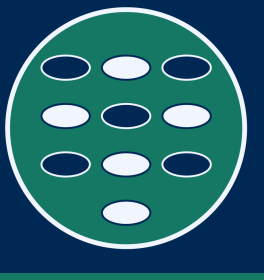
Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap



Digitalisering på Ås

INF203

June advanced programming project

2 Object orientation in practice

2.1 Object orientation
2.2 Encapsulation
2.3 E-R diagrams
2.4 Factories
2.5 Data-code co-design
2.6 Data interoperability

Get prepared: What will be submitted?

Norwegian University of Life Sciences

Submit the final complete project material through Canvas by Saturday, 21st June 2025, 17.30 CEST. This includes:

- Your **complete code** base.
- The **project report** in PDF format. The project report must be written using LaTeX, but do not submit the LaTeX sources, only the PDF.
- The **requirements register** as a spreadsheet in ODS or XLSX format, including columns stating whether the functionality was actually implemented (y/n), and whether it was validated or tested (y/n).
- Any simulation output or data that you would want to share as long as they are not very large. **Do not include any extremely large files.**

The above should be uploaded on Canvas in the form of a single zip (or tar.bz2 or tar.gz) archive.

Style guidelines for the report

1) The report's purpose is to help the examiners grade your submission.
2) It must be written in LaTeX – any style will be good. Any length in number of pages is allowed. The PDF must be less than 4 MB in size.
3) **Be concise.**
4) The examiners will read the code and the requirements register. But the report can help the examiners figure out where to look. So it helps to include pointers to a method and source filename, or to a labelled element of the requirements register when you are referring to it.
5) Do not include any code fragments longer than seven lines of code. Better explain yourselves using pseudocode or diagrams if you want to explain any parts of code that are more complex.

Style guidelines for the report

6) This is not an academic text, and it does not matter if the English is not perfect. Please don't run it through an LLM for language polishing.

7) You do need to cite *academic references* if you use them, *e.g.* when comparing to *literature data*, or when implementing *methods that we did not discuss* in the lecture. References are *not needed* for the *methods that we discussed in the lecture*, such as the test area method.

8) Visualize your simulation results using diagrams. These should follow the usual common-sense principles for good use of diagrams.

9) Do not include any screenshots.

10) The external examiner does not speak Norwegian, and the appeals examiner might not speak Norwegian. To make yourselves understood, all of the comments, requirements, and report must be English.

Suggested structure for the report

I. Introduction

(a) Group composition and responsibilities. **(b)** Shared repository and tools for collaboration. **(c)** Agile methodology and requirements analysis. **(d)** User guide.

II. Implementation

(a) Code and class structure. **(b)** Commenting. **(c)** Advanced OOP. **(d)** Python package.

III. Functionality and validation

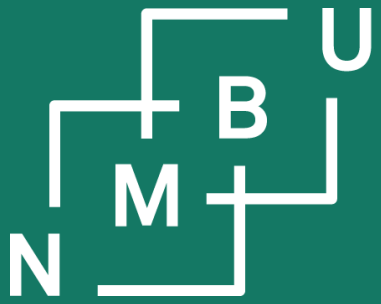
(a) Testing and validation strategy. **(b)** Initial configuration and I/O. **(c)** Pair potential and Monte Carlo method. **(d)** Test area method and surface tension calculation.

IV. Sampling and data analysis

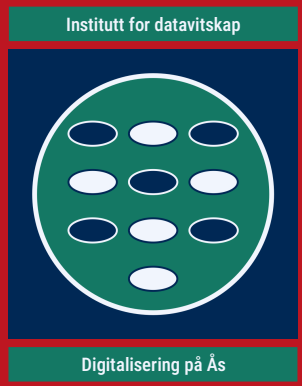
(a) Equilibration. **(b)** Uncertainty. **(c)** Console output. **(d)** Data processing for diagrams.

V. Simulation results

(a) Vapour-only reference system. **(b)** Vapour-liquid reference system. **(c)** Any other results. **(d)** Comparison to results from the literature.



