

# Лабораторная работа №15

## CGI

### 1 Цель работы

Изучить CGI и научиться применять полученные знания на практике.

### 2 Краткая теория

Проще всего создать динамические страницы на Python при помощи CGI-скриптов. CGI-скрипты – это исполняемые файлы, которые выполняются веб-сервером, когда в URL запрашивается соответствующий скрипт.

#### 2.1 Настройка локального сервера

В Python уже есть встроенный CGI сервер, поэтому его настройка элементарна.

Для запуска из консоли (для любителей linux-систем). Запускать нужно из той папки, где мы хотим работать:

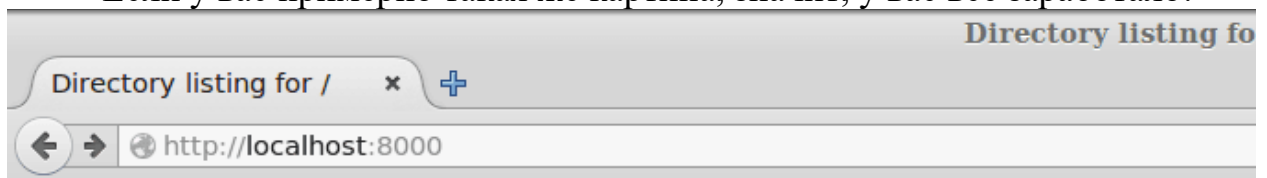
```
python3 -m http.server --cgi
```

Для сидящих на Windows чуть проще будет запуск Python файла (заметьте, что он должен находиться в той же папке, в которой мы планируем работать!):

```
from http.server import HTTPServer, CGIHTTPRequestHandler
server_address = ("", 8000)
httpd = HTTPServer(server_address, CGIHTTPRequestHandler)
httpd.serve_forever()
```

Теперь откройте браузер и в адресной строке наберите **localhost:8000**

Если у вас примерно такая же картина, значит, у вас все заработало!



## Directory listing for /

- [cgi-bin/](#)
- [server.py](#)

#### 2.2 Hello world

Теперь в той папке, где мы запустили сервер, создаём папку cgi-bin.

В этой папке создаём скрипт hello.py со следующим содержимым:

```
#!/usr/bin/env python3
```

```
print("Content-type: text/html")
print()
print("<h1>Hello world!</h1>")
```

Первая строка говорит о том, что это Python скрипт (CGI-скрипты можно писать не только на Python).

Вторая строка печатает заголовок. Он обозначает, что это будет html файл (бывает ещё css, javascript, pdf и куча других, и браузер различает их по заголовкам).

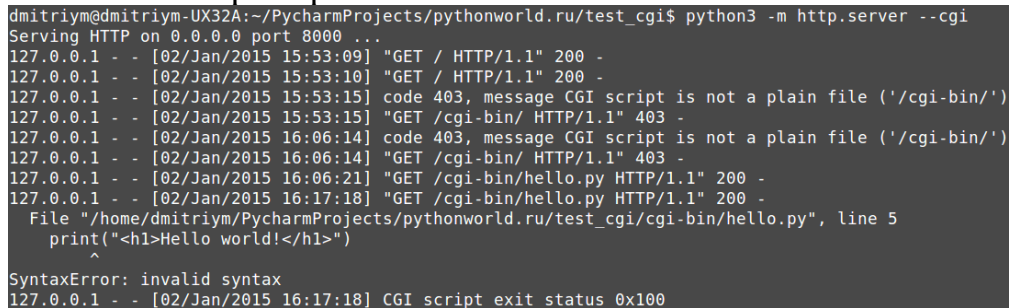
Третья строка (просто символ новой строки) отделяет заголовки от тела ответа.

Четвёртая печатает Hello world.

Теперь переходим на localhost:8000/cgi-bin/hello.py

Если у вас не работает, проверьте, установлены ли права на выполнение.

