СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

WEB-ПРОГРАММИРОВАНИЕ НА PYTHON

В. В. Янцев

ЛАНЬ

E.LANBOOK.COM

В. В. ЯНЦЕВ

WEB-ПРОГРАММИРОВАНИЕ НА PYTHON

Учебное пособие

ЛАНЬ

· САНКТ-ПЕТЕРБУРГ · МОСКВА · КРАСНОДАР ·
2022

УДК 004
ББК 32.973.4я723

Я 65 **Янцев В. В.** Web-программирование на Python : учебное пособие для СПО / В. В. Янцев. — Санкт-Петербург : Лань, 2022. — 180 с. : ил. — Текст : непосредственный.

ISBN 978-5-8114-9460-6

В книге рассматривается web-программирование на «чистом» Python — то есть без применения популярных фреймворков.

Автор последовательно проведет вас по всем этапам данной темы. Прочитав эту книгу, вы сможете настроить на своем компьютере полноценную среду разработки, состоящую из интерпретатора Python, сервера Apache и текстового редактора Notepad++. Научитесь работать с переменными, операторами, числами, строками, регулярными выражениями, списками, кортежами, множествами, датой и временем. Узнаете, как писать функции, читать содержимое каталогов, работать с файлами, подключать модули. На практике освоите создание несложных программ: вывода данных из форм, проверки адреса электронной почты, подсвечивания ссылок, бесконечной ленты новостей и других. Наконец, на завершающем этапе читателю предстоит написать простой, но вполне работоспособный сайт.

Соответствует современным требованиям Федерального государственного образовательного стандарта среднего профессионального образования и профессиональным квалификационным требованиям.

Рекомендовано в качестве дополнительной литературы для студентов вузов, обучающихся по направлению «Информатика и вычислительная техника».

УДК 004
ББК 32.973.4я723

Обложка
П. И. ПОЛЯКОВА

Оглавление

1. Введение

Не знаю, ошибочны ли эти впечатления, но, по моим наблюдениям, в последние годы книг по программированию на Python издано очень много. Намного больше, чем по любым другим языкам. Это свидетельствует о высоком уровне интереса к Python. В то же время авторы этих книг в большинстве своем рассказывают об одном и том же — о переменных, операторах, числах, строках, кортежах и списках. О файлах, каталогах, функциях, базах данных и ООП. Еще о библиотеках и фреймворках. Есть целый ряд изданий, рассказывающих о применении Python в научных дисциплинах. При этом процессы создания кода чаще всего описаны для оболочки командной строки или интерактивной оболочки Python. И практически нет никакой информации о web-программировании на Python. Точнее, она есть, но исключительно в двух видах:

1) либо достаточно короткий раздел, посвященный этой теме, где рассматриваются лишь отдельные и не самые актуальные вопросы web-программирования;

2) либо книги, целиком и полностью посвященные фреймворкам типа Django, F0lask или Pyramid.

И совершенно отсутствует литература о web-программировании на «чистом» Python — то есть без использования фреймворков. А ведь это очень интересная тема, открывающая богатые возможности и весьма широкое поле деятельности. Вообще, по моему мнению, умение писать сайты и серверные программы на «чистом» Python — навык очень полезный настоящему программисту. Но, увы, приходится еще раз повторить — книг на данную тему нет (во всяком случае, мне найти не удалось).

Ну что ж, раз есть такой пробел, его явно необходимо заполнить. Что и решил сделать автор.

1.1. О чем эта книга

Прежде чем купить какую-либо книгу по программированию, потенциальному читателю важно знать: для кого и о чем она написана. На этот вопрос я отвечу так: данная книга — для тех, кто уже начал или еще только собирается изучать азы очень интересного и модного языка Python. При этом материал в основном посвящен именно такой узкой, но очень востребованной теме, как web-программирование на Python.

Правда, тут есть две оговорки. Полноценная разработка программ для Интернета на своем компьютере невозможна без создания на нем локального хостинга. Поэтому часть книги будет посвящена этой теме. Современные сайты нельзя представить без использования сценариев на JavaScript. Этому моменту мы тоже уделим некоторое внимание.

Нужно сказать, что в данной книге читатель не найдет ряд интересных тем, которые автор не посчитал нужным рассмотреть на страницах издания. Это обработка исключений, итераторы, подключение к базам данных, объектно-

ориентированное программирование. Если вам интересны данные темы, советую прочесть какую-либо книгу о Python, где эти аспекты разобраны подробно. Более того, советую вам после прочтения моей книги ознакомиться еще с несколькими, где рассказывается о Python. Хороший программист никогда не стоит на месте и не довольствуется минимальными знаниями.

Хочу также обратить внимание читателей на то, что данная книга не является всеобъемлющим учебником по Python. В ней приведена лишь самая необходимая информация для первых опытов в web-программировании. Если вы хотите изучить Python глубже, вспомните про совет, данный в предыдущем абзаце.

Основные материалы по теме содержатся в третьей, четвертой и пятой главах. Мы изучим синтаксис языка, попробуем свои силы на нескольких примерах и напишем простой сайт студии звукозаписи. Во всех случаях мы станем использовать «чистый» Python 3 без применения каких-либо фреймворков. Все результаты нашей работы предстоит тестировать в браузерах. Можно в одном вашем любимом, но лучше в нескольких, наиболее популярных. Теме web-обозревателей у нас посвящен четвертый раздел «Введения».

Конечно, web-программирование на Python не ограничивается написанием простых сайтов. На этом языке можно создавать парсеры, боты, каталоги, справочники, Интернет-магазины, порталы и многое другое. Книга, которую вы держите в руках, — лишь первый шаг в данном направлении.

1.2. Особенности изложения материала

В книге семь глав.

Первая — «Введение». Здесь изложен ряд сведений, необходимых для понимания читателями того, что они найдут в данной книге. Например, как на страницах издания оформлен код или откуда скачать zip-архив с программами.

Вторая глава посвящена настройке среды разработки на персональном компьютере. Если вы хотите скачать zip-архив и экспериментировать на своем ПК, необходимо создать на нем локальный хостинг, установив несколько программ.

В третьей мы познакомимся с «фундаментом», на котором построено «здание» web-программирования на языке Python 3. Вы узнаете о переменных, операторах, условных выражениях, циклах и многом другом. При этом напоминаю, что все примеры нам предстоит тестировать в браузерах.

Четвертая глава посвящена созданию нескольких элементарных программ, иллюстрирующих отдельные возможности Python.

В пятой мы подробно разберем, как создать простой сайт, получать сообщения от посетителей и управлять ими.

Название шестой главы говорит само за себя — «Приложения — сценарии на JavaScript». Для создания целостной картины механизмов работы нашего сайта необходимо разобрать, какую роль играют в нем сценарии на JavaScript.

И в самой короткой, седьмой, главе подведем итоги нашей работы.

1.3. Оформление кода

В книге много примеров кода. Они имеют различное типографское оформление в соответствии с их размещением в тексте.

Если код выделен в отдельный блок, то он оформлен моноширинным шрифтом, например так:

```
def opfi(adr):
  fo = open(adr, encoding='utf-8')
  ti = fo.read()
  fo.close()
  return ti
```

Если фрагменты сценария внедрены непосредственно в текст, то в этом случае части кода выделены полужирным шрифтом, например так:

воспользуемся функцией **listdir** из модуля

Обратите внимание: в некоторых блоках программ сделан перенос части кода на вторую и даже на третью строку (из-за недостатка ширины страницы в книге). Как на самом деле выглядят строки, вы можете видеть, открыв соответствующий файл из zip-архива. Запомните правило: все переносы строк кода существуют только в их типографском воспроизведении. Если вы в дальнейшем столкнетесь с подобной ситуацией, учитывайте данный аспект.

Последний момент. Автор постарался, чтобы оформление всех примеров максимально соответствовало стандартам PEP8 — Python Enhancement Proposal 8 (**https://www.python.org/dev/peps/pep-0008/** — ссылка на эту страницу в файле **link.html** из папки «Глава1» zip-архива). Будет неплохо, если читатели ознакомятся с данным руководством.

1.4. Браузеры

Поскольку мы изучаем не просто Python, а web-программирование на нем, то все примеры нам предстоит тестировать в браузерах. Также надо отметить, что в главе 5 мы напишем простой сайт, в котором, помимо Python, будут HTML-разметка, таблицы стилей и сценарии на JavaScript. А это означает, что наш проект желательно тестировать не в одном, а в нескольких браузерах. Хотя современные web-обозреватели имеют меньше различий, чем в былые времена, все-таки некоторые расхождения в обработке HTML, CSS и JavaScript у них есть. Понятно, что наша задача — добиться совершенно одинаковой работы сайта в наиболее популярных браузерах.

На мой взгляд, обязательный набор разработчика должен включать минимум 5 web-обозревателей. Если вы решили серьезно заниматься web-программированием, советую установить на свой компьютер эти браузеры. Расскажу о них по порядку.

1. Браузер **Microsoft Edge**. Он входит в состав операционной системы Windows 10, поэтому не нуждается в скачивании и установке. Многие опытные программисты не любят данный браузер и игнорируют его. Я считаю, что это серьезная ошибка. Дело в том, что Microsoft Edge является браузером по умолчанию в операционной системе Windows 10. Многие пользователи, не такие продвинутые, как программисты, просто используют то, что у них изначально установлено на ПК. Следовательно, большой процент посетителей Интернета прибегает к услугам Microsoft Edge.

2. **Google Chrome**. На мой субъективный взгляд — лучший браузер. И, судя по некоторым опросам, наиболее популярный. Он моложе некоторых своих конкурентов, но снабжен самыми передовыми решениями и очень быстро развивается. И это не удивительно — ведь за ним стоит такой гигант, как компания Google. Скачать ПО можно здесь: **https://www.google.ru/chrome/**.

3. **Яндекс.Браузер**. После его создания компания «Яндекс» вложила заметные средства в рекламу и популяризацию своей разработки. Поэтому данный браузер стоит у многих владельцев компьютеров с ОС Windows. Этих людей обязательно надо учитывать. Скачать дистрибутив можно здесь: **https://browser.yandex.ru/**.

4. **Opera** — один из старейших браузеров, существовавших еще во времена борьбы за лидерство между Netscape Navigator и Internet Explorer. В России у него много поклонников. Недаром среди российских пользователей Интернета показатель его популярности в несколько раз выше, чем общемировой уровень. Кстати, браузер Opera был одним из первых, кто начал поддерживать таблицы стилей. Скачать программное обеспечение можно здесь: **https://www.opera.com/ru**.

5. **Mozilla Firefox**. В этом браузере сценарии необходимо проверять в обязательном порядке. У него достаточно высокий уровень популярности. При этом есть ряд отличий в обработке кода по сравнению с четырьмя перечисленными выше браузерами. Что-то Mozilla Firefox обрабатывает аналогично остальным браузерам, а что-то по-своему. Во всяком случае, я неоднократно сталкивался с ситуациями, когда код, отлично работавший в других браузерах, начинал «капризничать» в Mozilla Firefox. Скачать браузер можно здесь: **https://www.mozilla.org/ru/firefox/**.

Чтобы вручную не вводить адреса для скачивания браузеров, вы можете запустить страницу **browser.html** из папки «Глава1» zip-архива.

1.5. Zip-архив

Книга не имеет сайта поддержки с действующими программами. Вместо этого каждый читатель может скачать zip-архив, в котором есть все необходимое для комфортного освоения материала книги. Это не только учебные примеры, но и комплект готовых программ, а также HTML-страницы с адресами всех ссылок, использованных в учебнике.

Архив можно скачать по адресу: **https://testjs.ru/PWP.zip** (сокращение от Python web programming).

Состав zip-архива следующий.

1. В папке «Глава 1» два файла — **browser.html**, который содержит ссылки для скачивания рекомендуемых к использованию браузеров, и **link.html**, в котором имеется адрес страницы «Руководство по оформлению кода Python».

2. В папке «Глава 2» тоже два файла — **link.html**, где приведены ссылки на необходимое программное обеспечение и online-валидатор кода, а также скрипт **test.py**, предназначенный для тестирования работы интерпретатора Python.

3. В папке «Глава 3» несколько десятков файлов с учебными примерами, иллюстрирующими синтаксис Python.

4. Папка «Глава 4» предназначена для хранения первых экспериментальных программ, которые иллюстрируют некоторые возможности Python.

5. Папка «Глава 5» содержит полный набор программ, изображений, текстовых файлов, скриптов и таблиц стилей для сайта, который нам предстоит написать (и даже файл favicon).

Для удобства читателей все сценарии, приведенные в книге, имеют указание на их место в архиве и обозначены рисунком:

2. Среда разработки

Для написания и тестирования примеров и законченных программ нам в обязательном порядке необходимо создать на своем компьютере локальный хостинг.

Вообще, хостинг — это, упрощенно говоря, компьютер, предназначенный для размещения на нем сайтов и обеспечивающий к ним доступ из Интернета. Локальный хостинг — набор программ на вашем персональном компьютере, которые позволяют имитировать реальный хостинг и проводить тестирование ваших сайтов перед их размещением в сети. **Кстати, обратите внимание: локальный хостинг работает без подключения к Интернету!**

Чтобы получить результаты, описанные в данной книге, вам понадобится установить на компьютер распространяемый пакет компонентов **Visual C++** от Microsoft, сервер **Apache** и интерпретатор **Python**. Описание технологии их установки — в следующих разделах.

Полноценную среду разработки невозможно представить без установки на ваш компьютер настоящего текстового редактора, специально предназначенного для создания программ. Конечно, весь код можно писать в обычном «Блокноте». Но гораздо лучше и рациональнее пользоваться специализированными редакторами. Такие приложения намного удобнее: они подсвечивают код, предлагают синтаксические подсказки, позволяют менять кодировку документов, сохраняют файлы в разных форматах.

В нашем случае необходим редактор, который позволит:

– выполнять качественную HTML-разметку;

– создавать файлы с таблицами стилей;

– писать сценарии на Python;

– писать сценарии на JavaScript.

Этим принципам удовлетворяют многие редакторы. Но лучше всего, на мой взгляд, — **Notepad++**. Где его взять и как установить — рассказано в пятом разделе этой главы.

Напомню, кстати, что вы можете запустить все необходимые ссылки для скачивания программного обеспечения на странице **link.html** из папки «Глава 2» zip-архива.

Итак, вводная часть завершена, приступим к созданию локального хостинга. А начнем мы с выяснения разрядности вашей ОС.

Обратите внимание: все процедуры описаны для компьютера с операционной системой Windows 10.

2.1. Выясняем разрядность ОС

Перед тем как мы создадим на компьютере среду разработки, необходимо выяснить разрядность вашей операционной системы, чтобы знать, какие файлы скачивать для локального хостинга.

Включите компьютер, дождитесь полной загрузки операционной системы. Щелкните на кнопке «Пуск», а затем на кнопке «Параметры» (рис. 2.1.1). В открывшемся окне выберите пункт меню «Система». Откроется следующая вкладка, в которой необходимо кликнуть на строке «О системе» (или «О программе»). После этого на вкладке «Характеристики устройства» посмотрите строку «Тип системы» (рис. 2.1.2).

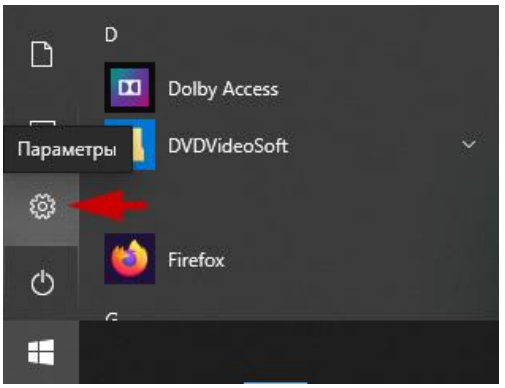

Рис. 2.1.1. Кнопка «Параметры» в меню

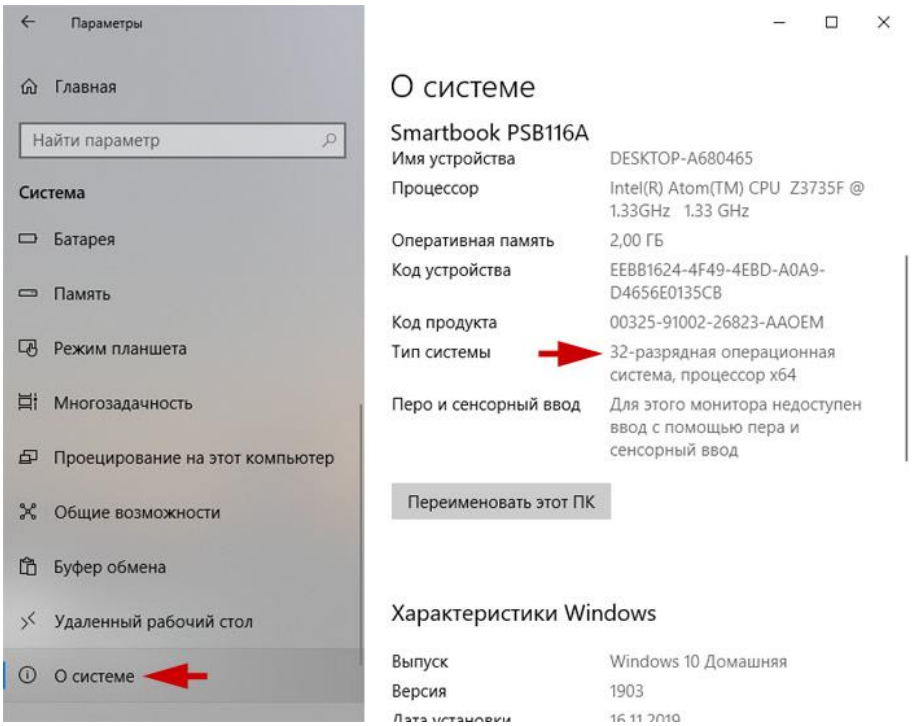

Рис. 2.1.2. Разрядность, или тип, операционной системы

Типов ОС Windows существует два — 32-разрядная и 64-разрядная. Запомните разрядность вашей. Это число понадобится трижды: при скачивании паке-

та Visual C++, zip-архива сервера Apache, а также установщика интерпретатора Python.

2.2. Установка пакета Visual C++

Откройте браузер и зайдите на страницу **https://www.apachelounge.com/download/**. Это сайт, с которого мы будем скачивать архив с сервером Apache. Но сначала надо установить на ваш компьютер компоненты Visual C++, без которых сервер не заработает.

Для этого найдите на указанной странице текст «**Be sure** you installed latest 14.29.30037.0 Visual C++ Redistributable for Visual Studio 2015–2019: **vc_redist_x64** or **vc_redist_x86** see **Redistributable**» (на момент подготовки книги данные строки были последними в описательной части страницы; со временем текст может несколько измениться). Для нас важны две ссылки, расположенные в тексте: **vc_redist_x64** и **vc_redist_x86**. Если у вас 64-разрядная ОС, нажмите ссылку **vc_redist_x64**. Если 32-разрядная, необходимо щелкнуть на ссылке **vc_redist_x86** (рис. 2.2.1).

Запустите скачанный файл (рис. 2.2.2), примите условия лицензионного соглашения и нажмите кнопку «Установить». Дождитесь завершения процесса установки (рис. 2.2.3). Нажмите кнопку «Закрыть» (рис. 2.2.4).

