# Kurs rozszerzony języka Python

## GTK+, cd

Marcin Młotkowski

1 grudnia 2017

# Plan wykładu

1. GTK+/GNOME, cd.
   - Wątki w GTK+
     - Zadania okresowe
     - Prawdziwe wątki
   - GNOME
2. Przechowywanie danych
   - Bazy danych DBM
   - Przechowywanie obiektów: shelve
   - Bazy danych SQL
   - Systemy NoSQL w Pythonie
     - Systemy zorientowane na dokumenty
     - Grafowe bazy danych
     - Bazy typu klucz–wartość

# Plan wykładu

# Timeouty

Uruchamianie zadań okresowych:

```
id = gobject.timeout_add(milisec, heart_beat, ...)
```

# Timeouty

Uruchamianie zadań okresowych:

```
id = gobject.timeout_add(milisec, heart_beat, ...)
```

Funkcja `heart_beat` powinna zwrócić True, w przeciwnym przypadku nie będzie dalej powtarzana.

# Timeouty

Uruchamianie zadań okresowych:

```
id = gobject.timeout_add(milisec, heart_beat, ...)
```

Funkcja `heart_beat` powinna zwrócić True, w przeciwnym przypadku nie będzie dalej powtarzana.

Wyłączenie zadania

```
gobject.source_remove(id)
```

# Funkcje idle

Co program robi jak nic nie robi?

# Implementacja funkcji

Uruchamianie zadań gdy nic się nie dzieje:

```
id = gobject.idle_add(milisec, heart_beat, ...)
```

# Implementacja funkcji

Uruchamianie zadań gdy nic się nie dzieje:

```
id = gobject.idle_add(milisec, heart_beat, ...)
```

Funkcja `heart_beat` powinna zwrócić True, w przeciwnym przypadku nie będzie dalej powtarzana.

# Implementacja funkcji

Uruchamianie zadań gdy nic się nie dzieje:

```
id = gobject.idle_add(milisec, heart_beat, ...)
```

Funkcja `heart_beat` powinna zwrócić True, w przeciwnym przypadku nie będzie dalej powtarzana.

Wyłączenie zadania

```
gobject.source_remove(id)
```

# Monitorowanie we/wy

Można kontrolować otwarte pliki (w tym pliki związane z sieciami, gniazdami czy potokami).

Co możemy monitorować (warunki)

| | |
|---|---|
| gobject.IO_IN | są dane gotowe do odczytu |
| gobject.IO_OUT | plik jest gotów do zapisu |
| gobject.IO_PRI | są pilne dane do odczytu |
| gobject.IO_ERR | błąd |
| gobject.IO_HUP | zerwanie połączenia |

# Jak monitorować

## Włączenie monitorowania

```
id = gobject.io_add_watch(plik(otwarty), warunek,
monitor)
```

# Jak monitorować

## Włączenie monitorowania

id = `gobject.io_add_watch(plik(otwarty), warunek, monitor)`

## funkcja monitorująca

```
def monitor(źródło, warunek):
```

Funkcja powinna zwrócić True jeśli ma dalej monitorować.

# Korzystanie z wątków

## Dobra wiadomość

`threading.Thread` działa.

# Korzystanie z wątków

**Dobra wiadomość**

`threading.Thread` działa.

**Zła wiadomość**

Ale nie jest to takie proste.

# Wersja 1.

Wątek nie odwołuje się do GUI

## Przed wywołaniem `gtk.main()`

```
import gobject

gobject.threads_init()
```

# Przykład

```
button.connect("clicked",
               lambda x:  threading.Thread(target=hello).start())
```

# Interakcja wątków z GUI

## Inicjowanie

```
gtk.gdk.threads_init()
```

# Uruchomienie aplikacji

```
gtk.threads_enter()
gtk.main()
gtk.threads_leave()
```

# Szkielet kodu w wątku

```
gtk.threads_enter()
try:
    myentry.set_text("""foo""")
finally:
    gtk.threads_leave()
```

# Szkielet kodu w wątku, wersja polecana

```
with gtk.gdk.lock:
    ...
```

# Co to jest

GNU Network Object Model Environment

# A co jest ważne

- Więcej kontrolek (rozbudowane GTK+);
- desktop, aplikacje mogą się integrować z desktopem;
- własne aplety (dla środowiska GNOME);
- wsparcie dla tworzenia róznych wersji językowych.