Также в консоли запущенного сервера появляются сообщения об ошибках. Например:



```
dmitym@dmitym-UX32A:~/PycharmProjects/pythonworld.ru/test_cgi$ python3 -m http.server --cgi
Serving HTTP on 0.0.0.0 port 8000 ...
127.0.0.1 - - [02/Jan/2015 15:53:09] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jan/2015 15:53:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [02/Jan/2015 15:53:15] code 403, message CGI script is not a plain file ('/cgi-bin/')
127.0.0.1 - - [02/Jan/2015 15:53:15] "GET /cgi-bin/ HTTP/1.1" 403 -
127.0.0.1 - - [02/Jan/2015 16:06:14] code 403, message CGI script is not a plain file ('/cgi-bin/')
127.0.0.1 - - [02/Jan/2015 16:06:14] "GET /cgi-bin/ HTTP/1.1" 403 -
127.0.0.1 - - [02/Jan/2015 16:06:21] "GET /cgi-bin/hello.py HTTP/1.1" 200 -
127.0.0.1 - - [02/Jan/2015 16:17:18] "GET /cgi-bin/hello.py HTTP/1.1" 200 -
  File "/home/dmitym/PycharmProjects/pythonworld.ru/test_cgi/cgi-bin/hello.py", line 5
    print("<h1>Hello world!</h1>")
          ^
SyntaxError: invalid syntax
127.0.0.1 - - [02/Jan/2015 16:17:18] CGI script exit status 0x100
```

Рассмотрим несколько более сложные вещи: обработку данных форм и cookies.

### 2.3 Получение данных из форм

Во-первых разберёмся с формами. В модуле CGI есть полезный класс: `FieldStorage`, который содержит в себе переданную в форме информацию. По сути дела этот класс представляет из себя словарь, обладающий теми же свойствами, что и обычный словарь в python.

У класса `FieldStorage` есть 2 метода получения значений данных формы:

**`FieldStorage.getfirst(name, default=None)`** – всегда возвращает только одно значение, связанное с именем поля формы. Метод возвращает только первое значение в том случае, если нехороший пользователь послал более одного значения. Обратите внимание, что порядок, в котором будут получены значения, могут отличаться от браузера к браузеру. Если нет такого поля формы или значение не существует, то метод возвращает `default`.

**`FieldStorage.getlist(name)`** – возвращает список значений, связанных с именем поля формы.

Разберём на примере: создадим в нашей папке файл `index.html` со следующим содержимым (это будет наша форма, данные из которой мы будем обрабатывать):

```

<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Обработка данных форм</title>
</head>
<body>
    <form action="/cgi-bin/form.py">
        <input type="text" name="TEXT_1">
        <input type="text" name="TEXT_2">
        <input type="submit">
    </form>
</body>
</html>

```

А в папке `cgi-bin/` – файл **form.py** (обработчик формы).

```

#!/usr/bin/env python3
import cgi

form = cgi.FieldStorage()
text1 = form.getfirst("TEXT_1", "не задано")
text2 = form.getfirst("TEXT_2", "не задано")

print("Content-type: text/html\n")
print("""<!DOCTYPE HTML>
    <html>
    <head>
        <meta charset="utf-8">
        <title>Обработка данных форм</title>
    </head>
    <body>""")

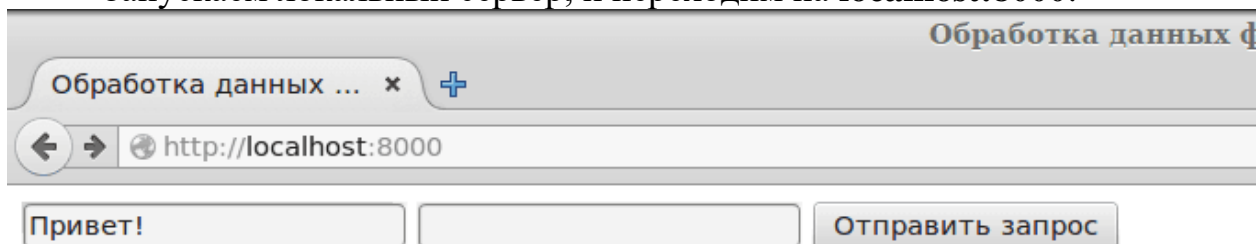
print("<h1>Обработка данных форм!</h1>")
print("<p>TEXT_1: {}</p>".format(text1))
print("<p>TEXT_2: {}</p>".format(text2))

print("""</body>
    </html>""")

```

Попробуем это в действии (кто сидит на linux, не забудьте поставить права на выполнение).

