

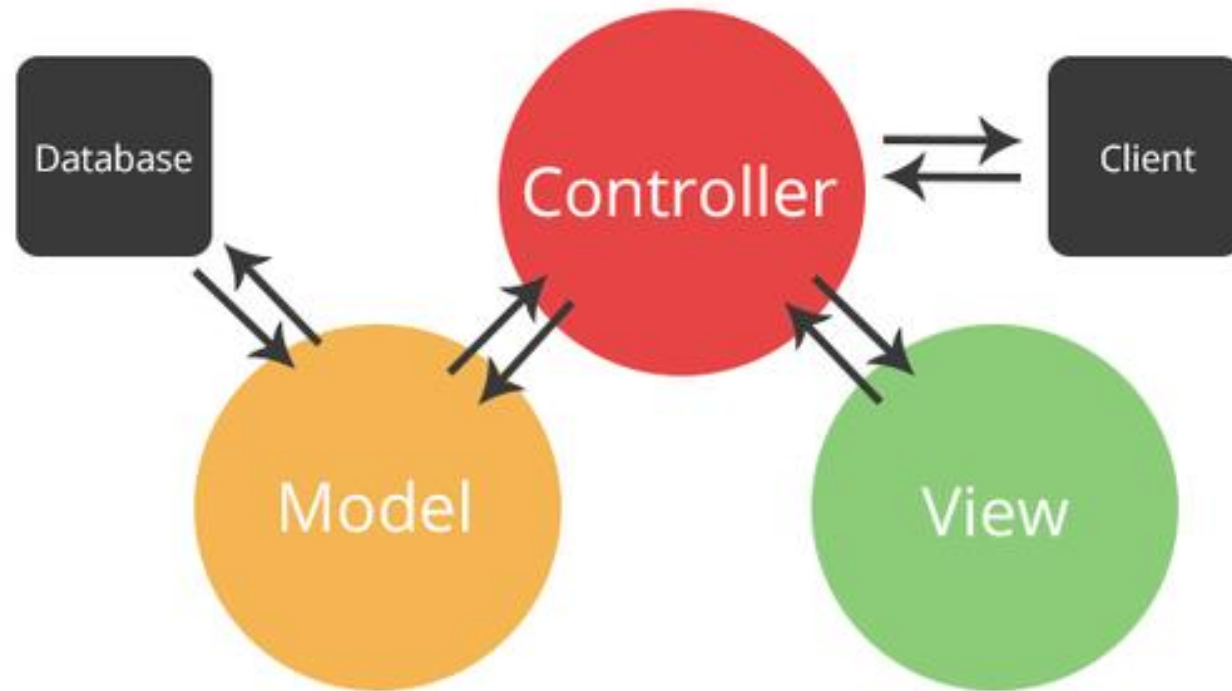


MVC
Python
Flask



MVC

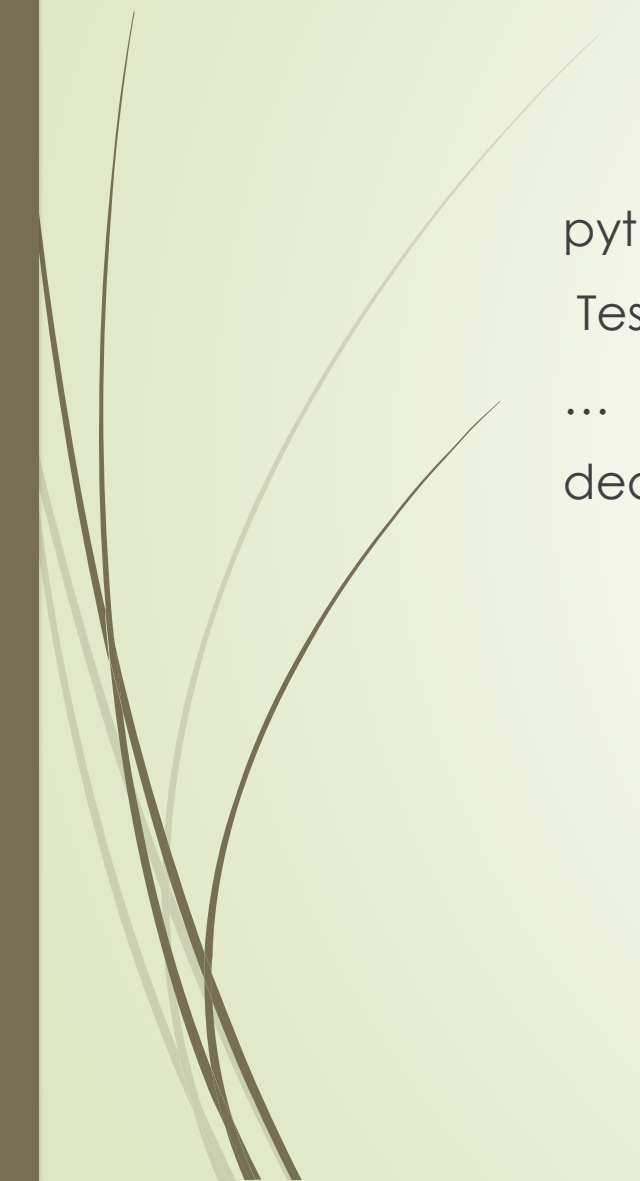

- Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента `model` будет определять список задач и отдельные задачи.
- Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента `view` определяет внешний вид приложения и способы его использования.
- Контроллер — этот компонент отвечает за связь между `model` и `view`. Код компонента `controller` определяет, как сайт реагирует на действия пользователя.





virtualenv

- Представим следующий сценарий, где у вас есть два проекта: **проект А** и **проект Б**, которые оба имеют зависимость от одной и той же библиотеки – проект В. Проблема становится явной, когда мы начинаем запрашивать **разные версии** проекта В. Может быть так, что проект А запрашивает версию 1.0.0, в то время как проект Б запрашивает более новую версию 2.0.0, к примеру.
- главная задача виртуальной среды Python – создание **изолированной среды** для проектов Python.



```
python -m venv Test
```

```
Test/Scripts/activate.bat // source Test/bin/activate
```


```
...
```

```
deactivate
```



Сквозная функциональность

- Идентификация пользователей/ролей, авторизация операций
- Функции защиты, логирования, репликации
- Сериализация/десериализация
- Трассировка
- Управление ресурсами
- Кэширование объектов
- Декорирование
- Изменение контекста выполнения
- Мониторинг, аудит




Аспектно-ориентированное программирование

- Аспектно-ориентированное программирование (АОП) — методология программирования, основанная на идее декомпозиции “сквозной” функциональности.
