Занятие 29

Flask

О примерной структуре презентации по работе

1. Титульный слайд – Название работы, автор, курс (зима 2023)
2. Содержание презентации или план
3. Цели
4. Задачи, спецификация, бизнес-проблема
5. Особенности, специфика
6. Функционал, реализованный и планируемый к реализации
7. Теория, подходы к решению
8. Что реализовано, что предстоит в планах реализовать
9. Структуры, схемы, алгоритмы и прочее
10. Использованные технические средства, модули, и т.д.
11. Демонстрация
12. Любые другие пункты, которые вы хотите включить. Может быть какие-то главные формулы или гениальные строки кода и т.д.

Комментарии.

1. Презентация не должна быть полноценным докладом с полным текстом
2. Графические представления приветствуются
3. Это скорее опорный конспект, но если вы не уверены в себе, как докладчик, то можно написать ключевые фразы
4. Обязательно проговорите ее перед зеркалом вслух. Выяснится много интересных деталей, засеките время – не больше 10 минут.
5. Если есть сокращения, то лучше их расшифровать, если это не общепринятые.
6. Это ваша презентация, сколько вам надо слайдов столько и делайте. Если надо больше, значит больше, если надо меньше, значит меньше

Задача 28-1

Дан список чисел a. Назовем пару (a[i], a[j]) инверсией, если i < j, а a[i] > a[j]. Напишите функцию, которая возвращает количество инверсий в списке.

Например:

[1,2,3,4,5] -> 0

[5,4,3,2,1] -> 10

Задача 28-2

Напишите функцию, результатом которой является расстояние Хемминга двух строк одинаковой длины, равное количеству несовпадающих букв на одинаковых позициях.

Например:

abc и abc – 0

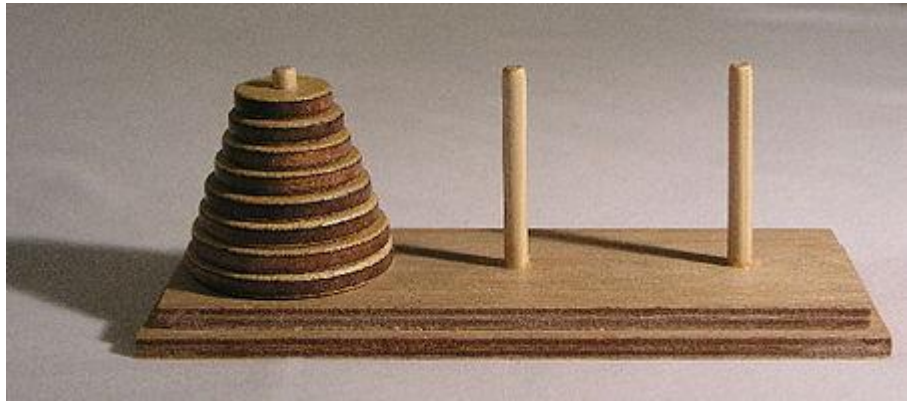
abc и abd – 1

abc и xyz - 3

Попробуйте написать эту функцию в одну строчку)

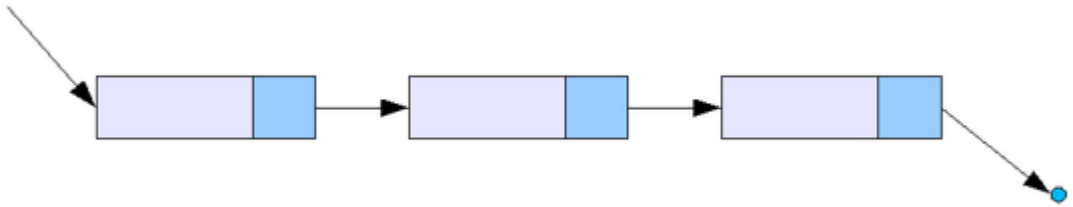
Задача 28-3

Напишите функцию, которая рассчитывает наименьшее число перестановок при перемещении Ханойской башни (n дисков насаженных на одном стержне). Требуется переместить эти диски на соседний стержень. Разрешается использовать третий стержень. Диски можно класть только на диски большего диаметра.
Для n = 1, число перестановок равно 1.
Для n = 2, число перестановок равно 3.



Динамические структуры данных - Связные списки

```
class Node:
    def __init__(self, value):
        self.value = value              # Значение узла
        self.next_node = None           # Ссылка на следующий узел
a = Node(1)
b = Node(22)
a.next_node = b
c = Node(333)
b.next_node = c
d = Node(4444)
c.next_node = d


x = a
while x.next_node != None:
    print(x.value)
    x = x.next_node
```



Связные списки – типичные функции

1. Добавить узел x в конец связного списка, головой которого является a
2. Сцепить два списка
3. Добавить узел в голову списка (поменять голову)
4. Сосчитать количество элементов связного списка.
5. Поменять местами два элемента.

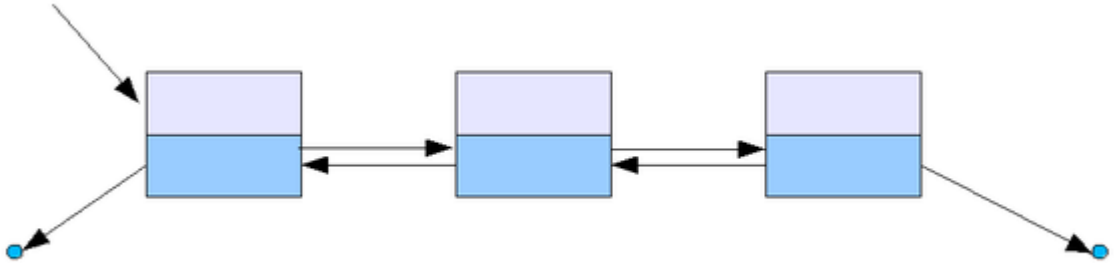
Давайте решим эти задачи.

```
class Node:
    def __init__(self, value):
        self.value = value                  # Значение узла
        self.next_node = None # Ссылка на следующий узел
```

a = Node(1) # голова списка, а где конец, мы и не знаем

x = Node(0) – это узел, который надо добавить или голова второго списка.