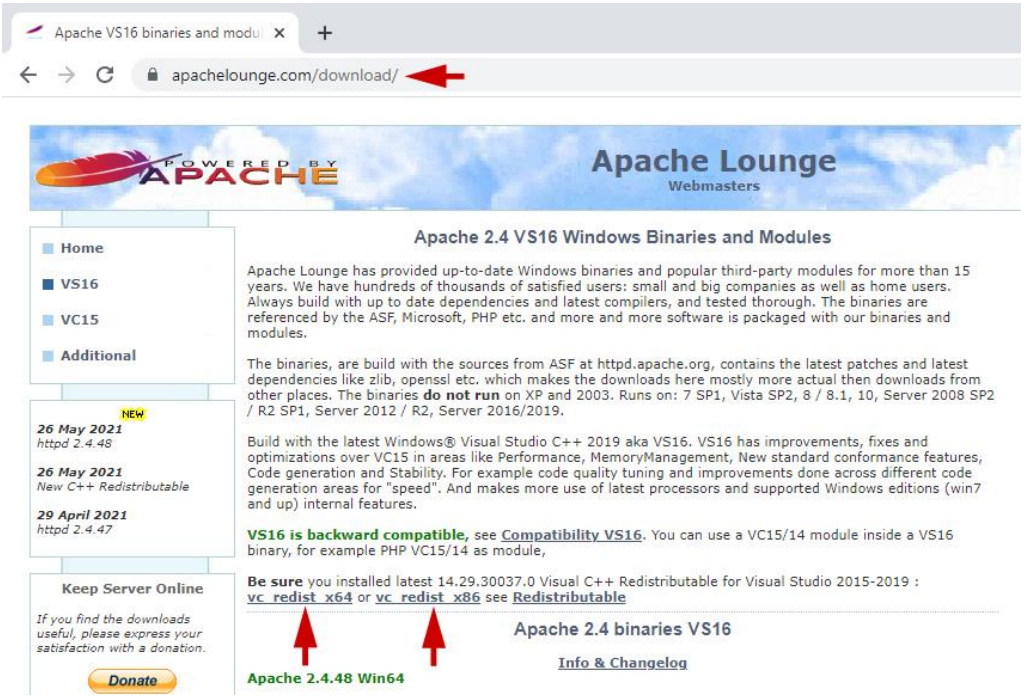

Рис. 2.2.1. Ссылки для скачивания пакетов Visual C++

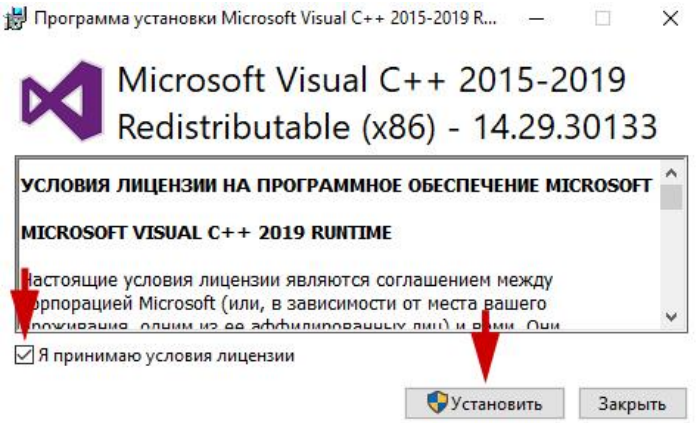

Рис. 2.2.2. Начало установки пакета

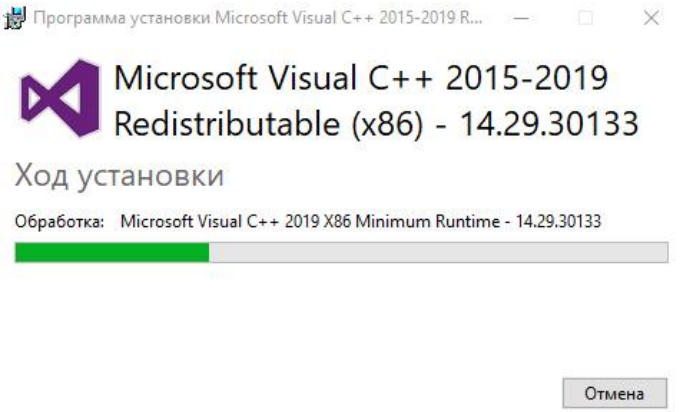

Рис. 2.2.3. Процесс установки

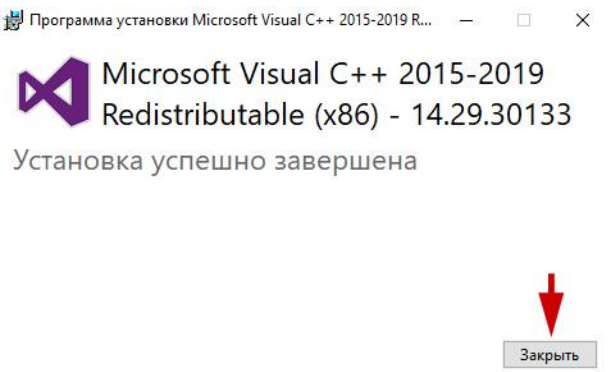

Рис. 2.2.4. Завершение установки

2.3. Установка сервера Apache 2.4

Теперь скачайте на рабочий стол компьютера с сайта **https://www.apache-lounge.com/download/** zip-архив сервера, соответствующий разрядности вашей ОС (рис. 2.3.1). Для 64-разрядной системы — архив с пометкой **Win64** (файл httpd-2.4.48-win64-VS16.zip), для 32-разрядной — с пометкой **Win32** (файл httpd-2.4.48-win32-VS16.zip). Распакуйте архив. Скопируйте папку **Apache24** (только ее) непосредственно на диск **C**, чтобы ее адрес был **C:\Apache24** (рис. 2.3.2).

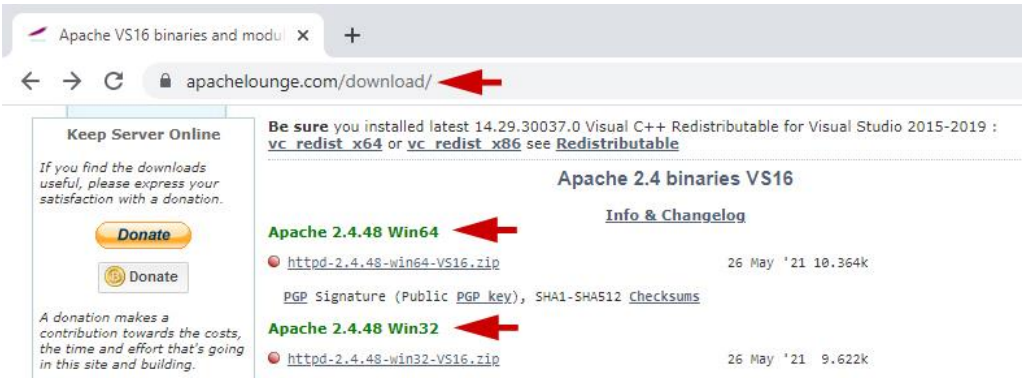

Рис. 2.3.1. Сайт с zip-архивом сервера

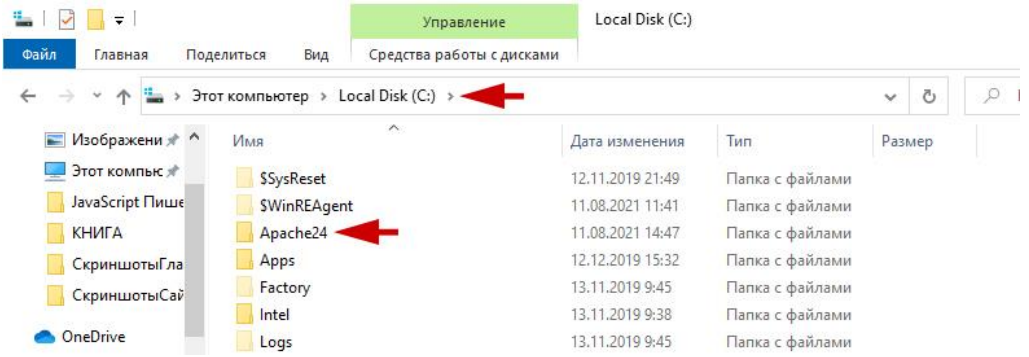

Рис. 2.3.2. Копируем папку Apache24 на диск C

Откройте приложение «Командная строка» от имени администратора. Для этого нажмите кнопку «Пуск», в меню выберите «Служебные — Windows», найдите пункт «Командная строка» и щелкните на нем правой (правой!) кнопкой мыши. В выпадающем списке выберите «Дополнительно», а затем «Запуск от имени администратора» (рис. 2.3.3). Откроется окно программы.

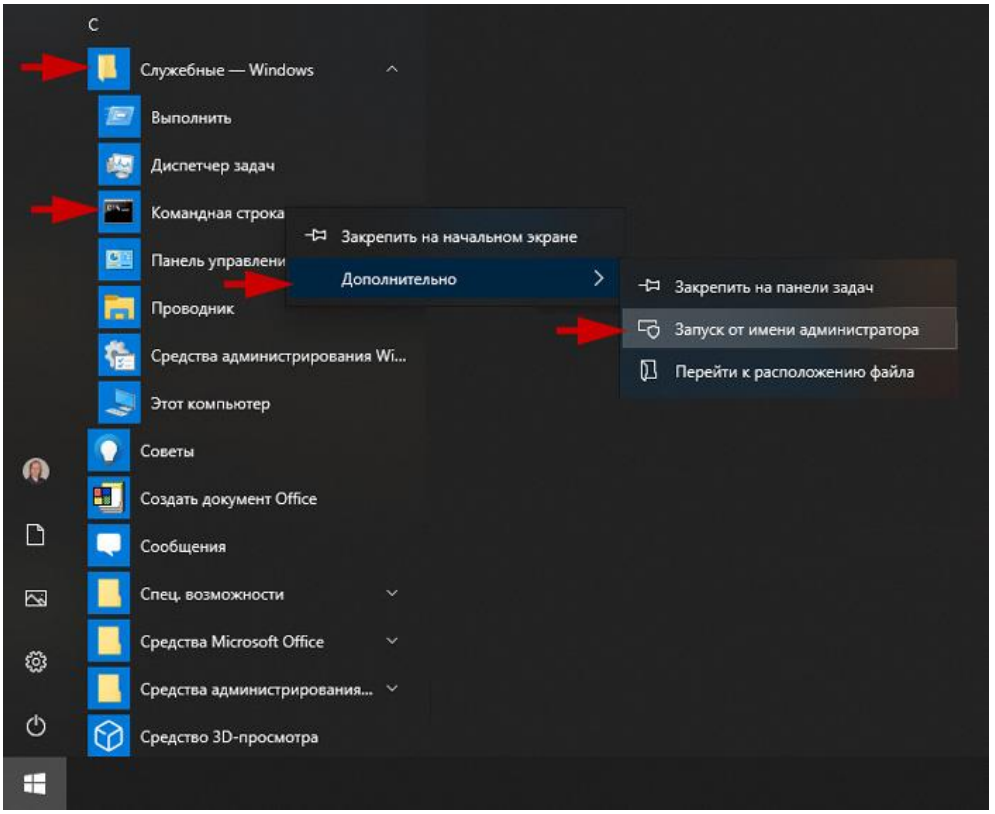

Рис. 2.3.3. Запуск командной строки

Теперь надо инсталлировать, а затем запустить сервер. Для этого в окне командной строки сразу после

```
C:\WINDOWS\System32>
```

наберите

```
C:\Apache24\bin\httpd.exe -k install
```

Должно получиться

```
C:\WINDOWS\System32>C:\Apache24\bin\httpd.exe -k install
```

Нажмите «Enter». Когда процесс инсталляции закончится (рис. 2.3.4), в окне программы вновь появится строка

```
C:\WINDOWS\System32>
```

Закройте окно командной строки и перезагрузите компьютер.

ВНИМАНИЕ! Может открыться окно брандмауэра с запросом на разрешение работы Apache. Разрешите доступ во всех сетях.

Нам надо убедиться, что сервер заработал. Для этого откройте ваш браузер и введите в строке адреса **http://localhost/**. Если появилось сообщение «It works!», значит все в порядке — сервер функционирует как положено (рис. 2.3.5).

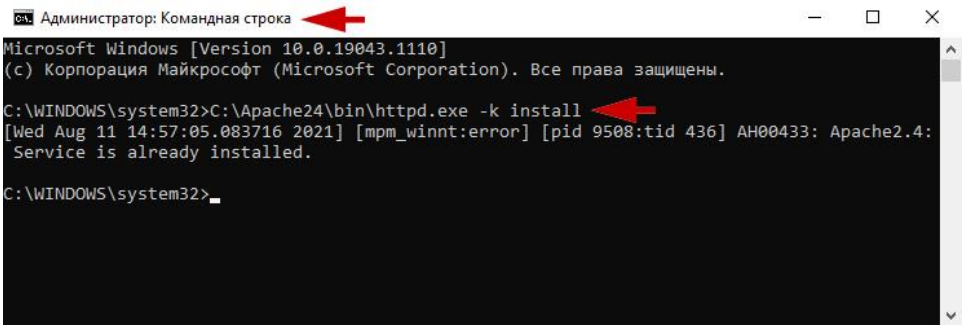

Рис. 2.3.4. Инсталляция сервера из приложения «Командная строка»

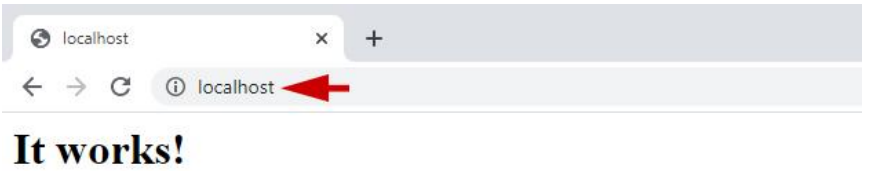

Рис. 2.3.5. Сервер подтверждает, что он работает

Теперь обратите внимание на один важный момент. Метод, которым мы установили сервер, позволяет запускать его автоматически при включении компьютера и завершать его работу при выключении компьютера. То есть, Apache продолжает функционировать, даже если в данное время он вам не нужен. Эта информация вызывает у вас беспокойство по поводу теоретической уязвимости ПК? Тогда переведите сервер в режим ручного запуска. Для этого выполните следующие действия:

– введите в строке поиска слово «службы» (рис. 2.3.6) и нажмите на соответствующей иконке, которая появится в списке результатов поиска;

– в открывшемся окне найдите строку «Apache2.4» и щелкните на ней правой (правой!) кнопкой мыши — появится меню управления службой (рис. 2.3.7);

– кликните на строке «Свойства» и в окне на вкладке «Общие» найдите список «Тип запуска»;

– выберите пункт «Вручную», затем последовательно нажмите кнопки «Применить» и «ОК» (рис. 2.3.8);

– теперь остановите службу, нажав соответствующую ссылку в левой стороне окна (рис. 2.3.9);

– дождитесь окончания процесса (рис. 2.3.10);

– закройте вкладку «Службы».

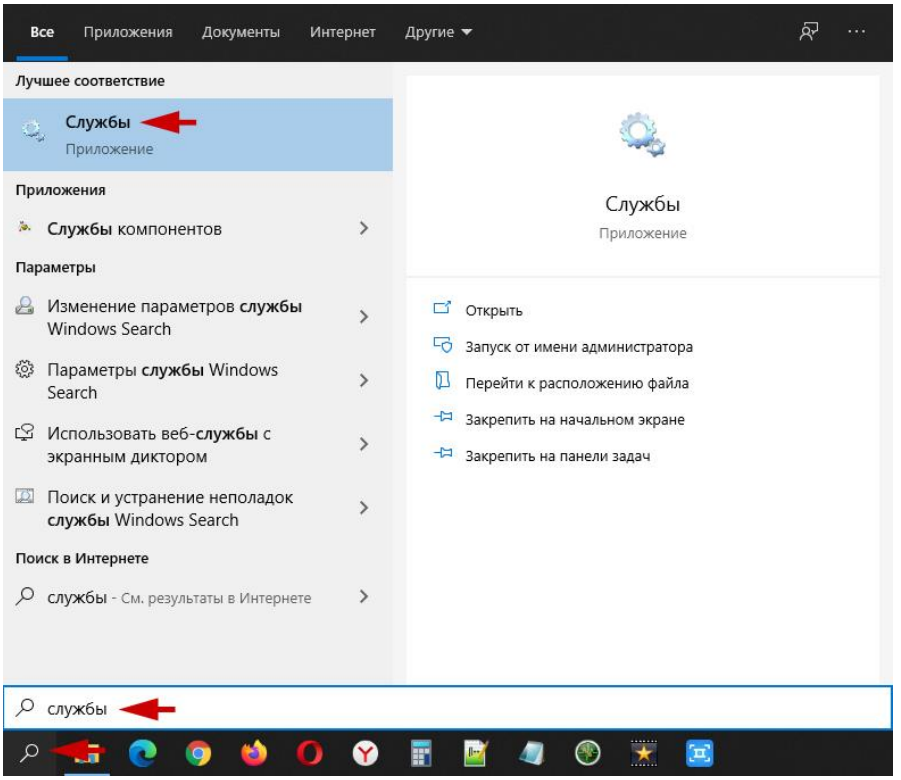

Рис. 2.3.6. Находим приложение «Службы»

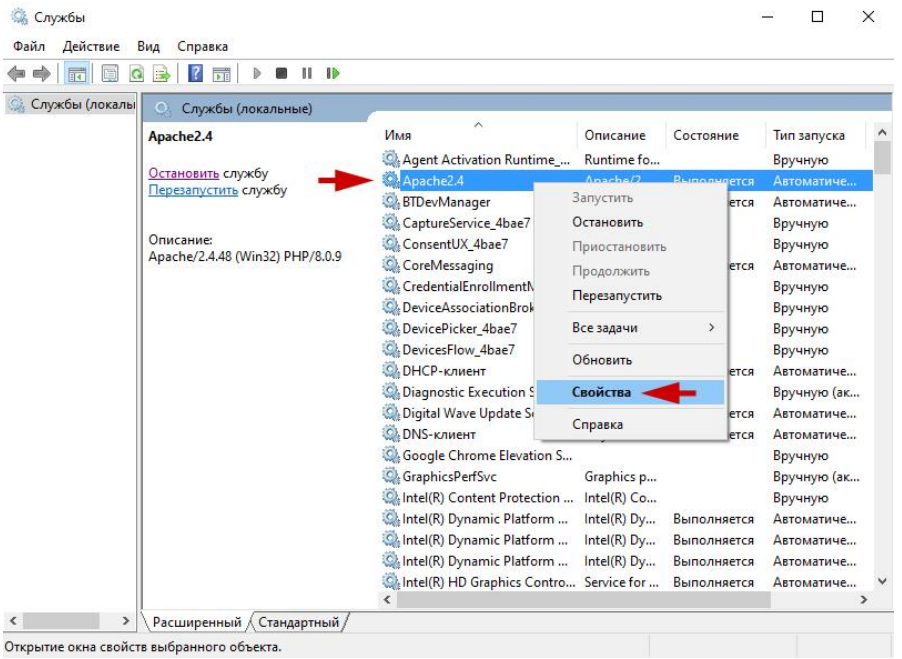

Рис. 2.3.7. Открываем меню управления службой

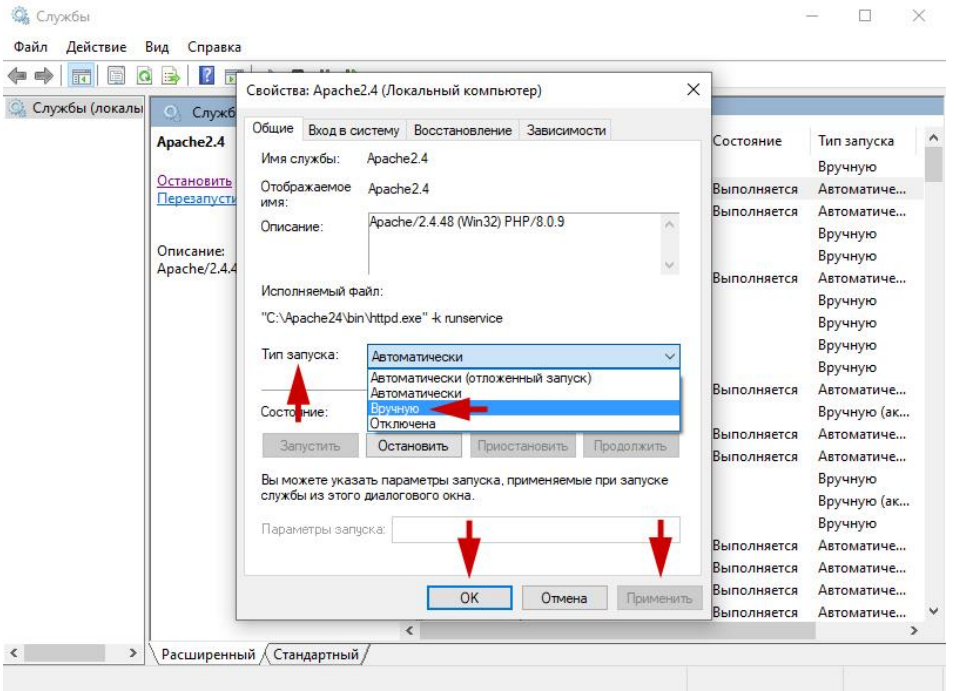

Рис. 2.3.8. Переводим службу в режим ручного запуска

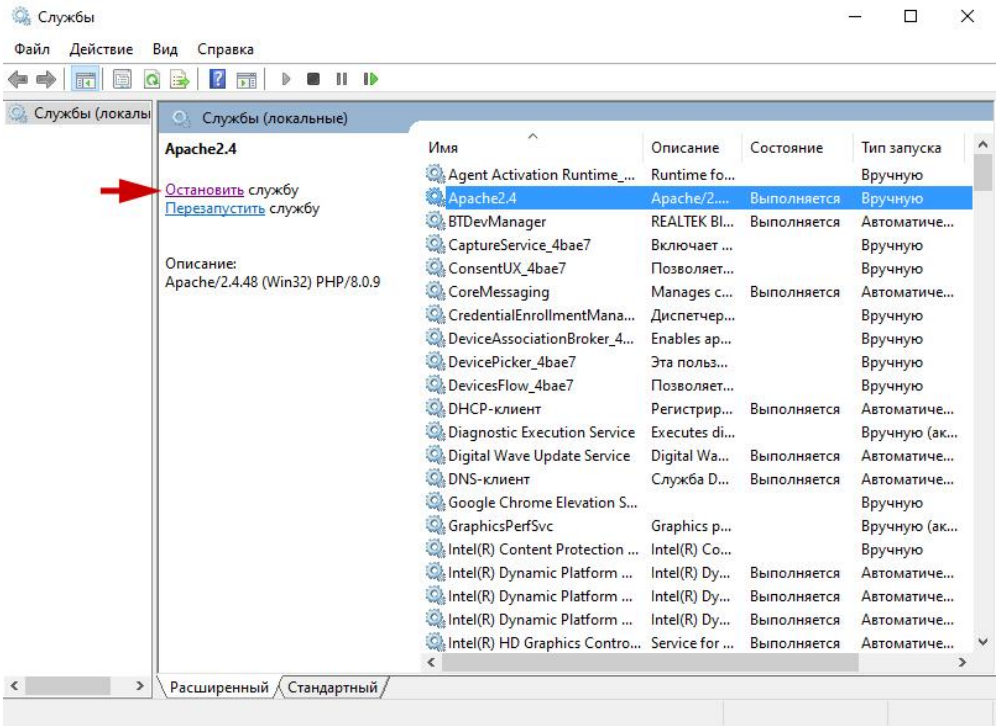

Рис. 2.3.9. Нажимаем ссылку «Остановить службу»

