

 < Урок flask

Введение во flask. Обработка HTML-форм

- 1 Инструменты для разработки веб-приложений
- 2 Первая страница на Flask
- 3 Статический контент
- 4 Наведем красоты (чуть-чуть)
- 5 Передача параметров в адресной строке
- 6 Обработка форм

Аннотация

Наконец-то мы добрались до момента, когда мы будем использовать не чужие веб-приложения, а писать свои. И уже очень скоро мы убедимся, что с помощью Python это делать достаточно просто (особенно с использованием дополнительных сторонних библиотек).

1. Инструменты для разработки веб-приложений

В письме про окончание первого курса мы просили вас за лето познакомиться с языком разметки HTML. Вот ссылки на курсы, которые мы рекомендовали:

- Интерактивные онлайн-курсы HTML Academy
- Документация по фреймворку Bootstrap
- Веб-разработка для начинающих: HTML и CSS от Stepik
- ОСНОВЫ Html & CSS от Stepik

Если вдруг вы еще этого не сделали, отложите другие дела и посмотрите эти курсы. Дальше без базового знания HTML-разметки вам будет очень сложно разрабатывать собственные веб-приложения.

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

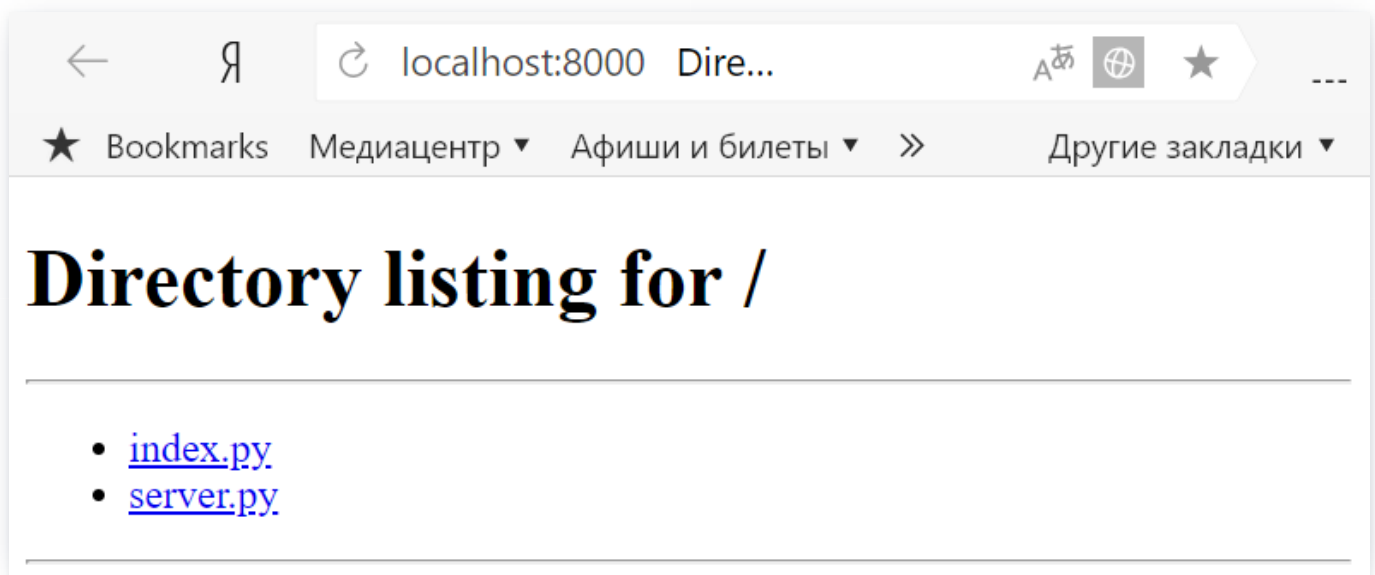
Многие разработчики мобильных приложений и приложений для десктопа «хитрят» и в целях экономии времени и сил создают веб-приложения просто завернутые в веб-компоненту (встроенный браузер), где вся функциональность реализована, как совокупность веб-страниц.

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 напомним 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 запущен наш сервер.