Noregs miljø- og biovitskaplege universitet



2 OOP in practice

2.5 Data-code co-design

2.6 Data interoperability

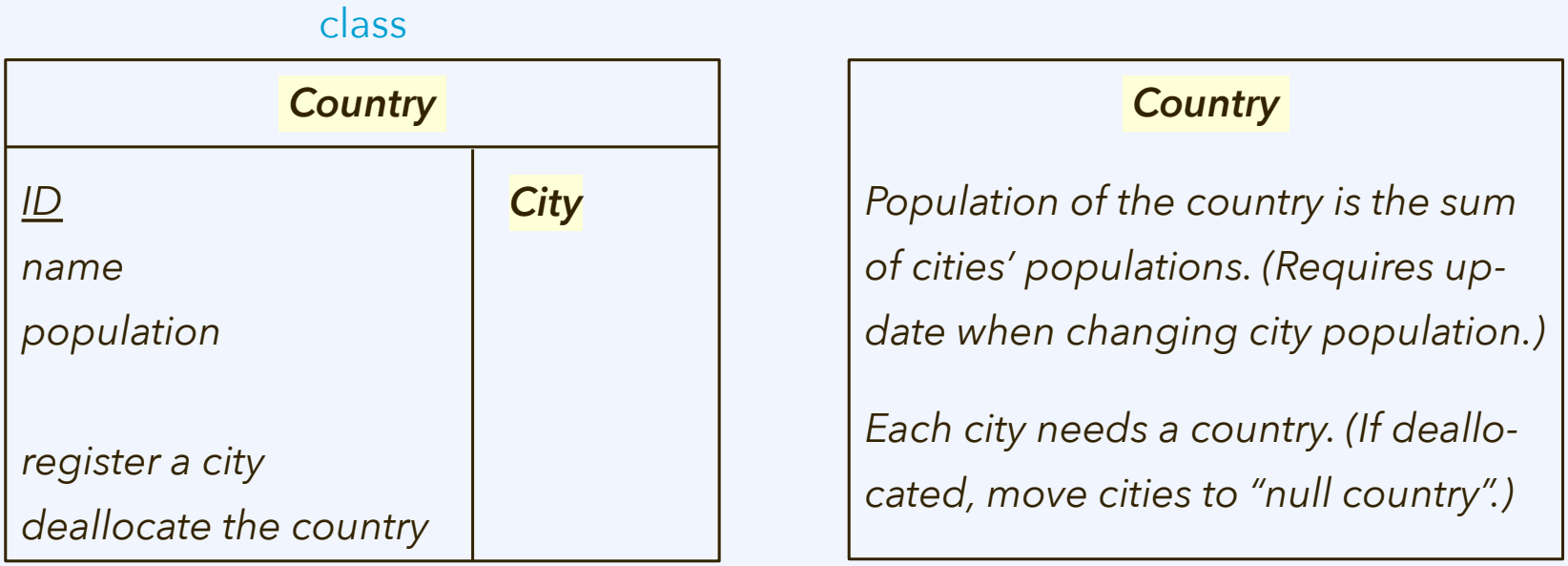
INF203

10th June 2025

CRC cards: Class – responsibilities – collaborations

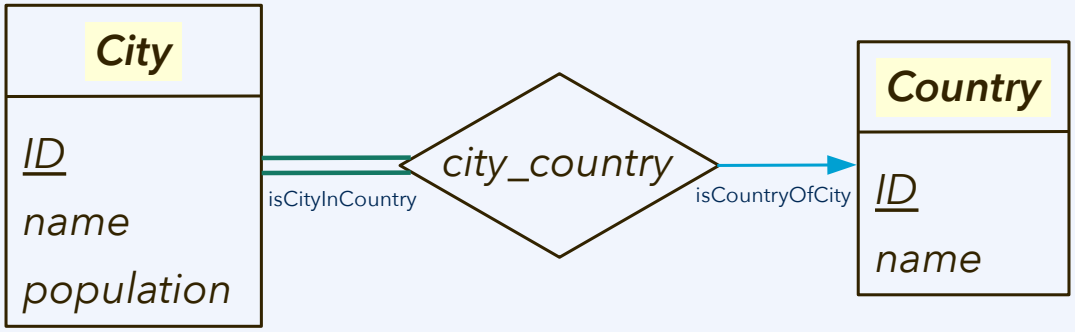
In structured or procedural programming, it is comparably easy to analyse a code even on a line-by-line basis and make statements about the execution state. These can take the form of **invariants** (conditions that are always met).

Classes in object oriented programming have a much more flexible control flow, as their methods can be called at any time and in any order. It can help to make the invariants explicit during design, *e.g.*, using **CRC cards** as a tool.



