
owl:Class
rdfs:Resource
ottr:IRI

List<xsd:string>
List<NEList<xsd:integer>>
```

stOTTR instances

stOTTR instances are instructions on how instance data will be serialised. We populate template signatures with actual data.

```
ottr:Triple(:subject, :predicate, :object) .
tpl:Satellite(ex:Mars, ex:Phobos) .
tpl:Star( , ) .
cross | tpl:Planet(ex:Mercury, ++("Merkur"@no, "Mercury"@en)) .

```

stOTTR template example

Remember our `df_planet` data frame, with selected columns: `planet`, `planet_uri`, `mean_temperature`, `length_of_day`, and `orbital_period`. **Template parameters must have the same name as data frame column headers.**

```
tpl:Planet[
  ! ottr:IRI ?planet_uri ,
  xsd:string ?planet ,
  ?mean_temperature ,
  ? ?length_of_day ,
  ?orbital_period
] :: {
  ottr:Triple(?planet_uri, rdf:type, :Planet),
  ottr:Triple(?planet_uri, rdf:type, owl:NamedIndividual),
  ottr:Triple(?planet_uri, rdfs:label, ?planet),
  ottr:Triple(?planet_uri, :meanTemperature, ?mean_temperature),
  ottr:Triple(?planet_uri, :lengthOfDay, ?length_of_day),
  ottr:Triple(?planet_uri, :orbitalPeriod, ?orbital_period)
} .
```

Need to kick-start your template?

With **maplib**, you can generate a stOTTR template based on your data frame. A perfect starting point for kicking off your template, so that you can focus on fine-tuning instead of layout.

```
tmp_tpl = m.expand_default(df_planets, "planet_uri")
```

`m.expand_default` takes the data frame, and the column containing subject URI as parameters, and returns a string containing the stOTTR template.

Mapping your data frame using OTTR

At core in **maplib**, is the mapping (surprise!).

```
with open("tpl.ttl", "r") as file:
  tpl = file.read()

# Init a model, and add template
m = Model()
m.add_template(tpl)

# Mapping: serialise data frame to RDF using templates
m.map("http://data.treehouse.example/tpl/Planet", df_planets())
```

- `m.add_template()` takes in the template as string or programmatically constructed `Template`.
- `m.map()` takes in template URI and the data frame to be serialised.

Results of mapping

Serialisation

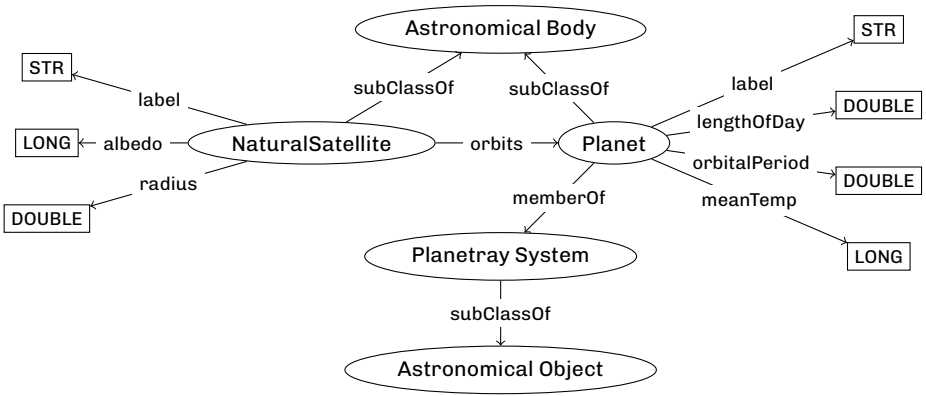
Using `m.write(output_file, format)` one can serialise `m` and write to file.
Supported formats: `"ntriples"`, `"turtle"`, and `"rdf/xml"`[1]

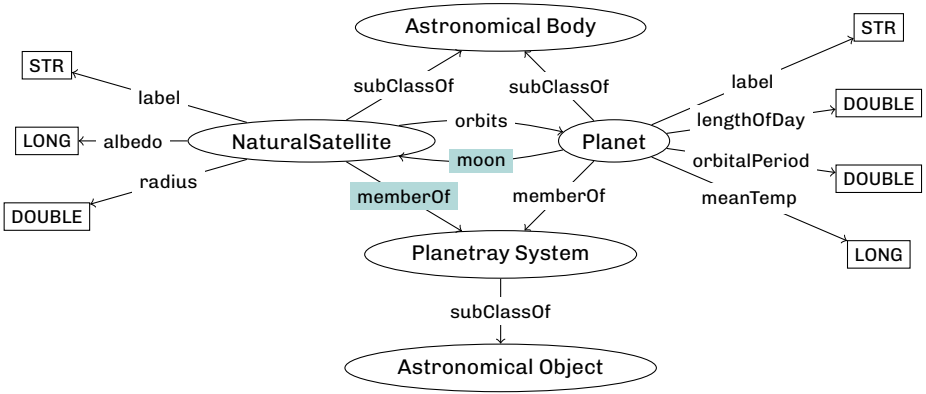
```
dt:Jupiter a dt:Planet,
  owl:NamedIndividual ;
rdfs:label "Jupiter" ;
dt:lengthOfDay 9.9e+00 ;
dt:meanTemperature "-110"^^xsd:long ;
dt:orbitalPeriod 4.331e+03 .

```

[1] *On the roadmap:* JSON-LD, pretty ttl

ONTOLOGY


Astronomical Body
STR
label
subClassOf
subClassOf
label
STR
LONG
albedo
NaturalSatellite
orbits
Planet
lengthOfDay
DOUBLE
orbitalPeriod
DOUBLE
radius
memberOf
meanTemp
DOUBLE
LONG
Planetray System
subClassOf
Astronomical Object


Astronomical Body
STR
label
subClassOf
subClassOf
label
STR
LONG
albedo
NaturalSatellite
orbits
Planet
lengthOfDay
DOUBLE
moon
orbitalPeriod
DOUBLE
radius
memberOf
memberOf
meanTemp
DOUBLE
LONG
Planetray System
subClassOf
Astronomical Object

SPARQL

Count

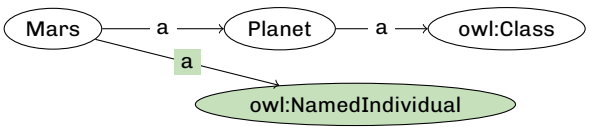
`m.query()` returns a data frame containing the query result. The bound variable `?count` is the column header in this example, and can be fetched using `["count"][0]` for first cell of first row in column `count`.