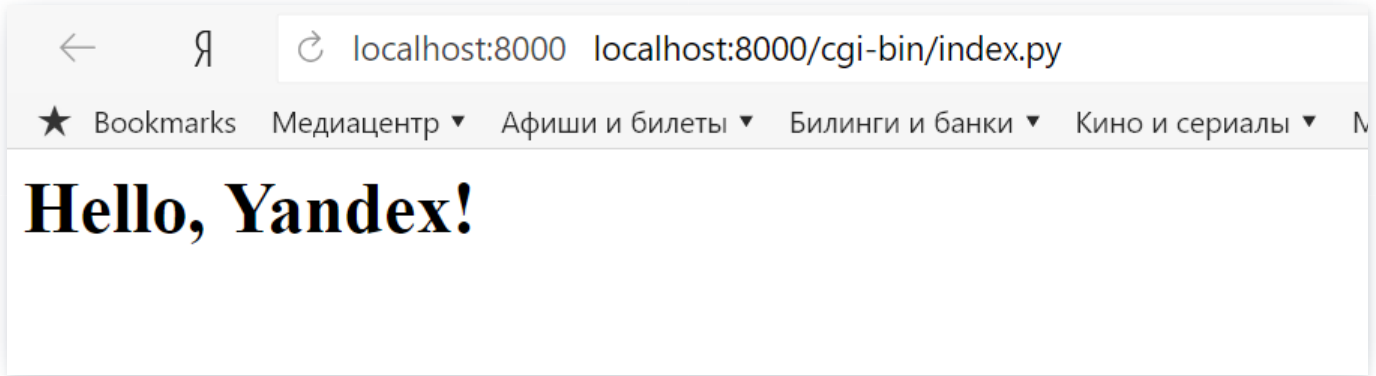
Адрес в командной строке `localhost` (или `127.0.0.1`) — это адрес вашего компьютера для самого этого компьютера, а цифры после двоеточия показывают номер порта. Порт необходим для того, чтобы на одной машине можно было запустить несколько приложений, которые ожидают сообщений. Если утрировать, то `ip-адрес` — это адрес только до многоквартирного дома, а порт — номер квартиры. Номер порта — целое 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:

```
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 увидеть результат выполнения нашего

скрипта.



Как вы могли заметить, на деле такой подход к созданию веб-приложений излишен и неудобен, поэтому мы его применять не будем, а будем использовать гораздо более удобные инструменты, которые построены в том числе поверх только что рассмотренного механизма.

Для Python разработано достаточно большое количество библиотек, которые значительно упрощают процесс создания веб-страниц. Некоторые из этих библиотек большие (Django, Twisted), а некоторые (Bottle, Flask) значительно меньше (их еще называют микро-фреймворками).

С чем связан размер библиотек? С тем, насколько сильно библиотека управляет архитектурой вашей системы и поддерживает ее, насколько развиты ее 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.

2. Первая страница на Flask

Разумеется, наши веб-приложения не будут огромными, сразу делать корпоративные **интранет** порталы — довольно-таки непростая задача. Мы начнем с ресурсов с небольшим количеством страниц и ограниченной функциональностью, чтобы рассмотреть лишь базовые механизмы того, как все работает. К счастью, ключевые принципы построения веб-приложений одинаковы вне зависимости от используемого фреймворка.

Для наших целей прекрасно подойдет микрофреймворк Flask. Он популярный, с хорошей документацией, легок в освоении и при этом вполне пригоден для создания больших промышленных веб-приложений. После его освоения вам не составит труда «пощупать» Bottle самостоятельно, так как принципы работы у них очень похожи, а затем перейти к другим библиотекам.

Flask не входит в стандартную библиотеку Python, чтобы установить его 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('/')

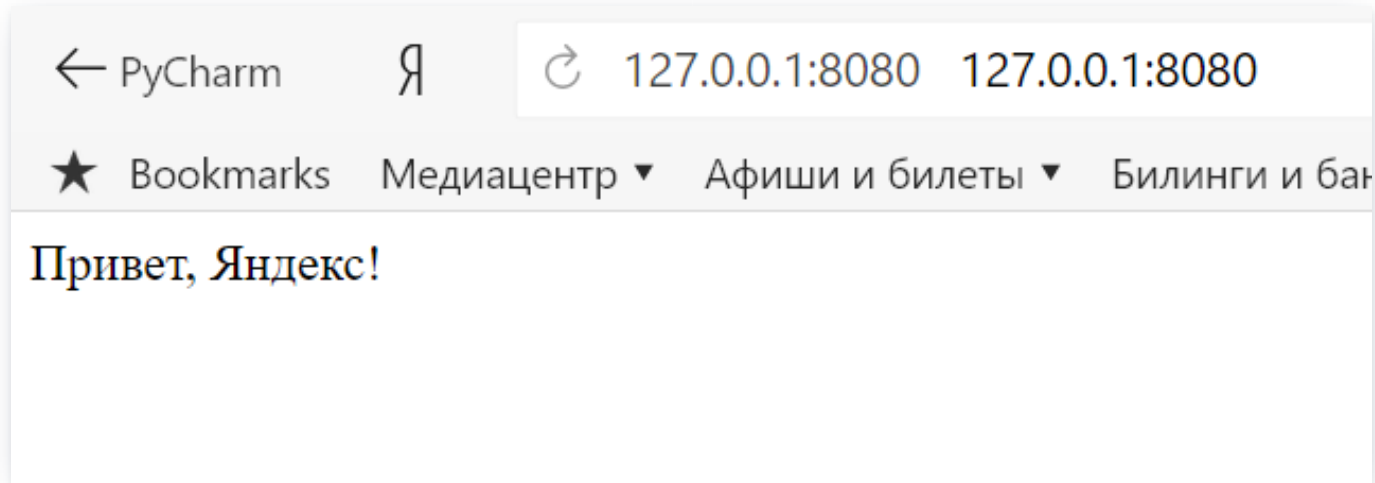
```