- JoinPoint — строго определённая точка выполнения программы, ассоциированная с контекстом выполнения (вызов метода, конструктора, доступ к полю класса, обработчик исключения, и т.д.)
- Pointcut — набор (срез) точек JoinPoint, удовлетворяющих заданному условию.
- Advice — набор инструкций, выполняемых до, после или вместо каждой из точек выполнения (JoinPoint), входящих в заданный срез (Pointcut).
- Aspect — основная единица модульности АОП, инкапсулирующая срезы точек выполнения (Pointcut), наборы инструкций (Advice).



Декораторы


```
def my_shiny_new_decorator(function_to_decorate):  
    def the_wrapper_around_the_original_function():  
        print("Код, который отработает до вызова функции")  
        function_to_decorate() # Сама функция  
        print("Код, срабатывающий после")  
    return the_wrapper_around_the_original_function  
  
def stand_alone_function():  
    print("Простая функция")  
  
stand_alone_function = my_shiny_new_decorator(stand_alone_function)
```

```
@my_shiny_new_decorator  
def another_stand_alone_function():  
    print("Декарируемая функция")
```

```
another_stand_alone_function =  
    my_shiny_new_decorator(another_stand_alone_function)
```

Один из важных фактов, которые следует понимать, заключается в том, что функции и методы в Python — это практически одно и то же, за исключением того, что методы всегда ожидают первым параметром ссылку на сам объект (*self*). Это значит, что мы можем создавать декораторы для методов точно так же, как и для функций, просто не забывая про *self*.