Двусвязные списки



```
class Node:
    def __init__(self, value):
        self.value = value                  # Значение узла
        self.next_node = None               # Ссылка на следующий узел
        self.prev_node = None               # Ссылка на предыдущий узел
```

Задание

Дан связный список.

```
class Node:
    def __init__(self, value):
        self.value = value                # Значение узла
        self.next_node = None             # Ссылка на следующий узел
 a = Node(1) # голова списка
```

1. Найти наибольший элемент в списке
2. (сложная) Отсортировать список по возрастанию value

Динамические структуры данных: стеки

Стек — динамическая структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого *вершиной стека*.

По определению, элементы извлекаются из стека в порядке, обратном их добавлению в эту структуру, т.е. действует принцип "последний пришёл — первый ушёл“ (LIFO – Last in First Out)

Наиболее наглядным примером организации стека служит детская пирамидка, где добавление и снятие колец осуществляется как раз согласно определению стека.

Стек

Стек можно организовать на базе любой структуры данных, где возможно хранение нескольких однотипных элементов и где можно реализовать определение стека: линейный массив, типизированный файл, однонаправленный или двунаправленный список.

Выделим типовые операции над стеком и его элементами:

- добавление элемента в стек; (push())
- удаление элемента из стека; (pop())
- проверка, пуст ли стек;
- просмотр элемента в вершине стека без удаления.

Используя стек, напечатайте символы данной строки в обратном порядке.

Очередь FIFO (First In First Out)

Очереди очень похожи на стеки. Они так же не дают доступа к произвольному элементу, но, в отличие от стека, элементы кладутся *(enqueue)* и забираются *(dequeue)* с разных концов.

Такой метод называется «первый вошел, первый вышел» *(First-In-First-Out или FIFO)*.

То есть забирать элементы из очереди мы будем в том же порядке, в котором и клали. Как реальная очередь или конвейер.

Задание

Реализуйте задачу хранения и выборки нескольких стопок тарелок, высотой не более 10 тарелок.

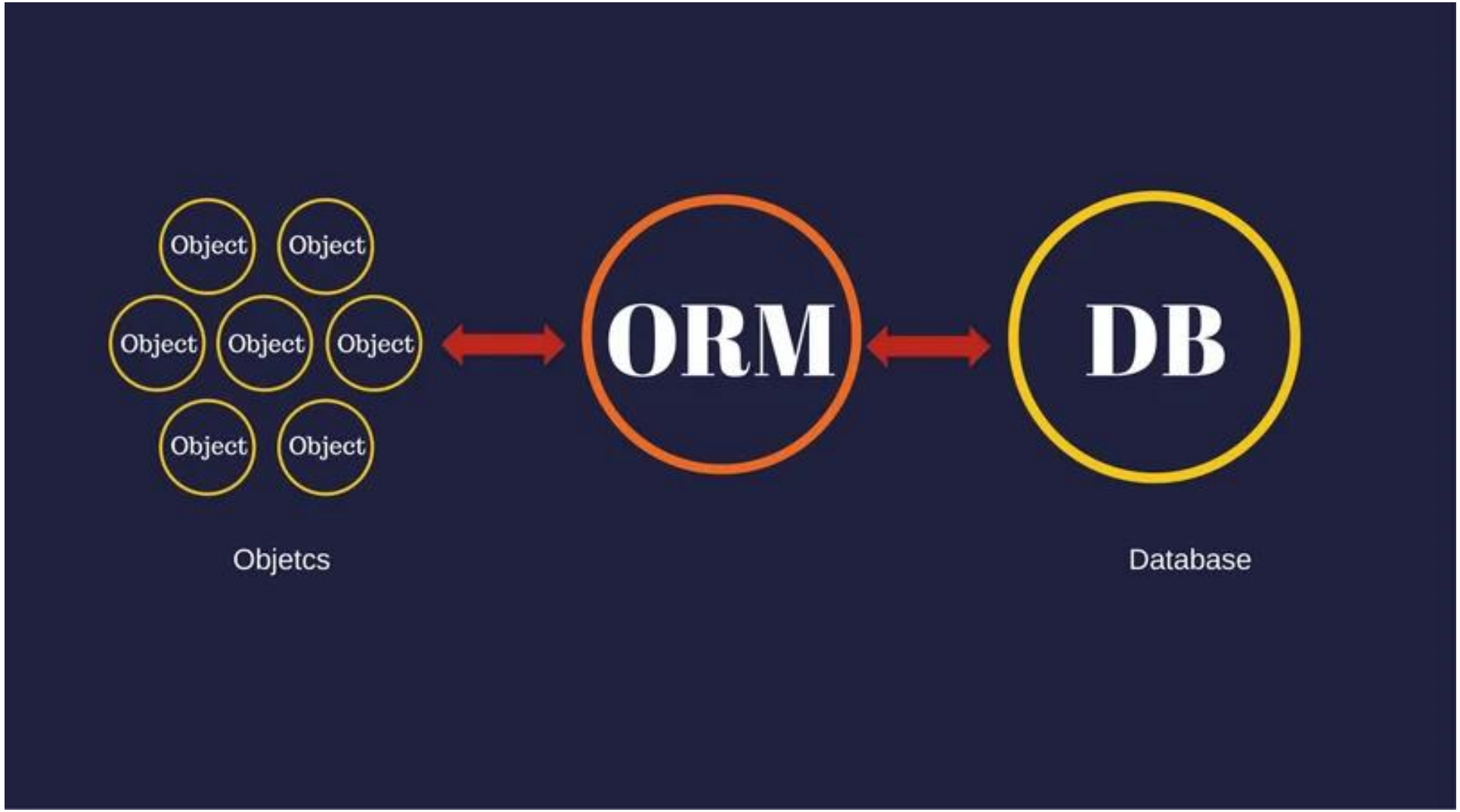
Если стопка достигла высоты 10, то рождается новая стопка тарелок.