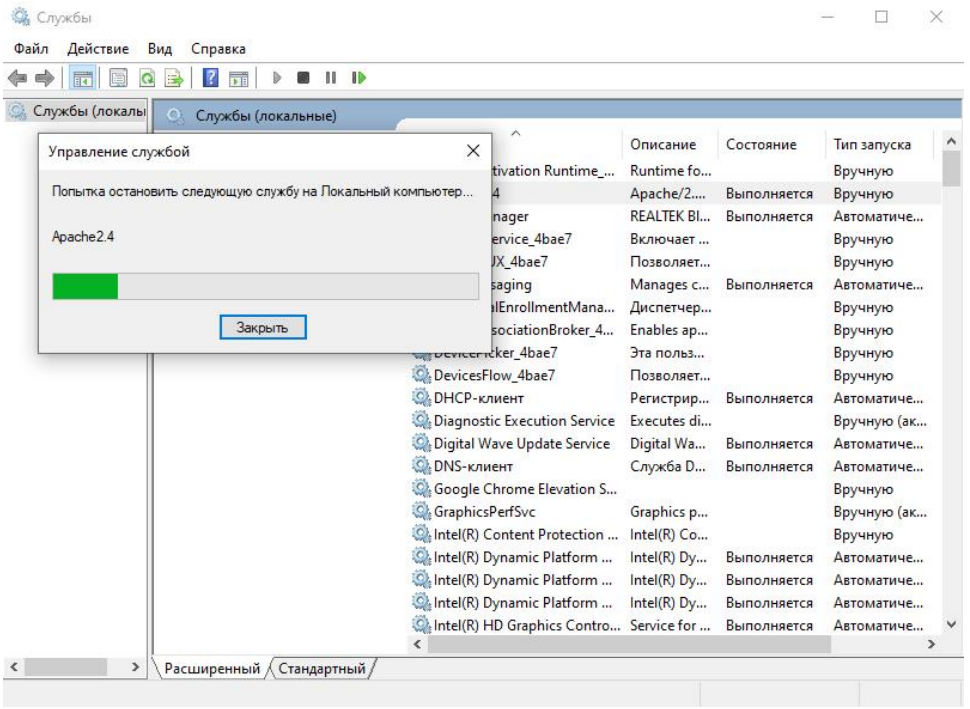

Рис. 2.3.10. Идет процесс остановки Apache

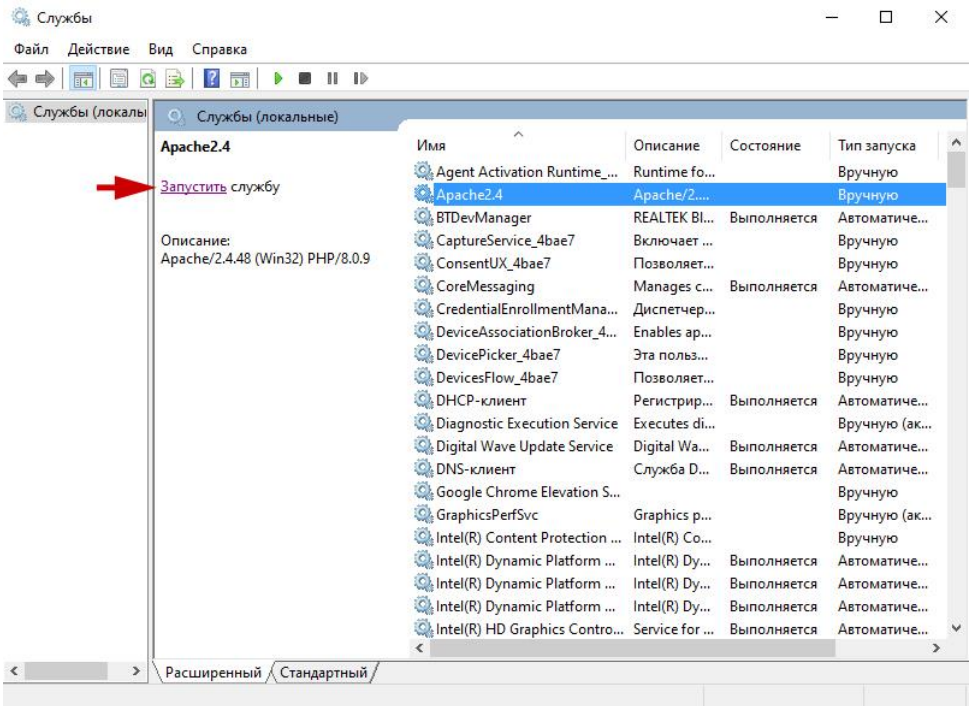

Рис. 2.3.11. Для запуска Apache нажмите ссылку «Запустить службу»

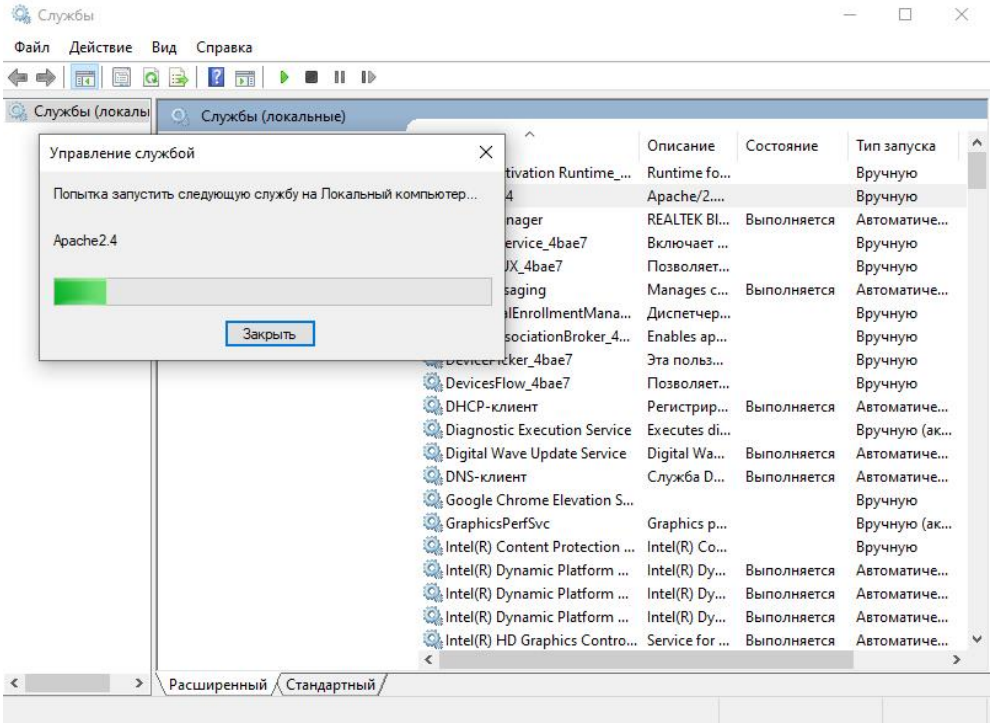

Рис. 2.3.12. Идет процесс запуска сервера

Теперь Apache остановлен и переведен в режим ручного запуска. При необходимости воспользоваться сервером снова зайдите в раздел «Службы», выделите строку «Apache2.4» и кликните на ссылке «Запустить службу» (рис. 2.3.11). Подождите окончания процесса (рис. 2.3.12).

Продолжаете беспокоиться о безопасности? Можете на время работы с сервером отключать свой ПК от Интернета. Напомню: локальный хостинг работает без подключения к сети.

2.4. Установка Python 3

Скачайте на рабочий стол компьютера с сайта **https://www.python.org/downloads/windows/** установочный файл, соответствующий разрядности вашей ОС (рис. 2.4.1). Запустите его.

Выберите вариант установки с настраиваемыми параметрами. Для этого нажмите ссылку «Customize installation» (рис. 2.4.2). В открывшемся окне ничего не меняйте, просто нажмите кнопку «Next» (рис. 2.4.3).

Основные настройки выполняются в следующем окне. Во-первых, поставьте «галочку» напротив пункта «Install for all users», чтобы интерпретатор был доступен всем пользователям. В строке «Customize install location» вручную введите адрес папки для интерпретатора: **C:\Python**. После этого нажмите кнопку «Install» (рис. 2.4.4).

Дождитесь окончания процесса (рис. 2.4.5). Нажмите кнопку «Close» (рис. 2.4.6).

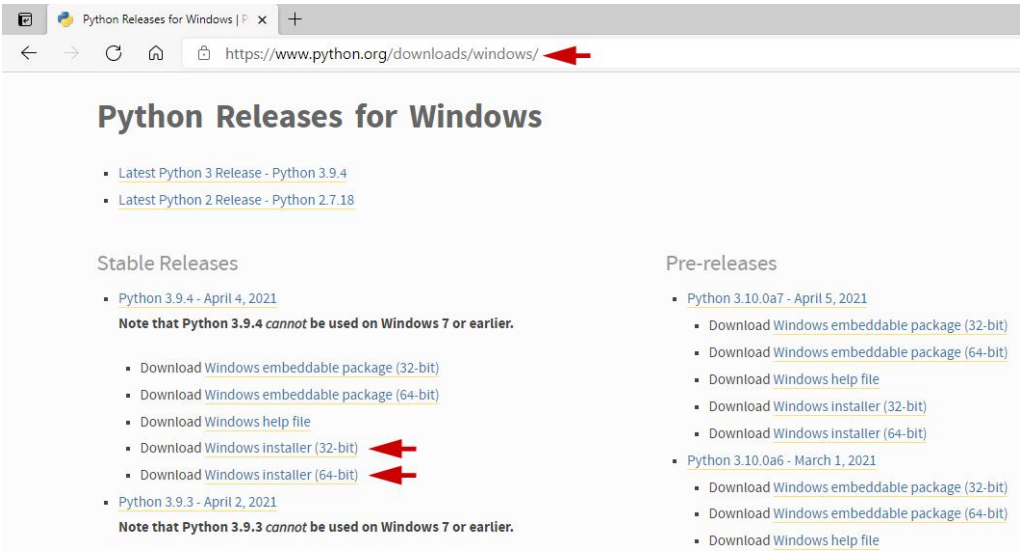

Рис. 2.4.1. Сайт с файлами интерпретатора Python

Рис. 2.4.2. Начало установки интерпретатора Python

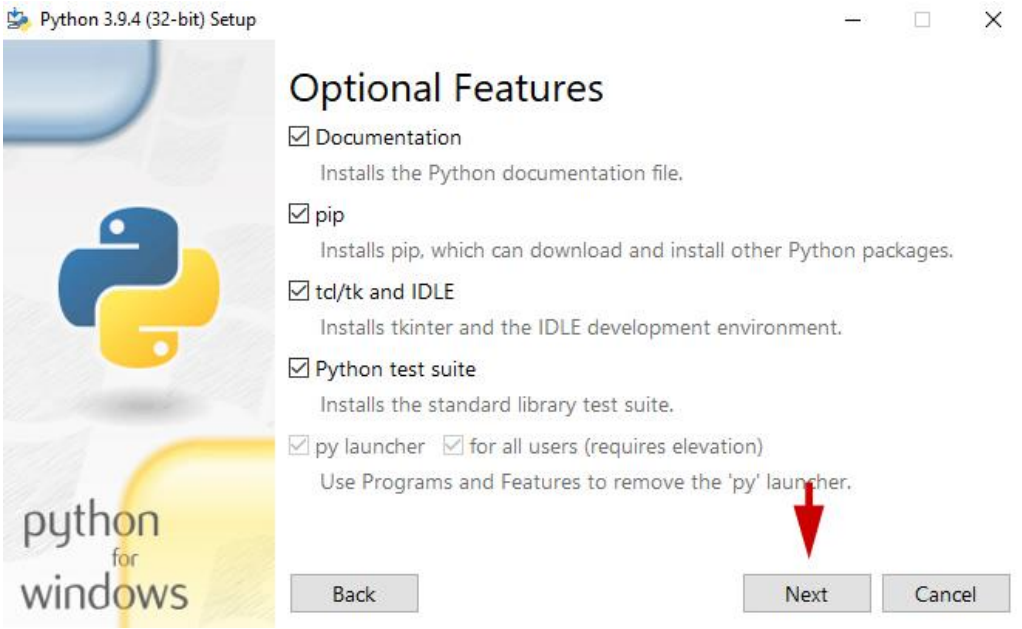

Рис. 2.4.3. Здесь просто жмем «Next»

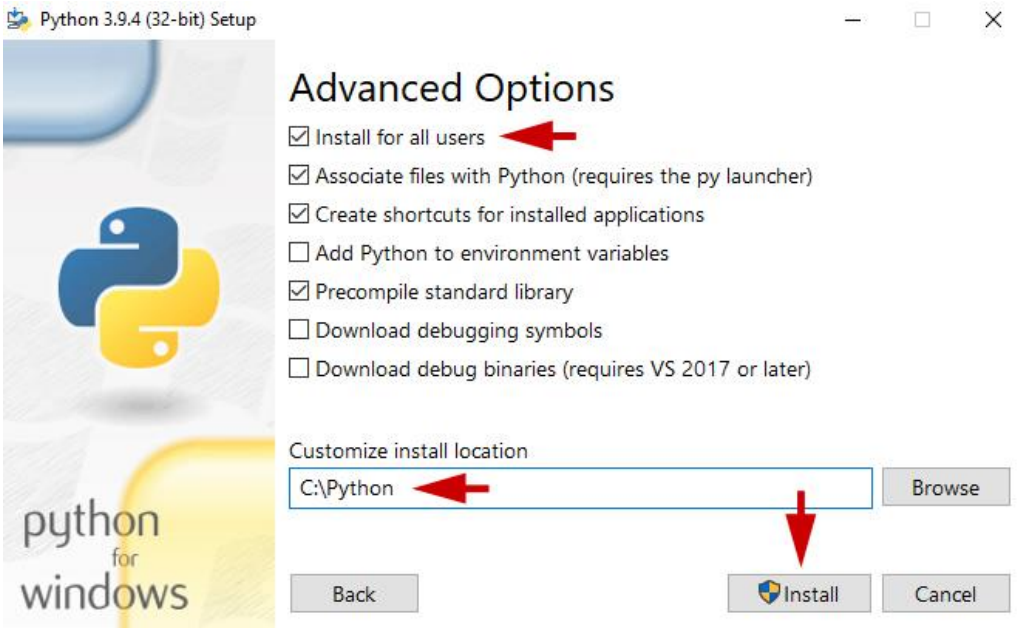

Рис. 2.4.4. Ставим «галочку» напротив пункта «Install for all users», меняем имя папки, в которую будет установлен интерпретатор, и начинаем инсталляцию

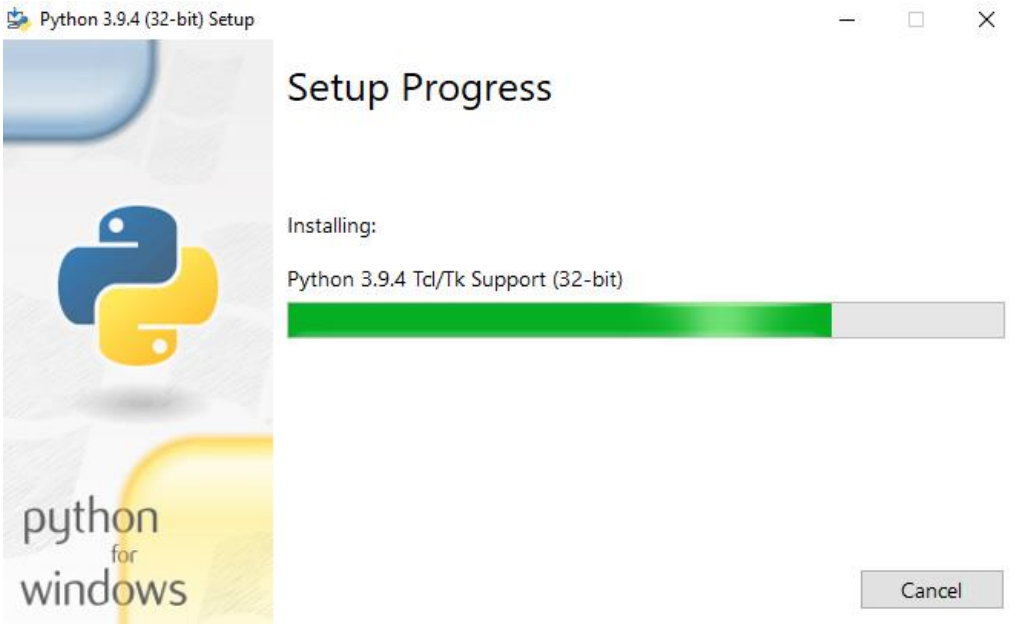

Рис. 2.4.5. Процесс установки файлов интерпретатора Python

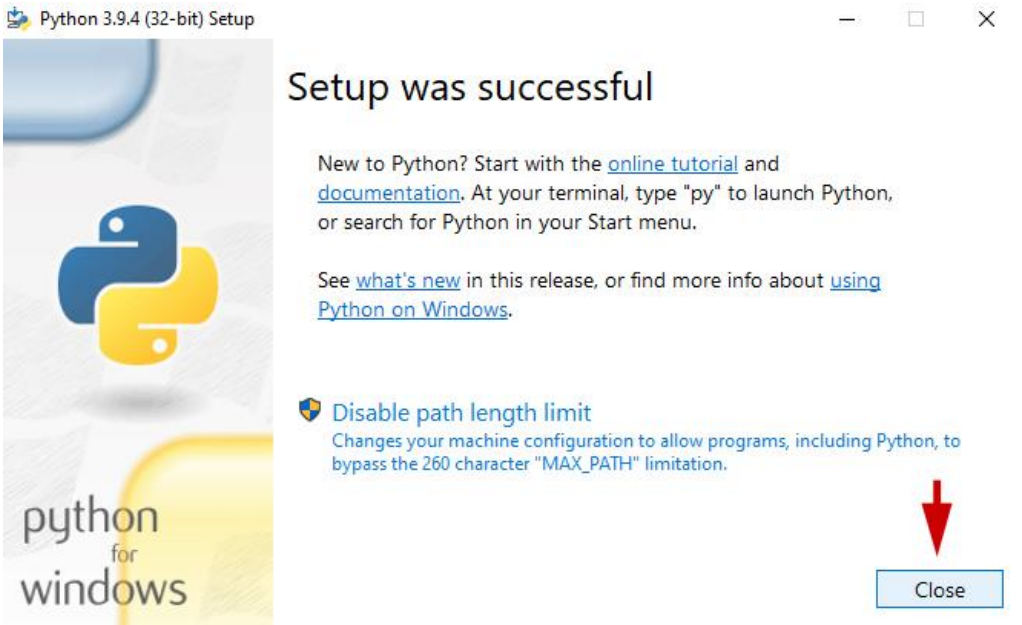

Рис. 2.4.6. Завершение установки

В папке **C:\Apache24\conf** с помощью текстового редактора «Блокнот» откройте файл **httpd.conf** (рис. 2.4.7), найдите в нем строку

```
DirectoryIndex index.html
```

и добавьте к ней

```
index.py
```

Должно получиться

```
DirectoryIndex index.html index.py
```

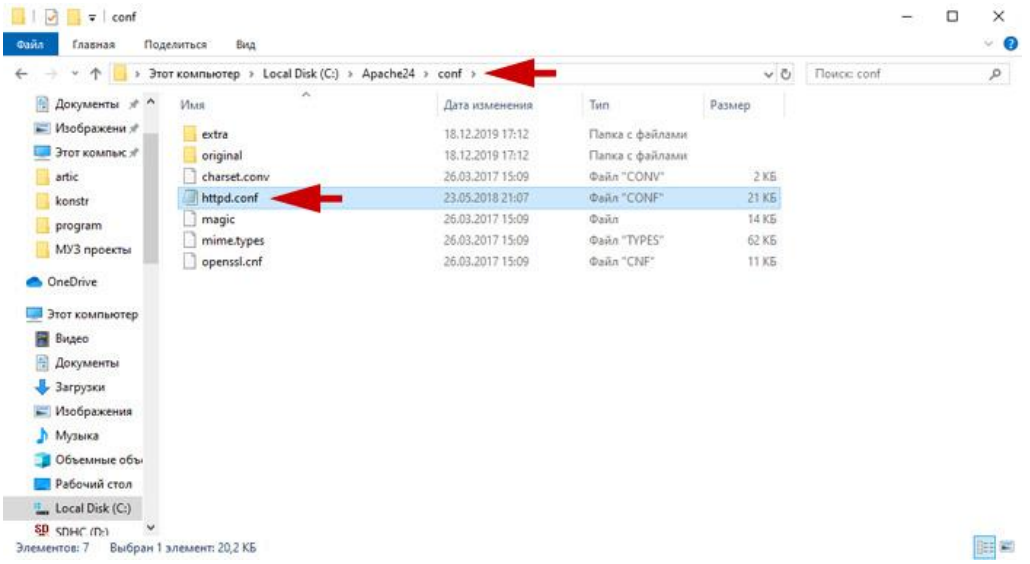

Рис. 2.4.7. Файл httpd.conf

Теперь найдите строку

```
Options Indexes FollowSymLinks
```

и добавьте в конце нее

```
ExecCGI
```

Должно получиться

```
Options Indexes FollowSymLinks ExecCGI
```

Дальше найдите строку

```
#AddHandler cgi-script .cgi
```

удалите в начале нее знак # и добавьте в конце строки

```
.py
```

Должно получиться

```
AddHandler cgi-script .cgi .py
```

И последняя операция: в самый конец файла **httpd.conf** добавьте строку

```
SetEnv PYTHONIOENCODING utf8
```

Сохраните изменения, закройте файл и перезагрузите компьютер.

Теперь убедимся, что Python заработал. Для этого в папке **C:\Apache24\htdocs** создайте текстовый файл и запишите в него следующий код:

```
#! C:/Python/python
print('Content-type: text/html\n\n')
print('PYTHON !')
```

Сохраните этот файл под именем **test.py**. Откройте ваш браузер и введите в строке адреса **http://localhost/test.py**. Если появилось сообщение «PYTHON !», значит, все в порядке (рис. 2.4.8).