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

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

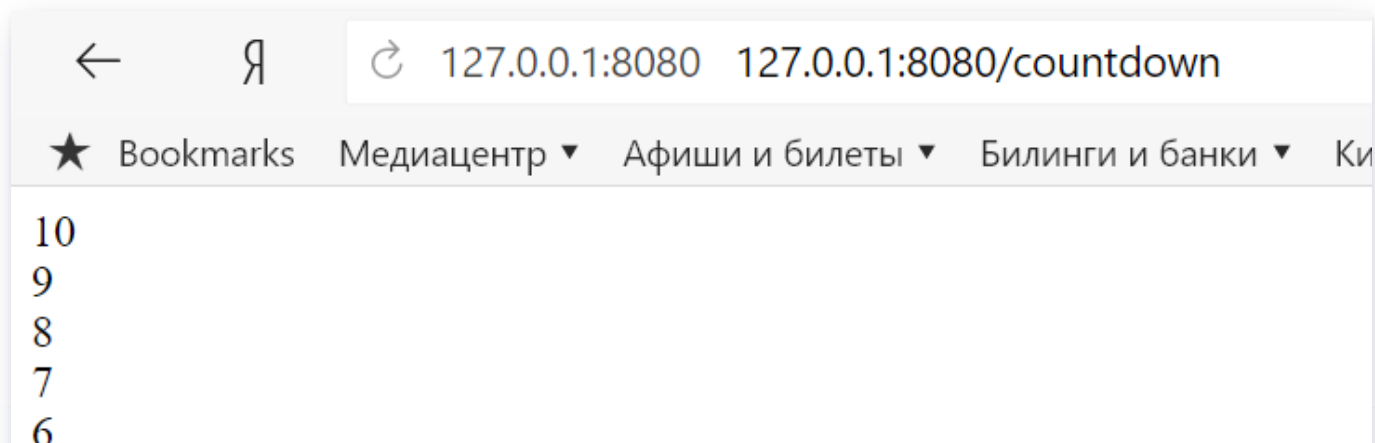
Наш скрипт создает объект приложения как экземпляра класса `Flask`, который мы предварительно импортировали из пакета `flask`. Декораторы `app.route` над функцией `index` используются для регистрации нашей функции, как функции обратного вызова для определенных событий. В нашем случае создается связь 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 передать возвращаемое значение обратно в браузер в качестве ответа.

Давайте запустим наш скрипт и перейдем 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 еще одну страницу с обратным отсчетом:

```
@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)
```



5
4
3
2
1
Пуск!

3. Статический контент

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

- Изображения
- Таблицы стилей CSS
- Шрифты
- Файлы для скачивания
- Файлы скриптов JavaScript
- Музыка, видео
- И т. д.

Всю подобную информацию Flask по умолчанию ищет в директории `static`, поэтому давайте ее 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` и т. д.

