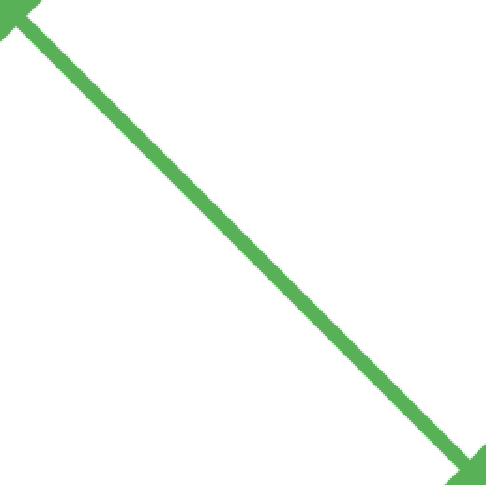
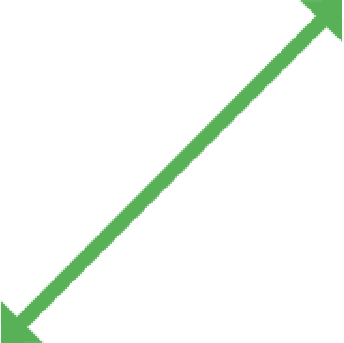


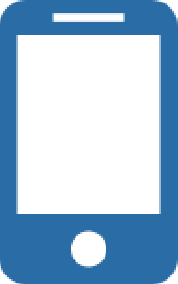
**План**

1. “Client-server” architecture
2. HTTP protocol
3. Flask vs Django
4. Routing
5. Variables Rules
6. Rendering Templates
7. Http methods
8. Static files
9. Request Data
10. File Uploads
11. Cookies
12. Redirects and Errors
13. Response
14. Session
15. Logging
16. Blueprint

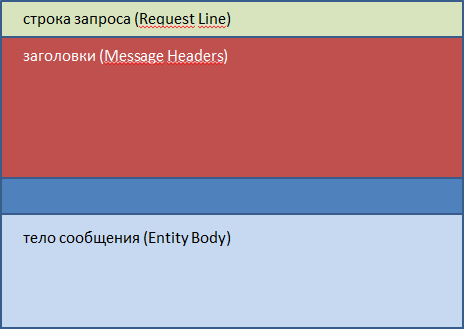
C[ient-Server Mode[



## Server

РС Smartphone Laptop

HyperText Transfer Protocol – протокол передачи данных в клиент серверной архитектуре.

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

**Строка запроса** – указывает метод передачи, URL- адрес, к которому нужно обратиться и версию протокола HTTP.

**Заголовки** – описывают тело сообщений, передают различные параметры и др. сведения и информацию.

**тело сообщения** — это сами данные, которые передаются в запросе. Тело сообщения – это необязательный параметр и может отсутствовать.

Тело обязательно должно отделяться от заголовков пустой строкой.

Тело сообщения может отсутствовать, но стартовая строка и заголовок являются обязательными элементами.

### Строка запроса

Заголовки

Тело сообщения

*Запрос от браузера:*

**GET / HTTP/1.1**

Host: webgyry.info

User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:18.0) Gecko/20100101 Firefox/18.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q

=0.8

Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3 Accept-Encoding: gzip, deflate

Cookie: wp-settings

Connection: keep-alive

<!DOCTYPE html>

<html>

<head>

<meta http-equiv=»Content-Type» content=»text/html;

charset=utf-8″ />

<title>Документ без названия</title>

</head>

<body></body>

</html>

### Ключевые различия:

* Flask предоставляет простоту, гибкость и аккуратность в работе, позволяя пользователю самому выбирать, как реализовать те или иные вещи.
* Django предоставляет пакет «*все включено*»: у вас есть панель админа, интерфейсы баз данных, ORM, и структура каталогов для ваших приложений и проектов.

### Выбирайте на основании следующего:

* Flask, если вы хотите получить больше опыта и возможности для обучения, или же если вам нужно больше контроля над используемыми компонентами (такие как базы данных, которые вам нужно использовать или взаимодействовать с ними).
* Django, если вас интересует конечный продукт. Особенно если вы работаете с прямолинейным приложением, таким как новостной сайт, магазин, блог, и если вам нужно, чтобы всегда один прозрачный способ выполнить ту или иную задачу.