Вы также можете упростить себе задачу, использовав готовый файл **test.py** из папки «Глава 2» zip-архива. Для этого файл необходимо скопировать и перенести в папку **C:\Apache24\htdocs**.

Осталось добавить, что файлы ваших проектов необходимо помещать в папку **htdocs** по адресу **C:\Apache24\htdocs**, а просматривать готовые страницы сайтов в браузере по адресу **http://localhost/** (для файла **index.py**) или **http://localhost/имя_страницы.py**, например **http://localhost/primer.py** или **http://localhost/test/primer.py**.

Рис. 2.4.8. Проверяем работу Python

2.5. Установка редактора Notepad++ 8

Скачать Notepad++ можно по адресу **https://notepad-plus-plus.org/downloads/** (рис. 2.5.1).

Устанавливается редактор следующим образом. Запустите скачанный файл. В первую очередь появится окно выбора языка программы. Оставляем русский (рис. 2.5.2) и нажимаем кнопку «ОК».

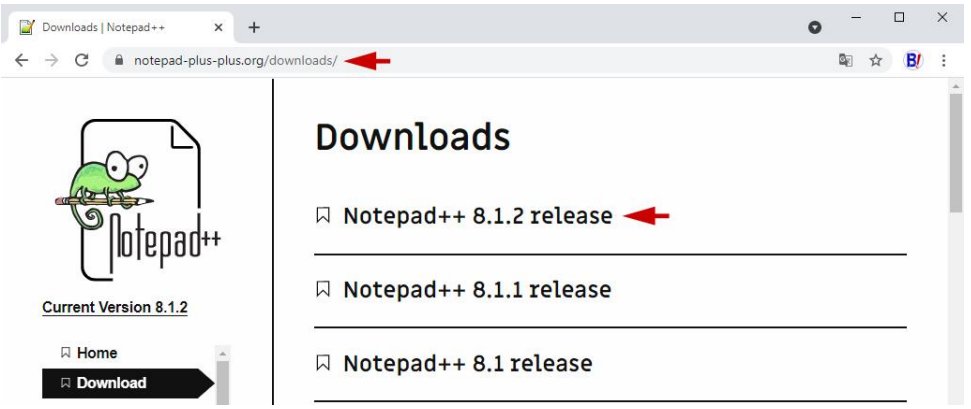

Рис. 2.5.1. Сайт с файлами редактора Notepad++

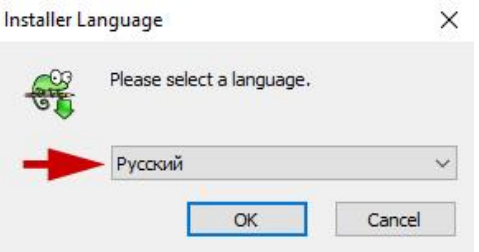

Рис. 2.5.2. Выбираем язык ПО

В следующем окне нажимаем кнопку «Далее» (рис. 2.5.3). Затем нажатием кнопки «Принимаю» соглашаемся с условиями лицензии (рис. 2.5.4).

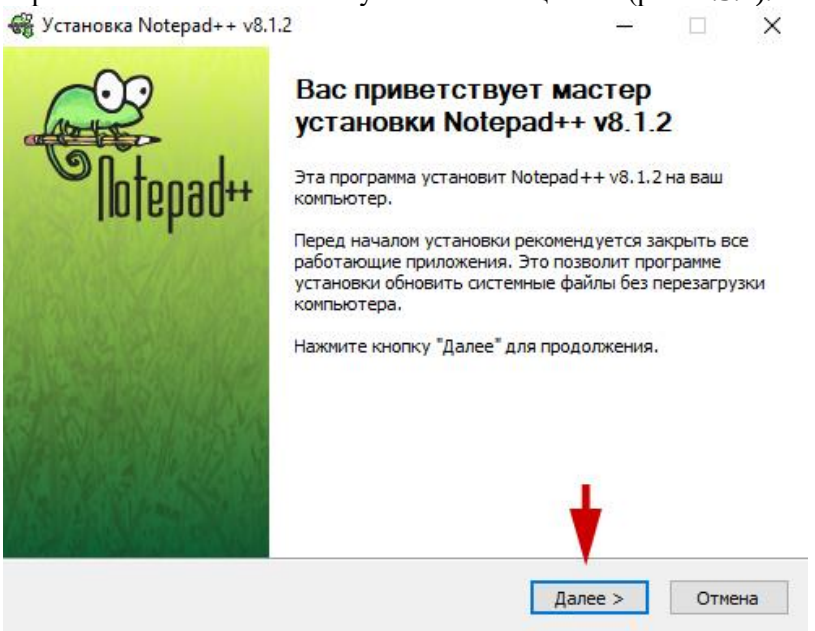

Рис. 2.5.3. Нажимаем кнопку «Далее»

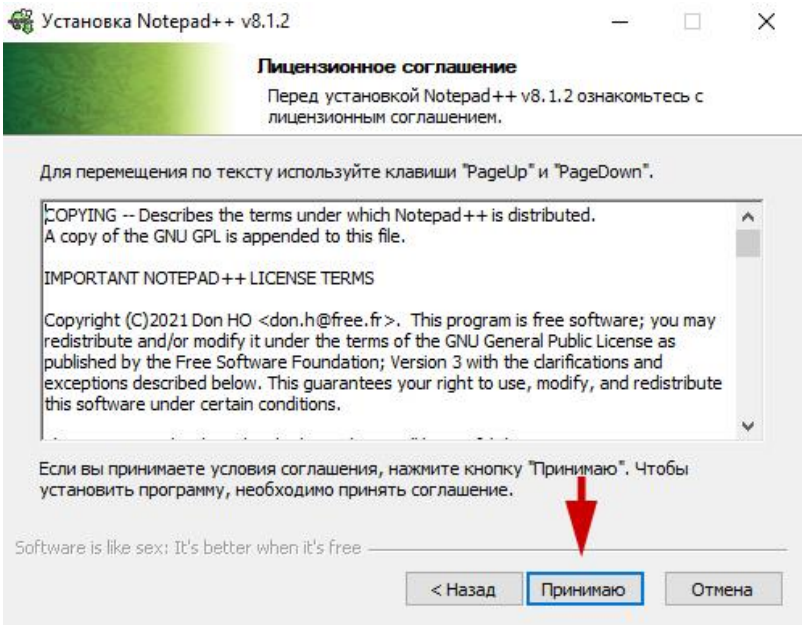

Рис. 2.5.4. Соглашаемся с условиями лицензии

Теперь нам необходимо выбрать папку для установки программы. По умолчанию предлагается **C:\Program Files\Notepad++**. Думаю, такой вариант установки подходит всем, поэтому данную папку оставляем без изменений (рис. 2.5.5).

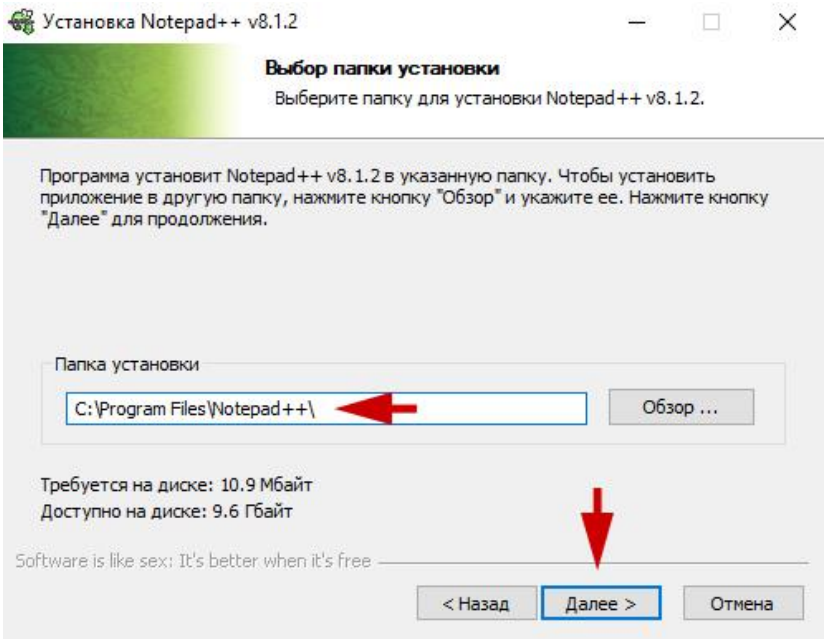

Рис. 2.5.5. Выбираем папку для установки программы

Дальше будет окно с выбором компонентов, которые можно установить вместе с программой. Этот раздел предназначен в первую очередь для опытных программистов, каковыми мы пока не являемся. Поэтому здесь ничего не меняем, жмем кнопку «Далее» (рис. 2.5.6).

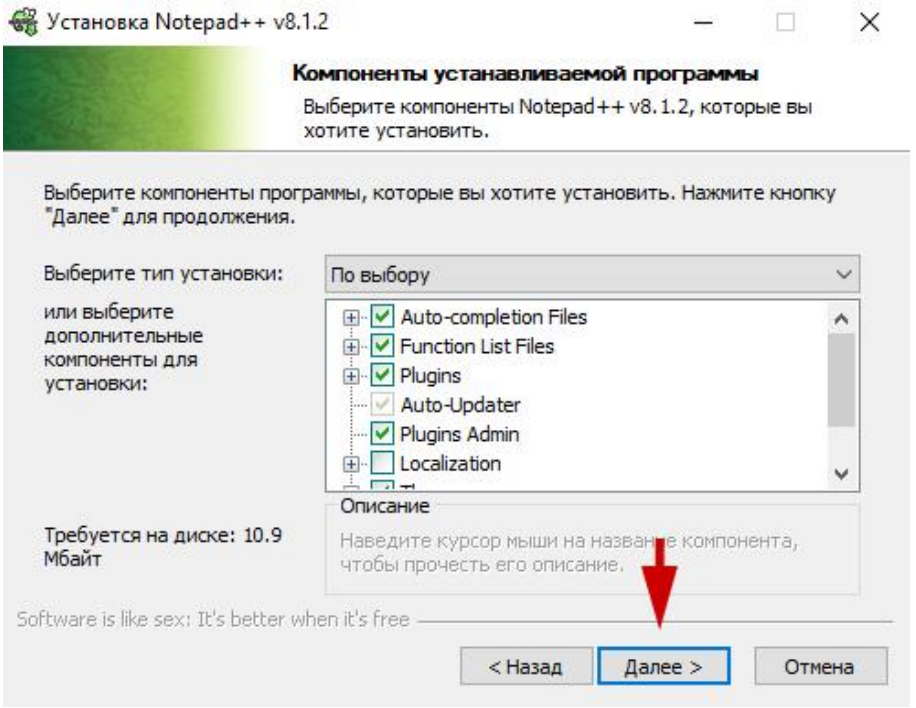

Рис. 2.5.6. Здесь ничего не меняем, жмем кнопку «Далее»

На последнем этапе перед началом непосредственной установки на вкладке появится пункт «Create Shortcut on Desktop» (рис. 2.5.7). То есть нам предлагают создать ярлык программы на рабочем столе. Делать это или нет — зависит от вашего желания. Замечу, что после установки редактора ссылка на него добавится в контекстное меню. Достаточно будет навести указатель мыши на файл, щелкнуть правой клавишей, и в списке возможных действий вы увидите строку «Edit with Notepad++». Кликните по ней — и файл будет открыт в редакторе.

Разобравшись с вопросом, необходим ли ярлык на рабочем столе или нет, нажмите кнопку «Установить». Дождитесь окончания процесса (рис. 2.5.8).

После завершения установки вам нужно будет принять еще одно решение: сразу запустить программу или отложить это дело «на потом». Для этого оставьте или снимите «галочку» в пункте «Запустить Notepad++» (рис. 2.5.9). Теперь нажмите кнопку «Готово». Поздравляю — отныне на вашем компьютере установлен профессиональный текстовый редактор!

Познакомимся с ним поближе. Как выглядит редактор, вы можете видеть на рисунке 2.5.10.

Notepad++ позволяет:

– выполнять поиск по файлу в разных направлениях;

– производить замену по шаблону;

– устанавливать кодировки, в том числе необходимую для нашего проекта UTF-8;

– менять страницы, форматируя их по стандартам разных операционных систем — Windows, Unix или macOS;

– подсвечивать код, выделяя разные по назначению фрагменты, операторы, функции, переменные, комментарии и т. д.;

– выбирать синтаксис документа (благодаря чему меняются варианты подсветки кода, например Python, HTML, CSS или JavaScript);

– сохранять файлы с разным расширением;

– менять стили оформления окна программы — вариантов достаточно, найдется любой по желанию;

– выполнять еще многие и многие операции.

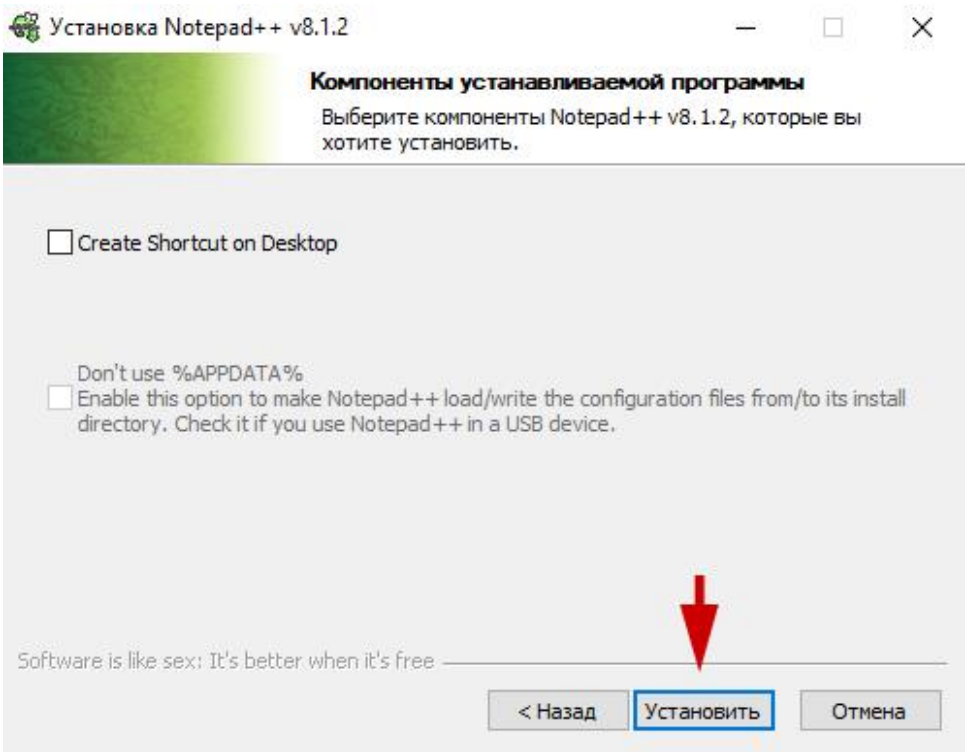

Рис. 2.5.7. Нажимаем кнопку «Установить»

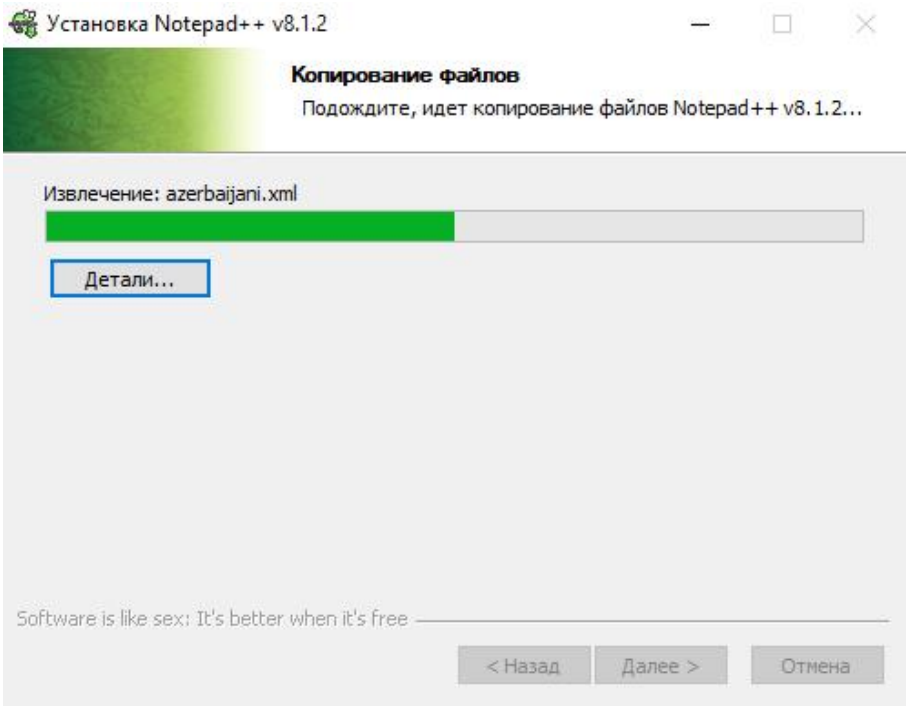

Рис. 2.5.8. Ждем окончания процесса

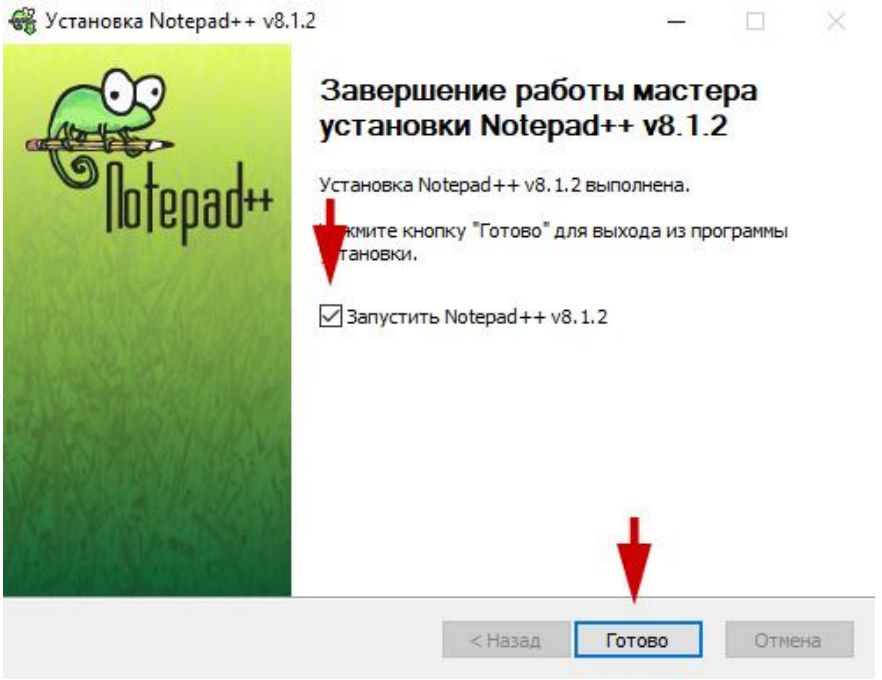

Рис. 2.5.9. Завершение установки

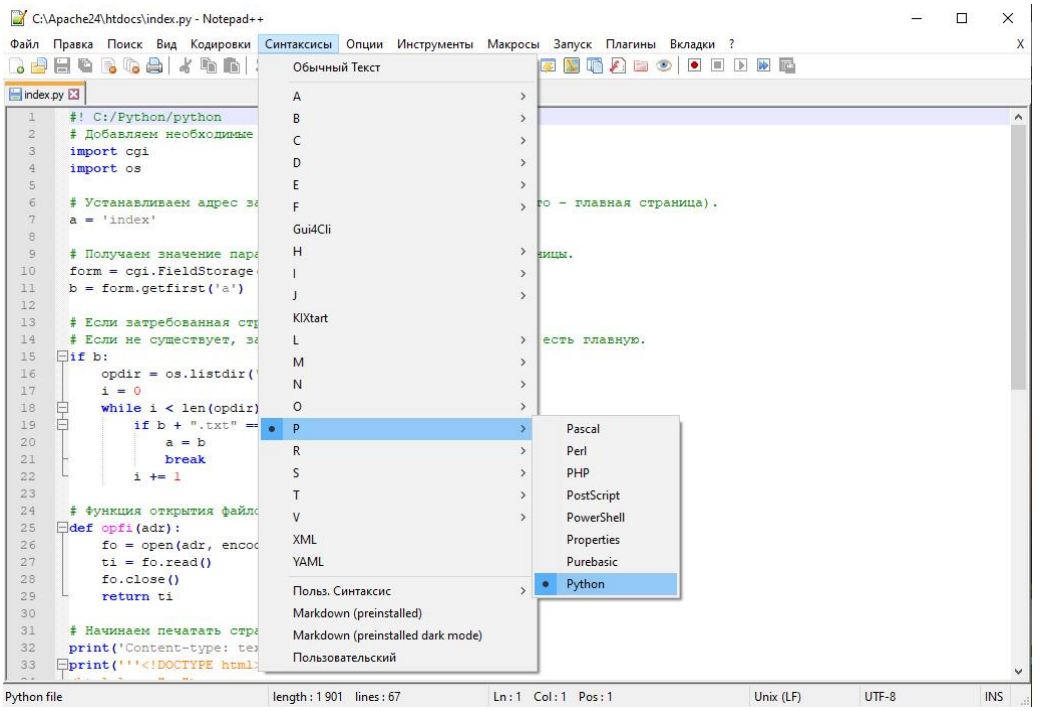

Рис. 2.5.10. Редактор Notepad++ с открытым в нем файлом Python

2.6. Среда разработки IDLE

Прочитав раздел 2.5 про редактор Notepad++, вы можете задать вопрос: а не удобнее ли использовать для написания программ редактор IDLE, поставляющийся вместе с интерпретатором Python? Автор даст на это отрицательный ответ. Мне кажется, что IDLE — слишком элементарное программное обеспечение. Notepad++ выигрывает по многим компонентам. Например, система автоматического завершения кода более удобна в Notepad++. Функциональность редакторов вообще несравнима: в IDLE она заметно ограничена по сравнению с Notepad++ — достаточно сравнить рисунки 2.5.10 (редактор Notepad++) и 2.6.1 (редактор IDLE). Наконец, Notepad++ — универсальный редактор. А это очень важно для наших проектов, ведь нам предстоит написать ряд примеров, в которых будет не только код Python, но также HTML, CSS и JavaScript.

