Kurs rozszerzony języka Python

Wykład 10.

Marcin Młotkowski

21 grudnia 2022

Plan wykładu

Plan wykładu

Pakiet `pickle`

Pakiet implementujący *serializację* i *deserializację* obiektów.

[1] A właściwie zbiór formatów

Pakiet `pickle`

Pakiet implementujący *serializację* i *deserializację* obiektów.

Format[1] natywny Pythona.

[1] A właściwie zbiór formatów

Pakiet `pickle`

Pakiet implementujący *serializację* i *deserializację* obiektów.

Format[1] natywny Pythona.
Użyteczny do przechowywania pojedynczych obiektów.

[1] A właściwie zbiór formatów

Jak korzystać

```
import picke
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
```

używamy 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.

Przykład

```
import shelve

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

Przydało mi się:

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

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

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 Pytona 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__ = "Osoby"

    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)
    ulica = 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__ = "Osoby"
    ...
    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

```
adr1 = Adres(ulica="Amphiteatre Parkway", miasto="Mountain
adr2 = Adres(ulica="One Microsoft Way", miasto="Redmond")

os = osoba(imie="Debeściak")
os.adresy = [adr1, adr2]

sesja.add_all([adr1, adr2])
sesja.add(os)

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 = "Noweimię"
```

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

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

Połączenie z MongoDB

```
from pymongo import MongoClient
```

Włożenie danych

```
with MongoClient() as client:
    db = client['wyklad']

    osoby = db['osoby']

    adr1 = {'ulica': "Amphiteatre Parkway",
            'miasto': 'Mountain View'}
    adr2 = {'ulica':
            "One Microsoft Way", 'miasto': 'Redmond'}
    osoba1 = { "imie": 'Debeściak',
               'adresy': [ adr1, adr2 ] }

    osoby.insert_one(osoba1) # inserted.id
```

Pobranie danych

```
with MongoClient() as client:
    db = client['wyklad']

    wynik = db.osoby.find({'imie': { '$regex' : "/*.ebeś.*'
    for elem in wynik:
        print(elem)
```

Grafowe bazy danych

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

Zastosowanie

Mapy, systemy geograficzne, dokumenty etc.

Plan wykładu