Проект WebServer. Введение

План урока

Аннотация

На этом занятии мы начинаем проектный блок, посвящённый созданию веб-приложений. Первый урок — знакомство с библиотекой Flask и работа со статическим контентом.

1. Введение

Мы с вами уже много-много раз писали программы, которые работают только на одном компьютере и никак не взаимодействуют с внешним миром. В последние годы приложения,

которые функционируют в пределах только одной машины и не смотрят «по сторонам», стали достаточно большой редкостью. Практически в каждую, даже самую маленькую утилиту автор или авторы старается встроить как минимум механизм автоматического обновления, который поможет быстро доставлять пользователям модули, в которых добавлен новый функционал или исправлены ошибки. Самыми массовыми приложениями для взаимодействия в сети Интернет и корпоративных сетях Интранетах являются веб-приложения, от небольших сайтов, до огромных порталов, которые обрабатывают информацию из сотен источников. Но прежде чем мы начнем писать собственные веб-сайты, небольшое «лирическое отступление».

Все из вас наверняка знают, что такое компьютерная сеть, но не все задумывались о механизмах взаимодействия устройств внутри сети, ведь даже физическая среда распространения сигналов может быть сильно различна:

- беспроводная среда (WiFi, 4G, Bluetooth);
- обычные провода;
- телефонные провода;
- оптические кабели и т.д.

Разумеется, что при написании своего приложения большинство программистов не задумывается о том, по каким физическим каналам будет происходить взаимодействие. Это стало возможным благодаря стандартизации в этой области и чётким разделениям уровней взаимодействия.

Сетевая модель OSI (open systems interconnection basic reference model) представляет собой набор **протоколов** (стандартов, описывающих правила взаимодействия разных частей систем при передаче данных), каждый из которых отвечает за определённый уровень взаимодействия. Модель имеет семь уровней:

Модель OSI			
Уровень (layer)	**Тип данных**	**Функции**	**Примеры**
7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, POP3
6. Представительский (представления) (presentation)		Представление и шифрование данных	ASCII, EBCDIC
5. Сеансовый (session)		Управление сеансом связи	RPC, PAP
4. Транспортный (transport)	Сегменты / Дейтаграммы	Прямая связь между конечными пунктами и надежность	TCP, UDP, SCTP, PORTS
3. Сетевой (network)	Пакеты(packet)	Определение маршрута и логическая адресация	IPv4, IPv6, IPsec, AppleTalk
2. Канальный (data link)	Биты / Кадры	Физическая адресация	PPP, IEEE 802.22, Ethernet, DSL, ARP
1. Физический (physical)	Биты	Работа со средой передачи, сигналами и двоичными данными	USB, "витая пара", оптоволоконный кабель, радиоканал

1. **Физический**, на котором происходит преобразование двоичных данных в вид, пригодный для передачи в среде (USB, витая пара, WiFi).

2. **Канальный**, на котором происходит физическая адресация, например, с использованием MAC-адресов сетевой платы.
3. **Сетевой**, на котором происходит логическая адресация. Сюда относятся хорошо известные протоколы IPv4 и IPv6.
4. **Транспортный** — для обеспечения связи между конечными точками и обеспечения надёжности.
5. **Сеансовый** — для обеспечения сеанса связи.
6. **Представительский (представления)** для представления и шифрования/дешифрования данных.

На вышеперечисленные уровни обычные разработчики прикладных программ забираются нечасто, обычно при написании специализированного программного обеспечения и сетевых игр. Протоколы же последнего, седьмого, уровня нужны в повседневной работе гораздо чаще.

7. **Прикладной**, на котором происходит доступ к сетевым службам (SMTP, FTP, HTTP).

Именно с использованием протоколов прикладного уровня построен «видимый» интернет, хотя при открытии любой странички по протоколу HTTP ваш запрос проходит по всем уровням модели, превращаясь из данных в физические сигналы, затем собираясь в точке приёма обратно для прикладного уровня. Ответ на запрос ждет такая же нелёгкая судьба.

Более подробно про модель OSI можете почитать, например, на Википедии.