Запускаем локальный сервер, и переходим на **localhost:8000**:



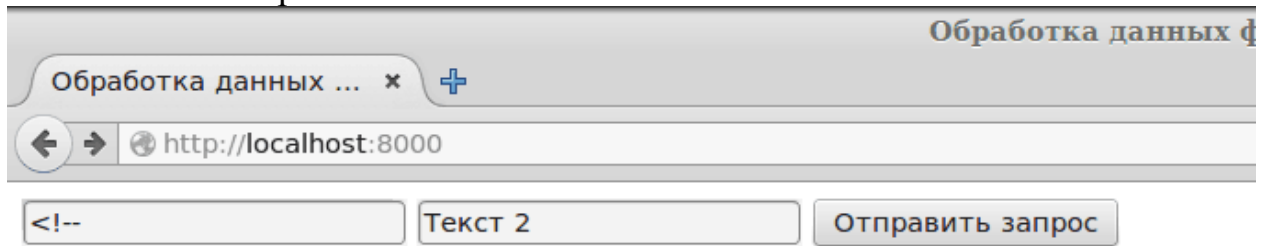
## Обработка данных форм!

TEXT\_1: Привет!

TEXT\_2: не задано

Но есть нюанс...

А если попробовать так?

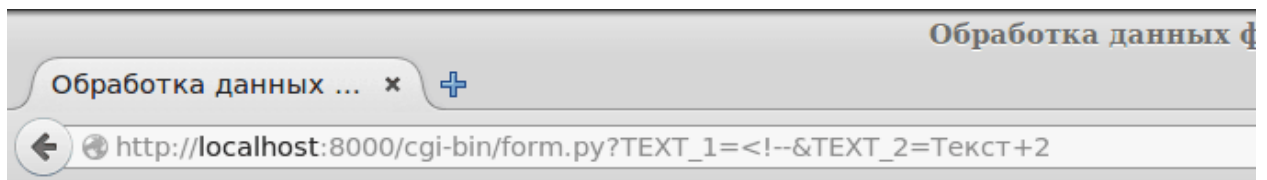


Обработка данных ф

Обработка данных ... x +

← → http://localhost:8000

<!-- Текст 2 Отправить запрос



Обработка данных ф

Обработка данных ... x +

← → http://localhost:8000/cgi-bin/form.py?TEXT\_1=<!--&TEXT\_2=Текст+2

## Обработка данных форм!

TEXT\_1:

Это серьёзная уязвимость, поэтому от неё нужно избавляться. Для этого нужно (в самом простом случае) экранировать все опасные символы. Это можно сделать с помощью функции `escape` из модуля `html`.

Перепишем `form.py`:

```
#!/usr/bin/env python3
import cgi
import html

form = cgi.FieldStorage()
text1 = form.getfirst("TEXT_1", "не задано")
text2 = form.getfirst("TEXT_2", "не задано")
text1 = html.escape(text1)
text2 = html.escape(text2)
```

```

print("Content-type: text/html\n")
print("""<!DOCTYPE HTML>
    <html>
    <head>
        <meta charset="utf-8">
        <title>Обработка данных форм</title>
    </head>
    <body>""")

print("<h1>Обработка данных форм!</h1>")
print("<p>TEXT_1: {}</p>".format(text1))
print("<p>TEXT_2: {}</p>".format(text2))

print("""</body>
    </html>""")

```

Результат можете проверить сами.

Вообще говоря, экранирование нежелательных символов везде, где нужно – очень большая проблема безопасности веб-приложений. Помните об этом.

## 2.4 Cookies

Cookies – небольшой фрагмент данных, отправленный веб-сервером и сохраняемый на компьютере пользователя. Браузер всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в составе HTTP-запроса.

Собственно, cookies – хороший способ сохранить некоторые данные о пользователях.

Отправка печенек осуществляется заголовком Set-cookie:

```

#!/usr/bin/env python3
print("Set-cookie: name=value; expires=Wed May 18 03:33:20 2033; path=/cgi-
bin/; httponly")

print("Content-type: text/html\n")
print("Cookies!!!")

```

Например, если сохранить этот скрипт в /cgi-bin/cookie.py и зайти на localhost:8000/cgi-bin/cookie.py, то вам поставится печенка с именем name и значением value. Срок её хранения до мая 2033 года, отправляется повторно на сервер только к скриптам, которые расположены в /cgi-bin/, и передается только http-запросами (её нельзя получить из браузера пользователя с помощью javascript).

Все эти параметры не являются обязательными. Можно написать так:

```

#!/usr/bin/env python3
print("Set-cookie: name=value")

print("Content-type: text/html\n")
print("Cookies!!!")