Designing classes: Entity-relationship diagrams

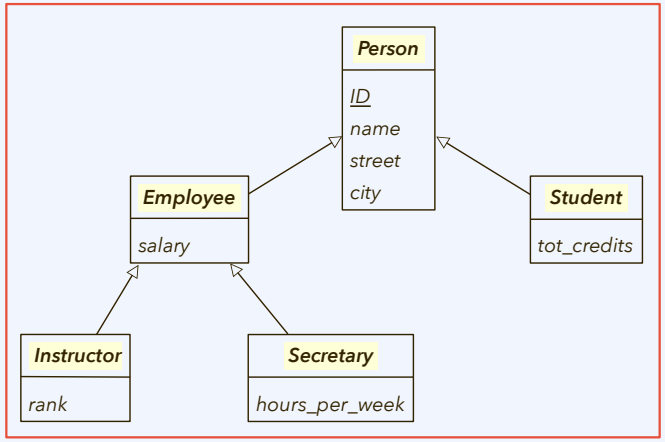
particular:	**object** (entity, individual)	**relationship**	**property** (sometimes: attribute)
universal:	**class** (entity type, concept)	**relation** (relationship type) (in OWL: ObjectProperty)	**attribute** (sometimes: attribute type) (in OWL: DatatypeProperty)



"every City is in such a relationship"

"it is an *N*-to-1 relation from Cities to Countries"

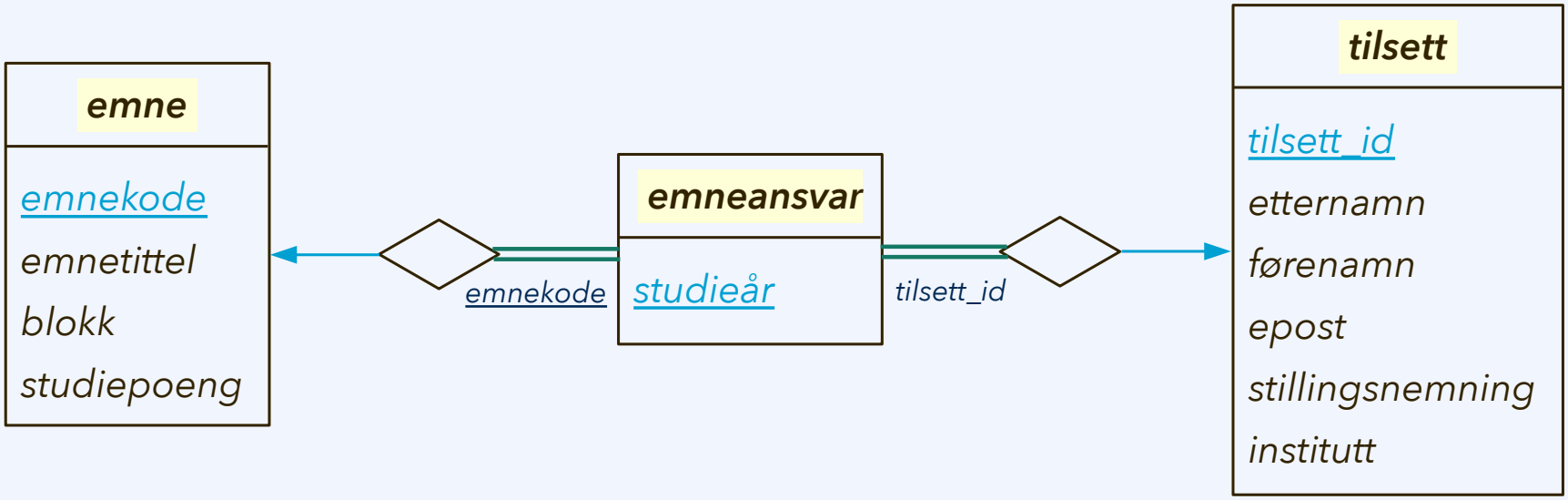
This was also an entity-relationship diagram:



More on entity-relationship diagrams:

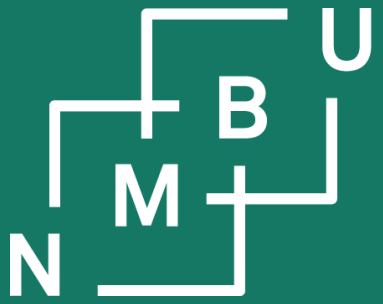
- Silberschatz *et al.*, *Database System Concepts*, Chapter 6
- https://en.wikipedia.org/wiki/Entity-relationship_model