Реализуйте функции push и pop, которые сигнализируют с какой стопкой они работают.

ORM SQLAlchemy

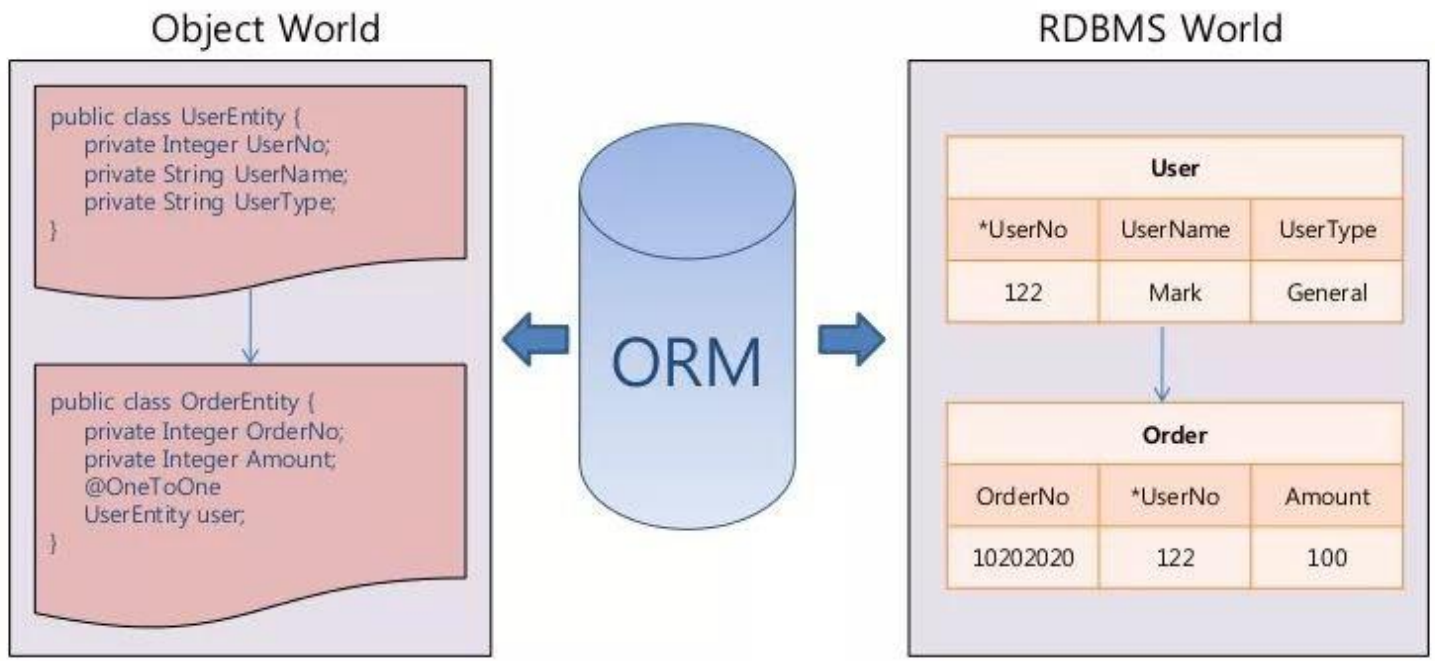
Определение

- Объектно-реляционное отображение (Object-Relational Mapping) — это метод, который позволяет вам запрашивать и манипулировать данными из базы данных, используя объектно-ориентированную парадигму.


Object
Object
Object
Object
Object
Object
Object
ORM
DB
Objetcs
Database

Object Relational Mapping

-Bridge between relational database and object world



Достоинства ORM

Безопасность запросов

Представление параметров типами данных основного языка (без преобразования в типы БД)

Автоматизация бэкапа и диплоя

Прозрачное кеширование данных и возможность выполнения отложенных запросов

Переносимость (использование разных СУБД (без дополнительных правок в коде)

Избавление программиста от необходимости вникать в детали реализации той или иной СУБД и синтаксиса соответствующего диалекта языка запросов.