```

Тогда храниться она будет до того момента, когда закроется браузер, будет отправляться на сервер для любых документов (и для /index.html тоже, в отличие от предыдущего случая). Также её можно будет получить средствами javascript (поскольку не был установлен флаг httponly).

## 2.5 Обработка Cookies

Теперь научимся получать cookies. Они передаются на сервер и доступны в переменной `os.environ` (словарь, cookies хранятся по ключу `HTTP_COOKIE`). Они передаются в виде пар ключ=значение, что не очень удобно при обработке. Для упрощения работы можно использовать модуль `http.cookies`.

Напишем простой скрипт (`/cgi-bin/cookie.py`), проверяющий, установлена ли кука, и если нет, устанавливает:

```
#!/usr/bin/env python3
```

```
import os
```

```
import http.cookies
```

```
cookie = http.cookies.SimpleCookie(os.environ.get("HTTP_COOKIE"))
```

```
name = cookie.get("name")
```

```
if name is None:
```

```
    print("Set-cookie: name=value")
```

```
    print("Content-type: text/html\n")
```

```
    print("Cookies!!!")
```

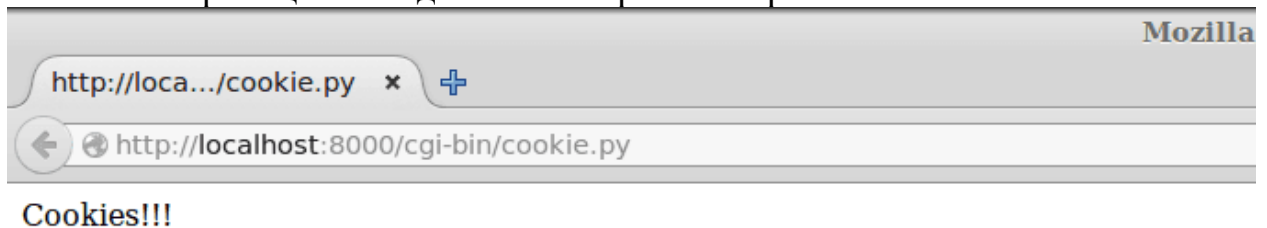
```
else:
```

```
    print("Content-type: text/html\n")
```

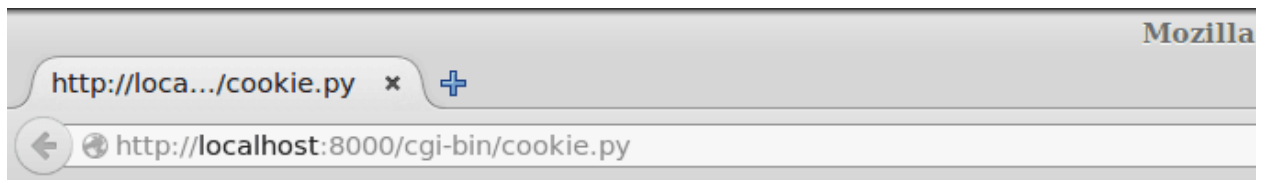
```
    print("Cookies:")
```

```
    print(name.value)
```

Так страница выглядит после первого запроса:



И после обновления страницы:



Cookies: value

Не следует хранить в cookies важные данные, и не полагайтесь на выставленный вами срок хранения. Cookies можно удалить или изменить вручную в браузере.

Мы уже научились обрабатывать формы и устанавливать cookies. Сегодня же мы посмотрим, что может из этого получиться.

Чтобы работать с пользовательскими данными, нужно где-то эти данные сохранять. Самый простой (но далеко не самый изящный и безопасный) – хранение данных в файлах. Более продвинутый способ – хранение в базе данных. Мы остановимся на первом способе, как на самом простом.

Собственно, ничего нового здесь объясняться не будет. Работу с файлами вы уже знаете, обрабатывать формы уже умеете.

Сегодня мы напишем прототип приложения типа «твиттер». Данные в файлах будем хранить в json.

Создадим 2 файла: один будет отвечать за обработку данных, вводимых пользователем, второй – вспомогательный модуль, который упростит код первого.