Difference between E-R use in databases and OOP

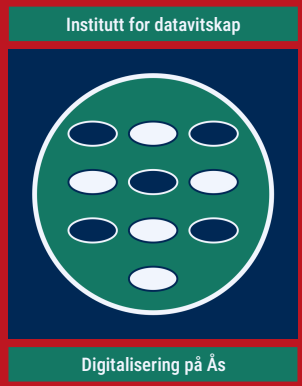


In a *database schema*, objects are identified through *primary keys* (IDs), and their relationship to others are give by *foreign keys*, *i.e.*, other objects' IDs. For example, above, an "emneansvar" entry contains a "tilsett_id" number.

In OOP, we *can* do this the same way, with a registry of objects, *e.g.* using a Python dictionary. But normally, that would be poor style. The standard way of doing this is: Holding an object reference to the object as an attribute.



Noregs miljø- og biovitskaplege universitet



2 OOP in practice

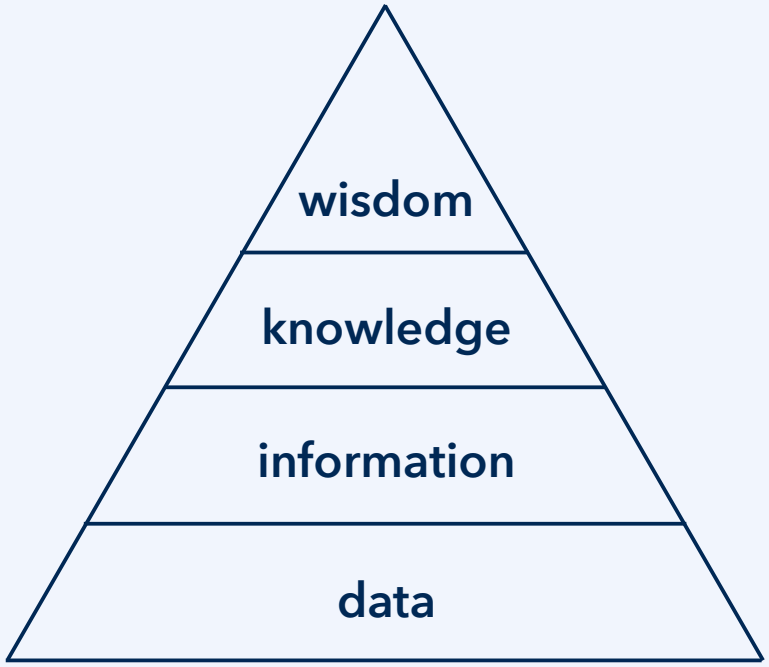
2.5 Data-code co-design

2.6 Data interoperability

INF203

10th June 2025

DIKW: Data, information, knowledge, and wisdom



Hierarchy of data, information, knowledge, and wisdom (**DIKW pyramid**[1, 2]).

[1] J. Rowley, *J. Inform. Sys.* **33**: 163–180, doi:10.1177/0165551506070706, **2009**.

[2] B. Schembera, in *Proc. DCLXVI 2024*, *pp.* 122–132, Springer, doi:10.1007/978-3-031-89274-5_9, **2025**.

Pragmatic competency and interoperability, including agreed good practices.

Epistemic metadata documentation: Establish the knowledge status.

Semantic interoperability: Data become information if their meaning is agreed.

Syntactic interoperability: Data exchanged in an agreed format.

Basic file I/O in Python

File objects are used for I/O from and to files. Opening:

f_obj = **open(**filename_init, **'r')** or f_obj = **open(**filename_init, **'w')** *etc.*