Вообще история Интернета довольно увлекательна сама по себе, рекомендуем вам ознакомиться с ней самостоятельно, начать можно, например, опять же на Википедии. Мы в неё вдаваться не будем, отметим лишь, что самый первый веб-сайт появился 6 августа 1991 года с доменным именем info.cern.ch. На этом сайте его создатель, Тим Бернерс-Ли, разместил описание новой технологии World Wide Web, основанной на протоколе HTTP, системе адресации URI и языке гипертекстовой разметки HTML.

2. Инструменты для разработки веб-приложений

В настоящее время разрабатывать веб-сайты можно практически на любом языке программирования, для каждого из которых создано не по одному десятку библиотек, облегчающих те или иные аспекты написания веб-приложений.

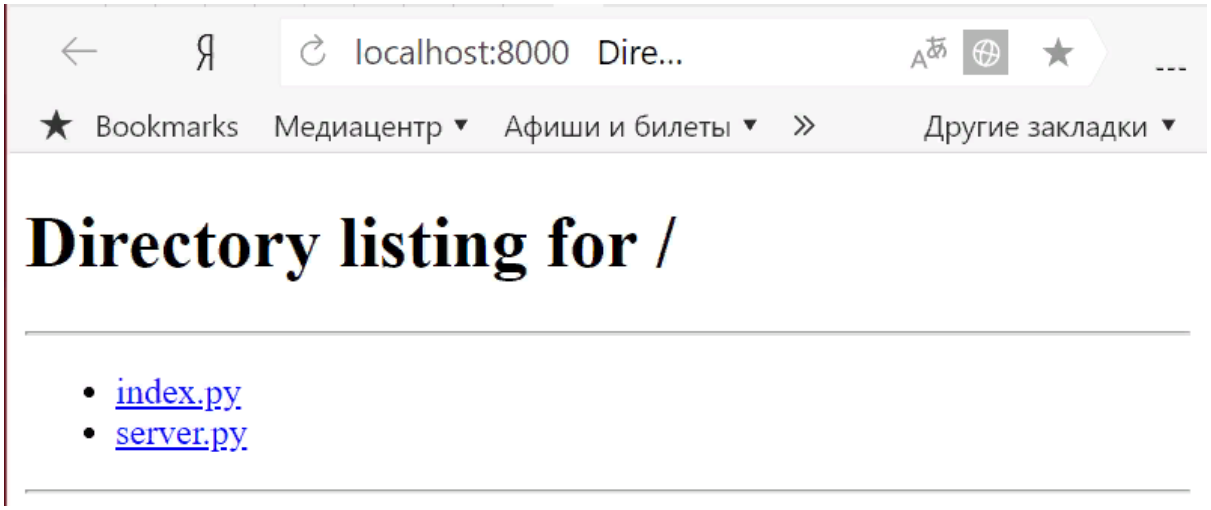
Кстати, многие разработчики мобильных приложений «хитрят» и в целях экономии времени и сил создают веб-приложения просто как завёрнутые в веб-компоненту (встроенный браузер), где вся функциональность реализована как совокупность веб-страниц.

Одной из сильных областей Python является достаточно простое создание веб-страниц. В принципе, веб-приложения на Python можно писать и без установки дополнительных библиотек, только средствами «из коробки», так как интерпретатор поставляется со встроенным CGI (стандарт интерфейса, применяемого для связи внешней программы с веб-сервером) сервером.

Для этого давайте создадим файл **server.py** и напишем там вот такой код:

```
from http.server import HTTPServer, CGIHTTPRequestHandler


server_address = ("", 8000)
httpd = HTTPServer(server_address, CGIHTTPRequestHandler)
httpd.serve_forever()
```

Этот код создает простейший веб-сервер, и если мы перейдем по адресу **localhost:8000** (или **127.0.0.1:8000**), то увидим список файлов той директории, в которой у нас запущен сервер.