# Используйте декоратор route (), чтобы привязать функцию к URL-адресу.

@app.route**(**'/’**) def** index**():**

**return** 'Index Page’

@app.route**(**'/hello’**) def** hello**():**

**return** 'Hello, World'

# Вы можете добавлять переменную в URL посредствам помещения её в

<>. Также вы можете указать тип аргумента, например:

<converter:variable\_name>

@app.route**(**'/user/<username>’**) def** show\_user\_profile**(**username**):**

*# show the user profile for that user*

**return** 'User %s' % 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** 'Post %d' % post\_id

@app.route**(**'/path/<path:subpath>’**) def** show\_subpath**(**subpath**):**

*# show the subpath after /path/*

**return** 'Subpath %s' % subpath

|  |  |
| --- | --- |
| string | (default) accepts any text without a slash |
| int | accepts positive integers |
| float | accepts positive floating point values |
| path | like string but also accepts slashes |
| uuid | accepts UUID strings |

Для того что бы вернуть html file вы можете использовать метод render\_template() Функция получает на вход параметры:

1. название темплейта который вы хотите вернуть
2. именные аргументы которые вы хотите в нее передать.

Flask будет искать этот файл в папке templates.

@app.route**(**'/hello/’**)** @app.route**(**'/hello/<name>’**) def** hello**(**name=**None):**

**return** render\_template**(**'hello.html'**,** name=name**)**

**Case 1**: a module

/application.py

/templates

/hello.html

**Case 2**: a package:

/application

/ init .py / templates /

hello.html

Вы можете использовать routes для POST, GET, DELTE, PUT, PATCH

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

**if** request.method == 'POST’**: return** do\_the\_login**()**

### else:

**return** show\_the\_login\_form**()**

Для того что бы получить url путь, вместо того что бы его хардкодить, лучше использовать url\_for(). Функция получает на вход параметры:

1. название функции url которой вы хотите получить
2. именные аргументы которые вы хотите в нее передать, если такого аргумента не будет, оно запишет его в параметры URL.

@app.route**(**'/user/<username>’**) def** show\_user\_profile**(**username**):**

*# show the user profile for that user*

**return** 'User %s' % username

# /user/nick?param=Hello

url\_for(‘show\_user\_profile’, username=‘nick’, param=‘Hello’)

Для указания статический файлов вы можете создать папку static в корне проекта и все файлы будут доступны по роуту ”static”

# static/style.css

url\_for**(**'static'**,** filename='style.css'**)**

Для веб-приложений крайне важно реагировать на данные, которые клиент отправляет на сервер. В Flask эта информация предоставляется request global object.

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

error = None

if request.method == 'POST':

if valid\_login(request.form['username'], request.form['password']):

return log\_the\_user\_in(request.form['username']) else:

error = 'Invalid username/password'

# the code below is executed if the request method # was GET or the credentials were invalid

return render\_template('login.html', error=error)

Вы можете получить доступ к загружаемым файлам в request.files. Это словарь ключ которого является название файла, а значение стандартный объект файла. Вы можете сохранять файл вызвав метод save() на объекте файла передав путь сохранения.

@app.route('/upload', methods=['GET', 'POST']) def upload\_file():

if request.method == 'POST': f = request.files['the\_file']

f.save('/var/www/uploads/file.txt’)

...

Если вы хотите сохранить файл с именем которое ему задал клиент, вы можете его вызвать У объекта файла f.filename, но имя файла может быть не корректным, поэтому можете Использовать метод secure\_filename() для того что бы получить безопасное имя файла.

>>> secure\_filename**(**"My cool movie.mov"**)** 'My\_cool\_movie.mov’

>>> secure\_filename**(**"../../../etc/passwd"**)** ‘ etc\_passwd’

>>> secure\_filename**(**u'i contain cool \xfcml\xe4uts.txt'**)** 'i\_contain\_cool\_umlauts.txt'

Это небольшой фрагмент данных отправленный веб- сервером и хранимый на компьютере пользователя.

Веб-клиент (обычно веб-браузер) всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP- запроса. Применяется для сохранения данных на стороне пользователя, на практике обычно используется для:

* аутентификации пользователя;
* хранения персональных предпочтений и настроек пользователя;
* отслеживания состояния сеанса доступа пользователя;
* ведения статистики о пользователях.

Чтобы получить доступ к файлам cookie, вы можете использовать атрибут cookie. Чтобы задать cookie, вы можете использовать метод set\_cookie для объекта response. Атрибут cookie объекта request- это словарь со всеми файлами cookie, которые передает клиент.

