

# Kurs rozszerzony języka Python

pickle, SQL, no-SQL

Marcin Młotkowski

4 grudnia 2019

# Plan wykładu

- 1 Przechowywanie obiektów
  - Pojedyncze obiekty
  - Kolekcje obiektów
- 2 Relacyjne bazy danych
- 3 Przykład ORM: SQLAlchemy
  - Definiowanie klas/tabel
  - Zapis i odczyt danych
  - Różności
- 4 Systemy NoSQL w Pythonie
  - Systemy zorientowane na dokumenty
  - Grafowe bazy danych
  - Bazy typu klucz-wartość

# Plan wykładu

- 1 Przechowywanie obiektów
  - Pojedyncze obiekty
  - Kolekcje obiektów
- 2 Relacyjne bazy danych
- 3 Przykład ORM: SQLAlchemy
  - Definiowanie klas/tabel
  - Zapis i odczyt danych
  - Różności
- 4 Systemy NoSQL w Pythonie
  - Systemy zorientowane na dokumenty
  - Grafowe bazy danych
  - Bazy typu klucz-wartość

# Pakiet pickle

Pakiet implementujący *serializację* i *deserializację* obiektów.

---

<sup>1</sup>A właściwie zbiór formatów

# Pakiet pickle

Pakiet implementujący *serializację* i *deserializację* obiektów.

Format<sup>1</sup> natywny Pythona.

---

<sup>1</sup>A właściwie zbiór formatów

# Pakiet pickle

Pakiet implementujący *serializację* i *deserializację* obiektów.

Format<sup>1</sup> natywny Pythona.

Użyteczny do przechowywania pojedynczych obiektów.

---

<sup>1</sup>A właściwie zbiór formatów

## Jak korzystać

```
import pickle
```

```
obj1 = {'uno': [1], 'duo': [2,3], 'tres': [4,5,6]}
```

### Zapis

```
with open('object.store', 'wb') as fh:  
    pickle.dump(obj1, fh)
```

### Odczyt

```
with open('object.store', 'rb') as fh:  
    obj2 = pickle.load(fh)
```

```
print(obj2)
```

## Co można przechowywać

- wartości proste (True, False, liczby);
- listy, stringi, krotki, słowniki;
- klasy, obiekty (spełniające pewne warunki), funkcje.



# json

```
import json
```

uzywamy dokładnie tak samo jak pickle

## Uwagi

- można używać kompresji;
- podatność na ataki (niezaufane pliki), można skorzystać z podpisywania, np. HMAC;
- jest sześć wariantów serializacji w zależności od wersji Pythona, można jawnie wskazywać której wersji się używa.

## Pakiet `shelve`

Pakiet do przechowywania w pliku większej ilości obiektów w postaci słownika.

- kluczem zawsze jest string;
- wartością jest obiekt zserializowany *picklem*;
- korzysta z tzw. *dbm*'ow, narzędzi dostępnych w bibliotekach uniksowych.

## Wykorzystanie

```
import shelve

with shelve.open('shelve') as db:
    for i in range(10):
        db['lista%i' % i] = [1,2,3, i]
    db.sync()
    for k in db:
        print("{}:{}".format(k, db[k]))
```

## Shelve: uwagi

- otwierając plik można wskazać, czy zapis ma być częsty (po zmianie/aktualizacji);
- nie ma wielodostępu;
- trzeba się pilnować:

```
db['lista'] = [1,2,3]
db['lista'].append(4)
```

nie zmienia listy w db, ale

```
tmp = db['lista']
tmp.append(4)
db['lista'] = tmp
```

działa dobrze.

## Po co mi to

### Cache zapytań SQL

```
SELECT * FROM dz_transakcje WHERE status ... : 389 504
SELECT * FROM dz_programy'                  :      868
select * from dz_kody_kasowe order by typ, i :      558
```

10-krotne przyspieszenie działania programu.

# Plan wykładu

- 1 Przechowywanie obiektów
  - Pojedyncze obiekty
  - Kolekcje obiektów
- 2 Relacyjne bazy danych
- 3 Przykład ORM: SQLAlchemy
  - Definiowanie klas/tabel
  - Zapis i odczyt danych
  - Różności
- 4 Systemy NoSQL w Pythonie
  - Systemy zorientowane na dokumenty
  - Grafowe bazy danych
  - Bazy typu klucz-wartość

# Silniki SQL

- Oracle
- DB/2
- MySQL
- PostgreSQL
- MSSQL
- ...



# DB API

## Python Database API Specification

Zunifikowany interfejs dostępu do różnych systemów BD. Obecna wersja: 2.0.