Важное замечание: если тут и далее после изменения что-то не отображается, возможно, вы не перезапустили ваше веб-приложение; запустили новую версию параллельно со старой и браузер посылает запросы именно предыдущей версии; или браузер закешировал какие-то данные (сохранил к себе, чтобы отдавать пользователю без запроса к серверу для ускорения работы). Чтобы это поправить, обновите страницу с очисткой кеша — Shift + F5.

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 функцию `url_for`. И добавим в наше веб-приложение следующий код:

```
@app.route('/image_sample')
def image():
    return f''''''
```

Если бы название файла с картинкой было на русском языке, оно перекодировалось бы в коды символов Unicode, что можно увидеть, посмотрев код страницы в браузере.

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

```
return '''
        <html lang="en">
            <head>
                <meta charset="utf-8">
                <title>Привет, Яндекс!</title>
            </head>
            <body>
                <h1>Первая HTML-страница</h1>
            </body>
        </html>"""
```

А теперь давайте добавим css-файл, который заменит цвет текста на красный. Для этого создадим в папке css внутри папки со статическим контентом файл `style.css` со следующим текстом:

```
h1 {
```

```
color: #d22e3a
}
```

Заменим возвращаемое значение на:

```
f"""<!doctype html>
    <html lang="en">
    <head>
        <meta charset="utf-8">
        <link rel="stylesheet" type="text/css" href="{url_for('static', filename='css/style.css')}">
        <title>Привет, Яндекс!</title>
    </head>
    <body>
        <h1>Первая HTML-страница</h1>
    </body>
</html>"""
```

Перезапустим приложение, обновим страницу и убедимся, что цвет текста изменился на указанный.

4. Наведем красоты (чуть-чуть)

Наш курс ориентирован на изучение технологий, которые используются при создании промышленных приложений, и, к сожалению, создание стилей и верстка HTML не входит в цели нашего курса (да и просто времени не хватит, так как это отдельная большая тема). Но, согласитесь, хочется как-то относительно просто делать веб-приложения, которые выглядят симпатично, поэтому 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. Кроме того, у него огромное сообщество, которое предлагает большое количество уже готовых тем и компонентов для Bootstrap, большая часть которых либо свободно распространяемая, либо бесплатная, либо стоит не запредельно много даже по меркам бюджета школьного/студенческого проекта.

В своем первоначальном виде Bootstrap частенько используется программистами как отправная точка для создания веб-приложения без участия дизайнеров и верстальщиков. Инструкцию по подключению и использованию Bootstrap можно найти на **официальном сайте**. Само подключение состоит из нескольких частей:

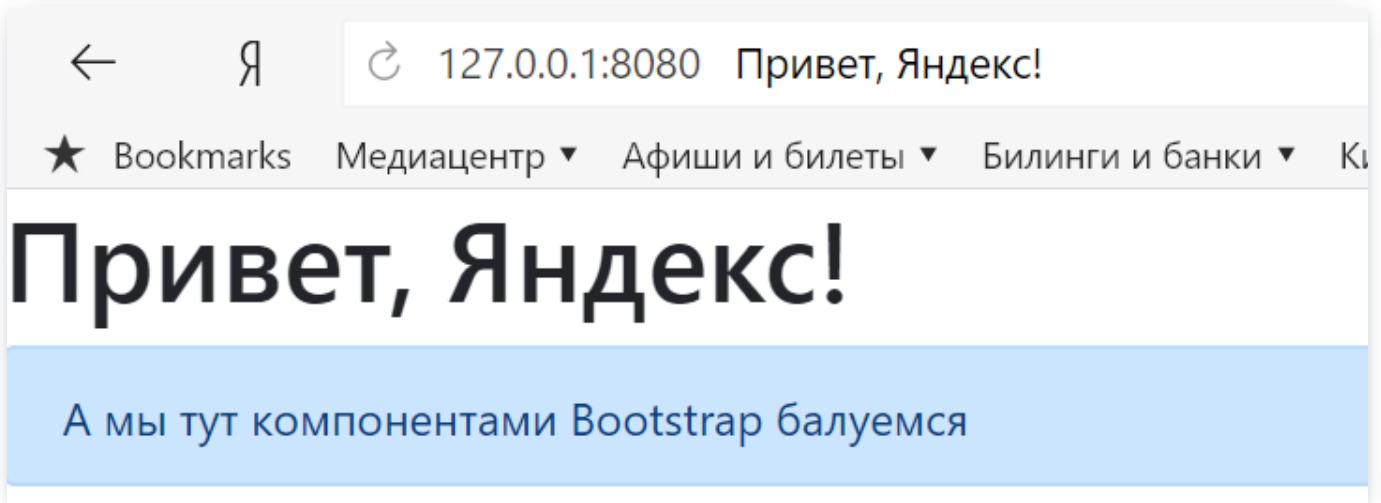
- Подключение стилей
- Подключение JavaScript

Для нашего следующего примера достаточно подключить только стили, но для использования всех возможностей Bootstrap необходимо выполнить оба шага.