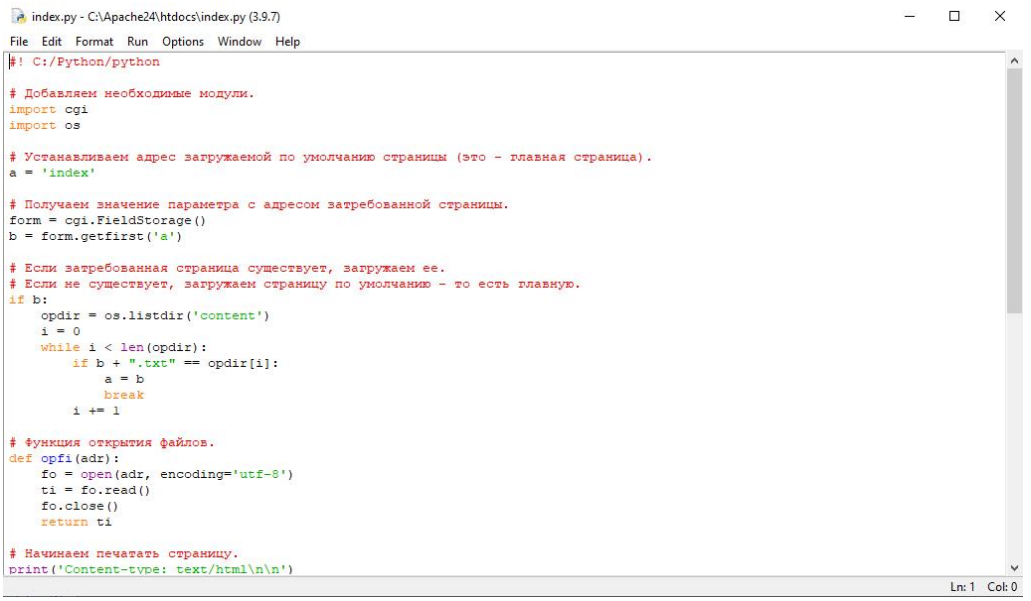

Рис. 2.6.1. Редактор IDLE с открытым в нем файлом Python

2.7. Валидация кода Python

Кроме проверки работоспособности примеров в браузерах автор тестировал код в online-валидаторах Python. На просторах Интернета существует множество таких систем. Автор экспериментировал примерно с полутора десятком из них. Но надежного валидатора мне обнаружить не удалось. В основном встречаются две крайности: валидаторы либо одобряют практически любой код (естественно, без ошибок), либо настолько придирчивы к элементам оформления, что цепляются к каждой мелочи, которая никак не влияет на работу программы.

В итоге свой выбор я остановил на одном валидаторе: **http://www.python-linter.com/** (ссылка — на странице **link.html** из папки «Глава 2» zip-архива). Он хорошо находит ошибки, достаточно строг к оформлению кода, но имеет свои недостатки, например: не признает никакие имена переменных. Подчеркиваю — именно никакие, даже самые что ни на есть правильные. Поэтому его можно использовать для обнаружения синтаксических ошибок, при этом не обращая внимания на претензии к переменным. Если читатели воспользуются этой ссылкой, учитывайте данные особенности валидатора. Как он выглядит, вы можете видеть на рисунке 2.7.1.

Неприятную особенность валидатора можно компенсировать проверкой имен переменных в редакторе IDLE. Откройте в нем файл вашей программы. Все переменные должны быть набраны черным шрифтом. Если это так, значит, имена всех переменных вполне корректны.

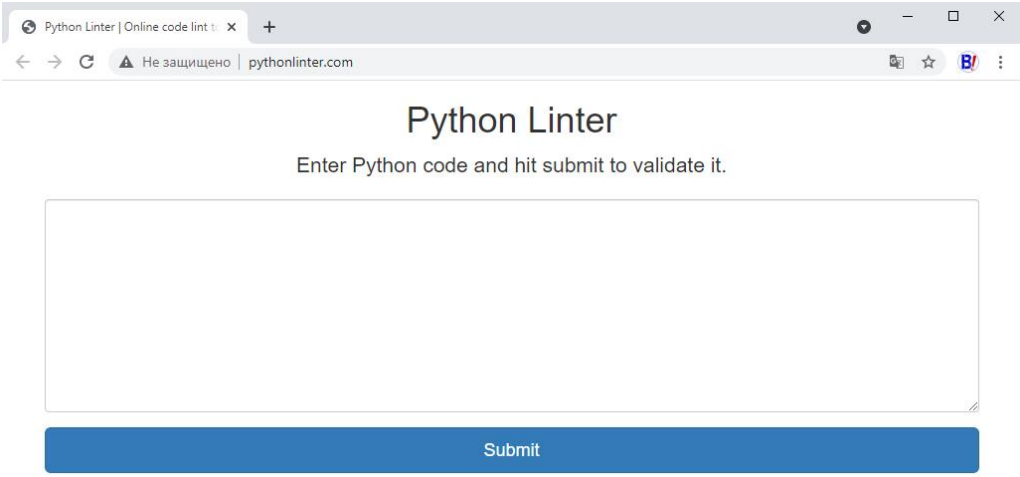

Рис. 2.7.1. Валидатор кода Python

3. Теория

В этой главе будет много примеров кода. Напомню, что все они собраны в папке «Глава 3» zip-архива. Чтобы запускать эти примеры в браузере, выполните следующие действия:

– удалите из папки **C:\Apache24\htdocs** тестовый файл **test.py**;

– скопируйте все содержимое папки «Глава 3» zip-архива;

– поместите скопированные папки и файлы в папку **C:\Apache24\htdocs**;

– чтобы запустить файл в браузере, в строке адреса введите **http://localhost/file_name.py**, где **file_name.py** — имя файла.

Python обладает большими функциональными возможностями. В этой главе мы рассмотрим далеко не все из них, а только те, что соответствуют двум основным требованиям.

1. Материал должен быть простым и легким в восприятии. Автор не ставил своей целью рассказать обо всех синтаксических особенностях этого языка. Если вы хотите узнать о Python больше, прочтите какую-нибудь более объемную и обстоятельную книгу.

2. Материал должен иметь первостепенную важность именно для web-программирования на Python. При этом уже в который раз констатируем, что все примеры кода (за одним исключением), иллюстрирующие наши рассуждения, мы будем тестировать в браузере. Писать в оболочке Python и запускать файлы из командной строки нам не придется.

3.1. Первая программа

Первую программу мы с вами создали, когда устанавливали интерпретатор Python (раздел 2.4). Как вы помните, она называлась **test.py**. Программа содержала три строки кода:

```
#! C:/Python/python
print('Content-type: text/html\n\n')
print('PYTHON !')
```

Разберемся, что за инструкции мы написали.

В первой строке мы указали путь к интерпретатору Python. Эта строка нужна, чтобы сервер Apache при запуске программы знал, где найти исполняемый файл. На компьютере он находится по адресу **C:/Python/python.exe**. Однако указывать расширение **.exe** совсем не обязательно, так как сервер ищет именно исполняемый файл, а не какой-либо другой. Поэтому первая строка у нас выглядит так:

```
#! C:/Python/python
```

Любой файл Python будет у нас всегда начинаться с этой строки. Подчеркну важность данного факта: инструкция с указанием пути к интерпретатору должна быть первой, какова бы ни была ваша программа!

Специальные символы **#!** указывают серверу, что данная строка содержит путь к исполняемому файлу.

Многие программисты утверждают, что первую строку с указанием пути к интерпретатору необходимо писать без пробела. На самом деле, это не более чем дань традиции. И на локальном хостинге, и на реальном (там путь к интерпретатору будет выглядеть по-другому) наличие или отсутствие пробела не играет никакой роли. Даже валидаторы кода Python не видят в пробеле никакого нарушения стандартов. Вы можете писать так, как вам больше нравится:

```
#! C:/Python/python
```

или

```
#!C:/Python/python
```

Продолжим. Вторая инструкция

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

нужна, чтобы сообщить браузеру, документ какого типа предстоит загрузить. В нашем случае это будет текст в формате HTML. Два символа перевода строки в конце инструкции отделяют указание типа документа от его содержимого.

Наконец, третья инструкция

```
print('PYTHON !')
```

думаю, понятна всем: выводим в окно браузера короткий текст, сообщающий о том, что мы приступаем к изучению Python.

Попутно хочу обратить внимание, что мы отступили от набившей оскомину практики демонстрировать в первой программе текст «Hello, World!».

Всем хороша наша программа — но только до тех пор, пока мы не захотим напечатать русские слова. Давайте, попробуем (файл **3_1_1.py**):

```
#! C:/Python/python
print('Content-type: text/html\n\n')
print('Изучаем язык программирования Python !')
```

Пишем приведенный выше код, сохраняем файл в кодировке **utf-8** и запускаем в браузере. Что у нас получилось, видно на рисунке 3.1.1.

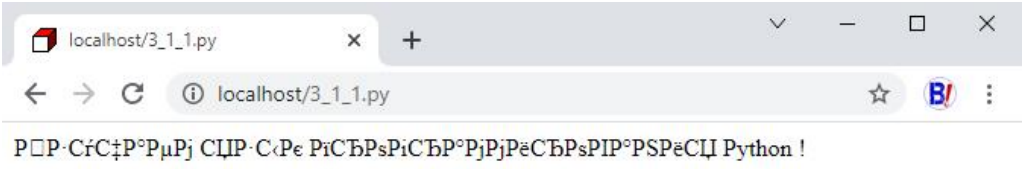

Рис. 3.1.1. Запускаем в браузере русскоязычный файл

Нужно признаться, что мы ожидали совершенно иного результата, а не этот набор непонятных символов. А все потому, что для букв русского алфавита необходимо явное указание кодировки в строке с описанием типа документа. Попробуем изменить ее:

```
print('Content-type: text/html; charset=utf-8\n\n')
```

Теперь наша программа выглядит следующим образом (файл **3_1_2.py**):

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')
print('Изучаем язык программирования Python !')
```

Явное указание кодировки дало нам желаемый результат: текст выводится правильно (рис. 3.1.2).

Рис. 3.1.2. Запускаем в браузере исправленный русскоязычный файл

Итак, разбор нашей первой, весьма элементарной программы закончен. Но это еще не все в данном разделе. Программы, написанные в качестве примеров к главе 3, в обязательном порядке будут содержать две строки с указаниями: пути к интерпретатору и типа загружаемого документа (вместе с кодировкой). Если вы планируете не просто запускать файлы из архива, а писать их самостоятельно, для программ главы 3 советую вам создать шаблон, который можно будет копировать в начало каждого файла Python:

```
#! C:/Python/python
```

```
print('Content-type: text/html; charset=utf-8\n\n')
```

Создавая первую программу, автор сохранял все три ее версии в кодировке utf-8. Это правило распространяется на любые примеры из книги. Если вы будете самостоятельно писать код из глав 3, 4 и 5, обязательно сохраняйте все файлы — Python, текстовые, CSS, JavaScript — в формате utf-8.

Как это сделать? Откройте редактор Notepad++ и наберите код программы. Затем в меню редактора кликните закладку «Кодировки» и посмотрите, в каком формате написан файл. Если не в **utf-8**, то в открывшемся меню щелкните на строке «Преобразовать в UTF-8» (рис 3.1.3). Сохраните результаты вашей работы.

Некоторые дополнительные факты о применении кодировки **utf-8** вы узнаете также в разделе 3.19.

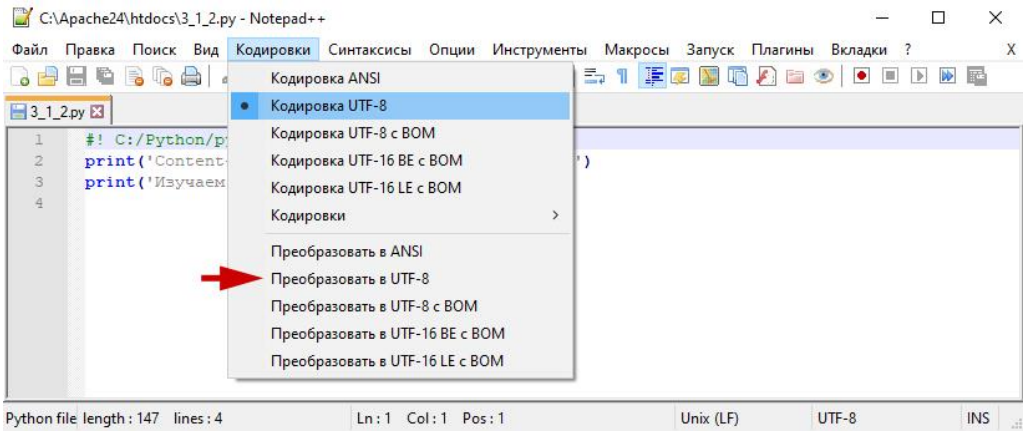

Рис. 3.1.3. Выбор кодировки для файла первой программы

3.2. Типы данных

Python оперирует данными самых разных типов. Мы рассмотрим двенадцать из них — наиболее важных, с точки зрения автора, для web-программирования. Ниже приведен их перечень с примерами.

str — строки. Могут содержать практически любые знаки и символы, включая HTML-теги. Пример:

```
'Python'
```

int — целые числа:

```
5
```

float — вещественные числа:

```
5.5
```

bool — логические данные (всего два значения):

```
True, False
```

NoneType — к этому типу принадлежат объекты, у которых значения отсутствуют:

```
None
```

list — списки (аналогичны массивам, например в PHP):

```
[0, 1, 2, 3]
```

tuple — кортежи (о них мы поговорим в разделе 3.13):

```
(0, 1, 2, 3)
```

set — множества (о них — в разделе 3.14):

```
{0, 1, 2, 3}
```

range — диапазоны (о них — в разделе 3.15):

```
range(1, 5)
```

dict — словари (аналогичны ассоциативным массивам, например в PHP)

```
{'x': 1, 'y': 'Python', 'z': 5.5}
```

function — функции:

```
def a(): 1 + 1
```

module — модули:

```
import cgi
```

Запомнить основные типы данных нам поможет программа **3_2_1.py**. Это будет единственный файл, который мы станем запускать не в браузере, а в интерактивной оболочке Python. Код программы показан ниже. Чтобы запустить ее, сделайте двойной щелчок мышью на иконке файла (рис. 3.2.1).

У файлов, запускаемых таким образом, есть один недостаток: вкладка оболочки откроется, программа моментально выполнится и вкладка снова исчезнет. Произойдет это так быстро, что вы ничего не увидите. Поэтому мы применим маленькую хитрость: добавим в конце файла функцию пользовательского ввода

input. В этом случае Python ожидает от нас ввода данных и не закрывает окно, позволяя нам увидеть результаты работы программы. Естественно, что на самом деле ничего больше вводить не надо, а просто изучите полученные результаты. Разобрать их очень просто: сначала идут данные, а строкой ниже — указание их типа. Пустая строка — и новые данные.

```
print("'Python'")
print(type('Python'))
print("")

print("5")
print(type(5))
print("")

print("5.5")
print(type(5.5))
print("")

print("True")
print(type(True))
print("")
```

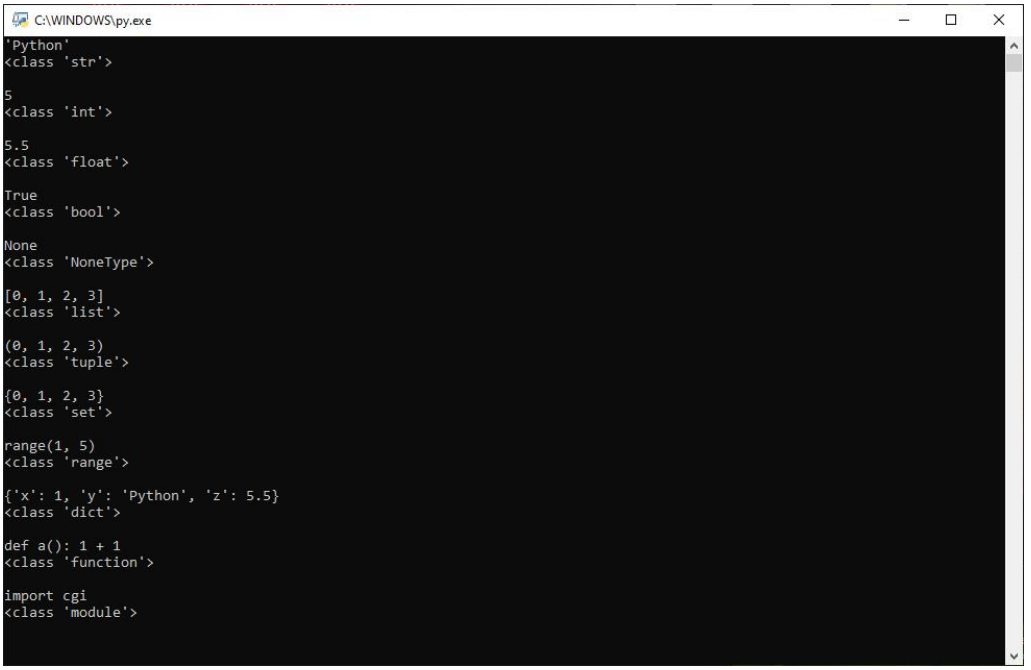

Рис. 3.2.1. Окно программы, демонстрирующей данные различных типов

```
print("None")
print(type(None))
print("")

print("[0, 1, 2, 3]")
print(type([0, 1, 2, 3]))
print("")
```

```
print("(0, 1, 2, 3)")
print(type((0, 1, 2, 3)))
print("")

print("{0, 1, 2, 3}")
print(type({0, 1, 2, 3}))
print("")

print("range(1, 5)")
print(type(range(1, 5)))
print("")

print("{'x': 1, 'y': 'Python', 'z': 5.5}")
print(type({'x': 1, 'y': 'Python', 'z': 5.5}))
print("")

def a(): 1 + 1
print("def a(): 1 + 1")
print(type(a))
print("")

import cgi
print("import cgi")
print(type(cgi))

input()
```

Нужно также отметить, что данные в Python делятся еще на две группы:

– **последовательности** — к ним относятся строки, списки, кортежи и диапазоны;

– **отображения** — к ним относятся словари.

3.3. Переменные

Переменные корректно сравнить с контейнерами, в которых временно или постоянно что-то хранится. В Python все данные являются объектами. Соответственно, переменные в Python — это контейнеры, в которых хранятся ссылки на определенные объекты. Эти объекты можно использовать, изменять или удалять программными методами.

Переменные в Python не требуют предварительного объявления и указания типа данных, которые они способны хранить. Они инициализируются в процессе присвоения им ссылки на какой-либо объект.

В Python существуют так называемые ключевые слова, которые нельзя использовать в качестве имен переменных. Вот их список:

```
False
None
True
__peg_parser__
and
as
assert
async
await
```

```
break
class
continue
def
del
elif
else
except
finally
for
from
global
if
import
in
is
lambda
nonlocal
not
or
pass
raise
return
try
while
with
yield
```

Правильные имена переменных могут состоять из латинских букв, цифр и символа подчеркивания. При этом они не могут начинаться с цифры. Кроме того, имена переменных чувствительны к регистру. **VAR**, **Var** и **var** — это разные переменные. Автор советует назначать всем переменным уникальные имена и не использовать одноименные в разных регистрах. Так вы избежите многих ошибок в процессе написания программ.

Правильные имена переменных:

```
a
adr
t2
n3b
varName
d_var
```

Присвоение значений переменным выполняется с помощью оператора =. Делается это так:

```
x = 5
y = 5.5
z = 'Python'
```

Существует также способ группового присвоения значений сразу нескольким переменным:

```
x = y = z = 1
```

или

```
x, y, z = 1, 2, 3
```

В результате присвоения значения переменной в ней сохраняется ссылка на объект.

Одна и та же переменная в разных блоках программы может неоднократно менять свое значение и поочередно хранить данные разного типа. Это хорошо иллюстрируется программой **3_3_1.py**. Вот ее код:

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')
a = 5
print(a)
print('<br>')
a = 'Python'
print(a)
print('<br>')
a = 5.5
print(a)
```

Как видите, в начале программы переменная **a** по типу сохраненных данных относится к **int**, затем ей присваивается строковое значение, а в конце в ней оказывается вещественное число.

Если запустить файл **3_3_1.py** в браузере, то вы можете проследить, как меняется значение переменной (рис. 3.3.1).