We can write into a file just the same way as to the command line:

print("Value of x:", x, sep='\t', end='\n', **file=**f_obj**)**

The most direct way of doing file input is to read the whole content as a string:

str = f_obj.**read()** # now str is the file's whole content as a string

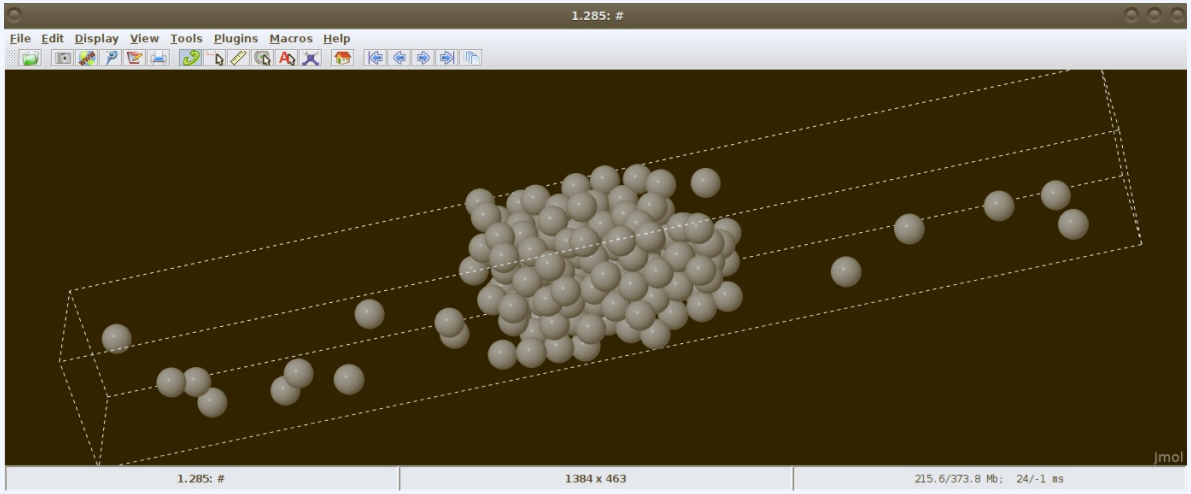
Close the file object once you are done: f_obj.**close()**

It is of course also possible to use libraries for I/O in various formats.

XYZ format for molecular configurations/trajectories

There is not much to say about the **XYZ format**. It is an easy-to-write format for configurations and trajectories (sequences of configurations) with one line per atom. See *e.g.* on Wikipedia: https://en.wikipedia.org/wiki/XYZ_file_format

The JMol tool,[1] among others, can be used to animate the trajectories.



You can use this to visually detect major problems with your simulation run. This is also a standard (if unelegant) format for exchanging trajectories.

[1]https://jmol.sourceforge.net/

JSON in digital infrastructures

JSON is often used for data ingest & extraction into/from DBs via RESTful APIs:

- JSON is a hierarchical format, in which one element can *contain* other elements; in this sense it is equivalent to XML, but with less overhead.
- JSON can be used as a type in *relational databases* including MySQL,[1] *i.e.*, JSON formatted data can be ingested without transformation.
- It is also used in *non-relational DBs*; *e.g.*, MongoDB is based on JSON.[2]
- The hierarchical structure – *transitivity* of the containment relation – limits the way in which objects can be connected to each other (*trees* only).
- Data in a JSON file are self-contained, there is no standard way to include *external resources*; except via "JSON linked data" (JSON-LD).

[1] https://dev.mysql.com/doc/refman/8.0/en/json.html

[2] https://docs.mongodb.com/guides/server/introduction/

JSON example input file

```
{
    "setup": {
        "Lx": 5,
        "Ly": 40,
        "Lz": 5,
        "compartments": [
            {
                "density": 0.02,
                "volume_fraction": 0.4
            }, {
                "density": 0.73,
                "volume_fraction": 0.2
            }, {
                "density": 0.02,
                "volume_fraction": 0.4
            }
        ]
    },
    "steps": {
        "total": 100000,
        "reset_sampling_at": [40, 10000, 20000]
    },
    "console_output": {
        "frequency": 50
    },
    …
    "configuration_output": {
        "initial": "data/vle-0.80-reference_config_init.xyz",
        "final": "data/vle-0.80-reference_config_final.xyz"
    },
    "trajectory_output": {
        "frequency": 200,
        "file": "data/vle-0.80-reference_trajectory.xyz"
    },
    "control_parameters": {
        "maximum_displacement": 0.125,
        "test_area_distortion": [
            {
                "sx": 0.999995,
                "sy": 1.00001,
                "sz": 0.999995
            }, {
                "sx": 1.000005,
                "sy": 0.99999,
                "sz": 1.000005
            }
        ]
    }
}
```

Loading structured data in JSON format

Explained here: https://docs.python.org/3/library/json.html

Just loading data from a JSON file is extremely simple, *e.g.*:

```
import json
jsonfile = open('vle-0.80-reference.json', 'r')
content = json.load(jsonfile)
jsonfile.close()
```

Now, content is a dictionary containing all the data from the JSON example.

```
print(content['console_output']['frequency'])
```

The above command will print the corresponding data item, which is 50.