@app.route**(**'/'**) def** index**():**

username = request.cookies.get**(**'username’**)**

*# use cookies.get(key) instead of cookies[key] to not get a # KeyError if the cookie is missing*

@app.route**(**'/’**) def** index**():**

resp = make\_response**(**render\_template**(**...**))** resp.set\_cookie**(**'username'**,** 'the username’**) return** resp

Иногда необходимо установить дополнительные headers во view. Поскольку views не обязательно возвращает response object, он может возвращать значение которое конвертируется в response object Flask-ом, и добавление параметров в response может быть запутаным.

Для того что бы добавить например headers в данный метод:

**def** index**():**

**return** render\_template**(**'index.html'**,** foo=42**)**

Вы можете обернуть render\_template в метод make\_response, что позволит вам добавлять headers в

response

**def** index**():**

response = make\_response**(**render\_template**(**'index.html'**,** foo=42**))** response.headers**[**'X-Parachutes'**]** = 'parachutes are cool’

**return** response

Для того что бы перенаправить пользователя на другой адрес используйте функцию redirect()

Для того что бы выкинуть ошибку используйте abort()

@app.route**(**'/’**) def** index**():**

**return** redirect**(**url\_for**(**'login’**))**

@app.route**(**'/login’**) def** login**():**

abort**(**401**)** this\_is\_never\_executed**()**

Стандартная ошибка выкинет вас на черно белую страницу, если вы хотите кастомизировать вывод ошибки можете использовать декаратор errorhandler(). Передав ему номер ошибки которую вы хотите кастомизировать.

@app.errorhandler**(**404**) def** page\_not\_found**(**error**):**

**return** render\_template**(**'page\_not\_found.html'**),** 404

В дополнение к объекту запроса есть также второй объект, называемый session, который позволяет вам хранить информацию, специфичную для пользователя, от одного запроса к следующему. Это выполняется поверх файлов cookie для вас и шифрует файлы cookie криптографически. Это означает, что пользователь может просматривать содержимое вашего файла cookie, но не изменять его, если только они не знают секретный ключ, используемый для подписания.

*# Set the secret key to some random bytes. Keep this really secret!*

app.secret\_key = b'\_5#y2L"F4Q8z\n\xec]/’ session**[**'username'**]** = request.form**[**'username’**]** session.pop**(**'username'**, None)**

### Generate key:

os.urandom(16)

Для отладки исполнения программы лучше всего использовать logger, он выведет в консоль сообщение об ошибке, warning и debug.

Выбор между debug, warning и error зависит от того что вы хотите сообщить пользователю. Debug логи будут видны если запустить программу в режиме debug=True

app.logger.debug**(**'A value for debugging'**)** app.logger.warning**(**'A warning occurred (%d apples)'**,** 42**)** app.logger.error**(**'An error occurred'**)**

https://docs.python.org/3/library/logging.html

Flask blueprints – это способ для вас организовать приложение в более мелкие приложения для пере использования.

Стоит отметить что blueprint не являет собой отдельное приложение и его нельзя запустить самостоятельно, каждый blueprint должен быть зарегистрирован во Flask приложении.

Основная концепция blueprints заключается в том, что они записывают операции, выполняемые при регистрации в приложении. Flask связывает view функции с blueprints при отправке запросов и генерации URL-адресов с одной конечной точки на другую.

simple\_page = Blueprint**(**'simple\_page'**,** name **,**

template\_folder='templates’**)** @simple\_page.route**(**'/'**,** defaults=**{**'page'**:** 'index’**})** @simple\_page.route**(**'/<page>’**)**

**def** show**(**page**): try:**

**return** render\_template**(**'pages/%s.html' % page**) except** TemplateNotFound**:**

abort**(**404**)**

Register blueprint:

**from** flask **import** Flask

**from** yourapplication.simple\_page **import** simple\_page app = Flask**(** name **)** app.register\_blueprint**(**simple\_page**)**

Rules registered on the application:

**[**<Rule '/static/<filename>' **(**HEAD**,** OPTIONS**,** GET**)** -> static>**,**

<Rule '/<page>' **(**HEAD**,** OPTIONS**,** GET**)** -> simple\_page.show>**,**

<Rule '/' **(**HEAD**,** OPTIONS**,** GET**)** -> simple\_page.show>**]**