**cgi-bin/wall.py:**

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
import cgi
import html
import http.cookies
import os
```

```
from _wall import Wall
wall = Wall()
```

```
cookie = http.cookies.SimpleCookie(os.environ.get("HTTP_COOKIE"))
session = cookie.get("session")
```



```

if session is not None:
    session = session.value
    user = wall.find_cookie(session) # Ищем пользователя по переданной куке

    form = cgi.FieldStorage()
    action = form.getfirst("action", "")

    if action == "publish":
        text = form.getfirst("text", "")
        text = html.escape(text)
        if text and user is not None:
            wall.publish(user, text)
    elif action == "login":
        login = form.getfirst("login", "")
        login = html.escape(login)
        password = form.getfirst("password", "")
        password = html.escape(password)
        if wall.find(login, password):
            cookie = wall.set_cookie(login)
            print('Set-cookie: session={}'.format(cookie))
        elif wall.find(login):
            pass # А надо бы предупреждение выдать
        else:
            wall.register(login, password)
            cookie = wall.set_cookie(login)
            print('Set-cookie: session={}'.format(cookie))

pattern = '''
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>Стена</title>
</head>
<body>
    Форма логина и регистрации. При вводе несуществующего имени
    зарегистрируется новый пользователь.
    <form action="/cgi-bin/wall.py">
        Логин: <input type="text" name="login">
        Пароль: <input type="password" name="password">
        <input type="hidden" name="action" value="login">
        <input type="submit">
    </form>

    {posts}

    {publish}
</body>
</html>
'''

if user is not None:
    pub = '''
    <form action="/cgi-bin/wall.py">
        <textarea name="text"></textarea>

```

```

        <input type="hidden" name="action" value="publish">
        <input type="submit">
    </form>
'''
else:
    pub = ''

print('Content-type: text/html\n')

print(pattern.format(posts=wall.html_list(), publish=pub))

```

Здесь мы используем форматирование строк для формирования страницы.

Не забудьте дать этому файлу права на выполнение (второму файлу эти права не нужны).

**cgi-bin/\_wall.py** (здесь определены функции publish, login и другие):

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import json
import random
import time

class Wall:
    USERS = 'cgi-bin/users.json'
    WALL = 'cgi-bin/wall.json'
    COOKIES = 'cgi-bin/cookies.json'

    def __init__(self):
        """Создаём начальные файлы, если они не созданы"""
        try:
            with open(self.USERS, 'r', encoding='utf-8'):
                pass
        except FileNotFoundError:
            with open(self.USERS, 'w', encoding='utf-8') as f:
                json.dump({}, f)

        try:
            with open(self.WALL, 'r', encoding='utf-8'):
                pass
        except FileNotFoundError:
            with open(self.WALL, 'w', encoding='utf-8') as f:
                json.dump({"posts": []}, f)

        try:
            with open(self.COOKIES, 'r', encoding='utf-8'):
                pass
        except FileNotFoundError:
            with open(self.COOKIES, 'w', encoding='utf-8') as f:
                json.dump({}, f)

```

```

def register(self, user, password):
    """Регистрирует пользователя. Возвращает True при успешной
    регистрации"""
    if self.find(user):
        return False # Такой пользователь существует
    with open(self.USERS, 'r', encoding='utf-8') as f:
        users = json.load(f)
    users[user] = password
    with open(self.USERS, 'w', encoding='utf-8') as f:
        json.dump(users, f)
    return True

def set_cookie(self, user):
    """Записывает куку в файл. Возвращает созданную куку."""
    with open(self.COOKIES, 'r', encoding='utf-8') as f:
        cookies = json.load(f)
    cookie = str(time.time()) + str(random.randrange(10**14)) #
    Генерируем уникальную куку для пользователя
    cookies[cookie] = user
    with open(self.COOKIES, 'w', encoding='utf-8') as f:
        json.dump(cookies, f)
    return cookie

def find_cookie(self, cookie):
    """По куке находит имя пользователя"""
    with open(self.COOKIES, 'r', encoding='utf-8') as f:
        cookies = json.load(f)
    return cookies.get(cookie)

def find(self, user, password=None):
    """Ищет пользователя по имени или по имени и паролю"""
    with open(self.USERS, 'r', encoding='utf-8') as f:
        users = json.load(f)
    if user in users and (password is None or password == users[user]):
        return True
    return False

def publish(self, user, text):
    """Публикует текст"""
    with open(self.WALL, 'r', encoding='utf-8') as f:
        wall = json.load(f)
    wall['posts'].append({'user': user, 'text': text})
    with open(self.WALL, 'w', encoding='utf-8') as f:
        json.dump(wall, f)

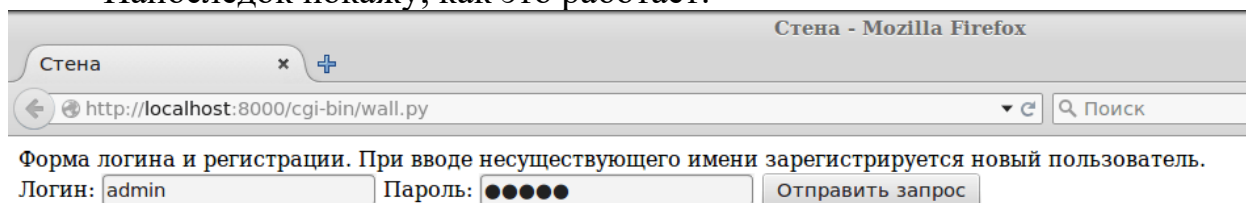
def html_list(self):
    """Список постов для отображения на странице"""
    with open(self.WALL, 'r', encoding='utf-8') as f:
        wall = json.load(f)
    posts = []
    for post in wall['posts']:
        content = post['user'] + ' : ' + post['text']
        posts.append(content)
    return '<br>'.join(posts)