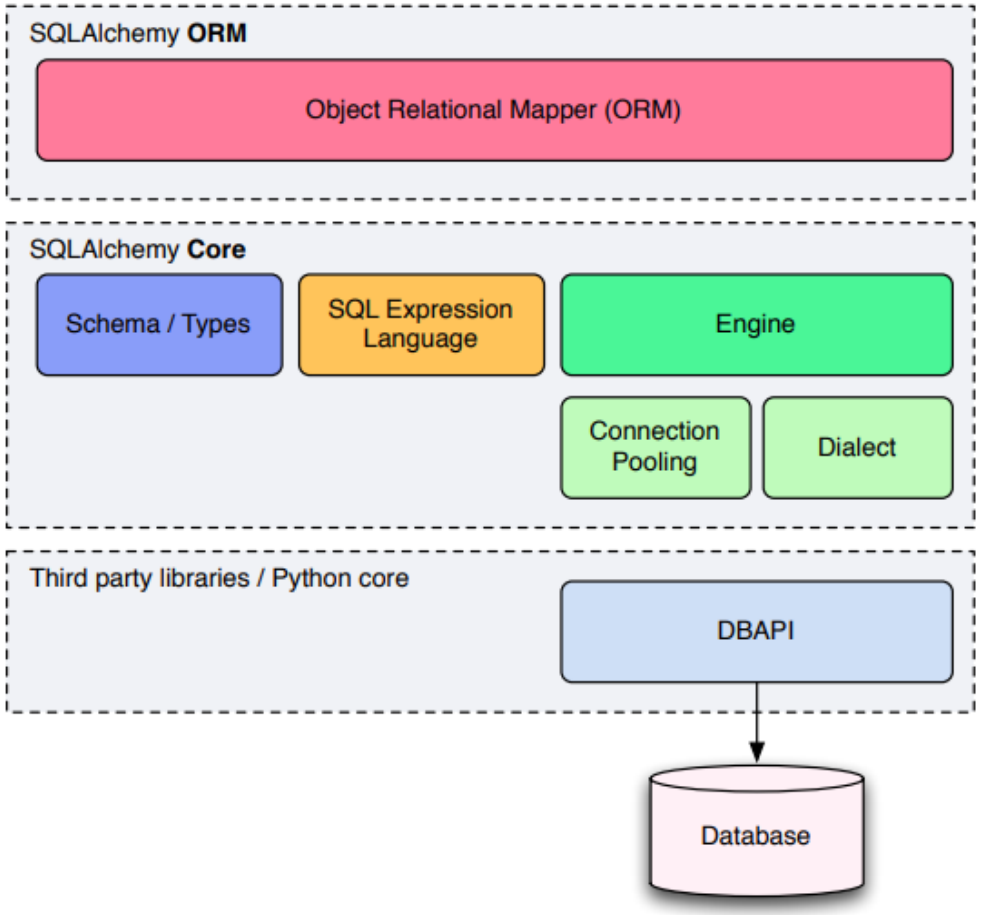
Недостатки ORM

- Абстрагирование от языка SQL. Во многих случаях выборка данных перестает быть интуитивно понятной и очевидной.
- Дополнительные накладные расходы на конвертацию запросов и создание внутренних объектов самого ORM.
- Возможна потеря производительности

Философия SQLAlchemy

- Привести использование различных баз данных и адаптеров к максимально согласованному интерфейсу
- Никогда не "скрывайте" базу данных или ее концепции разработчики должны знать / продолжать думать на языке SQL.
- Обеспечить автоматизацию рутинных операций CRUD
- Разрешить выразить синтаксис DB/SQL в декларативном шаблоне.

Архитектура SQLAlchemy



- SQAlchemy состоит из двух компонентов ORM и CORE

Компонента SQLAlchemy - Core

- **SQLAlchemy Core** - это абстракция над традиционным SQL. Он предоставляет SQL Expression Language, позволяющий генерировать SQL-инструкции с помощью конструкций Python.
- • **Engine** - механизм, который обеспечивает подключение к конкретному серверу базы данных.
- • **Dialect** - интерпретирует разные диалекты SQL и команды базы данных в синтаксис конкретного DBAPI и серверной части базы данных.
- • **Connection Pool** - хранит коллекцию подключений к БД для быстрого повторного использования
- • **SQL Expression Language** - позволяет писать SQL запрос с помощью выражений Python
- • **Schema/Types** - использует объекты Python для представления таблиц, столбцов и типов данных.

SQLAlchemy - ORM

- Позволяет создавать объекты Python, которые могут быть сопоставлены с таблицами реляционной базы данных
- Предоставляет систему запросов, которая загружает объекты и атрибуты с использованием SQL, сгенерированного на основе сопоставлений.
- Выстроена поверх Core - использует Core для создания SQL и обращений с базой данных
- Представляет несколько более объектно-ориентированную перспективу,в отличие от перспективы, ориентированной на схему

Создание БД

```
from sqlalchemy import create_engine

import psycopg2
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT

connection = psycopg2.connect(user="postgres", password=“Вашпароль”)
connection.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)

cursor = connection.cursor()
sql_create_database = cursor.execute('create database sqlalchemy_tuts')

cursor.close()
connection.close()
```

Создание таблиц

```
from sqlalchemy import create_engine, MetaData, Table, Integer, String, \
    Column, DateTime, ForeignKey, Numeric, SmallInteger
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from datetime import datetime
engine = create_engine("postgresql+psycopg2://postgres:Вашпароль@localhost/sqlalchemy_tuts")

Base = declarative_base()
class Customer(Base):
    __tablename__ = 'customers'
    id = Column(Integer(), primary_key=True)
    first_name = Column(String(100), nullable=False)
    last_name = Column(String(100), nullable=False)
    username = Column(String(50), nullable=False)
    email = Column(String(200), nullable=False)
Base.metadata.create_all(engine)
```

```
from sqlalchemy import create_engine, MetaData, Table, Integer, String,  Column, DateTime, ForeignKey, Numeric, SmallInteger


from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship


engine = create_engine("postgresql+psycopg2://postgres:Вашпароль@localhost/sqlalchemy_tuts")


Base = declarative_base()
class Customer(Base):
    __tablename__ = 'customers'
    id = Column(Integer(), primary_key=True)
    first_name = Column(String(100), nullable=False)
    last_name = Column(String(100), nullable=False)
    username = Column(String(50), nullable=False)
    email = Column(String(200), nullable=False)
Base.metadata.create_all(engine)
```