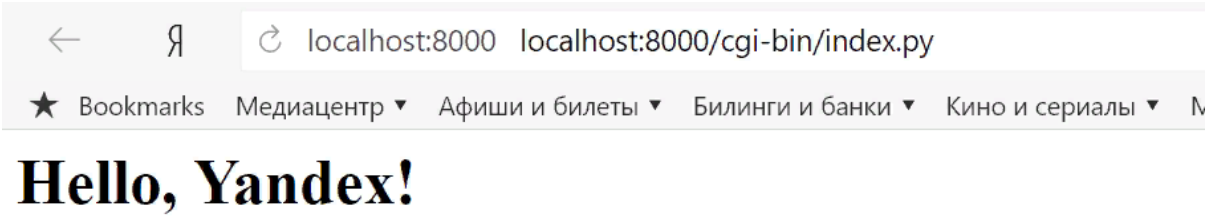
localhost или **127.0.0.1** — это адрес вашего компьютера для самого этого компьютера, а цифры после двоеточия показывают номер порта. **Порт** нужен для того, чтобы на одной машине можно было запустить несколько приложений, которые ожидают сообщений. Если утрировать, то ip-адрес — это адрес только до многоквартирного дома, а порт — номер квартиры. Номер порта — число от 0 до 65536. Для многих протоколов прикладного уровня есть свои стандартные порты, например для HTTP чаще всего используются 80, 8000 и 8080.

Для создания первой веб-страницы надо сделать еще несколько шагов. Давайте создадим рядом с нашим сервером папку с именем **cgi-bin**, внутри которой создадим скрипт с именем **index.py** следующего содержания:

```
#!/usr/bin/python
```

```
print("Content-type: text/html; charset=utf-8")
print()
print("<h1>Hello, Yandex!</h1>")
```

Теперь мы можем перейти по адресу **localhost:8000/cgi-bin/index.py** и увидим результат выполнения нашего скрипта.



На деле такой подход к созданию веб-приложений излишен и неудобен, поэтому мы его применять не будем. Для Python разработано несколько библиотек, которые значительно упрощают процесс создания веб-страниц. Некоторые из этих библиотек большие (Django, Twisted), а некоторые (Bottle, Flask) значительно меньше (их ещё называют микро-фреймворками).

С чем связан размер библиотек? С тем, насколько сильно библиотека управляет архитектурой вашей системы и поддерживает её, насколько развиты её подсистемы, например, интеграция с разными базами данных, настройками, файловыми хранилищами и т.д. Небольшие библиотеки всего лишь скрывают от вас сетевой уровень, давая вам возможность управлять формированием страницы в зависимости от параметров. Остальное вы должны реализовывать сами.

3. Первая страница на Flask

Из-за того, что промышленные порталы мы делать не будем, и у нас будут ресурсы с небольшим количеством страниц, мы ограничимся только микро-фреймворком **Flask** как наиболее популярным. После его освоения вам не составит труда пощупать Bottle самостоятельно, так как принципы работы у них очень похожи.

Чтобы установить Flask, необходимо выполнить команду:

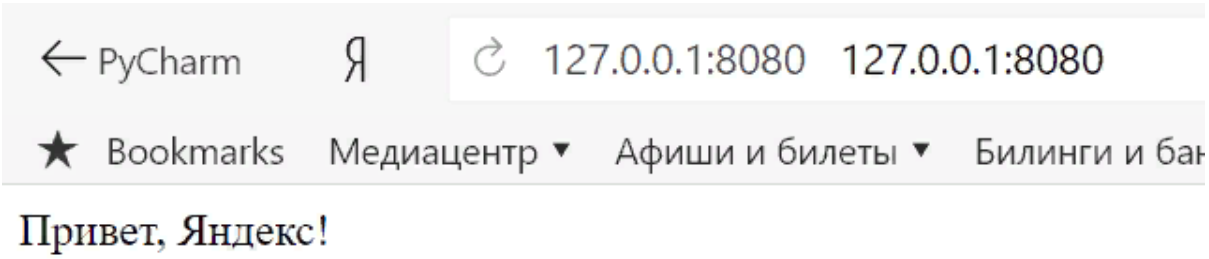
```
pip install flask
```

После установки библиотеки давайте создадим наше первое веб-приложение на Flask:

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
@app.route('/index')
def index():
    return "Привет, Яндекс!"


if __name__ == '__main__':
    app.run(port=8080, host='127.0.0.1')
```

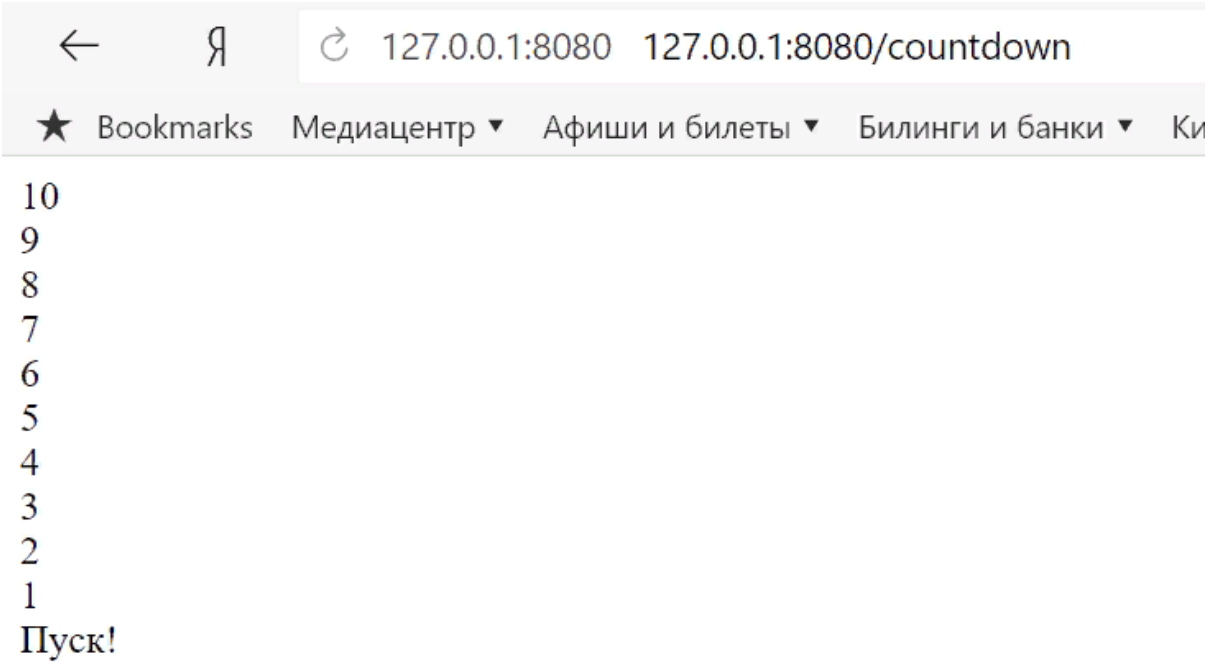
Наш скрипт создаёт объект приложения как экземпляр класса **Flask**, который мы предварительно импортировали из пакета **flask**. Декораторы **app.route** над функцией **index** используются для регистрации нашей функции как функции обратного вызова для определённых событий. В нашем случае создаётся связь между адресом (URL) в браузере (’/’ и ’/index’) и функцией **index()**. Это означает, что когда веб-браузер запрашивает один из этих двух URL-адресов, Flask будет вызывать эту функцию и передавать возвращаемое значение обратно в браузер в качестве ответа.

Давайте запустим наш скрипт и перейдём на страницу http://127.0.0.1:8080, что соответствует URL ’/’ или на страницу http://127.0.0.1:8080/index, чтобы увидеть результат выполнения функции index.



С помощью декоратора app.route мы можем создавать сколько угодно страниц со своими собственными адресами. Давайте добавим ещё одну страницу с обратным отсчётом:

```
@app.route('/countdown')
def countdown():
    countdown_list = [str(x) for x in range(10, 0, -1)]
    countdown_list.append('Пуск!')
    return '</br>'.join(countdown_list)
```



4. Статический контент

Практически любой веб-сайт содержит большое количество разнообразного статического контента, к которому могут относится следующие виды информации:

- изображения;
- таблицы стилей CSS;
- шрифты;
- файлы для скачивания;
- файлы скриптов на языке JavaScript;
- музыка, видео
- и т.д.

Всю подобную информацию Flask ищет в директории **static**, поэтому давайте её создадим. Для более аккуратной организации файлов рекомендуется создавать подпапки в static для каждого типа статического контента, который у вас есть, например, static/img, static/fonts.

Давайте разместим хорошо известную нам сову Риану на отдельной странице.

Для этого в папке static создадим подпапку img, разместим там наше изображение, импортируем из модуля flask **url_for**. И напишем следующий код:

```
@app.route('/image_sample')
def image():
    return '''<img src="{}" alt="здесь должна была быть картинка,
    но не нашлась">'''.format(url_for('static', filename='img/Риана.jpg'))
```

Если вы посмотрите код страницы в браузере, то заметите, что имя картинки, которое было на русском языке, перекодировалось в коды символов Unicode.

В принципе, никто нам не запрещает написать путь к файлу вот так:

```
return '''<img src="/static/img/Риана.jpg" alt="здесь должна была быть
картинка, но не нашлась">'''
```

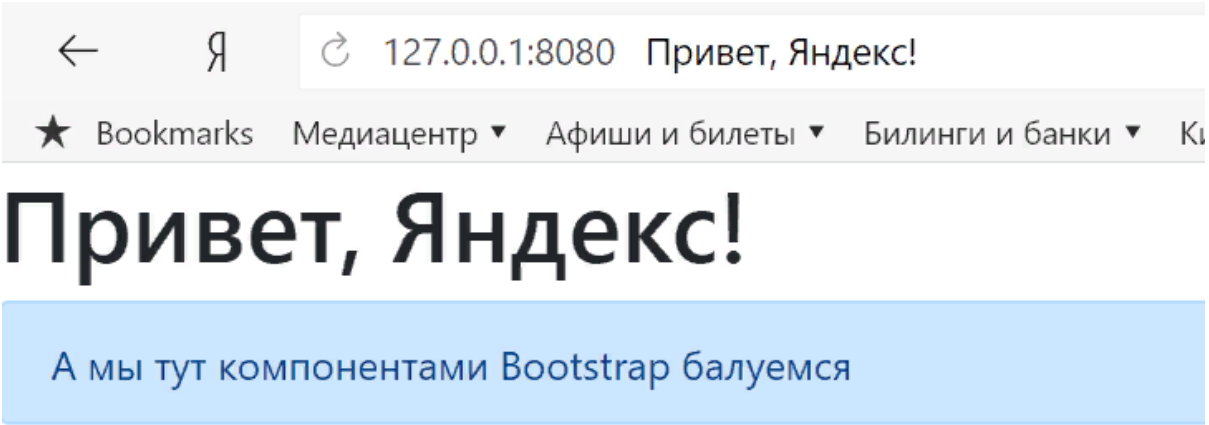
Однако при создании относительно большого проекта, вы можете столкнуться с тем, что ваши пути перестанут работать, так как кто-то из ваших коллег переопределил у микро-фреймворка параметр, отвечающий за местонахождение директории со статическими файлами, а наш первоначальный вариант сохранит работоспособность.

Точно по такому же принципу работает и остальной статический контент. Давайте рассмотрим еще ситуацию с таким распространенным типом файлов, как **css**, где хранятся стили нашей страницы. Так как создание стилей и вёрстка не входит в цели нашего курса, давайте обратимся к такому набору инструментов, как **Bootstrap**.

Bootstrap — это свободный набор инструментов для создания сайтов и веб-приложений, который включает в себя скрипты, стили, иконки и многое другое. Набор расширяемый и достаточно гибкий. В своём первоначальном виде частенько используется программистами для того, чтобы создать веб-приложение без участия дизайнеров и верстальщиков.

Инструкцию по подключению и использованию Bootstrap можно найти на официальном сайте. Давайте сделаем функцию, которая будет нам отдавать простую страничку с подключением Bootstrap и несколькими элементами на ней. Примерно вот так:

```
@app.route('/bootstrap_sample')
def bootstrap():
    return '''<!doctype html>
                <html lang="en">
                  <head>
                    <meta charset="utf-8">
                    <meta name="viewport" content="width=device-width,
                    initial-scale=1, shrink-to-fit=no">
                    <link rel="stylesheet"
                    href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/boo
                    integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmF
                    crossorigin="anonymous">
                    <title>Привет, Яндекс!</title>
                  </head>
                  <body>
                    <h1>Привет, Яндекс!</h1>
                    <div class="alert alert-primary" role="alert">
                      А мы тут компонентами Bootstrap балуемся
                    </div>
                  </body>
                </html>'''
```



Как вы могли заметить, даже создание простых страниц напрямую из кода выглядит громостко, а при небольшом увеличении сложности страницы трудоёмкость написания и поддержки возрастает значительно. Поэтому вариант, который мы сегодня рассматривали, годится только

для того, чтобы попрактиковаться с библиотекой и языком разметки HTML. На следующем уроке мы рассмотрим, как решается эта проблема.

Кроме того, мы добавим в наше приложение некоторую динамику путём передачи информации от пользователя на сервер. В общем-то, мы можем влиять на состояние сервера и сейчас, например, изменяя значение глобальной переменной:

```
from flask import Flask, url_for
i = 0
app = Flask(__name__)


@app.route('/i')
def show_i():
    global i
    i += 1
    return str(i)
```

Но это плохая практика, не делайте так, если от этого не зависит ваша жизнь. Единственное, на что можно обратить внимание в этом примере, это тот факт, что если мы попробуем возвратить этой функцией просто число, то получим страницу с ошибкой следующего содержания: **"TypeError: 'int' object is not callable. The view function did not return a valid response. The return type must be a string, tuple, Response instance, or WSGI callable, but it was a int."**, что недвусмысленно намекнёт нам на то, каким может быть возвращаемое значение у функций, украшенных декоратором app.route.

Напоследок всё же добавим немного интерактива. Декоратор @app.route умеет принимать на вход ещё и значение параметра, который пишется после завершающего слэша в пути внутри треугольных скобок <>, вот так:

```
@app.route('/greeting/<username>')
def greeting(username):
    return '''<!doctype html>
                <html lang="en">
                  <head>
                    <meta charset="utf-8">
                    <meta name="viewport" content="width=device-width,
                    initial-scale=1, shrink-to-fit=no">
                    <link rel="stylesheet"
                    href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/boo
                    integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmF
                    crossorigin="anonymous">
                    <title>Привет, {}</title>
                  </head>
                  <body>
                    <h1>Привет, {}!</h1>
```

```
                </body>
            </html>'''.format(username, username)
```

Обратите внимание, что в данном случае меняется и сигнатура функции, которая теперь тоже принимает на вход одноимённый параметр. Кроме того, этот параметр — обязательный.

Flask позволяет указывать несколько параметров, а также явно определять тип параметра указанием конвертера. Рассмотрим обе эти возможности в одном примере:

```
@app.route('/two_params/<username>/<int:number>')
def two_params(username, number):
    return '''<!doctype html>
            <html lang="en">
              <head>
                <meta charset="utf-8">
                <meta name="viewport" content="width=device-width,
                initial-scale=1, shrink-to-fit=no">
                <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
                integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmF
                crossorigin="anonymous">
                <title>Пример с несколькими параметрами</title>
              </head>
              <body>
                <h2>{}</h2>
                <div>Это первый параметр и его тип: {}</div>
                <h2>{}</h2>
                <div>Это второй параметр и его тип: {}</div>
              </body>
            </html>'''.format(username, str(type(username))[1:-1],
                              number, str(type(number))[1:-1])
```

Всего таких конвертеров пять:

Имя конвертера	**Описание**
string	(по умолчанию) любой текст без слэшей
int	положительное целое число
float	положительное дробное число
path	строка, но может содержать слэши
uuid	стандарт строк-идентификаторов из 16 байт в шестнадцатеричном представлении, выглядит примерно так: 550e8400-e29b-41d4-a716-446655440000

Помощь