

< Teach  
Me  
Skills />

# Занятие 31.

1

# Знакомство с Flask

# Минимальное приложение на flask

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Привет, это минимальное приложение на Flask!'

if __name__ == '__main__':
    app.run()
```

# Что такое `app_context`

В Flask **`app_context` (контекст приложения)** - это контекст, который предоставляет доступ к текущему экземпляру приложения Flask во время выполнения приложения. Контекст обеспечивает правильное связывание текущего приложения с областью выполнения кода, позволяя вам получить доступ к различным компонентам Flask, таким как текущий запрос (`request`), текущий ответ (`response`), конфигурация приложения (`app.config`), и многое другое

Когда приложение Flask запускается, оно создает экземпляр Flask и связывает его с глобальным контекстом приложения. Этот глобальный контекст доступен в течение всего времени выполнения приложения и может быть использован в различных частях кода. (фоновые задачи, расширения)

```
with app.app_context():  
    log_db_queries(app)
```

# Что такое app.config

- app.config в Flask представляет собой объект, который хранит конфигурационные переменные приложения. Он используется для хранения различных параметров настройки, таких как параметры базы данных, секретные ключи, пути к файлам, параметры отладки и многое другое.
- app.config является экземпляром класса Config из Flask, который предоставляет методы для установки и получения значений конфигурационных переменных. Эти переменные можно настраивать либо напрямую в коде вашего приложения, либо с помощью внешнего файла конфигурации, который может быть загружен при запуске приложения.

```
app = Flask(__name__)  
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'  
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

# Как работает request во flask

- В Flask объект **request** представляет текущий HTTP-запрос, который был отправлен на сервер при обработке запроса клиента. Он содержит информацию о различных аспектах запроса, таких как URL, метод запроса, параметры, заголовки и тело запроса.
- Когда клиент отправляет HTTP-запрос на ваш сервер Flask, Flask автоматически создает объект request и делает его доступным в контексте запроса. Это означает, что вы можете обращаться к объекту request в любой обработчик маршрута или представлении без явной передачи этого объекта как аргумента функции.



The background is a vibrant yellow field filled with a repeating pattern of various geometric shapes. These include circles, squares, triangles, and lines in different shades of yellow and orange. Some shapes are solid, while others are outlined or have a dotted texture. The overall effect is a modern, energetic, and abstract design.

2

# SQLAlchemy

# SQLAlchemy

Основные компоненты SQLAlchemy:

1. **Core (Ядро SQLAlchemy):** Ядро предоставляет низкоуровневый интерфейс для работы с базой данных, позволяя выполнять SQL-запросы и взаимодействовать с таблицами и структурами базы данных напрямую.
2. **ORM (Object-Relational Mapping):** SQLAlchemy ORM предоставляет высокоуровневый интерфейс для работы с базой данных через классы Python. Он позволяет определять классы, которые соответствуют таблицам в базе данных, и автоматически создавать соответствующие SQL-запросы для взаимодействия с данными.



# SQLAlchemy

- **Table (Таблица):** Класс Table представляет таблицу в базе данных и используется для определения ее структуры, таких как имена столбцов и типы данных.
- **Mapper (Отображение):** Маппер - это объект, который связывает класс Python с таблицей в базе данных. Он определяет отображение между атрибутами класса и столбцами таблицы.
- **Session (Сессия):** Сессия представляет собой контекст взаимодействия с базой данных. Она позволяет выполнять операции чтения, записи и изменения данных в базе данных с помощью объектов классов, а также управлять транзакциями

# SQLAlchemy Mapper

**Маппер (Mapper)** в SQLAlchemy является одним из ключевых компонентов ORM и представляет собой механизм, который связывает классы Python с таблицами в базе данных. Маппер определяет отображение (mapping) между атрибутами класса и столбцами таблицы, что позволяет вам взаимодействовать с данными в базе данных, используя объектно-ориентированный подход.

# SQLAlchemy Mapper

- 1. Отображение классов на таблицы:** Маппер позволяет определить соответствие между атрибутами и методами класса Python и столбцами в таблице базы данных. Это позволяет ORM автоматически создавать SQL-запросы и выполнять операции CRUD (создание, чтение, обновление, удаление) над данными с использованием объектов Python, а не напрямую с использованием SQL.
- 2. Управление связями:** Маппер позволяет определять отношения между разными таблицами и управлять связями между ними. Например, вы можете определить отношение "один-к-одному", "один-ко-многим" или "многие-ко-многим" между классами и таблицами.
- 3. Преобразование типов данных:** Маппер также обеспечивает преобразование данных между типами Python и типами данных базы данных. Например, он может автоматически преобразовывать даты и времена из объектов Python в строки в формате, поддерживаемом БД.