```

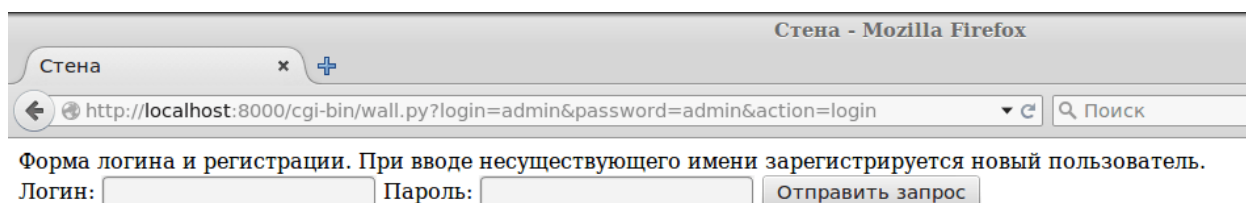
Разумеется, в нашем простом «твиттере» очень много недостатков: не выводятся предупреждения пользователю, регистрация при несуществующем имени, пароли хранятся в открытом виде, использованные куки не удаляются, и многие другие. Кто хочет, может усовершенствовать.

Но есть и преимущество: поскольку у нас теперь 2 разных файла (почти независимых), то можно поменять систему хранения данных (например, база данных вместо файлов), вообще не затрагивая wall.py.

Напоследок покажу, как это работает:

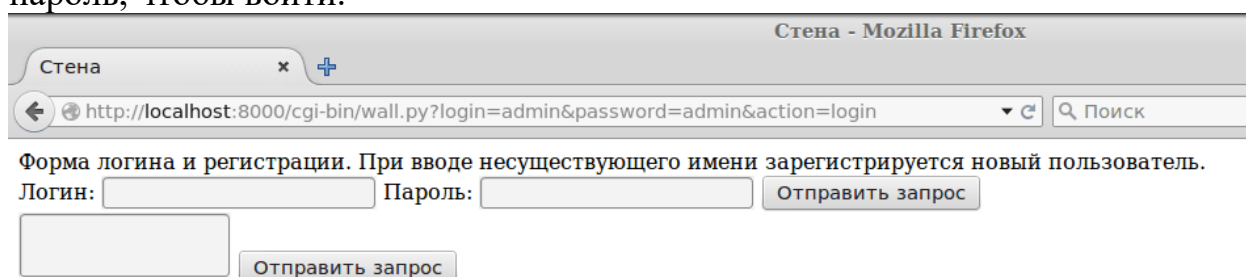


Скриншот веб-браузера Mozilla Firefox. Вкладка «Стена». Адресная строка: `http://localhost:8000/cgi-bin/wall.py`. Текст на странице: «Форма логина и регистрации. При вводе несуществующего имени зарегистрируется новый пользователь.» Форма содержит поле «Логин» с текстом «admin», поле «Пароль» с шестью черными точками, и кнопку «Отправить запрос».