Декораторы, принимающие параметры

```
def decorator_maker_with_arguments(decorator_arg1, decorator_arg2):  
    print("Создание декоратора, аргументы:", decorator_arg1,  
          decorator_arg2)  
    def my_decorator(func):  
        print("Декоратор аргументы:", decorator_arg1,  
              decorator_arg2)  
        def wrapped(function_arg1, function_arg2):  
            print ("Обёртка вокруг декорируемой функции")  
            return func(function_arg1, function_arg2)  
        return wrapped  
    return my_decorator
```

Микрофреймворк Flask

■ Установка

```
pip install Flask
```

■ Пример приложения

```
from flask import Flask
```

```
app = Flask(__name__)
```

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

```
app.run(debug = True, host='127.0.0.1', port='5050')
```



Изменяемые правила

```
from markupsafe import escape
```

```
@app.route('/user/<username>')  
def show_user_profile(username):  
    return 'User %s' % escape(username)
```

```
@app.route('/post/<int:post_id>')  
def show_post(post_id):  
    return 'Post %d' % post_id
```

```
@app.route('/path/<path:subpath>')  
def show_subpath(subpath):  
    return 'Subpath %s' % escape(subpath)
```



Поддерживаемые типы

string	Любая строка без слэшей
int	Положительное целое число
float	Положительное вещественное число
path	Строка, может содержать слэши
uuid	UUID-строка

uuid (universally unique identifier)

- ▶ Любой узел распределенной системы может создать UUID и использовать его для идентификации чего-либо с приемлемым уровнем уверенности, что данный идентификатор непреднамеренно никогда не будет использован для чего-то ещё. Поэтому информация, помеченная с помощью UUID, может быть помещена позже в общую базу данных, без необходимости разрешения конфликта имен.
- ▶ Пример: 123e4567-e89b-12d3-a456-426655440000

Название поля	Длина (в байтах)	Длина (число 16-ричных цифр)	Содержимое
time_low	4	8	целое число, обозначающее младшие 32 бита времени
time_mid	2	4	целое число, обозначающее средние 16 бит времени
time_hi_and_version	2	4	4 старших бита обозначают версию UUID, младшие биты обозначают старшие 12 бит времени
clock_seq_hi_and_res clock_seq_low	2	4	1-3 старших бита обозначают вариант UUID, остальные 13-15 бит обозначают clock sequence
node	6	12	48-битный идентификатор узла



JSON

- Для ответа в формате json используется метод:
`jsonify(data)`
- В качестве данных можно передавать любой сериализуемый тип
- Метод создает ответ со статусом 200 и заголовком, в котором указан mime-тип данных:

`mimetype='application/json'`



Методы HTTP

```
from flask import request
```

```
@app.route('/login', methods=['GET', 'POST'])
```

```
def login():
```

```
    if request.method == 'POST':
```

```
        return do_the_login()
```

```
    else:
```

```
        return show_the_login_form()
```



request

- ▶ method

- ▶ form

```
request.form['username']
```

- ▶ args

```
searchword = request.args.get('key', '')
```

- ▶ files

```
f = request.files['the_file']
```

```
f.save('/var/www/uploads/uploaded_file.txt')
```

- ▶ url

- ▶ cookies



session

```
app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'
```

```
@app.route('/login', methods=['GET', 'POST'])
```

```
def login():
```

```
    if request.method == 'POST':
```

```
        session['username'] = request.form['username']
```

```
        return redirect(url_for('index'))
```

```
    return "<form method='post' action='login'>
```

```
        <p><input type='text' name='username'></p>
```

```
    </form>''
```

```
@app.route('/logout')
```

```
def logout():
```

```
    session.pop('username', None)
```

```
    return redirect(url_for('index'))
```



g

```
from flask import g
```

```
def get_db():  
    if 'db' not in g:  
        g.db = connect_to_database()  
    return g.db
```

```
@app.teardown_appcontext  
def teardown_db():  
    db = g.pop('db', None)  
    if db is not None:  
        db.close()
```



Кастомные страницы ошибок

Общий формат ответа: (response, status, headers)

```
@app.errorhandler(404)
def page_not_found(e):
    return '404', 404
```

Генерация в контроллере:

```
abort(404)
```



Встроенные декораторы

- `before_first_request`: этот декоратор выполняет функцию еще до обработки первого запроса
- `before_request`: выполняет функцию до обработки запроса
- `after_request`: выполняет функцию после обработки запроса. Такая функция не будет вызвана при возникновении исключений в обработчике запросов. Она должна принять объект ответа и вернуть тот же или новый ответ.
- `teardown_request`: этот декоратор похож на `after_request`. Но вызванная функция всегда будет выполняться вне зависимости от того, возвращает ли обработчик исключение или нет.