# Django ORM vs SQL Alchemy (Django)

1. Скорость разработки
2. Хорошая документация и комьюнити
3. Из коробки защищает от CSRF, SQL-инъекций, XSS
4. Активно разрабатывается и поддерживается
5. Active record - каждая строка в бд будет обернута в отдельный python-объект, нет необходимости определять отдельно схему бд, она должна быть понятна просто при взгляде на код моделей
6. Не справится со сложными запросами
7. Нельзя использовать отдельно от django
8. Много магии
9. Есть интеграция форм, подходит больше для веба
10. Тяжелый

# Django ORM vs SQL Alchemy (Alchemy)

1. Data mapper - позволяет не привязывать строку к python объекту, то есть он более гибкий, это средний уровень между приложением и бд и передает данные между этими двумя, сохраняя при это одно независимо от другого, это помогает составлять сложные запросы не через sql код
2. Позволяет создавать очень сложные запросы
3. Сложнее настроить



# Alembic

**Alembic** - это инструмент для управления миграциями баз данных в Python. Он предоставляет средства для создания и применения изменений схемы базы данных, что делает процесс обновления структуры базы данных более управляемым и автоматизированным.

- **Миграции (Migrations):** Миграции представляют собой изменения структуры базы данных, такие как добавление новых таблиц, столбцов, индексов или удаление существующих объектов. Alembic позволяет создавать миграции, которые представляют эти изменения в виде кода на языке Python.
- **Версионирование (Versioning):** Alembic управляет версиями миграций, что позволяет легко контролировать последовательность применения изменений и обеспечивает согласованность между структурой базы данных и кодом приложения.

# Alembic

- **Скрипты миграций (Migration Scripts):** Миграции в Alembic представляют собой Python-скрипты, которые определяют изменения структуры базы данных. Эти скрипты могут быть применены или отменены (откаты), чтобы перейти к определенной версии базы данных.
- **Интеграция с SQLAlchemy:** Alembic тесно интегрирован с SQLAlchemy, что позволяет легко использовать SQLAlchemy для определения структуры базы данных и создания соответствующих миграций.

ЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫ  
ЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫ  
ЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫ  
ЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫ  
ЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫ  
ЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫ  
ЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫ  
ЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫЫ  
333333333333333333333333  
333333333333333333333333  
3

Elena Dejkun



3

**Marshmallow**

# Marshmallow

Библиотека для сериализации и десериализации данных в Flask и других веб-приложениях на Python. Основное предназначение Marshmallow - преобразование сложных структур данных, таких как объекты Python, в форматы данных, которые могут быть легко переданы по сети (например, JSON) и обратно

Marshmallow часто используется в Flask для обработки данных, передаваемых в формате JSON или из форм на веб-страницах. Он позволяет легко определить схемы данных (так называемые "схемы Marshmallow"), которые определяют структуру данных, валидацию и преобразование между сложными объектами Python и простыми форматами данных, такими как JSON.

# Marshmallow - главные функции

1. **Сериализация (Serialization):** Преобразование сложных объектов данных в простые данные, такие как строки JSON. Это полезно, когда вам нужно отправить данные клиенту в формате, который клиент понимает.
2. **Десериализация (Deserialization):** Преобразование простых данных (например, строки JSON) обратно в сложные объекты Python. Это полезно при получении данных от клиента и преобразовании их обратно в структуру, с которой вы можете работать в вашем приложении.
3. **Валидация (Validation):** Проверка данных на соответствие определенным правилам или схемам для обеспечения их целостности и безопасности.



# Пример использования

```
from marshmallow import Schema, fields

1 usage
class UserSchema(Schema):
    id = fields.Int()
    username = fields.Str()
    email = fields.Email()

user_data = {
    'id': 1,
    'username': 'john_doe',
    'email': 'john@example.com'
}

user_schema = UserSchema()
result = user_schema.dump(user_data)
```

4

# Домашнее задание

# Домашнее задание

- Создать модель пользователя, форму для создания пользователя и страницу для отображения всех пользователей
- При отображении списка пользователей у каждого пользователя должно быть две кнопки: просмотреть идеи, сгенерировать идею
- При нажатии на кнопку сгенерировать идею будет запрашиваться <https://www.boredapi.com/api/activity> и идея будет сохраняться в бд и привязываться к пользователю (идея - отдельная модель в бд)
- Кнопка “просмотреть идеи” будет переводить на страницу с идеями пользователя

The background is a vibrant yellow color, covered with a dense, repeating pattern of various geometric shapes. These shapes include circles, squares, triangles, and lines, some of which are filled with a fine grid pattern. The shapes are scattered across the entire surface, creating a dynamic and modern aesthetic.

**Спасибо за внимание!**