

# Werkzeuge für das wissenschaftliche Arbeiten

Python for Machine Learning and Data Science

Abgabe: 15.12.2023

---

## Contents

<b>1</b>	<b>Projektaufgabe</b>	<b>2</b>
1.1	Einleitung . . . . .	2
1.2	Aufbau . . . . .	3
1.3	Methoden . . . . .	3
<b>2</b>	<b>Abgabe</b>	<b>4</b>

# 1 Projektaufgabe

In dieser Aufgabe beschäftigen wir uns mit Objektorientierung in Python. Der Fokus liegt auf der Implementierung einer Klasse, dabei nutzen wir insbesondere auch Magic Methods.

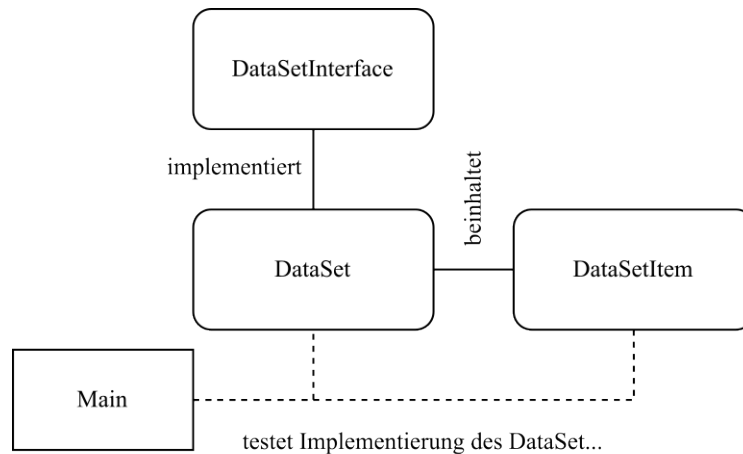


Figure 1: Abbildung 1: Darstellung der Klassenbeziehungen.

## 1.1 Einleitung

Ein Datensatz besteht aus mehreren Daten, ein einzelnes Datum wird durch ein Objekt der Klasse "DataSetItem" repräsentiert. Jedes Datum hat einen Namen (Zeichenkette), eine ID (Zahl) und bel. Inhalt.

Nun sollen mehrere Daten, Objekte vom Typ **DataSetItem**, in einem Datensatz zusammengefasst werden. Sie haben sich schon auf eine Schnittstelle und die benötigten Operationen, die ein Datensatz unterstützen muss, geeinigt. Es gibt eine Klasse **DataSetInterface**, die die Schnittstelle definiert und Operationen jedes Datensatzes angibt. Bisher fehlt aber noch die Implementierung eines Datensatzes mit allen Operationen.

Implementieren Sie eine Klasse **DataSet** als eine Unterklasse von **DataSetInterface**.

## 1.2 Aufbau

Es gibt drei Dateien, **dataset.py**, **main.py** und **implementation.py\***. In der **dataset.py** befinden sich die Klassen **DataSetInterface** und **DataSetItem**, in der Datei **implementation.py** muss die Klasse **DataSet** implementiert werden. Die Datei **main.py** nutzt die Klassen **DataSet** und **DataSetItem** aus den jeweiligen Dateien und testet die Schnittstelle und Operationen von **DataSetInterface**.

## 1.3 Methoden

Bei der Klasse **DataSet** sind insbesondere folgenden Methoden zu implementieren, die genaue Spezifikation finden Sie in der **dataset.py**:

1. `__setitem__(self, name, id_content)` Hinzufügen eines Datums mit Name, ID und Inhalt.
2. `__iadd__(self, item)` Hinzufügen eines **DataSetItem**.
3. `__delitem__(self, name)` Löschen eines Datums basierend auf dem Namen. Der Name eines Datums ist ein eindeutiger Schlüssel und kann nur einmal vorkommen.
4. `__contains__(self, name)` Prüfung, ob ein Datum mit diesem Namen im Datensatz vorhanden ist.
5. `__getitem__(self, name)` Abrufen des Datums über seinen Namen.
6. `__and__(self, dataset)` Bestimmung der Schnittmenge zweier Datensätze und Rückgabe als neuen Datensatz.
7. `__or__(self, dataset)` Bestimmung der Vereinigung zweier Datensätze und Rückgabe als neuen Datensatz.
8. `__iter__(self)` Iteration über alle Daten des Datensatzes. Es ist möglich, eine Sortierung für die Reihenfolge der Iteration anzugeben.
9. `filtered_iterate(self, filter)` Gefilterte Iteration über einen Datensatz mithilfe einer Lambda-Funktion mit den Parametern Name und ID.
10. `__len__(self)` Abrufen der Anzahl der Daten im Datensatz.

## 2 Abgabe

Programmieren Sie die Klasse `DataSet` in der Datei **implementation.py** zur Lösung der oben beschriebenen Aufgabe im VPL. Sie können auch direkt auf Ihrem Computer programmieren, dazu finden Sie alle drei benötigten Dateien zum Download im Moodle.

Das VPL nutzt den gleichen Code, wobei die **main.py** um weitere Testfälle und Überprüfungen erweitert wurde. Die Überprüfungen dienen dazu sicherzustellen, dass Sie die richtigen Klassen nutzen.

---

\* Dateien befinden sich im Ordner `/code/` dieses Git-Repositories.