Использование шаблонизатора

```
@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

```
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
    <h1>Hello {{ name }}!</h1>
{% else %}
    <h1>Hello, World!</h1>
{% endif %}
```


- 
- Переменные, выражения и вызовы функций

```
{{ }}
```

- Комментарии

```
{# комментарий #}
```

- Объявление переменных

```
{% set fruit = 'apple' %}
```

- Условный оператор

```
{% if user.newbie %}
```

```
    <p>Display newbie stages</p>
```

```
{% elif user.pro %}
```

```
    <p>Display pro stages</p>
```


```
{% elif user.ninja %}
```

```
    <p>Display ninja stages</p>
```

```
{% else %}
```

```
    <p>You have completed all stages</p>
```

```
{% endif %}
```

- 
- Запись в одну строку

```
{{ "User is logged in" if loggedin else "User is not logged in" }}
```

- Цикл for

```
{% for user in user_list %}
```

```
    <li>{{ user }}</li>
```

```
{% else %}
```

```
    <li>user_list is empty</li>
```

```
{% endfor %}
```



Переменная loop

- Цикл for предоставляет специальную переменную loop для отслеживания прогресса цикла.

loop.index - возвращает номер текущей итерации

loop.index0 - то же самое что и loop.index, но с индексом 0, то есть, начинает считать с 0, а не с 1.

loop.revindex - возвращает номер итерации с конца цикла (считает с 1).

loop.revindex0 - возвращает номер итерации с конца цикла (считает с 0).

loop.first - возвращает True, если итерация первая. В противном случае — False.

loop.last - возвращает True, если итерация последняя. В противном случае — False.

loop.length - возвращает длину цикла(количество итераций).

Фильтры

- Фильтры изменяют переменные до процесса рендеринга. Синтаксис использования фильтров следующий:

`variable_or_value | filter_name`

upper	делает все символы заглавными
lower	приводит все символы к нижнему регистру
capitalize	делает заглавной первую букву и приводит остальные к нижнему регистру
escape	экранирует значение
safe	предотвращает экранирование
length	возвращает количество элементов в последовательности
trim	удаляет пустые символы в начале и в конце
random	возвращает случайный элемент последовательности



Вложенные шаблоны

```
<nav>
  <a href="/home">Home</a>
  <a href="/blog">Blog</a>
  <a href="/contact">Contact</a>
</nav>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Title</title>
  </head>
  <body>
    {% include 'nav.html' %}
  </body>
</html>
```



Макросы

```
{% import "macros.html" as macros %} #или {% from "macros.html" import render_posts %}
```

```
{% macro render_posts(post_list, sep=False) %}
```

```
<div>
```

```
  {% for post in post_list %}
```

```
    <h2>{{ post.title }}</h2>
```

```
    <article>
```

```
      {{ post.html | safe }}
```


```
    </article>
```

```
  {% endfor %}
```

```
  {% if sep %}<hr>{% endif %}
```

```
</div>
```

```
{% endmacro %}
```



```
{% macro custom_renderer(para) %}
```

```
<p>{{ para }}</p>
```

```
<p>varargs: {{ varargs }}</p>
```

```
<p>kwargs: {{ kwargs }}</p>
```

```
{% endmacro %}
```

```
{{ custom_renderer("some content", "apple", name='spike', age=15) }}
```




Наследование шаблонов

```
<body>
```

```
  {% block content %}
```

```
  {% endblock %}
```

```
</body>
```

```
{% extends 'base.html' %}
```

```
{% block content %}
```

```
  {% for bookmark in bookmarks %}
```

```
    <p>{{ bookmark.title }}</p>
```

```
  {% endfor %}
```

```
{% endblock %}
```



Перезапись блока

```
{% block nav %}
```

```
{{ super() }}
```

```
<li><a href="/contact">Contact</a></li>
```

```
<li><a href="/career">Career</a></li>
```

```
{% endblock %}
```



make_response

При возвращении данных методом `render_template` невозможно добавить к данным какие-либо заголовки, установить куки и т.д.