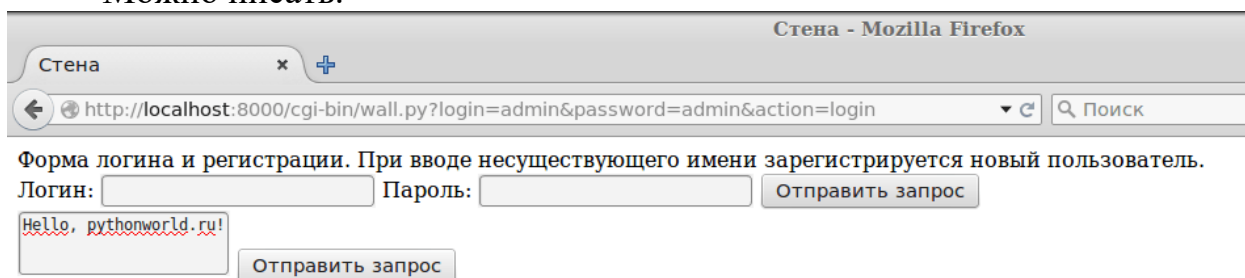
Скриншот веб-браузера Mozilla Firefox. Вкладка «Стена». Адресная строка: `http://localhost:8000/cgi-bin/wall.py?login=admin&password=admin&action=login`. Текст на странице: «Форма логина и регистрации. При вводе несуществующего имени зарегистрируется новый пользователь.» Форма содержит пустые поля «Логин» и «Пароль», и кнопку «Отправить запрос».

Сначала зарегистрировались, теперь нужно ещё раз ввести логин-пароль, чтобы войти.



Скриншот веб-браузера Mozilla Firefox. Вкладка «Стена». Адресная строка: `http://localhost:8000/cgi-bin/wall.py?login=admin&password=admin&action=login`. Текст на странице: «Форма логина и регистрации. При вводе несуществующего имени зарегистрируется новый пользователь.» Форма содержит пустые поля «Логин» и «Пароль», кнопку «Отправить запрос», и ниже — ещё одно пустое текстовое поле и кнопку «Отправить запрос».

Можно писать.



Сцена - Mozilla Firefox

Сцена

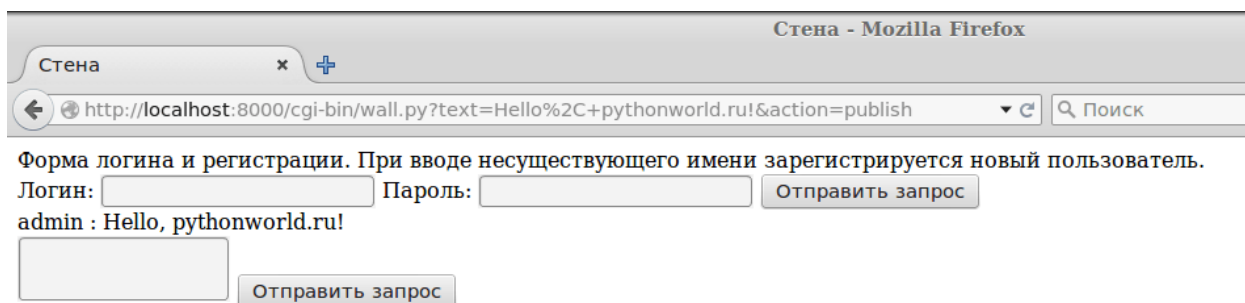
http://localhost:8000/cgi-bin/wall.py?login=admin&password=admin&action=login

Поиск

Форма логина и регистрации. При вводе несуществующего имени зарегистрируется новый пользователь.

Логин:  Пароль:

Hello, pythonworld.ru!



Сцена - Mozilla Firefox

Сцена

http://localhost:8000/cgi-bin/wall.py?text=Hello%2C+pythonworld.ru!&action=publish

Поиск

Форма логина и регистрации. При вводе несуществующего имени зарегистрируется новый пользователь.

Логин:  Пароль:

admin : Hello, pythonworld.ru!

### 3 Задание для выполнения работы

На основе приведенных примеров создать по 3 формы для каждой из таблиц, созданных в предыдущей лабораторной работе (их должно было быть не менее 10): 1 форма для ввода записей в таблицу, 1 для вывода ВСЕХ записей, 1 для обновления КАКОЙ-НИБУДЬ КОНКРЕТНОЙ записи. Также предусмотреть возможность удаления КАКОЙ-НИБУДЬ КОНКРЕТНОЙ записи. Если какое-либо поле формы заполнено неправильно, стоит немедленно сообщить об этом, а не ждать, пока форма будет отправлена. Если форма заполнена корректно, то отправить ее и сообщить, что форма была отправлена успешно.