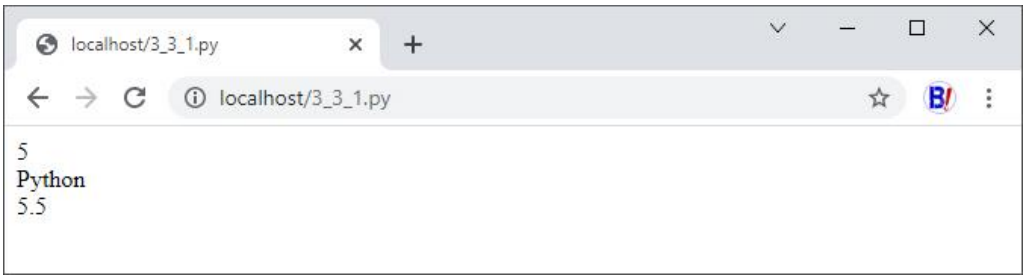

Рис. 3.3.1. Изменение значения переменной

Переменные также подразделяются на локальные и глобальные. Локальными называют те переменные, которые объявлены внутри функции. Доступны локальные переменные тоже только внутри функции. Соответственно, понятно, что глобальные переменные объявляются вне функций. Глобальные переменные доступны в любой части программы, в том числе и в теле функции. Если

имена глобальной и локальной переменных совпадают, то функция оперирует локальной переменной, не изменяя значения глобальной. И снова автор советует присваивать всем переменным уникальные имена, чтобы не запутаться в них.

Важное место в работе с переменными занимает изменение типа данных. Для этого существуют несколько функций. Мы упомянем пять из них.

str — преобразует данные в строку. Например, есть переменная

```
a = 1
```

Если мы захотим напечатать строку со следующим текстом:

```
print('Значение переменной a - ' + a)
```

то у нас ничего не выйдет, так как мы пытаемся в одной функции использовать данные разного типа — строку и число. Чтобы процесс завершился успешно, надо преобразовать число в строку, записав функцию печати следующим образом:

```
print('Значение переменной a - ' + str(a))
```

int — преобразует объект в целое число. Здесь можно привести обратный пример:

```
a = '1'
print(1 + a)
```

Ошибку вызовет тот факт, что в выражении одно слагаемое — число, а второе — строка. Необходимо привести строку '**1**' к числовому значению:

```
print(1 + int(a))
```

float — преобразует данные в вещественное число. Например, операция

```
float(1)
```

даст результат

```
1.0
```

list — преобразует набор отдельных элементов в список. Вот такая операция

```
list('Python')
```

завершится возникновением следующего списка:

```
['P', 'y', 't', 'h', 'o', 'n']
```

bool — приводит объект к логическому типу данных. Каждое из перечисленных ниже выражений будет истолковано как **False**:

```
bool(0)
bool('')
```

А вот эти выражения дадут результат **True**:

```
bool(1)
bool('Python')
```

3.4. Кавычки

В Python, как и во многих других языках программирования, строки заключаются в кавычки. Мы уже использовали их в трех предыдущих разделах данной главы. Теперь дадим некоторые полезные разъяснения.

Существует два вида кавычек:

– '' — одинарные (апострофы);

– "" — двойные.

С точки зрения синтаксиса строка, заключенная в двойные или одинарные кавычки, — одно и то же. Главное, чтобы открывающие и закрывающие кавычки были одного типа.

Оба типа удобно использовать одновременно, когда необходимо заключить в кавычки какой-то фрагмент строки. Например:

```
print('Web-программирование на "чистом" Python')
```

Кавычки, уже использованные в качестве ограничителей, могут применяться в строке только вместе с обратными слешами перед ними:

```
print('Web-программирование на \'чистом\' Python')
```

Такой прием называется экранированием. Однако ставить множество слешей — непродуктивно. Гораздо удобнее использовать именно разные кавычки.

Данная тема особенно актуальна для web-программирования, где принято значения атрибутов тегов заключать в кавычки.

В HTML-разметке страницы чаще всего используют двойные кавычки. Например, так:

```
<img src="abs/1.jpg" alt="Фото">
```

Чтобы напечатать такую строку, надо заключить ее в одинарные кавычки:

```
print('<img src="abs/1.jpg" alt="Фото">')
```

Мы с вами в этой книге будем придерживаться следующего правила: двойные кавычки — для HTML-разметки, одинарные — для ограничения строки в коде Python.

3.5. Комментарии

В языке Python существует 2 способа размещать сопровождающую информацию внутри программы: однострочные комментарии и строки документирования.

Однострочный комментарий начинается со знака решетки:

```
# Это комментарий
```

Например:

```
# Получаем данные из формы
form = cgi.FieldStorage()
```

Строки документирования обрамляются тройными кавычками:

```
'''Это пояснения
  к программе.
  Они состоят
  из нескольких строк'''
```

Например:

```
def isImageType(t):
  '''
  Проверяем, является ли объект изображением.
  Параметр t - проверяемый объект
  Возвращает True, если объект является изображением
  '''
  return hasattr(t, 'im')
```

Как вы понимаете, комментарии размещают в коде программы, чтобы пояснить ее отдельные фрагменты. Также комментарии могут использоваться в качестве различных временных заметок, необходимых в процессе написания программы. Эти заметки по завершении разработки обычно удаляют.

Пояснение фрагментов кода целесообразно по двум причинам.

1. По прошествии времени разработчику иногда трудно вспомнить, что делают те или иные части программы. Наличие комментариев и строк документирования упрощает этот процесс.

2. Если вы работаете в IT-компании, необходимо, чтобы ваш код понимали не только вы, но и другой программист, которому, вполне возможно, придется использовать его в качестве составной части какого-либо проекта.

Дам несколько советов по использованию комментариев.

1. Не загромождайте код комментариями к тем частям, назначение которых понятно без лишних слов. Пример:

```
# Присваиваем счетчику начальное значение
i = 0
```

Здесь комментарий явно лишний — любой программист, увидев данную строку, сразу поймет ее назначение.

2. Если код достаточно сложный, можно дать комментарии не только к отдельным фрагментам, но и ко всей программе в целом, поместив соответствующие разъяснения в самом начале.

3. Старайтесь вставлять комментарии к блокам программы, а не к каждой ее строке. Пример неудачного расположения комментариев:

```
# Получаем данные из формы
form = cgi.FieldStorage()
# Если данные существуют
if form:
  # то получаем значение параметра
  sum = int(form.getfirst('s'))
  # Вводим переменную для сохранения результатов
  rs = 1
  # Вводим счетчик
  i = 0
  # Запускаем цикл
  while i < sum - 1:
```

В этом фрагменте комментируется каждое действие, в результате чего строки кода «теряются» среди строк пояснений. А это, в свою очередь, усложняет восприятие программы. Лучше сгруппировать все пояснения в несколько строк, расположенных перед блоком кода или внутри него:

```
# Получаем данные из формы.
# Если они существуют
# то выясняем значение параметра.

form = cgi.FieldStorage()
if form:
  sum = int(form.getfirst('s'))

  # Вводим переменные:
  # rs - для сохранения результатов;
  # i – счетчик.
  # Запускаем цикл.

  rs = 1
  i = 0
  while i < sum - 1:
```

При таком «построении» и тело функции «как на ладони», и сами комментарии вполне понятны.

Другой вариант — добавлять комментарии в конце строк кода:

```
            # Вводим переменные:
rs = 1         # для сохранения результатов
i = 0         # счетчик
while i < sum - 1:  # Запускаем цикл
```

Получилось очень удобно: и функция выглядит понятно, и комментарии привязаны к нужным строкам. Единственное, на что надо обратить внимание, — количество знаков в таких строках. PEP 8 рекомендует длину строки не более 79 символов. В самом крайнем случае — 99. Поэтому текст комментариев в строках кода должен быть коротким.

Строки документирования чаще всего используют для описания функций. Впрочем, существует еще один случай, кода строки документирования оказываются полезными — мы увидим это в разделе 4.1.

Между комментариями и строками документирования наблюдается принципиальная разница. Комментарий игнорируется интерпретатором, а для строки документирования создается объект. Это слегка замедляет программу, поэтому строки документирования желательно писать лаконичным языком.

3.6. Операторы

В Python для данных разных типов есть различные операторы.

Начнем с математических. Их перечень приведен в таблице 3.6.1.

Таблица 3.6.1. Математические операторы

Оператор	Действие	Пример	Примечание
+	Сложение	**x + y**	
–	Вычитание	**x – y**	
*	Умножение	**x * y**	
/	Деление	**x / y**	
//	Деление с округлением вниз	**x // y**	Результатом будет число, у которого отброшен остаток
%	Остаток от деления	**x % y**	Результатом будет остаток от деления
**	Возведение в степень	**x ** y**	

В работе с последовательностями применяют операторы, которые перечислены в таблице 3.6.2.

Таблица 3.6.2. Операторы для работы с последовательностями

Оператор	Действие	Пример	Примечание
+	Конкатенация	**'Строка1' + 'Строка2'**	Результатом будет новая строка
*	Повторение	**'Строка' * y**	Результатом будет новая строка, состоящая из **y** повторений исходной строки

Продолжение табл. 3.6.2

Оператор	Действие	Пример	Примечание
in	Проверка на вхождение	**'Стр' in 'Строка'**	**True**, в случае вхождения элемента в последовательность
not in	Проверка на невхождение	**'Стр' not in 'Строка'**	**True**, если элемент отсутствует в последовательности

Существует несколько способов присвоить значение переменной. Эти способы объяснены в таблице 3.6.3.

Таблица 3.6.3. Операторы присваивания

Оператор	Действие	Пример
=	Простое присваивание	**x = 1**
+=	Увеличение переменной на указанное значение	**x += 1**
-=	Уменьшение переменной на указанное значение	**x -= 1**
*=	Умножение переменной на указанную величину	**x *= 2**
/=	Деление переменной на указанную величину	**x /= 2**
//=	Деление переменной на указанную величину с округлением вниз	**x //= 2**
%=	Деление переменной на указанную величину и отбрасывание остатка	**x %= 2**
**=	Возведение переменной в степень, величина которой указана справа	**x **= 2**

Для логических выражений существуют операторы сравнения, которые приведены в таблице 3.6.4.

Таблица 3.6.4. Операторы сравнения

Оператор	Сравнение	Пример
==	Равно	**x == y**
!=	Не равно	**x != y**
>	Больше	**x > y**
<	Меньше	**x < y**
>=	Больше или равно	**x >= y**
<=	Меньше или равно	**x <= y**
and	Логическое **И**	**x > 1 and y < 10**
not	Логическое **НЕ**	**not x == y**
or	Логическое **ИЛИ**	**x > 1 or y < 10**

Продолжение табл. 3.6.4

Оператор	Сравнение	Пример
is	Проверяет, содержат ли две переменные ссылки на один и тот же объект	**x is y**
is not	Проверяет, содержат ли две переменные ссылки на разные объекты	**x is not y**

3.7. Числа

Мы с вами уже рассмотрели две функции — **int** и **float**, выполняющие преобразования данных в числа. Добавим в ряд полезных функций еще 7.

round — возвращает число, округленное до целого согласно математическим правилам: если дробная часть меньше 0.5, выполняется округление до ближайшего меньшего числа, если дробная часть равна или больше 0.5, округляется до ближайшего большего числа. Пример:

```
round(3.5)
```

Результат:

```
4
```

abs — возвращает абсолютное значение числа (иначе говоря, если число с отрицательным знаком, то вернется с положительным).

```
abs(-5)
```

```
5
```

pow — возводит число в степень, указанную вторым параметром.

```
pow(5, 3)
```

```
125
```

max — возвращает максимальное значение из набора чисел.

```
max(0, 1, 2, 3)
```

```
3
```

min — возвращает минимальное значение из набора чисел.

```
min(0, 1, 2, 3)
```

```
0
```

sum — возвращает сумму значений элементов последовательности.

```
sum([0, 1, 2, 3])
```

```
6
```

is_integer — проверяет, содержит ли число дробную часть. Возвращает **True**, если число целое.

```
(3.5).is_integer()
```

```
False
```

Теперь коснемся модулей, хотя предметный разговор о них состоится лишь в последнем разделе этой главы. Тем не менее имеет смысл забежать вперед и рассказать о том, какие полезные функции есть в модулях **math** и **random**.

Начнем с **math**. Многие его функции дублируют возможности, которые в Python реализуются с помощью различных операторов, рассмотренных нами в предыдущем разделе. Поэтому выберем лишь несколько функций, не имеющих аналогов.

pi — возвращает значение числа ***π***.

```
math.pi
```

```
3.141592653589793
```

e — возвращает значение константы ***e***.

```
math.e
```

```
2.718281828459045
```

sqrt — извлекает квадратный корень.

```
math.sqrt(49)
```

```
7.0
```

Floor — возвращает число, округленное до ближайшего меньшего.

```
math.floor(5.9)
```

```
5
```

factorial — вычисляет факториал — произведение всех натуральных чисел от 1 до указанного значения.

```
math.factorial(7)
```

```
5040
```

Теперь перейдем к модулю **random**.

random — возвращает случайное вещественное число от 0.0 до 1.0. Пример:

```
random.random()
```

```
0.45109179654967557
```

uniform — возвращает случайное вещественное число из заданного диапазона.

```
random.uniform(2, 7)
```

```
3.942466835198392
```

randint — возвращает случайное целое число из заданного диапазона.

```
random.randint(2, 7)
```

```
6
```

choice — возвращает случайный элемент из последовательности.

```
random.choice[1, 2, 3, 4, 5, 6, 7]
```

```
2
```

shuffle — случайным образом перемешивает элементы списка, не возвращая результата.

```
gr = [1, 2, 3, 4, 5, 6, 7]
random.shuffle(gr)
print(gr)
```

```
[1, 4, 6, 2, 3, 7, 5]
```

sample — возвращает список из указанного количества элементов, которые случайным образом выбраны из последовательности.

```
random.sample([1, 2, 3, 4, 5, 6, 7], 3)
```

```
[3, 6, 7]
```

3.8. Строки

Создать строку очень просто. Добавьте в программу переменную и присвойте ей некий текст (строковое значение):

```
s = 'Текст'
```

Кроме того, как вы помните, можно прибегнуть к помощи функции **str**, чтобы преобразовать число в строку.

А еще строку можно создать, ограничив ее тройными апострофами слева и справа. Об этом вы тоже уже знаете.

Если надо узнать количество символов в строке, то воспользуйтесь функцией **len**. Например:

```
s = 'Текст'
len(s)
```

Результат:

```
5
```

В строках можно использовать специальные символы. Наиболее важные для нас:

\n — перевод строки;

\' — апостроф;

\\ — обратный слеш.

Иногда надо вывести текст строго в том виде, как он написан (программисты часто называют такую строку «сырой»). В Python для подобных случаев предусмотрен модификатор **r**. Зачем он нужен? Допустим, вы хотите показать на странице фрагмент какого-то кода, в котором есть спецсимвол **\n**. Если не поставить перед строкой с таким символом модификатор, то браузер посчитает, что это команда на перевод строки, и символ не напечатает. Если поставить модификатор перед строкой, то символ **\n** будет отображен на странице. Запустите программу **3_8_1.py**:

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')
print('1 \n 2')
print('<br><br>')
print(r'1\n2')
```

и убедитесь в этом сами (рис. 3.8.1).

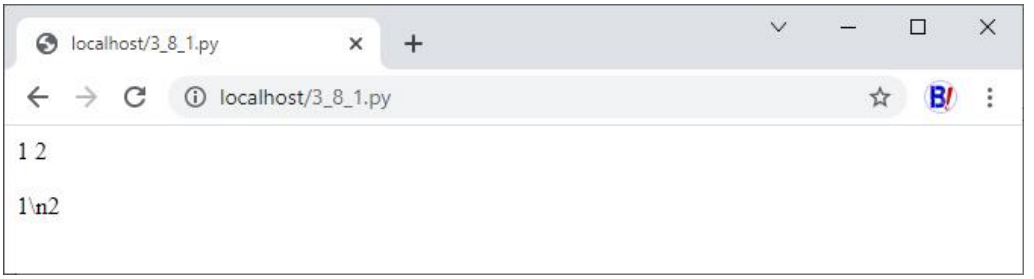

Рис. 3.8.1. Вывод обработанной и «сырой» строки

Всякая строка является последовательностью, а это значит, что любой ее символ можно получить по индексу вхождения. Пример:

```
s = 'Текст'
s[3]
```

Результат:

```
с
```

Обратите внимание: индексы отсчитываются, начиная с **0**. Поэтому самый большой индекс будет на **1** меньше количества символов в строке.

В примере выше отсчет ведется от начала строки. Если указать отрицательный индекс, то отсчет будет вестись с конца строки:

```
s = 'Текст'
s[-3]
```

```
к
```

Из строки можно получить не только отдельные символы, но и **срез**. У этой операции следующий формат записи:

```
s[start_index : end_index : step]
```

где **start_index** — индекс начала среза, **end_index** — индекс конца среза (в результат не входит), **step** — шаг, с которым получают номер следующего индекса. Если значение шага не указано, то по умолчанию оно равно **1**. Если не указан индекс начала, то его значение будет **0**, если не указан индекс конца, то срез будет выполнен до конца строки. Если вообще не указать никакие параметры, то вернется строка в исходном виде (что вряд ли имеет какое-то практическое значение).

Срез хорошо иллюстрирует следующая программа (файл **3_8_2.py**):

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')
```

```
s = 'Текст строка'
print(s[2 : 8 : 2])
print('<br><br>')
print(s[2 : 8])
print('<br><br>')
print(s[:])
```

Результаты ее работы показаны на рисунке 3.8.2.

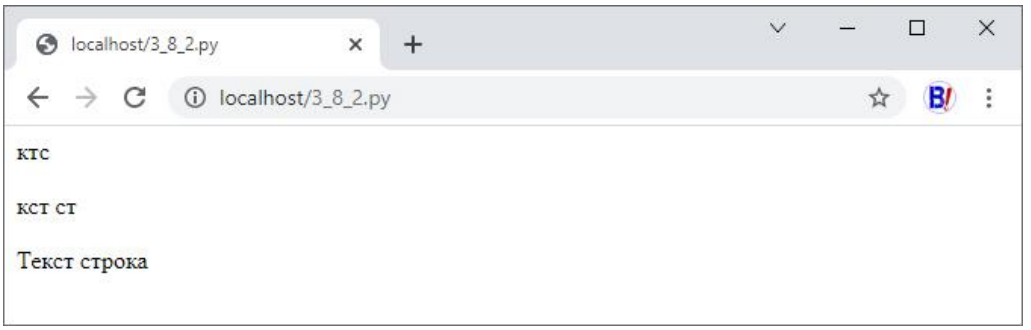

Рис. 3.8.2. Иллюстрация к разным срезам строки

Для работы со строками есть несколько полезных методов. Большинство из них чувствительны к регистру символов (за исключением **splitlines** и **join**).

strip — удаляет указанные символы в начале и в конце строки. Если символы не заданы, то удаляются пробелы и знаки перевода строки **n**. Пример:

```
s = 'Текст'
s.strip('т')
```

Результат:

```
Текс
```

Как видите, метод чувствителен к регистру символов. Так же, как и три следующих:

lstrip — удаляет указанные символы или пробелы в начале строки.

```
s = 'Текст'
s.lstrip('Т')
```

```
Текст
```

rstrip — удаляет указанные символы или пробелы и знаки перевода строки **\n** в конце строки.

```
s = 'Текст'
s.rstrip('ст')
```

```
Тек
```

split — разделяет строку по указанным символам и возвращает список из подстрок.

```
s = 'Текст'
s.split('к')
```

```
['Те', 'ст']
```

splitlines — разделяет строку по символам перевода строки.

```
s = 'Текст\nстрока'
s.splitlines()
```

```
['Текст', 'строка']
```

partition — разбивает строку по указанным символам и возвращает кортеж из трех элементов: до разделителя, разделитель, после разделителя.

```
s = 'Текст\nстрока'
s.partition('\n')
```

```
('Текст', '\n', 'строка')
```

join — преобразует последовательность в строку, элементы которой разделены указанными символами.

```
s = ['Текст', 'Строка', 'Абзац']
', '.join(s)
```

```
Текст, Строка, Абзац
```

Для поиска и замены в строке используйте методы, перечисленные далее. Обратите внимание: все они чувствительны к регистру символов.

find — ищет заданную подстроку в строке и возвращает номер позиции первого вхождения. Если подстрока не найдена, возвращается **–1**. Например:

```
s = 'Текст текст'
s.find('кс')
```

Результат:

```
2
```

rfind — ищет заданную подстроку в строке и возвращает номер позиции последнего вхождения. Если подстрока не найдена, возвращается **–1**.

```
s = 'Текст текст'
s.rfind('кс')
```

```
8
```

count — возвращает количество вхождений заданной подстроки в строку.

```
s = 'Текст текст'
s.count('кс')
```

```
2
```

startswith — проверяет, начинается ли строка с заданной подстроки.

```
s = 'Текст текст'
s.startswith('кс')
```

```
False
```

endswith — проверяет, заканчивается ли строка заданной подстрокой.

```
s = 'Текст текст'
s.endswith('ст')
```

```
True
```

replace — заменяет все вхождения указанной подстроки на заданную подстроку.

```
s = 'Текст текст'
s.replace('кс', 'н')
```

```
Тент тент
```

3.9. Условия

Условный оператор **if** необходим для проверки определенных данных на соответствие некоторым критериям.

Вариант наиболее простого использования условного оператора **if** выглядит так:

```
if условие:
  инструкции
```

Здесь инструкции будут выполняться в случае истинности условия. Пример:

```
a = 1
if a == 1:
  print('Yes')
```

Результат:

```
Yes
```

Если необходимо предусмотреть альтернативные инструкции на случай ложности условия, то используют следующую форму записи:

```
if условие:
  инструкции 1
else:
  инструкции 2
```

Инструкции блока **else** будут выполняться в случае ложности основного условия.

```
a = 1
if a == 2:
  print('Yes')
else:
  print('No')
```

```
No
```

Бывает надо проверить несколько условий. В этой ситуации используют такую конструкцию:

```
if условие 1:
  инструкции 1
elif условие 2:
  инструкции 2
...
elif условие n:
  инструкции n
else:
  инструкции на случай ложности всех условий
```

Простой пример:

```
a = 4
if a == 1:
  print('Один')
elif a == 2:
  print('Два')
elif a == 3:
  print('Три')
else:
  print('Сопоставление не найдено')
```

```
Сопоставление не найдено
```

Иногда требуется проверить несколько условий, но при этом не выполнять никаких действий, если все условия окажутся ложными. Представим, что у нас есть переменная **a**, которая может принимать разные числовые значения, но для

нас важны только три из них: **1**, **2** и **3**. В этом случае условные выражения можно записать так:

```
if a == 1:
  инструкции 1
if a == 2:
  инструкции 2
if a == 3:
  инструкции 3
```

Вот пример:

```
a = 4
if a == 1:
  print('Один')
if a == 2:
  print('Два')
if a == 3:
  print('Три')
```

В результате выполнения этого блока проверок ничего напечатано не будет.