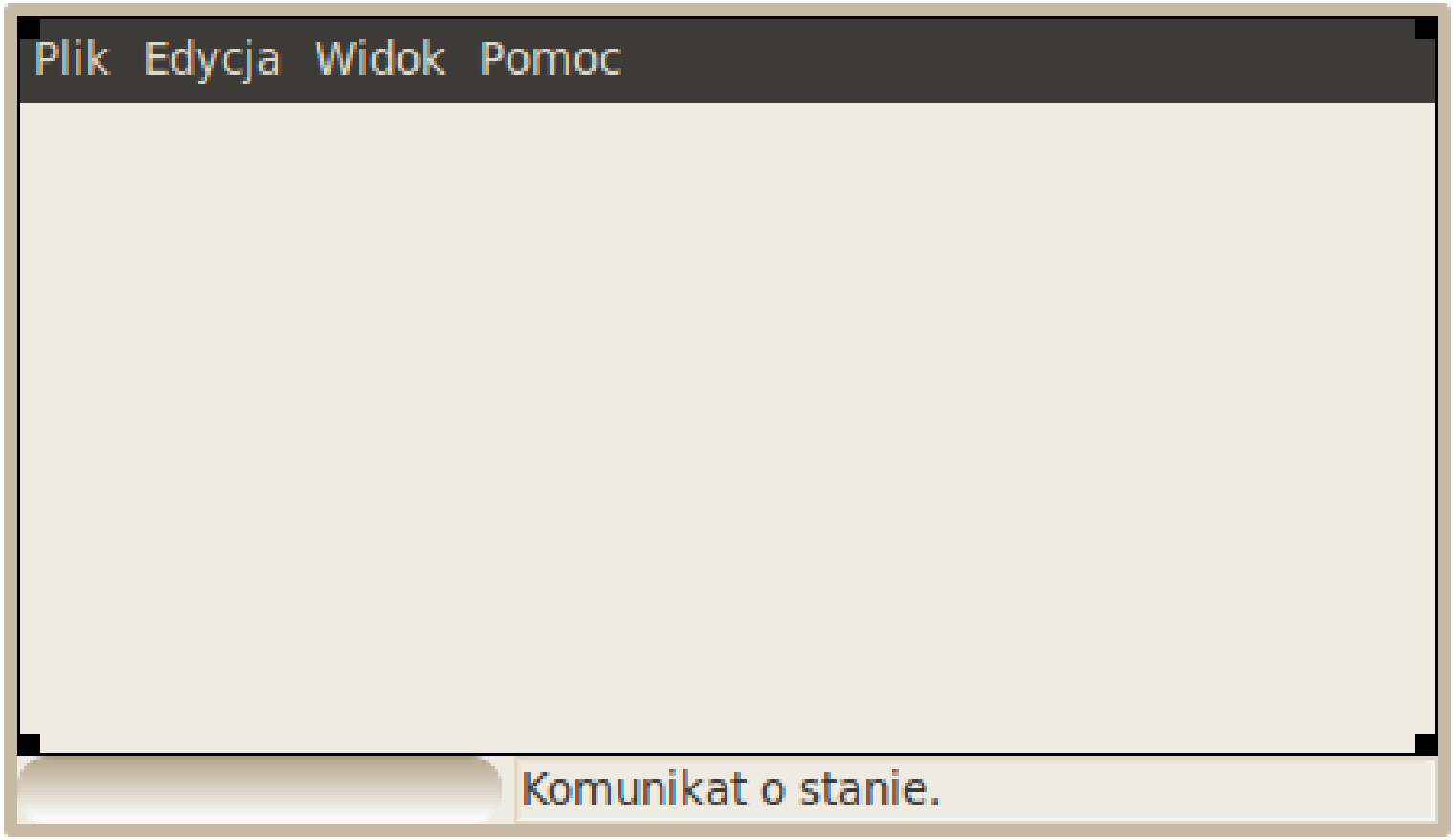
# GLADE i GNOME

GNOME używa LibGlade

# Przykład

```
import gtk
import gnome.ui
import gtk.glade

gnome.init(''Wykład'', ''0.1'')
aplikacja = gtk.glade.XML(''gnome.glade'')
gtk.main()
```


Plik Edycja Widok Pomoc
Komunikat o stanie.