Внесение данных в таблицы

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session, sessionmaker
from datetime import datetime
from sqlalchemy import create_engine, MetaData, Table, Integer, String, Column, DateTime, ForeignKey, Numeric, SmallInteger

from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship

engine = create_engine("postgresql+psycopg2://postgres:1111@localhost/sqlalchemy_tuts")
session = Session(bind=engine)
Base = declarative_base()

class Customer(Base):
    __tablename__ = 'customers'
    id = Column(Integer(), primary_key=True)
….
```

```
c1 = Customer(
    first_name = 'Dmitriy',
    last_name = 'Yatsenko',
    username = 'Moseend',
    email = 'moseend@mail.com'
)
c2 = Customer(
    first_name = 'Valeriy',
    last_name = 'Golyshkin',
    username = 'Fortioneaks',
    email = 'fortioneaks@gmail.com'
)

print(c1.first_name, c2.last_name)
session.add(c1)
session.add(c2)

print(session.new)

session.commit()
```

Работа с данными

```
print(session.query(Customer).all())
print(session.query(Customer))
```

```
q = session.query(Customer)          # Просмотр
for c in q:
    print(c.id, c.first_name, c.last_name, c.email)
```

```
print(session.query(Customer).count()) # Сколько
```

```
i = session.query(Customer).get(2)
i.email = "OldEmail"                 # Изменение
session.add(i)
```

```
q = session.query(Customer)
for c in q:
    print(c.id, c.first_name, c.last_name, c.email)
```

```
# i = session.query(Item).filter(Item.name == 'Monitor').one()
# session.delete(i)                  # Удаление
session.commit()
```

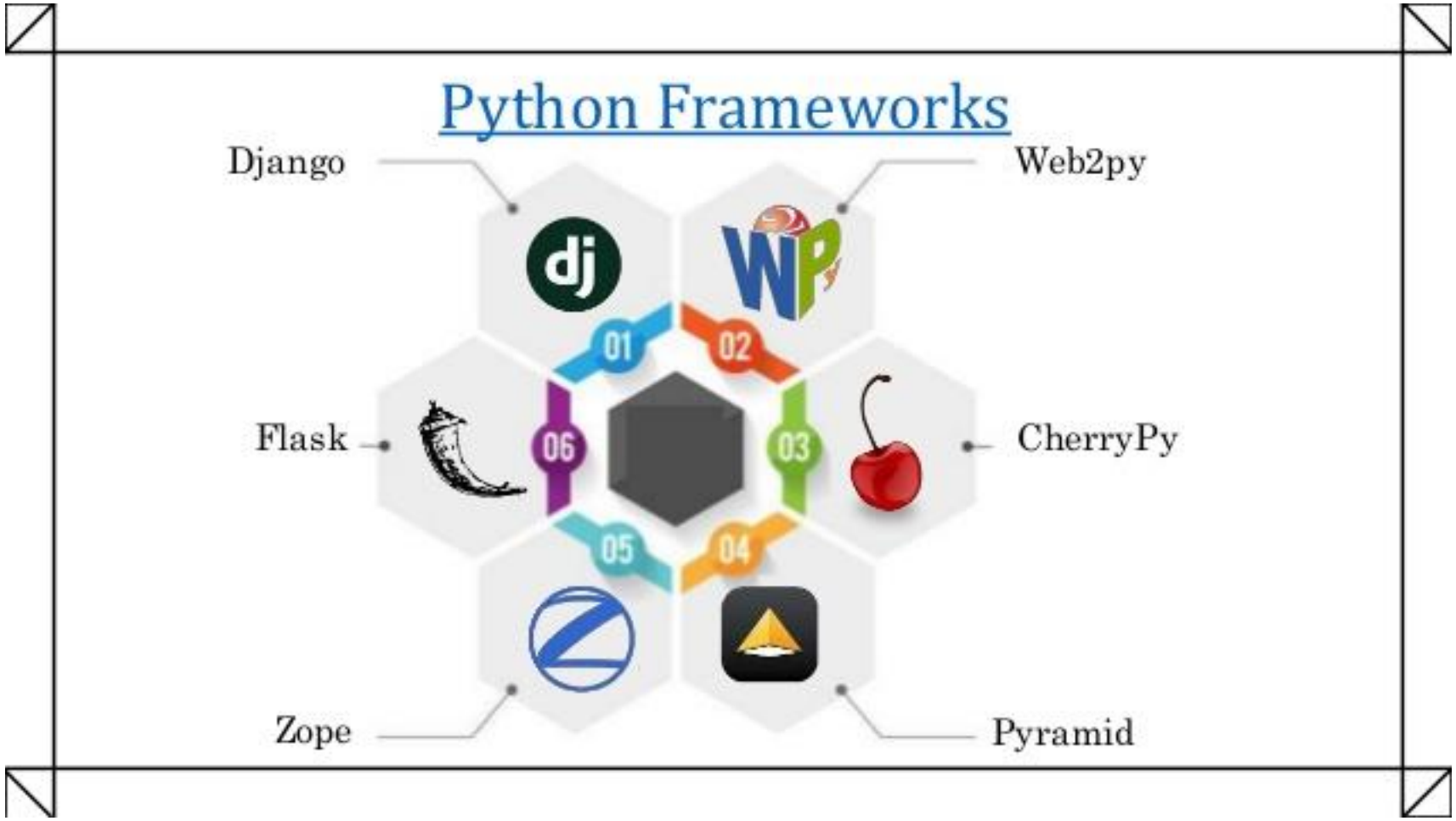
BUILDING A PYTHON APP IN

FLASK

Web service

- Web service – сервис (набор методов), предоставляемый приложением и доступный по сети
- Стандартизированный способ взаимодействия разнородных приложений
- Представление услуг для любого приложения

Выбор микрофреймворка



Установка Flask, Flask-Alchemy

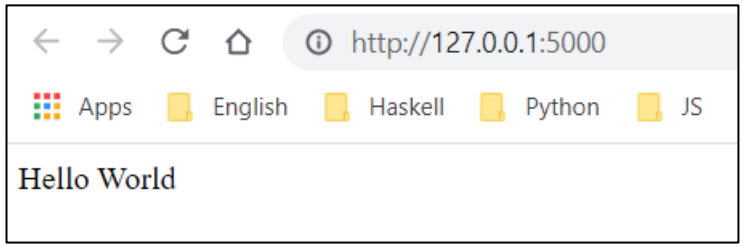
Pycharm: File / Settings / Project … / Interpretator / + / имя модуля

или

pip install имя модуля

Первая программа

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello World'

if __name__ == "__main__":
    app.run(debug = True)
```