Python позволяет использовать и «усеченные» блоки условий:

```
if a == 1:
  инструкции 1
elif a == 2:
  инструкции 2
elif a == 3:
  инструкции 3
```

Здесь оператор **else** отсутствует. Пример:

```
a = 2
if a == 1:
  print('Один')
elif a == 2:
  print('Два')
elif a == 3:
  print('Три')
```

```
Два
```

И еще одно интересное замечание. В качестве проверяемых выражений в условный оператор **if** можно передавать значения, возвращаемые из функций. Такой пример показан ниже (файл **3_9_1.py**):

```
#! C:/Python/python
```

```
print('Content-type: text/html; charset=utf-8\n\n')

def func(pr):
  return pr*2

a = 4
```

```
b = 2

if a == func(b):
  print('Равно')
```

Здесь значение переменной **a** сравнивается со значением, возвращаемым функцией **func**:

```
if a == func(b):
```

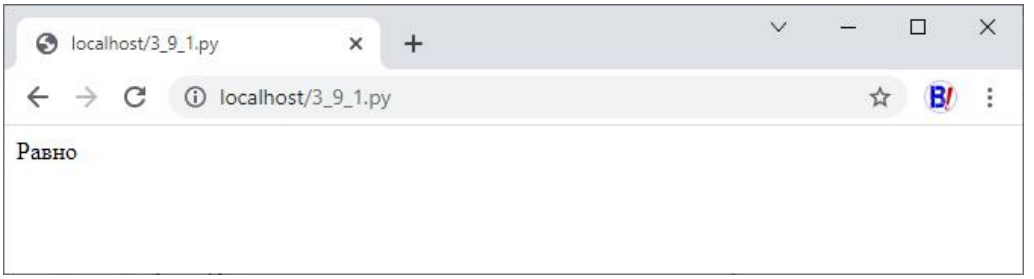

Рис. 3.9.1. Результат сравнения значения переменной со значением, возвращаемым функцией

Если запустить в браузере указанную страницу, вы увидите, что сравнение было успешным и появилась надпись «Равно» (рис. 3.9.1).

3.10. Циклы

Операторы циклов необходимы для перебора некоторых значений в заданных пределах и поочередного выполнения определенных инструкций, иногда зависящих от значения на очередном проходе, а иногда нет. Циклы удобно использовать в тех случаях, когда какую-то часть программного кода надо выполнить многократно.

Начнем с цикла **for**. Его структура обычно выглядит так:

```
for переменная in набор значений:
  инструкции
```

Здесь **переменная** — параметр, через который доступно значение последовательности или ключ словаря на каждом проходе. **Набор значений** — строка, список, кортеж, диапазон или словарь.

Посмотрим, например, как с помощью цикла **for** можно найти квадраты чисел из списка (файл **3_10_1.py**):

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')

t = [0, 3, 4, 7, 8]
for i in t:
```

```
  print(i**2)
  print('<br>')
```

Результат показан на рисунке 3.10.1.

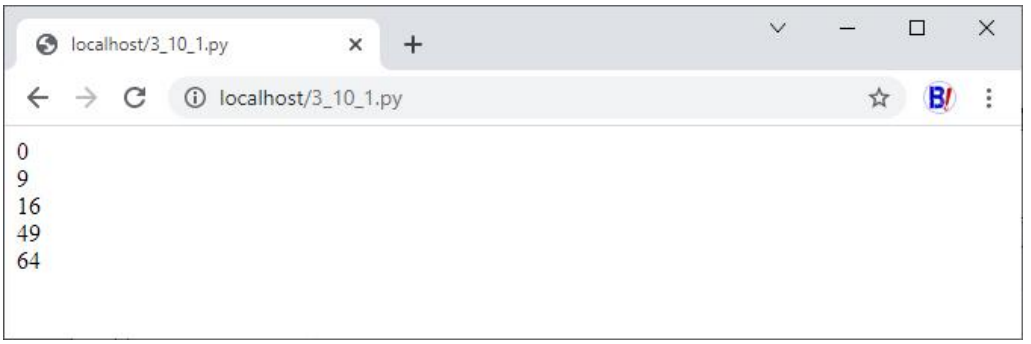

Рис. 3.10.1. Выводим квадраты заданных чисел в цикле for

В цикле **for** можно перебрать значения словаря по их ключам. Вот пример (файл **3_10_2.py**):

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')
s = {'o': 'один', 'd': 'два', 't': 'три'}
for i in s:
  print(i, ' - ', s[i])
  print('<br>')
```

Что выведет такая программа, можно видеть на рисунке 3.10.2.

Рис. 3.10.2. Выводим ключи и значения из словаря в цикле for

Еще один полезный оператор цикла — **while**. Он устроен несколько проще:

```
while условие:
  инструкции
```

Пример записи цикла в реальной программе (файл **3_10_3.py**):

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')

i = 0

while i <= 5:
  print(i*2)
  print('<br>')
  i += 1
```

В этом цикле мы последовательно умножаем значение из каждого прохода на **2**. Что из этого получилось, видно на рисунке 3.10.3.

В предыдущем разделе мы узнали, что в условия можно передавать значения из функций. Такой же «фокус» возможен и с циклом **while** (файл **3_10_4.py**):

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')

def func(pr):
  if pr < 6:
    pr += 1
    return pr

i = 0

while i < func(i):
  print(i)
  print('<br>')
  i += 1
```

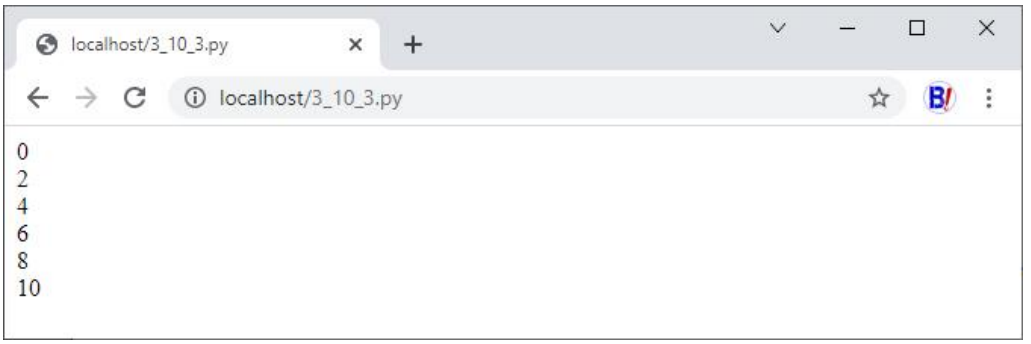

Рис. 3.10.3. Выводим результаты вычислений в цикле while

Функция **func** получает в качестве параметра значение переменной **i**, которое увеличивается при каждом проходе цикла на **1**:

```
while i < func(i):
  ...
  i += 1
```

В функции задано ограничение — она работает, пока переданное значение меньше **6**:

```
def func(pr):
  if pr < 6:
```

При этом возвращаемое функцией значение постоянно оказывается больше значения переменной **i**:

```
pr += 1
return pr
```

Останавливается цикл лишь после того, как условие

```
if pr < 6:
```

в функции становится ложным.

Таким образом, управление циклом на самом деле производится из внешней функции.

Что мы получим, используя подобный подход, видно на рисунке 3.10.4.

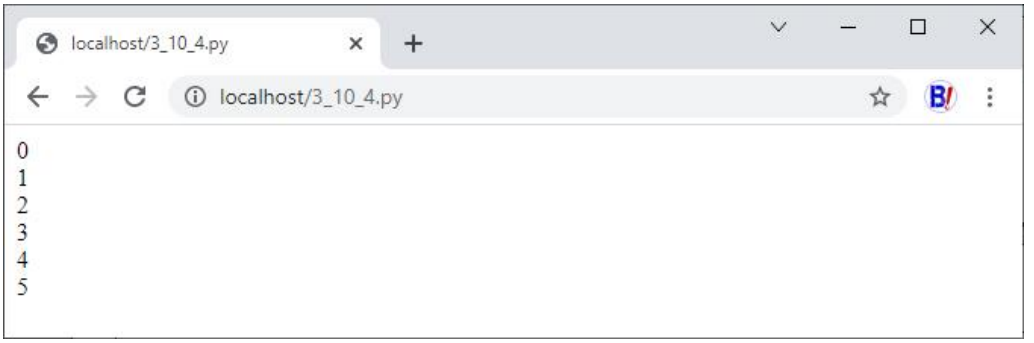

Рис. 3.10.4. Выводим числа в цикле while, условие для которого получаем из функции

Для циклов предусмотрено два полезных оператора.

Первый из них — **continue** — служит для перехода на очередную итерацию цикла, минуя определенный участок. Пример:

```
for i in [1, 2, 3, 4, 5]:
  if i == 4:
    continue
  print(i)
```

На печать будут выведены все цифры, исключая **4**:

```
1 2 3 5
```

Второй полезный оператор — **break**. Он необходим для полной остановки цикла в случае выполнения определенных условий. Следующая программа

```
i = 1
while i < 6:
  if i == 4:
    break
  print(i)
  i += 1
```

напечатает только цифры

```
1 2 3
```

после чего цикл прервется.

3.11. Регулярные выражения

Регулярные выражения в Python применяются в виде откомпилированных шаблонов:

```
переменная = re.compile(регулярное_выражение, модификатор)
```

Чтобы задействовать модификаторы и откомпилировать выражение, требуется подключение модуля **re**.

К сказанному выше добавим, что модификаторов может быть несколько. В этом случае они записываются через оператор |. Нам интересны два из них:

I — поиск без учета регистра;

M — поиск в объекте, состоящем из нескольких подстрок, разделенных символом **\n**.

Покажем, как работают регулярные выражения на примере программы **3_11_1.py**:

```
#! C:/Python/python
```

```
import re

print('Content-type: text/html; charset=utf-8\n\n')

it = re.compile('[a-z]+')
ar = ['Python', '12345']

for i in [0, 1]:
  res = it.search(ar[i])
  if res:
    print(ar[i], ' - это текст<br>')
  else:
    print(ar[i], ' - это не текст<br>')
```

У нас есть откомпилированный шаблон

```
it = re.compile('[a-z]+')
```

и список из двух элементов

```
ar = ['Python', '12345']
```

В первом проходе цикла

```
for i in [0, 1]:
```

мы проверяем значение из списка **ar** с нулевым индексом — 'Python'. Сравнивая эту строку с регулярным выражением (с помощью метода **search**, о котором будет сказано позже)

```
res = it.search(ar[i])
```

программа обнаруживает, что поиск букв был успешен, и выводит об этом сообщение:

```
if res:
  print(ar[i], ' - это текст<br>')
```

На втором проходе совпадения с шаблоном не произойдет и будет выдан отрицательный ответ:

```
else:
  print(ar[i], ' - это не текст<br>')
```

После запуска в браузере программа выдаст результаты, показанные на рисунке 3.11.1.

Рис. 3.11.1. Результат проверки строк с помощью регулярного выражения

Обратите внимание: шаблоны бывают гораздо сложнее, чем мы видели в примере. В них часто и во множестве употребляются слеши. Чтобы не экранировать каждый слеш, разработчики ставят перед шаблоном модификатор **r**, уже знакомый нам по разделу, посвященному строкам.

Если выполняется проверка строки, которая полностью должна совпадать с регулярным выражением, то используется привязка регулярного выражения к началу и концу строки. Делается это с помощью таких метасимволов:

^ — привязка к началу строки;

$ — привязка к концу строки.

Есть еще один метасимвол, полезный для первых экспериментов с регулярными выражениями:

\b — привязка к началу слова (началом слова считается пробел или любой символ, кроме буквы, цифры и знака подчеркивания).

В регулярных выражениях удобно использовать диапазоны символов. Они собраны в таблицу 3.11.1.

Таблица 3.11.1. Диапазоны символов

Диапазон	Комментарий
[0–9]	Любая цифра
[a–z]	Любая латинская буква в нижнем регистре
[A–Z]	Любая латинская буква в верхнем регистре
[A–Za–z]	Любая латинская буква независимо от регистра
[а–яё]	Любая кириллическая буква в нижнем регистре
[А–ЯЁ]	Любая кириллическая буква в верхнем регистре
[А–Яа–яЁё]	Любая кириллическая буква независимо от регистра

Чтобы искать любые символы, кроме указанных в диапазоне, необходимо в квадратные скобки вписать знак ^ перед описанием диапазона. Например, так:

```
[^0-9]
```

В этом конкретном случае будет искаться любой символ, кроме цифры.

Диапазоны или перечисления символов можно заменить специальными классами (табл. 3.11.2).

Таблица 3.11.2. Классы символов

Класс	Комментарий
\d	Любая цифра
\D	Не цифра

Продолжение табл. 3.11.2

Класс	Комментарий
\w	Любая латинская буква, цифра или символ подчеркивания
\W	Не латинская буква, цифра или символ подчеркивания
\s	Пробельный символ
\S	Не пробельный символ

Если необходимо искать в строке точку, то это можно сделать двумя способами:

– заключив ее в квадратные скобки (например, вместе с другими символами) – **[.]**;

– поставив перед ней слеш — **\.**

Если необходимо искать в строке дефис, то отменить его специальное значение можно также двумя способами:

– заключив его в квадратные скобки (например, вместе с другими символами) и расположив последним в перечислении — **[-]**;

– поставив перед ним слеш — \-.

Квантификаторы задают количество вхождений символа или группы символов в строку. Они перечислены в таблице 3.11.3.

Таблица 3.11.3. Квантификаторы

Квантификатор	Комментарий
{n}	**n** вхождений
{n, }	**n** или более вхождений
{n, m}	Не менее **n** и не более **m** вхождений
+	Одно или более вхождений
?	Ни одного или одно вхождение
*****	Ноль или более вхождений

Для поиска совпадений с шаблоном регулярного выражения используются следующие методы.

match — выполняет проверку совпадения регулярного выражения с началом строки.
search — проверяет совпадение регулярного выражения с любой частью строки.
fullmatch — проверяет, совпадает ли строка целиком с регулярным выражением.
findall — выполняет поиск всех совпадений с шаблоном. Возвращает список с фрагментами совпадений.

Для замены в строке всех совпадений с шаблоном регулярного выражения используется метод **sub**. Его формат:

```
шаблон.sub(заменяющая_строка, заменяемая_строка)
```

Пример:

```
it = re.compile('\d')
it.sub('A', '123456789')
```

Результат:

```
AAAAAAAAA
```

3.12. Списки

Список — это изменяемый набор объектов. Позиция элемента в списке называется индексом.

Создать список можно следующими способами:

– заключив все элементы списка в квадратные скобки. Например:

```
a = [0, 1, 2, 3, 4, 5]
```

– преобразовав последовательность в список посредством функции **list**:

```
a = list('Python')
```

Результат:

```
['P', 'y', 't', 'h', 'o', 'n']
```

Есть еще один способ, о котором речь пойдет несколько позже.

Чтобы получить определенный элемент списка, надо рядом с его именем указать индекс, заключенный в квадратные скобки. Пример:

```
a = ['P', 'y', 't', 'h', 'o', 'n']
a[2]
```

Результат:

```
t
```

Если указать отрицательный индекс, то отсчет элементов будет вестись от конца списка:

```
a = ['P', 'y', 't', 'h', 'o', 'n']
a[-3]
```

```
h
```

Для изменения элемента списка необходимо присвоить ему новое значение. Пример:

```
a = [0, 1, 2, 3, 4, 5]
a[2] = 'два'
```

Результат:

```
[0, 1, 'два', 3, 4, 5]
```

Узнать количество элементов в списке можно с помощью функции **len**:

```
a = [0, 1, 2, 3, 4, 5]
len(a)
```

```
6
```

Списки могут состоять не только из отдельных элементов, но также из других последовательностей, например тех же списков. Вот пример многомерного списка:

```
a = [[0, 1, 2], [3, 5, 7], [4, 6, 8], [10, 9]]
```

А это список смешанного типа:

```
a = ['a', 5, [0, 1, 2], 'n', [4, 6, 8]]
```

Чтобы получить элемент из многомерного списка, необходимо указать индекс внутреннего списка и индекс элемента во внутреннем списке. Это одиночный элемент из смешанного списка:

```
a = ['a', 5, [0, 1, 2], 'n', [4, 6, 8]]
a[1]
```

```
5
```

А это элемент из списка, вложенного в смешанный:

```
a = ['a', 5, [0, 1, 2], 'n', [4, 6, 8]]
a[2][1]
```

```
1
```

В некоторых программах списки создаются «на лету» в зависимости от различных факторов. Для создания таких списков удобно пользоваться методом **append**:

```
a = []
a.append('Py')
a.append('thon')
a.append('3')
```

Получим

```
['Py', 'thon', '3']
```

Для удаления и добавления элементов в списках есть несколько методов. Метод **append** мы уже проиллюстрировали. Посмотрим остальные.

Операция конкатенации:

```
a = [0, 1, 2]
b = [9, 8, 7]
c = a + b
```

```
[0, 1, 2, 9, 8, 7]
```

extend — добавляет элемент в конец списка:

```
a = [0, 1, 2]
a.extend('Python')
```

```
[0, 1, 2, 'P', 'y', 't', 'h', 'o', 'n']
```

insert — добавляет элемент в указанную позицию. Остальные элементы смещаются вправо. Формат метода:

```
a.insert(индекс, элемент)
```

Пример:

```
a = [0, 1, 2]
a.insert(1, 'Python')
```

```
[0, 'Python', 1, 2]
```

pop — удаляет элемент с указанным индексом. Остальные элементы смещаются влево. Если индекс не указан, удаляется последний элемент. Пример удаления элемента по индексу:

```
a = [0, 'Python', 1, 2]
a.pop(1)
```

```
[0, 1, 2]
```

Пример удаления последнего элемента:

```
a = [0, 'Python', 1, 2]
a.pop()
```

```
[0, 'Python', 1]
```

remove — удаляет первый элемент с указанным значением.

```
a = [0, 1, 0, 2, 1]
a.remove(1)
```

```
[0, 0, 2, 1]
```

clear — удаляет все элементы из списка.

```
a = [0, 1, 0, 2, 1]
a.clear()
```

```
[]
```

Необходимо сказать еще о двух полезных методах.

Reverse — «переворачивает» список на «180 градусов», то есть меняет последовательность элементов на обратную.

```
a = ['P', 'y', 't', 'h', 'o', 'n']
a.reverse()
```

```
['n', 'o', 'h', 't', 'y', 'P']
```

sort — сортирует список. Если надо выполнить сортировку по возрастанию, то форма записи будет следующая:

```
a = [3, 1, 9, 5, 7, 2]
a.sort()
```

Результат:

```
[1, 2, 3, 5, 7, 9]
```

Если необходимо отсортировать список по убыванию, применяют такую запись:

```
a = [3, 1, 9, 5, 7, 2]
a.sort(reverse = True)
```

```
[9, 7, 5, 3, 2, 1]
```

Также напомню, что в разделе 3.7 мы познакомились с двумя интересными функциями из модуля **random**, которые можно использовать для работы с последовательностями: **choice** и **sample**. А в разделе 3.8 — с методом **join**, с помощью которого легко преобразовать последовательность в строку.

3.13. Кортежи

Кортеж — это тоже список, только неизменяемый. Иначе говоря, кортеж доступен исключительно для чтения.

Создать кортеж можно двумя способами.

1. Перечислив все элементы через запятую внутри круглых скобок:

```
a = (3, 1, 9, 5, 7, 2)
```

2. Преобразовав последовательность с помощью функции **tuple**:

```
a = [3, 1, 9, 5, 7, 2]
tuple(a)
```

Результат:

```
(3, 1, 9, 5, 7, 2)
```

Если у вас в кортеже всего один элемент, после него обязательно должна стоять запятая:

```
a = (3,)
```

Получить значение элемента из кортежа можно по его индексу:

```
a = (3, 1, 9, 5, 7, 2)
a[2]
```

```
9
```

Для работы с кортежами можно использовать уже знакомые нам функции **count** и **len**.

Пример использования функции **count**:

```
a = (3, 1, 9, 5, 7, 2, 9)
a.count(9)
```

```
2
```

Пример использования функции **len**:

```
a = [0, 1, 2, 3, 4, 5]
len(a)
```

```
6
```

Наконец, последний пример — конкатенация кортежей:

```
a = (3, 1, 9, 5, 7, 2)
a += (8, 0, 4, 6)
print(a)
```

```
(3, 1, 9, 5, 7, 2, 8, 0, 4, 6)
```

3.14. Множества

Множество — это последовательность уникальных элементов, в которой нет дубликатов значений.