Resource description framework (RDF)

Norwegian University of Life Sciences

Semantic technology can facilitate the integration of data and software into a coherent framework, permitting multiple components to become interoperable.

On the semantic web, data and metadata are provided as RDF triples:

Triples: Individual Relation Individual. (Subject Predicate Object.)

Example: The fox *f* *eats* the chicken *c*.

(Other kind of triples: Individual "a" Concept. Example: *f* a Fox.)

RDF is the Resource Description Framework, which specifies the semantic web.

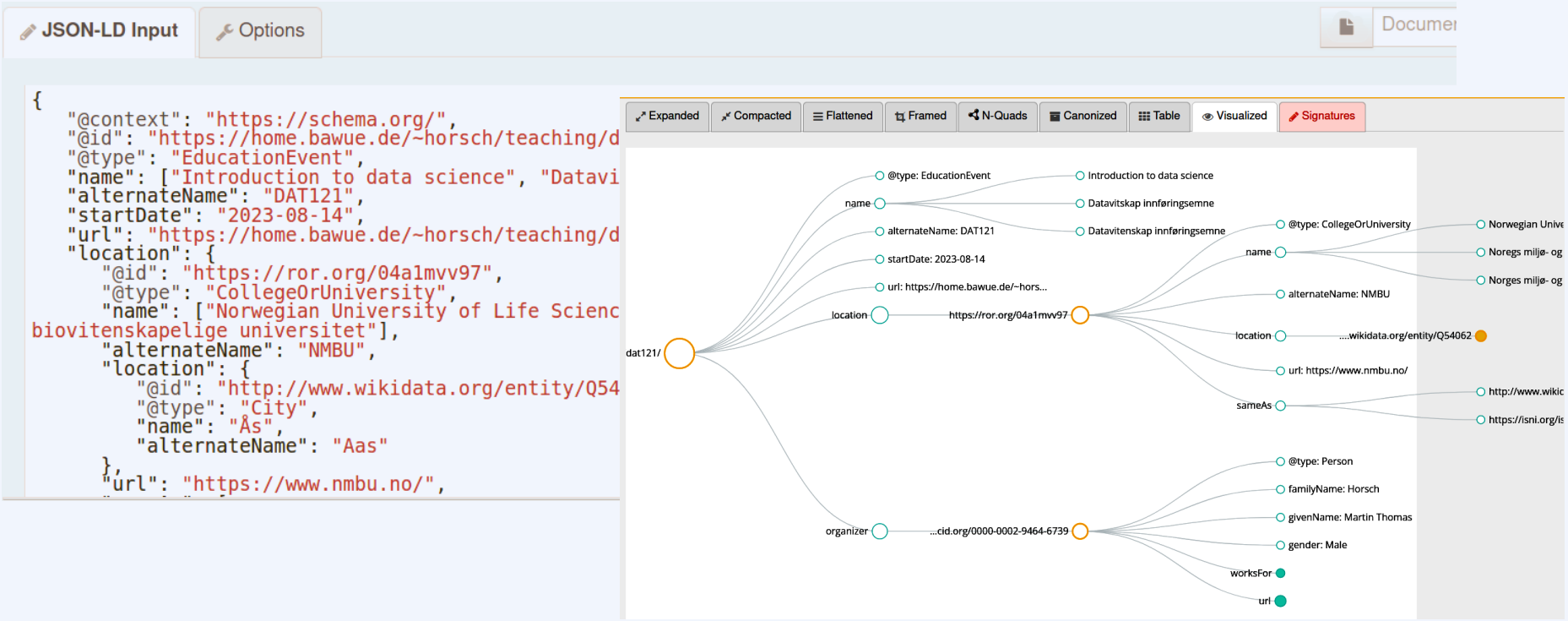
JSON-LD is an extension of JSON by which it can express RDF triples.