# Zła wiadomość

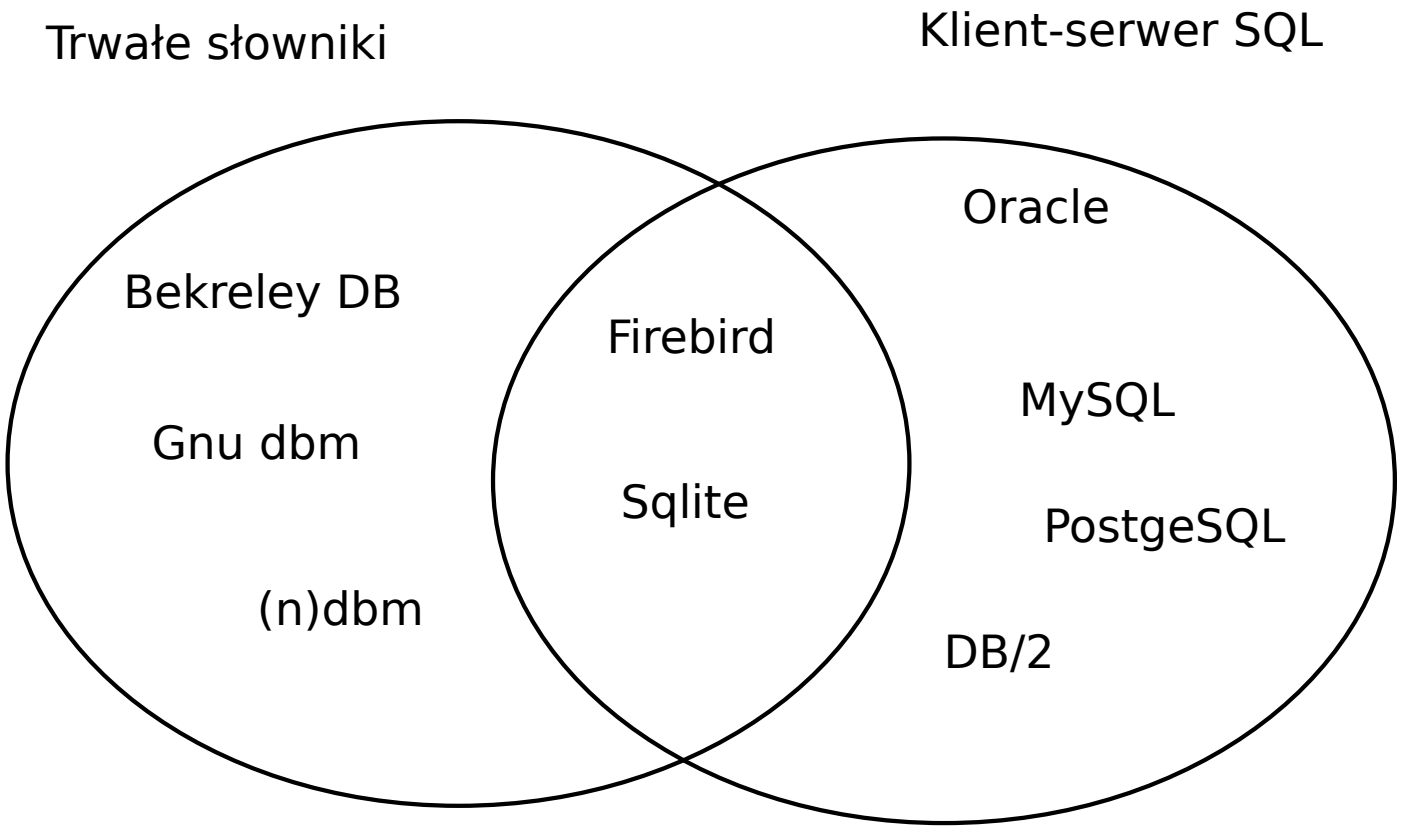
Są błędy, ale libglade nie jest już naprawiane.