Маршруты (Routing)

- from markupsafe import escape

- @app.route('/user/<username>')
- **def** show_user_profile(username):
- # show the user profile for that user
- **return** f'User {escape(username)}'

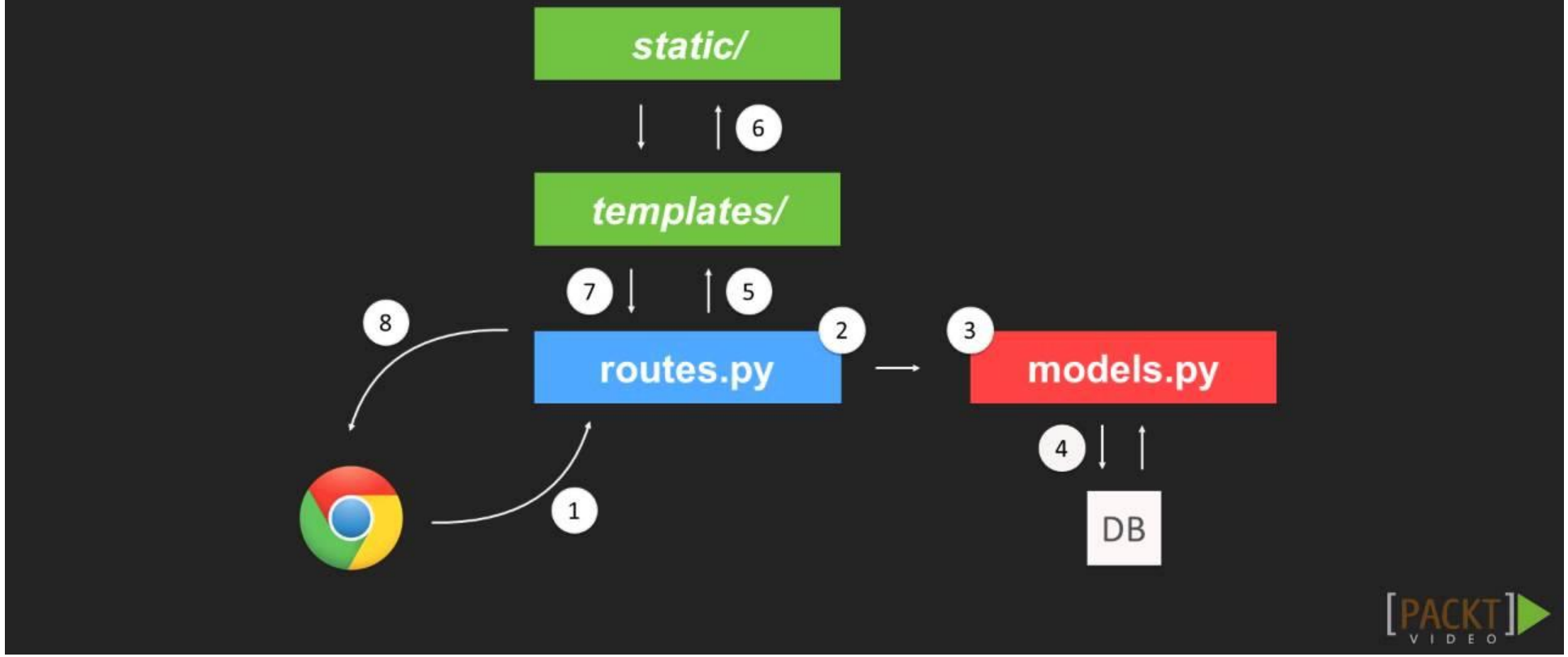
- @app.route('/post/<int:post_id>')
- **def** show_post(post_id):
- # show the post with the given id, the id is an integer
- **return** f'Post {post_id}'

- @app.route('/path/<path:subpath>')
- **def** show_subpath(subpath):
- # show the subpath after /path/
- **return** f'Subpath {escape(subpath)}'

Вторая программа

```
from flask import Flask, render_template

menu = ["Первый", "Второй", "Третий"]

app = Flask(__name__)

@app.route('/index')
@app.route('/')
def index():
    return render_template('index.html', title = 'Про Flask', menu = menu)
@app.route('/about')
def about():
    return render_template('about.html', title = 'О сайте')

if __name__ == "__main__":
    app.run(debug=True)
```