## Otwarcie połączenia z serwerem BD

```
connect("parametry") # zwraca obiekt Connection
```

## Otwarcie połączenia z serwerem BD

```
connect("parametry") # zwraca obiekt Connection
```

### MySQL

```
import MySQLdb  
db = MySQLdb.connect(host='localhost',  
    db='testing',  
    user='user',  
    passwd='123456')
```

## Zamknięcie połączenia

```
db.close()
```

## Komunikacja z bd

wysłanie zapytania

```
wynik = db.cursor()
```

```
wynik.execute('SELECT * FROM Studenci')
```

## Komunikacja z bd

### wysłanie zapytania

```
wynik = db.cursor()  
wynik.execute('SELECT * FROM Studenci')
```

### pobranie wyniku

```
row = wynik.fetchone()  
while row:  
    print row  
    row = wynik.fetchone()
```

### Opcjonalnie

```
wynik.close()
```

## Wynik: obiekt klasy Cursor

Atrybuty wyniku:

- description: opisuje kolumny
- rowcount: liczba przetworzonych wierszy (np. INSERT czy UPDATE)
- ...

## DB API: dodatkowe informacje

Standardowe wyjątki:

Warning, DatabaseError, NotSupportedError, ...



# SQLite

- 'Plikowa' baza danych, bez zewnętrznego serwera, żadnego kontaktu z adminem;
- moduł: `sqlite3`
- Implementuje DB API 2.0 z rozszerzeniami

# Użycie Sqlite

## Dostęp

```
import sqlite3  
db = sqlite3.connect("/tmp/temp.db")
```

albo

## Ciekawostka

```
db = sqlite.connect(':memory:')
```

# Plan wykładu

- 1 Przechowywanie obiektów
  - Pojedyncze obiekty
  - Kolekcje obiektów
- 2 Relacyjne bazy danych
- 3 Przykład ORM: SQLAlchemy
  - Definiowanie klas/tabel
  - Zapis i odczyt danych
  - Różności
- 4 Systemy NoSQL w Pythonie
  - Systemy zorientowane na dokumenty
  - Grafowe bazy danych
  - Bazy typu klucz-wartość

# Po co SQLAlchemy

To zajęcia Pythona a nie z SQL'a!

# Po co SQLAlchemy

To zajęcia Pythona a nie z SQL'a!

## Object-Relational Mapping (ORM)

Sposób odwzorowania świata obiektów w programie na świat relacyjny w bazie danych.

# Co daje nam ORM

Używając tylko Pythona możemy

- utworzyć tabele w bazie danych;
- tworzyć, odczytywać, aktualizować i usuwać dane (CRUD: Create, Read, Update, Delete);
- definiować różne sposoby komunikacji danych (leniwość/gorliwość, transakcyjność, etc).

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Table, Column, Integer,
    ForeignKey, String, DateTime
```

```
Base = declarative_base()
```

```
class Osoba(Base):
```

```
    __tablename__ = 'Osoba'
```

```
    id = Column(Integer, primary_key=True)
```

```
    imie = Column(String(20), nullable=False)
```

```
    wiek = Column(Integer, default=18)
```

```
    created = Column(DateTime,
        default=datetime.datetime.utcnow)
```

```
# albo lepiej: created =
```

```
    Column(DateTime, server_default=func.now())
```

## Każdy gdzieś mieszka

```
class Adres(Base):  
    __tablename__ = 'Adresy'  
    id = Column(Integer, primary_key=True)  
    email = Column(String)  
    miasto = Column(String)
```



## Każdy gdzieś mieszka

```
class Adres(Base):  
    __tablename__ = 'Adresy'  
    id = Column(Integer, primary_key=True)  
    email = Column(String)  
    miasto = Column(String)  
  
    @validates('email')  
    def validate_email(self, key, value):  
        assert "@" in value  
        return value
```

## Związki między klasami/tabelami

Jedna osoba ma wiele adresów (One-To-Many):

```
class Osoba(Base):
    __tablename__ = 'Osoba'
    ...
    adresy = relationship('Adres')

class Adres(Base):
    __tablename__ = 'Adres'
    ...
    mieszkaniec = Column(Integer, ForeignKey("Osoba.id"))
```

## Utworzenie tabeli

```
from sqlalchemy import create_engine  
engine = create_engine('sqlite:///wyklad.db', echo=True)  
Base.metadata.create_all(engine)
```

# Migracje

A co ze zmianą struktury bazy danych?

# Migracje

A co ze zmianą struktury bazy danych?

**alembic**

# Sesja

Operacje odbywają się w ramach sesji:

```
from sqlalchemy.orm import sessionmaker
engine = create_engine('sqlite:///wyklad.db', echo=True)
Session = sessionmaker(bind=engine)
sesja = Session()
```

## Dodawanie danych

```
o = Osoba(imie='Debeściak')  
adr1 = Adres(email='Joliot@Curie', miasto='Wrocław')  
o.adresy = [adr1]  
sesja.add(o)  
sesja.add(adr1)  
sesja.commit()
```

# Odpytywanie

```
lista = sesja.query(Osoba).filter(Osoba.imie.in_(['Debeściak'])).all()  
lista = sesja.query(Osoba).filter(Osoba.imie == 'Debeściak').all()
```



# Usuwanie