JSON-LD based data exchange

JSON: JavaScript Object Notation

JSON-LD: Extension of JSON to deal with linked data (knowledge graphs)

JSON-LD Playground (https://json-ld.org/playground/)



JSON-LD based data exchange

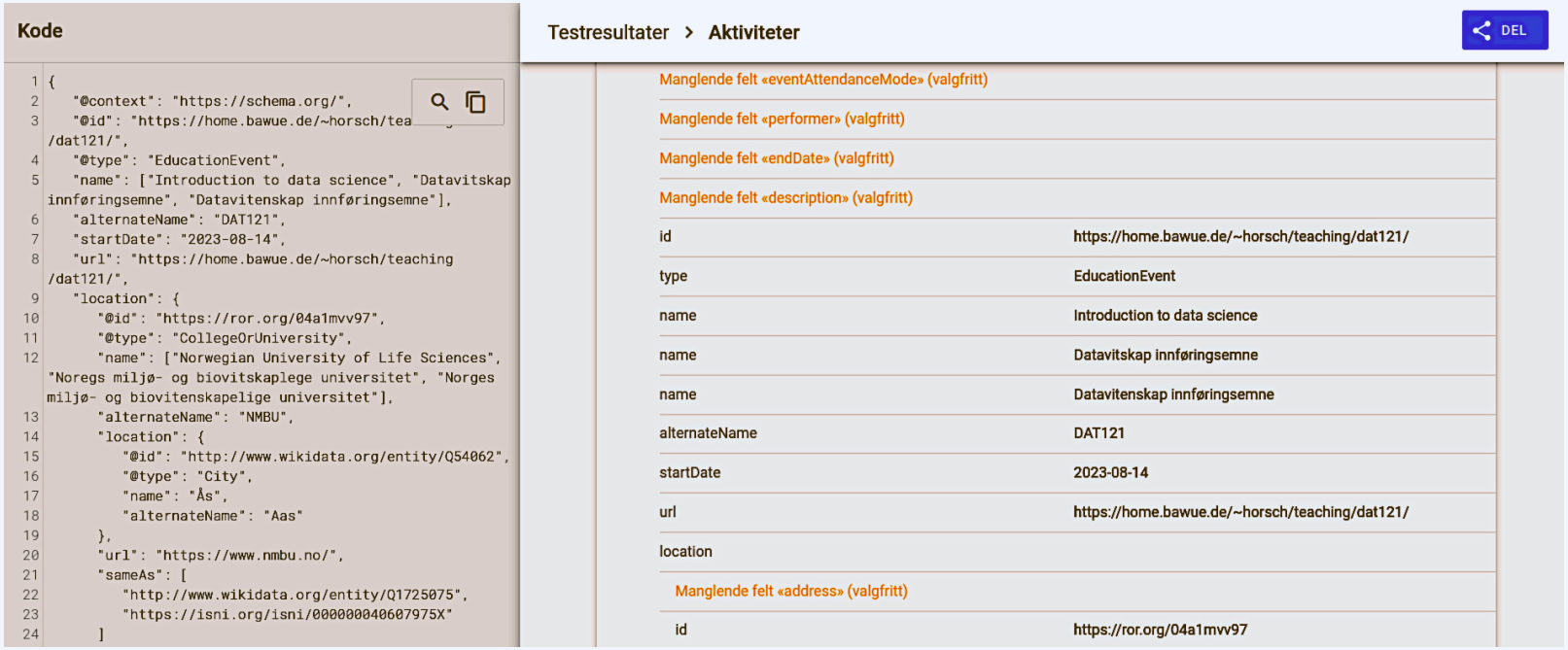
JSON: JavaScript Object Notation

JSON-LD: Extension of JSON to deal with linked data (knowledge graphs)

JSON-LD Playground (https://json-ld.org/playground/)

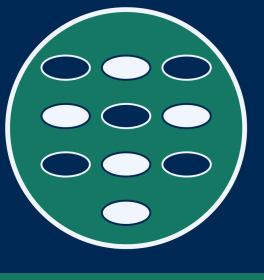
Google's Schema Markup Validator (https://validator.schema.org/)

Google's Rich Results Test (https://search.google.com/test/rich-results)



Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap



Digitalisering på Ås

INF203

June advanced programming project

2 Object orientation in practice

2.1 Object orientation
2.2 Encapsulation
2.3 E-R diagrams
2.4 Factories
2.5 Data-code co-design
2.6 Data interoperability