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://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"
                integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsre"
                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 громоздко, а при небольшом увеличении сложности страницы трудоемкость 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.

Сейчас же мы добавим в наше приложение некоторую динамику путем 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 возратить 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.
```

Данное сообщение недвусмысленно наводит нас на то, каким может быть возвращаемое значение у функций, украшенных декоратором `app.route`.

5. Передача параметров в адресной строке

Статичные страницы делать не интересно, поэтому давайте рассмотрим то, как мы можем взаимодействовать с пользователем нашего веб-приложения. Таких способов несколько, давайте начнем с передачи параметров в адресной строке.

Декоратор `app.route` умеет принимать на вход значение одного или нескольких параметров, которые пишется после завершающего слэша в пути внутри треугольных скобок `<>`, вот так:

```
@app.route('/greeting/<username>')
def greeting(username):
    return f'''<!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://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"
                integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrehsFOv3Gnu2O3wZPc"
                crossorigin="anonymous">
            <title>Привет, {username}</title>
        </head>
        <body>
            <h1>Привет, {username}!</h1>
        </body>
        </html>'''
```

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

Flask позволяет указывать несколько параметров, а также явно определять их тип с помощью конвертера. Рассмотрим обе этих возможности в следующем примере:

```
@app.route('/two_params/<username>/<int:number>')
def two_params(username, number):
    return f'''<!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://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"
integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrehsFOv3Gnu2O3wZjtSc4/sQjeaRku3152RT86g/Yuy"
crossorigin="anonymous">
                <title>Пример с несколькими параметрами</title>
            </head>
            <body>
                <h2>{username}</h2>
                <div>Это первый параметр и его тип: {str(type(username))[1:-1]}</div>
                <h2>{number}</h2>
                <div>Это второй параметр и его тип: {str(type(number))[1:-1]}</div>
            </body>
        </html>'''
```

Всего таких конвертеров пять:

Имя конвертера	Описание
string	(по умолчанию) любой текст без слешей
int	положительное целое число
float	положительное дробное число
path	строка, но может содержать слеш, для передачи некоторого URL-пути
uuid	стандарт строк-идентификаторов из 16 байт в шестнадцатеричном представлении. Выглядит примерно так: 550e8400-e29b-41d4-a716-446655440000

Передача параметров в адресе командной строки используется достаточно часто для динамического формирования однотипных страниц. Например, на сайте ruri.org такой способ применяется для формирования страниц с информацией о каждом конкретном модуле.

6. Обработка форм

Передача параметра в адресной строке — это скорее не инструмент взаимодействия с пользователем, а динамическое формирование для него ссылок на разные объекты внутри веб-приложения. Для непосредственного общения с пользователем такой способ неудобен, так как в общем случае параметров может быть много, а предлагать пользователю заносить их в адресную строку руками — не лучшая идея.

Для взаимодействия с пользователем применяется другой механизм — формы. HTML-разметка позволяет

создавать элементы для ввода данных разных типов, с которыми Flask, конечно, умеет работать. Давайте вспомним, какие типы элементов ввода поддерживает HTML.

Вообще говоря, разных типов полей ввода довольно много, и периодически в новые версии языка разметки добавляются дополнительные. Вот некоторые из таких типов:

- button — кнопка
- checkbox — множественный выбор
- color — поле выбора цвета
- date, datetime, datetime-local, month, time, week — ввод даты и времени
- email — поле для ввода адреса электронной почты
- file — поле для выбора файла
- number — поле для ввода числовой информации
- password — поле для ввода пароля
- radio — выбор одного из нескольких вариантов
- range — ползунок (как в музыкальном или видео-плеере)
- submit — кнопка для отправки формы
- tel — поле для ввода телефона
- text — поле для ввода текста
- url — поле для ввода адреса в Интернете

Давайте сделаем форму с несколькими самыми распространенными типами полей ввода, для их стилизации используем Bootstrap. Чтобы увидеть, в каком виде информация из этих полей придет на сервер нашего веб-приложения, напомним такой код (предварительно импортировав request из Flask):