```
def index():  
    response = make_response(render_template('index.html', foo=42))  
    response.headers['X-Parachutes'] = 'parachutes are cool'  
    return response
```



url_for

- `url_for('books', genre='biography', page=2, sort_by='date-published')`
`'/books/biography/?page=2&sort_by=date-published'`
- `url_for()` можно использовать внутри шаблона
- Чтобы сгенерировать абсолютной URL, нужно передать функции `url_for()` аргумент `external=True`
- Вместо того чтобы прописывать URL в функции `redirect()`, стоит всегда использовать `url_for()` для этого.



Статические файлы

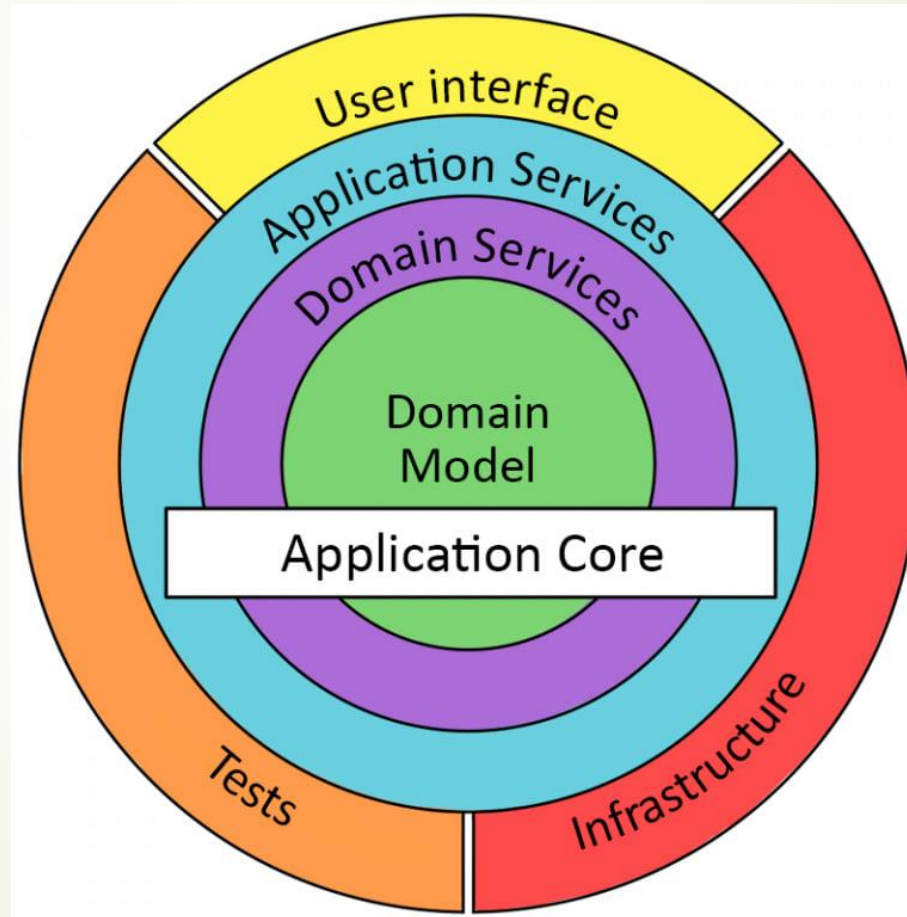
- По умолчанию директория static, обращение в шаблоне:

```
<script src="{{ url_for('static', filename='js/jquery.js') }}"></script>
```

- Смена директорий по умолчанию:

```
app = Flask(__name__, static_folder="static_dir",  
            template_folder="template_dir")
```

Onion architecture





Структура приложения

/

static/

css/

js/

templates/

__init__.py (инициализация приложения)

controllers.py

model.py

dao.py

run.py (создание экземпляра приложения)



init

```
from flask import Flask, render_template, request  
from flaskext.mysql import MySQL
```

```
app = Flask(__name__)  
from app import controllers  
app.config['SECRET_KEY'] = 'TestKey1'
```

```
# MySQL configurations  
app.config['MYSQL_DATABASE_USER'] = 'sysdba'  
app.config['MYSQL_DATABASE_PASSWORD'] = 'password'  
app.config['MYSQL_DATABASE_DB'] = 'db'  
app.config['MYSQL_DATABASE_HOST'] = 'localhost'  
mysql.init_app(app)  
app.conn = mysql.connect()
```

run.py

```
from app import app  
app.run(debug = True)
```