```
sesja.delete(o)
```

# Aktualizacja

```
o.imie = 'Inne'
```

## Uwagi

- wycofywanie zmian: `session.rollback()`;
- na końcu dobrze jest zrobić `sesja.close()`;
- sesja nie jest dla wielu wątków;
- są dedykowane warianty typów kolumn i zapytań związanych ze specyfiką poszczególnych silników.

# Plan wykładu

- 1 Przechowywanie obiektów
  - Pojedyncze obiekty
  - Kolekcje obiektów
- 2 Relacyjne bazy danych
- 3 Przykład ORM: SQLAlchemy
  - Definiowanie klas/tabel
  - Zapis i odczyt danych
  - Różności
- 4 Systemy NoSQL w Pythonie
  - Systemy zorientowane na dokumenty
  - Grafowe bazy danych
  - Bazy typu klucz-wartość

# NoSQL

## NOSQL (not only SQL)

Systemy baz danych o elastycznej strukturze danych. Czasem mówi się że są to ustrukturalizowane zasoby.

# NoSQL

## NOSQL (not only SQL)

Systemy baz danych o elastycznej strukturze danych. Czasem mówi się że są to ustrukturalizowane zasoby.

## Do czego się używa

Proste, lecz wielkie bazy danych przetwarzane na wielu komputerach.

# Systemy zorientowane na dokumenty

## Dokument

Dokument zawiera jakąś informację. Dokument może być w formacie XML, YAML, JSON, PDF, MS Office. Dokumenty nie muszą mieć jednego schematu.

# Systemy zorientowane na dokumenty

## Dokument

Dokument zawiera jakąś informację. Dokument może być w formacie XML, YAML, JSON, PDF, MS Office. Dokumenty nie muszą mieć jednego schematu.

## Przykłady danych

Imie="Adam"

Imie="Janina", Adres="ul. Cicha 132 m. 16", Dzieci=["Staś", "Krzyś" ]



## Inne cechy

Dokumenty mają unikatowe klucze (string, URI).

## Inne cechy

Dokumenty mają unikatowe klucze (string, URI).

### Wyszukiwanie

Wyszukiwanie oparte na kluczu lub zawartości.

# Przykłady systemów

CouchDB, MongoDB, Redis

# MongoDB

## System MongoDB

- system zorientowany na dokumenty;
- kolekcja: coś w rodzaju tabeli;
- nazwa modułu: `pymongo`;
- wymaga uruchomionego serwera `mongod`.

## Połączenie z bd

```
from pymongo import Connection
```

## Połączenie z bd

```
from pymongo import Connection  
connection = Connection()
```

## Połączenie z bd

```
from pymongo import Connection
connection = Connection()
db = connection["studenckie"]
```

# Kolekcje

## Pobranie kolekcji (leniwe)

```
kolekcja = db["protokoly"]  
# alternatywnie: kolekcja.protokoly
```



## Dodanie elementu do kolekcji

```
import datetime
python_prot = { "prowadzacy": 'Marcin Młotkowski',
                'przedmiot': 'Zaawansowany kurs Pythona',
                'pracownie': ['Marcin Młotkowski', 'Mateusz
Lewandowski'],
                'data': datetime.datetime.utcnow() }
```

## Dodanie elementu do kolekcji

```
import datetime
python_prot = { "prowadzacy": 'Marcin Młotkowski',
                'przedmiot': 'Zaawansowany kurs Pythona',
                'pracownie': ['Marcin Młotkowski', 'Mateusz
Lewandowski'],
                'data': datetime.datetime.utcnow()}
kolekcja.insert(python_prot)
# Zwraca ObjectId(..)
```

## Wyszukiwanie elementów

```
for prot in kolekcja.find({ 'prowadzacy' : 'Marcin Młotkowski' }):  
    # prot jest słownikiem
```

# Grafowe bazy danych

Dane są trzymane w postaci grafów: węzły reprezentują obiekty, a krawędzie: związki między obiektami.

# Zastosowanie

Mapy, systemy geograficzne, dokumenty etc.

# Bazy danych typu klucz-wartość

## Implementacje

- BigTable (Google)
- Apache Hadoop (Facebook, Amazon, HP, IBM, Microsoft)
- BerkeleyDB
- dbm

# MapReduce

## Map

Map( (klucz, wartość) ): przekształca każdą parę w listę par (kl, wart), a następnie łączy w grupy wartości związane z tym samym kluczem. Operacja jest wykonywana równoległe na każdej parze.

# MapReduce

## Map

Map( (klucz, wartość) ): przekształca każdą parę w listę par (kl, wart), a następnie łączy w grupy wartości związane z tym samym kluczem. Operacja jest wykonywana równoległe na każdej parze.

## Reduce

Reduce(kl, lista(wartości)): operacja wykonywana na każdej grupie i dająca w wyniku listę kolejnych wartości.



## Przykład (Hadoop)

```
# zliczanie słów we wszystkich dokumentach
from pydoop.pipes import Mapper, Reducer, Factory, runTask
class WordCountMapper(Mapper):
    def map(self, context):
        words = context.getInputValue().split()
        for w in words:
            context.emit(w, "1")
class WordCountReducer(Reducer):
    def reduce(self, context):
        s = 0
        while context.nextValue():
            s += int(context.getInputValue())
            context.emit(context.getInputKey(), str(s))
runTask(Factory(WordCountMapper, WordCountReducer))
```

# HDFS: Hadoop Distributed File System

Rozproszony, skalowalny, przenośny, obsługujący replikację i wielkie pliki.