```
@app.route('/form_sample', methods=['POST', 'GET'])
def form_sample():
    if request.method == 'GET':
        return f'''<!doctype html>
            <html lang="en">
            <head>
                <meta charset="utf-8">
                <meta name="viewport" content="width=device-width, initial-scale=1,
                <link rel="stylesheet"
                href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/b
                integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddV
                crossorigin="anonymous">
                <link rel="stylesheet" type="text/css" href="{url_for('static', file
                <title>Пример формы</title>
            </head>
            <body>
                <h1>Форма для регистрации в суперсекретной системе</h1>
                <div>
                    <form class="login_form" method="post">
```

```

<input type="email" class="form-control" id="email" aria-des
<input type="password" class="form-control" id="password" p
<div class="form-group">
    <label for="classSelect">В каком вы классе</label>
    <select class="form-control" id="classSelect" name="clas
        <option>7</option>
        <option>8</option>
        <option>9</option>
        <option>10</option>
        <option>11</option>
    </select>
</div>
<div class="form-group">
    <label for="about">Немного о себе</label>
    <textarea class="form-control" id="about" rows="3" name=
</div>
<div class="form-group">
    <label for="photo">Приложите фотографию</label>
    <input type="file" class="form-control-file" id="photo"
</div>
<div class="form-group">
    <label for="form-check">Укажите пол</label>
    <div class="form-check">
        <input class="form-check-input" type="radio" name="se
        <label class="form-check-label" for="male">
            Мужской
        </label>
    </div>
    <div class="form-check">
        <input class="form-check-input" type="radio" name="se
        <label class="form-check-label" for="female">
            Женский
        </label>
    </div>
</div>
<div class="form-group form-check">
    <input type="checkbox" class="form-check-input" id="acce
    <label class="form-check-label" for="acceptRules">Готов
</div>
<button type="submit" class="btn btn-primary">Записаться</bu
</form>
</div>
</body>
</html>'''
elif request.method == 'POST':
    print(request.form['email'])
    print(request.form['password'])
    print(request.form['class'])

```

```
print(request.form['file'])
print(request.form['about'])
print(request.form['accept'])
print(request.form['sex'])
return "Форма отправлена"
```

Заодно давайте настроим нашу форму в style.css, который подключим после стилей Bootstrap:

```
form.login_form {
    margin-left: auto;
    margin-right: auto;
    max-width: 450px;
    background-color: #ffcc00;
    border: 1px solid gray;
    border-radius: 5px;
    padding: 10px;
}
```

Если мы все сделали правильно, увидим вот такой впечатляющий результат.

Форма для регистрации в суперсекретной системе

Введите адрес почты

Введите пароль

В каком вы классе

7

Немного о себе

Приложите фотографию

Выберите файл Файл не выбран

Укажите пол

☒ Мужской

☐ Женский

☐ Готов быть добровольцем

Записаться

Мы дополнили наш декоратор `app.route` новым параметром — списком методов протокола HTTP, с которыми он работает. Всего методов довольно много, но мы будем говорить только о пяти из них:

1. GET — запрашивает данные, не меняя состояния сервера («прочитать»).
2. POST — отправляет данные на сервер («отправить»).
3. PUT — заменяет все текущие данные сервера данными запроса («заменить»).

4. DELETE — удаляет указанные данные («удалить»).

5. PATCH — используется для частичного изменения данных («изменить»).

Таким образом, наша функция `form_sample` работает с двумя методами. Если мы хотим получить данные с сервера, тогда отработывает ветка условия, в которой мы отправляем пользователю форму для заполнения. Когда пользователь заполнил форму и нажал на кнопку «Записаться», данные формы заворачиваются в специальный аналог словаря в сущности `request`, которая хранит всю информацию о пользовательском запросе.

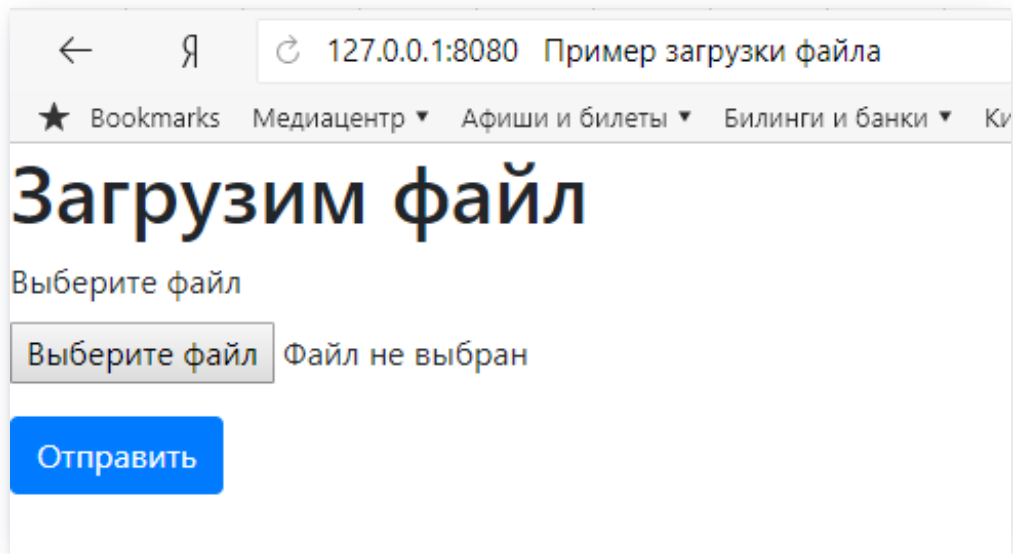
Обратите внимание: поскольку данные формы заворачиваются в аналог словаря, может случиться ситуация, когда по ключу мы не сможем найти значение (например, если пользователь не отметит чекбокс), и тогда выбросится исключение. Чтобы этого избежать, лучше обращаться к данным не напрямую по ключу, а через метод `get`. Таким образом, более корректная обработка значения чекбокса будет выглядеть так:

```
request.form.get('accept')
```

Как вы могли заметить, произошла небольшая странность с приложенным файлом: мы достали только его название, а не содержимое. Это произошло потому, что содержимое файла хранится в другом месте. Давайте напишем еще один пример:

```
@app.route('/sample_file_upload', methods=['POST', 'GET'])
def sample_file_upload():
    if request.method == 'GET':
        return f'''<!doctype html>
            <html lang="en">
            <head>
                <meta charset="utf-8">
                <meta name="viewport" content="width=device-width, initial-scale=1,
                <link rel="stylesheet"
                href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/t
                integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKhr8RbDVdd\
                crossorigin="anonymous">
                <link rel="stylesheet" type="text/css" href="{url_for('static', file
                <title>Пример загрузки файла</title>
            </head>
            <body>
                <h1>Загрузим файл</h1>
                <form method="post" enctype="multipart/form-data">
                    <div class="form-group">
                        <label for="photo">Выберите файл</label>
                        <input type="file" class="form-control-file" id="photo" name
                    </div>
                        <button type="submit" class="btn btn-primary">Отправить</button>
                </form>
            </body>
            </html>'''
    elif request.method == 'POST':
        f = request.files['file']
```

```
print(f.read())  
return "Форма отправлена"
```



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8080' and the page title 'Пример загрузки файла'. The browser's bookmark bar is visible with links to 'Bookmarks', 'Медиацентр', 'Афиши и билеты', 'Билинги и банки', and 'Ки'. The main content area features a large heading 'Загрузим файл'. Below the heading is the text 'Выберите файл'. There is a text input field containing the placeholder text 'Выберите файл' and the status text 'Файл не выбран' to its right. At the bottom of the form is a blue button labeled 'Отправить'.

Обратите внимание: кроме доступа к файлу в `request.files` мы немного изменили саму разметку формы, добавив в нее параметр

```
enctype="multipart/form-data"
```

Иначе форма так и продолжит отправлять только имена файлов, а при попытке доступа к самому файлу мы получим ошибку.

Все достаточно просто, не так ли? Пока самая большая сложность, которая нас преследует — это громоздкая и неудобная генерация HTML-разметки внутри функции, от этого недостатка мы избавимся уже на следующем уроке.

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках сервиса, принадлежат АНО ДПО «Образовательные технологии Яндекса». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «Образовательные технологии Яндекса».

[Пользовательское соглашение.](#)

© 2018 – 2024 ООО «Яндекс»