# Jak ktoś potrzebuje

```
aplikacja = gnome.ui.App('''Aplikacja''', '''Okno aplikacji''')
aplikacja.show()
gtk.main()
```

# Plan wykładu

# Rodzaje baz danych



# Bazy danych typu DBM

Database manager

- Dane przechowywane są w pliku, który przypomina słownik
- Dostęp do danych jest po kluczu
- Wartość zwykle może być tylko napisem
- Implementacja: zwykle tablice haszujące i b-drzewa
- Brak odrębnego serwera, dane pamiętane są w lokalnym pliku
- Szybkie!!!

# Korzystanie z DBM

```
db['jeden'] = 'one'
db['dwa'] = 'two'

if 'trzy' in db:
    del db['trzy']
```

# Korzystanie z DBM

```
import <modul>
db = <modul>.open("storage.dbm", 'c')
```

```
db['jeden'] = 'one'
db['dwa'] = 'two'

if 'trzy' in db:
    del db['trzy']
```

```
db.close()
```

# Dostępne moduły

| Python 2.* | Python 3.* | Opis |
|---|---|---|
| dumbdb | dbm.dumb | wolny, ale czysto Pythonowa implementacja |
| bsddb (dbhash) | — | implementacja Berkeley DB |
| dbm | dbm.ndbm | interfejs uniksowej `(n)dbm` |
| gdbm | dbm.gnu | rozszerzenie dbm |

# Przetwarzanie całej kolekcji

```
for key in db.keys():
    print(db[key])
```

# Przetwarzanie całej kolekcji

```
for key in db.keys():
    print(db[key])
```

**Ostrzeżenie**

Wymaga odczytu do pamięci całego pliku

# Przetwarzanie kolekcji, dumbdbm

```
for key in db:
    print("db[", key, "] =", db[key])
```

```
db.close()
```

# Przetwarzanie kolekcji

## dbhash

```
print db.first()
while True:
    try:
        el = db.next()
        print el
    except KeyError:
        break
```

## gdbm

```
k = db.firstkey()
while k != None:
    print k
    k = db.nextkey(k)
```

# Dostępne moduły

| | for–in | firstkey(), nextkey() | first(), next() |
|---|---|---|---|
| dumbdbm | X | — | — |
| (n)dbm | — | — | — |
| gdbm | — | X | — |
| dbhash | — | — | X |

# Jak sobie radzić w tym bałaganie

Używać generycznego pakietu `anydbm`

# Jak sobie radzić w tym bałaganie

Używać generycznego pakietu `anydbm`

Używać pakietu `whichdbm`:

```
>>> import anydbm
>>> whichdb.whichdb('plik.db')
```

# Berkeley DB

## Gdzie można spotkać Berkeley DB

OpenLDAP, Subversion, Spamassasin, KDevelop

Właściwości:

- transakcje;
- replikacje;
- blokowanie rekordów;
- kluczami są liczby naturalne.

# A jak przechowywać obiekty

## Pakiet `shelve`

pliki są słownikami przechowującymi pary (string, obiekt)

# Pakiet shelve

## przykład

```
import shelve
db = shelve.open('dane.db')
db['lista'] = [2,3,5,7,11]
db.sync()
del db['obiekt']
db.close()
```

# 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

## Tworzenie tabeli

```
import sqlite3
db = sqlite3.connect("/tmp/temp.db")
kursor = db.cursor()
kursor.execute("'create table Library
        (Author text, Title text, publish_year int, price real)"')
db.commit()
```

# Użycie Sqlite

## Wstawienie tabeli do wiersza

```
kursor.execute(''insert into Library values
        ('Shakespeare', 'Hamlet', 2003, 25.5)'')
```

# Użycie Sqlite

## Pobranie danych bd (rozszerzenie DB API 2.0)

```
kursor.execute('SELECT * FROM Library')
for row in kursor:
    print(row)
```

# Sqlite

## Ciekawostka

db = sqlite.connect(":memory:")

# 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 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', 'Jan Otop'],
                'data' : datetime.datetime.utcnow()}
```

# Dodanie elementu do kolekcji

```
import datetime
python_prot = { "prowadzacy": 'Marcin Młotkowski',
                'przedmiot': 'Zaawansowany kurs Pythona',
                'pracownie': ['Marcin Młotkowski', 'Jan Otop'],
                '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 równolegle 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 równolegle 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.