```
# Print number of triples
count = """SELECT (COUNT(?s) AS ?count) WHERE { ?s ?p ?o . }"""
print("Graph size after mapping: ", m.query(count)["count"][0])
```

m.insert

```
# Insert everything that has some rdf:type to anything except for owl:Class
# to be rdf:type of owl:NamedIndividual

CONSTRUCT {
  ?s rdf:type owl:NamedIndividual .
}
WHERE {
  ?s rdf:type ?o .
  FILTER(?o != owl:Class)
}
```



m.insert

`m.insert()` expect a SPARQL `CONSTRUCT` query as string, and inserts the constructed statements to the `Model`.

```
with open("queries/insert_individual.rq", "r") as file:
  insert_individual = file.read()

m.insert(insert_individual)
```

DATALOG

What is Datalog?

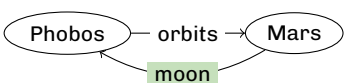
Datalog is a declarative logic programming language. It's a bit similar to Prolog, but resembles more a query language than Prolog does.

In **maplib**, Datalog is used for *logical inference* over the knowledge graph[1].

[1] *On the roadmap:* RDFS eintailment

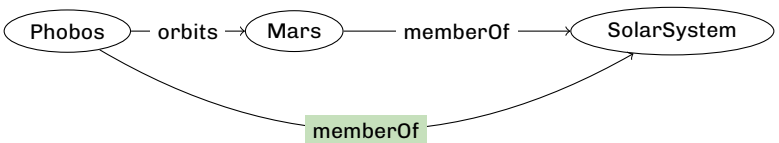
Declaring statements

```
1 [?x, :moon, ?y] :- [?y, :orbits, ?x] .
2
```



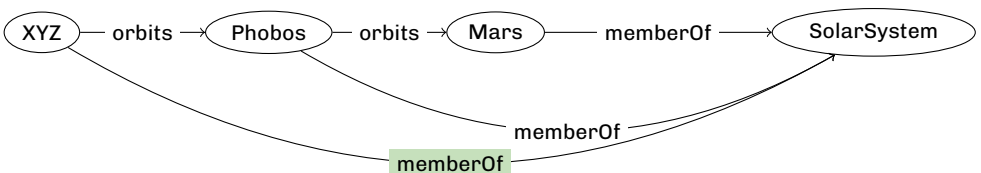
Declaring statements

```
1 [?x, :memberOf, ?z] :- [?x, :orbits, ?y], [?y, :memberOf, ?z] .
2
```



Declaring statements

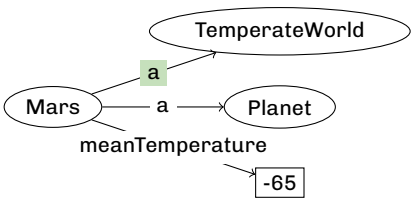
```
1 [?x, :memberOf, ?z] :- [?x, :orbits, ?y], [?y, :memberOf, ?z] .
2
```



Classification

```
[?planet, rdf:type, :TemperateWorld] :-
  [?planet, rdf:type, :Planet] ,
  [?planet, :meanTemperature, ?temp] ,
  FILTER(?temp >= -100 && ?temp <= 100) .
```

All things that are *Temperate World*s are things that are of type `Planet`, and has a `meanTemperature` between `-100` and `100`.



infer in maplib

```
with open("ttl/rule.dlog", "r") as file:
  rules = file.read()

m.infer(rules)
```

Result after reasoning

```
dt:Mars a dt:Planet,
  dt:TemperateWorld, owl:NamedIndividual ; #
  rdfs:label "Mars" ;
  dt:lengthOfDay 2.47e+01 ;
  dt:meanTemperature "-65"^^xsd:long ;
  dt:memberOf dt:SolarSystem ;
  dt:moon dt:Deimos, dt:Phobos ; #
  dt:orbitalPeriod 6.87e+02 .

dt:Deimos a dt:NaturalSatellite, owl:NamedIndividual ;
  rdfs:label "Deimos" ;
  dt:memberOf dt:SolarSystem ; #
  dt:orbits dt:Mars .
```

SHACL

Constraining `:memberOf`

Astronomical bodies shall be a member of either a planetary system or a star cluster.

```
rule:AstronomicalBody a sh:NodeShape ;
  sh:targetClass :AstronomicalBody ;
  sh:property rule:AstronomicalBody-memberOf .

rule:AstronomicalBody-memberOf a sh:PropertyShape ;
  sh:path :memberOf ;
  sh:minCount 1 ;
  sh:or (
      [ sh:class :PlanetarySystem ]
      [ sh:class :StarCluster ]
  ) .

```

TREEHOUSE EXPLORER