The Request-Response Cycle



Шаблон – index.html, about.html (в \templates)

```
<!DOCTYPE html>
<head>
    <title> {{ title }} </title>
</head>
<body>
<H1> The very main page </H1>
<H2> Главная страница </H2>
<H3> {{title}} </H3>
<u1>
{% for m in menu %}
<li> {{m}} </li>
{% endfor %}
</u1>
</body>
</html>
```

```
<!DOCTYPE html>
<head>
    <title> О сайте (about)
</title>
</head>
<body>
<H1> О сайте (about)
</H1>
</body>
</html>
```

Лучший способ воспользоваться шаблонизатором

- # создадим файл в папке   /templates/index.html
- <html>
- <head>
- <title>{{title}} - microblog</title>
- </head>
- <body>
- <h1>Hello, {{user.nickname}}!</h1>
- </body>
- </html>

HTML5

HTML5 – стандарт языка гипертекстовой разметки. Служит для структурирования и представления материалов в сети WWW

```
# пример простой  страницы
```

```
<!DOCTYPE html>          <!-- тип документа -->
<html>                   <!-- начало документа -->
  <head>                   <!-- начало заголовка -->
    <meta charset="utf-8">
    <title>Главная страница</title>
  </head>
  <body>                   <!-- тело документа -->
      Привет
  </body>
</html>                    <!-- окончание документа -->
```

CSS3

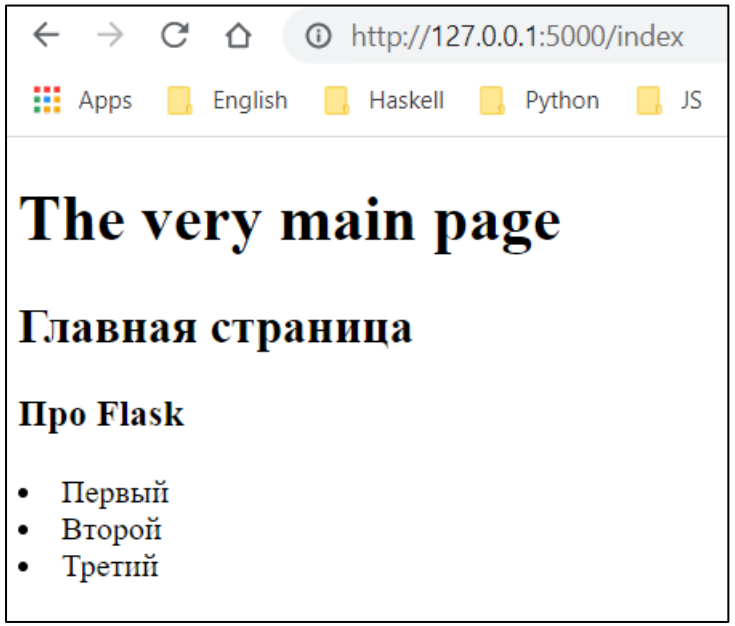
- CSS3 – каскадные таблицы стилей предназначены для задания элементам оформления: размеры блоков, фон, цвета, рамки, отступы, шрифты, эффекты и т.д.
- <!DOCTYPE HTML>
- <html>
- **<head>**
- <meta charset="utf-8">
- <title>Глобальные стили</title>
- <style> <!-- способ 1 , в загловке html –>
- H1 {
- font-size**:** 120%;
- font-family: Verdana, Arial, Helvetica, sans-serif;
- color: #333366;
- }
- </style>
- **</head>**
- <body>
- <h1>Hello, world!</h1>
- </body>
- </html>

Добавим функцию render_template

- В функцию render_template передаем шаблон и данные. Функция сама сопоставляет данные с метками в шаблоне и в итоге отдает сгенерированную страницу.

- from flask import **render_template**
- from app import app
- @app.route('/')
- @app.route('/index')
- def index():
- user = { 'nickname': 'Miguel' } # выдуманный пользователь
- return **render_template**("index.html",
- title = 'Home',
- user = user)

Вторая программа



Измените что-нибудь в обоих шаблонах, затем нажмите «Обновить»

Где что и как

- Создаем шаблоны страницы в html и храним их в папке **templates**
- Медия ресурсы css, img, audio, video и т.д храним в папке **static**
- Отдаем красивую страницу через шаблонизатор Jinja методом **render_template**

Задание

Добавьте страницу help с заголовком, с текстом.

Добавьте в программу вызов этой страницы.

Проверьте работоспособность программы, переключите с index на help и т.д.

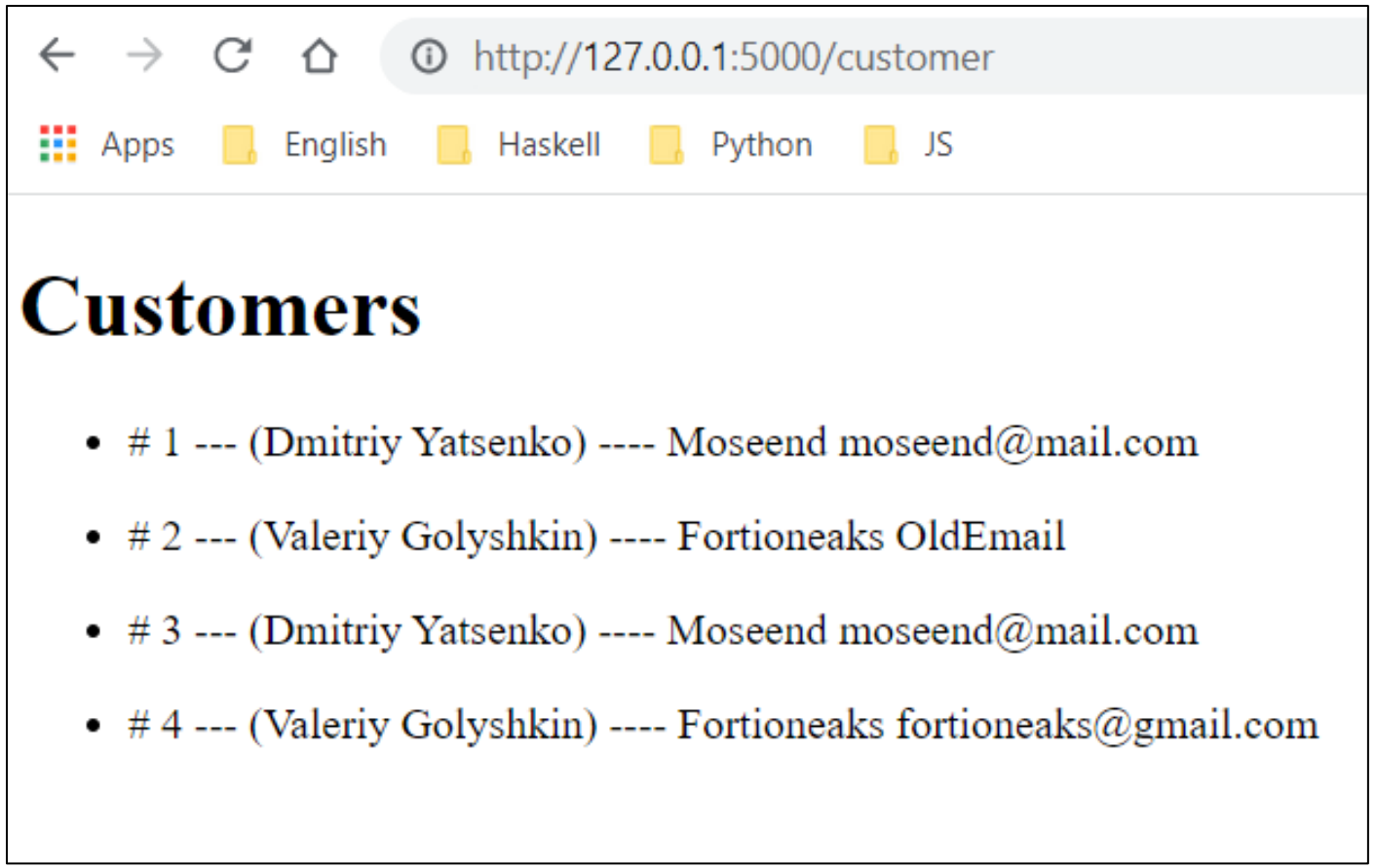
Flask, SQLAlchemy, Postgresql

```
from flask_sqlalchemy import SQLAlchemy
...
engine = create_engine("postgresql+psycopg2://postgres:1111@localhost/sqlalchemy_tuts")
...
q = session.query(Customer)
...
app = Flask(__name__)

@app.route('/index')
@app.route('/')
def index():
    return render_template('index.html', title = 'Про Flask', menu = menu)
@app.route('/about')
def about():
    return render_template('about.html', title = 'О сайте')
@app.route('/customer')
def customer():
    return render_template('customer.html', title = 'Customers', list = q)
```

Customer.html

```
<!DOCTYPE html>
<head>
   <title>Customers</title>
</head>
<body>
<H1> Customers </H1>
<u1>
{% for m in list %}
   <ul>
      <li> # {{m.id}}  ---  ({{m.first_name}} {{m.last_name}})  ----  {{m.username}} {{m.email}} </li>

   </ul>
{% endfor %}
</u1>
</body>
</html>
```

Страница Customers



Давайте разобъем строку на две строки

Что происходит и в какой последовательности

HTTP запрос (request) приходит на сервер (web server) разбирается под капотом URL, а затем вызывается конкретный роутер который этот запрос обрабатывает.

Далее идет процесс извлечения данных из БД с помощью модели, это может быть (SQLAlchemy) или простые SQL запросы через библиотеку psycopg2.

Как только мы получили данные, идет вызов шаблонизатора, файлы которого находятся в папке templates.

Файлы как правило имеют расширение .html

Данные файлы включают в себя код разметки, стили , ссылки на медия ресурсы. Все что относится к медия ресурсам хранится в папке static.

После формирования страницы (шаблон + данные из БД) идет ответ сервера (request) из роутера браузеру.

Задание

Создайте сайт о себе любимом.

- Биографические данные
- Профессиональный опыт
- Хобби
- Прочее

Задание 29-1

Дан список, который состоит из одинаковых чисел за исключением одного.

Найдите это число.

Задача 29-2

Дана квадратная матрица, напишите функцию, которая возвращает матрицу, полученную вращением по или против часовой стрелки.

matrix = [[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]

rotate(matrix, «по часовой») # -----> [[7, 4, 1], [8, 5, 2], [9, 6, 3]]

Задача 29-3

Напишите функцию, которая проверяет, являются ли два слова изоморфными.

Два слова изоморфны, если буквам одного слова можно сопоставить (map) буквам другого слова.

True:

CBAABC DEFFED

XXX YYY

RAMBUNCTIOUSLY THERMODYNAMICS

False:

AB CC

XXY XYY

ABAB CD