Создается множество функцией **set**:

```
st = set(['Python', 'версия', '3'])
st
```

Результат:

```
{'3', 'Python', 'версия'}
```

Если в списке, из которого создается множество, есть дубликаты значений, то они будут отброшены, что хорошо иллюстрирует программа **3_14_1.py**:

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')

st = set(['Python', 'версия', '3', 'Python', '3'])
print(st)
```

Результат:

```
{'версия', 'Python', '3'}
```

Получившееся множество можно видеть на рисунке 3.14.1.

Рис. 3.14.1. Множество из трех элементов, образованное из пятиэлементного списка

Узнать количество элементов во множестве можно с помощью функции **len**:

```
st = set(['Python', 'версия', '3'])
len(st)
3
```

Для работы со множествами можно использовать операторы и аналогичные им методы.

a | b или **a.union(b)** — объединение двух или более множеств:

```
a = set([0, 1, 2])
b = set([3, 4, 5])
c = set([6, 7, 8, 9])
a | b | c
```

Результат:

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Другой случай:

```
a = set([0, 1, 2])
b = set([3, 4, 5])
a.union(b)
```

Результат:

```
{0, 1, 2, 3, 4, 5}
```

a |= b или **a.update(b)** — добавление элементов множества **b** во множество **a**:

```
a = set([0, 1, 2])
b = set([3, 4, 5])
a |= b
```

или

```
a = set([0, 1, 2])
b = set([3, 4, 5])
a.update(b)
```

Результат:

```
{0, 1, 2, 3, 4, 5}
```

a & b или **intersection** — получение элементов, входящих в оба множества.

```
a = set([0, 1, 2, 3, 4])
b = set([3, 4, 5, 6, 7])
a & b
```

или

```
a = set([0, 1, 2, 3, 4])
b = set([3, 4, 5, 6, 7])
a.intersection(b)
```

Результат:

```
{3, 4}
```

Для сравнения множеств можно применять следующие операторы.

== — проверяет идентичность двух множеств:

```
a = set([0, 1, 2, 3, 4])
b = set([3, 4, 5, 6, 7])
a == b
```

```
False
```

!= — проверяет, различаются ли два множества:

```
a = set([0, 1, 2, 3, 4])
b = set([3, 4, 5, 6, 7])
a != b
```

```
True
```

a > **b** — проверяет, входят ли элементы множества **b** во множество **a**:

```
a = set([0, 1, 2, 3, 4])
b = set([3, 4, 5, 6, 7])
a > b
```

```
False
```

a < **b** — проверяет, входят ли элементы множества **a** во множество **b**:

```
a = set([3, 4])
b = set([3, 4, 5, 6, 7])
a < b
```

```
True
```

Для сравнения множеств можно также использовать метод **isdisjoint**, который проверяет, являются ли два множества совершенно разными:

```
a = set([0, 1, 2])
b = set([3, 4, 5, 6, 7])
a.isdisjoint(b)
```

```
True
```

Чтобы добавить элемент во множество, воспользуйтесь методом **add**:

```
st = set(['Python', 'версия', '3'])
st.add('!')
```

```
{'3', 'Python', 'версия', '!'}
```

Чтобы удалить элемент из множества, примените метод **remove**:

```
st = set(['Python', 'версия', '3'])
st.remove('версия')

{'Python', '3'}
```

Чтобы удалить все элементы из множества, используйте метод **clear**:

```
st = set(['Python', 'версия', '3'])
st.clear()

set()
```

Чтобы получить тот или иной элемент, преобразуйте множество в список. Пример такой программы (файл **3_14_2.py**):

```
#! C:/Python/python

print('Content-type: text/html; charset=utf-8\n\n')

st = set(['Python', 'версия', '3'])
a = list(st)

print(a)
print('<br><br>')
print(a[1])
```

С помощью функции **list** мы сохранили элементы множества в список, а также смогли получить отдельный элемент с индексом 1 (рис. 3.14.2). При этом учтите, что при следующем запуске программы у элемента с данным индексом может оказаться иное значение, так как множество — это неупорядоченная последовательность.

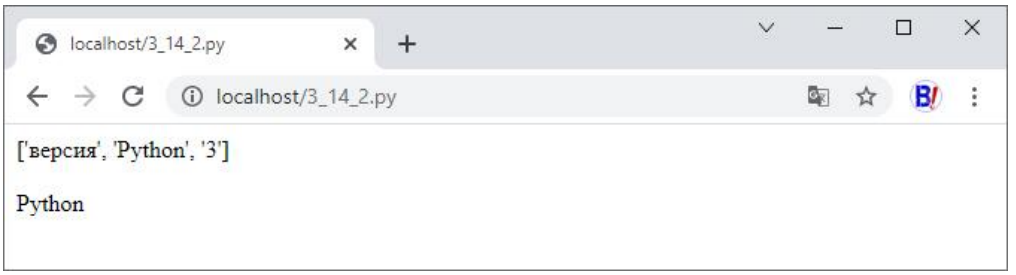

Рис. 3.14.2. Преобразование множества в список

3.15. Диапазоны

Диапазон — это последовательность целых чисел с заданными начальным и конечным значениями. Разность между соседними числами называется шагом. Диапазоны относятся к неизменяемым типам данных.

Чтобы создать диапазон, необходимо применить функцию **range**. Ее формат:

```
range(начальное_значение, конечное_значение, шаг)
```

Если шаг не указан, то он автоматически выбирается равным **1**. Обратите внимание: конечное значение не входит в диапазон!

Наиболее частая сфера применения диапазонов — это циклы **for**. Напишем простой пример (файл **3_15_1.py**):

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')
for i in range(1, 8, 2):
  print(i**3)
  print('<br>')
```

Эта программа вычисляет и печатает кубы нечетных чисел от 1 до 7, получая данные для расчетов из диапазона. Результат ее применения показан на рисунке 3.15.1.

Диапазоны можно сравнивать между собой с помощью операторов == и **!=**.

Рис. 3.15.1. Результат применения диапазона в цикле

3.16. Словари

Словарь — это набор элементов, доступ к которым выполняется по ключу. Как мы уже упоминали, словари относятся к отображениям.

Для создания словаря можно использовать несколько способов.

Самый простой — указав в фигурных скобках пары ключ-значение:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
```

Можно создать словарь, используя функцию dict:

```
di = dict(a = 'Python', b = 'версия', c = '3')
```

Словарь можно заполнить, поочередно создавая пары ключ-значение. Пример (файл **3_16_1.py**):

```
#! C:/Python/python
print('Content-type: text/html; charset=utf-8\n\n')

di = {}
di['a'] = 'Python'
di['b'] = 'версия'
di['c'] = '3'

print(di)
```

Получившийся словарь показан на рисунке 3.16.1.

Рис. 3.16.1. Словарь, образованный методом поочередного заполнения

Получить то или иное значение из словаря можно по ключу:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
di['b']
```

Результат:

```
версия
```

Для работы со словарями существует много различных приемов. Мы рассмотрим лишь несколько из них. Для начала вспомним про операторы **in** и **not in**.

In — с его помощью проверяют существование ключа в словаре:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
'b' in di
```

```
True
```

not in — с его помощью проверяют отсутствие ключа в словаре:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
'g' not in di
```

```
True
```

А теперь посмотрим несколько методов.

items — возвращает объект в виде списка, состоящего из кортежей, в которые входят ключи и значения:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
di.items()
```

```
dict_items([('a', 'Python'), ('b', 'версия'),
                    ('c', '3')])
```

Чтобы получить список из объекта **dict_items**, воспользуйтесь функцией **list**:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
ndi = di.items()
list(ndi)
```

```
[('a', 'Python'), ('b', 'версия'), ('c', '3')]
```

pop — удаляет элемент с указанным ключом:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
di.pop('b')
```

```
{'a': 'Python', 'c': '3'}
```

update — добавляет элемент в словарь:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
di.update(d = '!')
```

```
{'a': 'Python', 'b': 'версия', 'c': '3', 'd': '!'}
```

clear — удаляет все пары ключ-значение из словаря:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
di.clear()
```

```
{}
```

Узнать количество пар ключ-значение в словаре можно с помощью функции **len**:

```
di = {'a': 'Python', 'b': 'версия', 'c': '3'}
len(di)
3
```

3.17. Дата и время

Web-приложению бывает важно получить текущую дату и время. Например, чтобы зафиксировать ваш визит на сайт или при оформлении Интернет-заказа. В этом программисту Python поможет модуль **time**. Мы рассмотрим лишь одну его функцию — **localtime**, которая позволяет получить дату и время на сервере в момент обращения к программе.

Результатом работы функции **localtime** будет объект **struct_time**. Если его вывести на печать, то он будет выглядеть примерно следующим образом:

```
time.struct_time(tm_year=2021, tm_mon=10, tm_mday=27,
         tm_hour=18, tm_min=11, tm_sec=22,
         tm_wday=2, tm_yday=300, tm_isdst=0)
```

Из данных объекта легко получить текущее время. Напишем для этого специальную программу (файл **3_17_1.py)**:

```
#! C:/Python/python
import time

dw = ['понедельник', 'вторник', 'среда', 'четверг',
   'пятница', 'суббота', 'воскресенье']

tm = time.localtime()

print('Content-type: text/html; charset=utf-8\n\n')

print(tm)
print('<br><br>Сегодня: ')
print(dw[tm[6]], str(tm[2]), str(tm[1]),
                 str(tm[0]), 'г.')
print('<br>Время: ')
print(str(tm[3]), ':', str(tm[4]), ':', str(tm[5]))
```

Здесь мы сначала для наглядности напечатали данные объекта **struct_time**, а потом сформировали на их основе две строки: с датой и временем запуска программы (рис. 3.17.1).

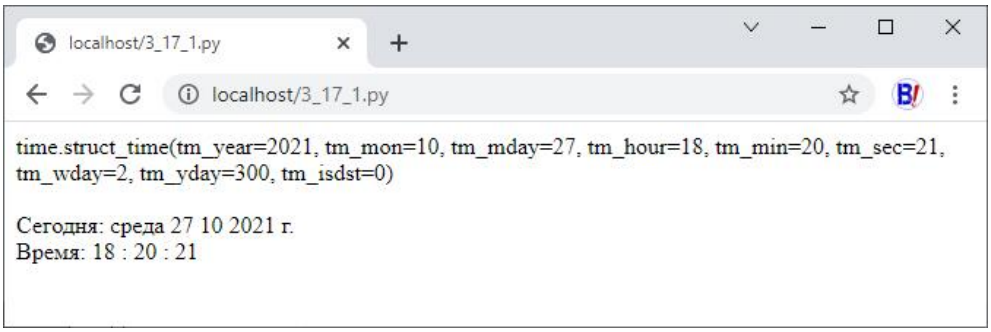

Рис. 3.17.1. Выводим на страницу браузера дату и время

3.18. Файлы

Программисту часто в своей работе приходится иметь дело с файлами. Мы с вами рассмотрим, как можно прочитать, дозаписать и перезаписать текстовый файл. А также попробуем переименовать, скопировать, переместить и удалить файл.

Начнем с того, что для открытия файлов применяется функция **open** следующего формата:

```
open('адрес_файла', 'режим', encoding='utf-8')
```

Параметр **encoding** указывает кодировку, в которой нужно представить данные из файла. Параметр **режим** можно не указывать, если ваша цель — только прочитать содержимое файла.

Существует несколько режимов открытия файла. Они перечислены в таблице 3.18.1.

Таблица 3.18.1. Режимы открытия файлов

Режим	Комментарий
r	Чтение файла
r+	Чтение и запись
w	Запись файла с первоначальным удалением из него всего содержимого
w+	Чтение и запись с первоначальным удалением из файла всего содержимого
a	Запись в конец файла
a+	Чтение и запись в конец файла
x	Создание файла для записи

Если необходимо открыть бинарный файл, то после указания режима надо добавить модификатор **b**:

```
open('img.jpg', 'wb')
```

После чтения или записи файл необходимо закрыть с помощью метода **close**:

```
fo = open('text.txt', 'режим', encoding='utf-8')
чтение или запись
fo.close()
```

Прочитать текстовый файл можно несколькими способами. Нам интересны два из них.

1. Сразу весть файл с помощью метода **read** (файл **3_18_1.py**):

```
#! C:/Python/python
fo = open('test/text.txt', encoding='utf-8')
tx = fo.read()
fo.close()

print('Content-type: text/html; charset=utf-8\n\n')

print(tx)
```

В результате текст из файла окажется в окне браузера (рис. 3.18.1).

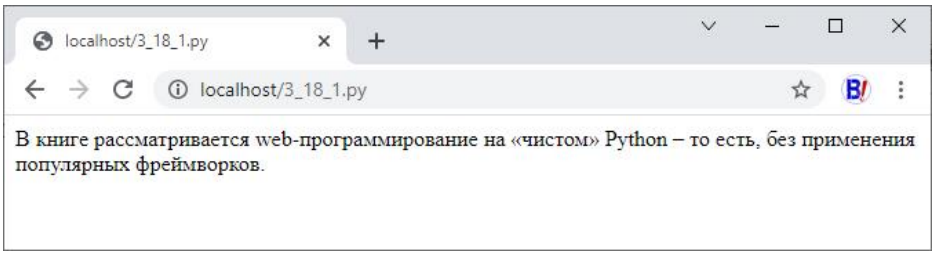

Рис. 3.18.1. Чтение файла

1. Построчно с помощью метода **readlines**:

```
fo = open('test/text.txt', encoding='utf-8')
tx = fo.readlines()
fo.close()
```

Рассмотрим также два метода для записи в файл.

1. **write** — запись строки в файл (программа **3_18_2.py**):

```
#! C:/Python/python
```

```
fo = open('test/text.txt', 'a', encoding='utf-8')
fo.write('\nНовый текст')
fo.close()

fo = open('test/text.txt', encoding='utf-8')
tx = fo.read()
fo.close()

print('Content-type: text/html; charset=utf-8\n\n')

print(tx)
```

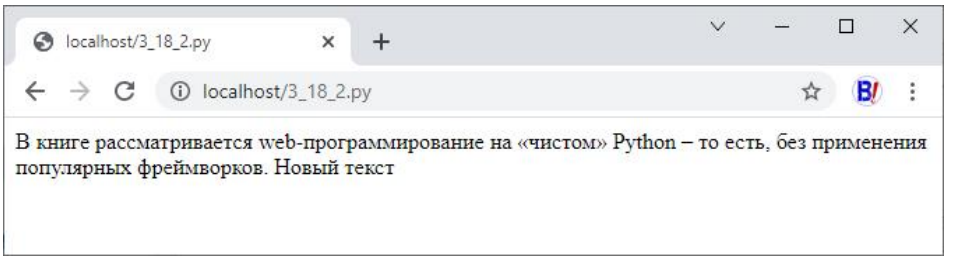

Рис. 3.18.2. Запись в конец файла

Результат можно посмотреть на рисунке 3.18.2.

В предыдущем примере мы выполняли запись в конец файла. Теперь полностью перезапишем этот же файл (программа **3_18_3.py**):

```
#! C:/Python/python
```

```
fo = open('test/text.txt', 'w', encoding='utf-8')
fo.write('\nНовый текст')
fo.close()

fo = open('test/text.txt', encoding='utf-8')
tx = fo.read()
fo.close()

print('Content-type: text/html; charset=utf-8\n\n')

print(tx)
```

Результат можно посмотреть на рисунке 3.18.3.

Рис. 3.18.3. Перезапись файла

2. **writelines** — запись последовательности (файл **3_18_4.py**):

```
#! C:/Python/python
```

```
fo = open('test/text.txt', 'w', encoding='utf-8')
fo.writelines(['0', 'Python', '3', 'Текст'])
fo.close()

fo = open('test/text.txt', encoding='utf-8')
tx = fo.read()
fo.close()

print('Content-type: text/html; charset=utf-8\n\n')

print(tx)
```

Результат на рисунке 3.18.4.

Рис. 3.18.4. Перезапись файла с помощью writelines

Теперь рассмотрим несколько функций для манипулирования файлами.

Для начала возьмем две полезные функции из модуля **os**.

rename — переименование файла:

```
os.rename('test/text.txt', 'test/1.txt')
```

remove — удаление файла:

```
os.remove('test/1.txt')
```

Теперь на очереди две функции из модуля **shutil**.

copy — копирование файла:

```
shutil.copy('test/test.txt', 'exper/new.txt')
```

move — перемещение файла с удалением исходного:

shutil.move('exper/new.txt', 'test')

3.19. Кодировка символов

Этот раздел касается всего одного пункта — указания кодировки при открытии файла.

Например, в PHP или Perl для корректного чтения и записи в текстовый файл достаточно того, что сама программа и текстовый файл были созданы в кодировке **utf-8**. В Python дело обстоит иначе.

Если вы помните, устанавливая интерпретатор, мы открывали файл **httpd.conf** из папки сервера **C:\Apache24\conf** и добавляли в самый конец строку

```
SetEnv PYTHONIOENCODING utf8
```

Тем самым мы указывали серверу кодировку, с которой предстояло работать программам на Python.

Более того, и программы, и текстовые файлы мы сохраняли в **utf-8**.

И это еще не все. В строке с указанием типа выводимого документа мы тоже установили кодировку: **charset=utf-8**.

Однако всех этих усилий оказалось недостаточно, чтобы правильно считать символы кириллицы. Нам пришлось в список параметров функции **open** добавить **encoding='utf-8'**.

Попробуйте запустить программу **3_19_1.py**, в которой функция **open** «избавлена» от этого параметра:

```
#! C:/Python/python

fo = open('content/text.txt')
tx = fo.read()
fo.close()

print('Content-type: text/html; charset=utf-8\n\n')

print(tx)
```

Вы увидите картину, показанную на рисунке 3.19.1.

Рис. 3.19.1. Попытка открыть текстовый файл без указания его кодировки

В сети есть много сообщений и статей о проблемах с кодировкой кириллицы в файлах Python. Для web-программирования на Python рецепт преодоления этих трудностей следующий:

– добавить в конфигурационный файл сервера Apache строку

```
SetEnv PYTHONIOENCODING utf8
```

– сохранить все файлы в формате **utf-8**;

– в файлах Python указать кодировку либо в заголовке:

```
print('Content-type: text/html; charset=utf-8\n\n')
```

либо в HTML-коде страницы:

```
print('Content-type: text/html\n\n')
print('<!DOCTYPE html>')
...
print('<meta charset="utf-8">')
...
```

– обязательно указать кодировку в качестве параметра при открытии файла:

```
open(adr, encoding='utf-8')
```

3.20. Каталоги

Для работы с каталогами (папками) необходимо импортировать в программу модуль **os** или **shutil** (или оба — в зависимости от того, какие действия требуется выполнить). Как это сделать, вы узнаете в последнем разделе этой главы.

Модуль **os** предоставляет следующие полезные для наших задач функции.

Listdir — возвращает список вложенных каталогов и файлов. Пример чтения содержимого каталога (**файл 3_20_1.py**):

```
#! C:/Python/python
import os
opdir = os.listdir('content')
print('Content-type: text/html; charset=utf-8\n\n')
print(opdir)
```

Запустите его в браузере — и вы получите список из двух файлов, содержащихся в папке content (рис. 3.20.1) — **test.txt** и **text.txt**.

Рис. 3.20.1. Результат прочтения содержимого каталога

mkdir — создает новый пустой каталог. С помощью приведенной ниже программы мы создадим пустой каталог **empty** (**файл 3_20_2.py**).

```
#! C:/Python/python
import os
os.mkdir('empty')
print('Content-type: text/html; charset=utf-8\n\n')
print('Каталог создан!')
```

rmdir — удаляет пустой каталог. Если в нем есть вложенные папки и файлы, сначала необходимо удалить их. В следующем примере мы удалим созданный на предыдущем шаге пустой каталог **empty** (**файл 3_20_3.py**).

```
#! C:/Python/python
```

```
import os

os.rmdir('empty')

print('Content-type: text/html; charset=utf-8\n\n')

print('Каталог удален!')
```

Из модуля **shutil** нам интересна одна функция. Как вы обратили внимание, удалять заполненный каталог с помощью функции **rmdir** из модуля **os** — хлопотное занятие: сначала надо удалить все содержимое самого каталога, а только потом его. И тут нам на помощь приходит модуль **shutil**.

rmtree — удаляет каталог со всеми вложенными папками и файлами. Экспериментируя с этой функцией, будьте осторожны и внимательны, чтобы случайно не удалить какой-либо каталог с важными файлами ваших программ или операционной системы компьютера! В следующем примере (**файл 3_20_4.py**) мы удалим каталог **test**, внутри которого находятся два файла. Перед этим сохраните копию папки **test** — возможно, она пригодится вам для повторных опытов.

```
#! C:/Python/python
```

```
import shutil

shutil.rmtree('test')

print('Content-type: text/html; charset=utf-8\n\n')

print('Каталог удален!')
```

3.21. Функции

Функция — это набор инструкций, объединенных в общий блок, который выполняет те или иные задачи и многократно используется программой.

Мы уже упоминали функции и даже пользовались некоторыми из них — например, **print** — для вывода в браузер страниц с примерами. Но в этом разделе нам предстоит поговорить не о встроенных функциях, а о тех, которые программист пишет сам.

Такие функции начинаются с ключевого слова **def**, после которого пишутся имя функции, круглые скобки и двоеточие. Если функция принимает какие-либо параметры, их указывают в круглых скобках через запятую. Все инструкции внутри функции выделяются одинаковым количеством пробелов в начале строки. Согласно PEP 8 лучше ставить четыре пробела.

В имени функции можно использовать латинские буквы, цифры и символы подчеркивания. При этом имя не должно начинаться с цифры и не может совпадать ни с одним ключевым словом. Как и переменные, имена функций чувствительны к регистру.

Если функция должна возвращать какой-то результат, то в последней строке необходимо записать такую инструкцию:

```
return Возвращаемое_значение
```

Переменные, созданные в теле функции, являются локальными и не доступны в других частях программы.

Прежде чем вызвать функцию, ее необходимо описать (дать определение). Вызывают функцию следующим образом:

```
# Определение функции
def func():
  ...
  инструкции
  ...

# Вызов функции
func()
```

Вот пример функции без параметров и ее вызов (**файл 3_21_1.py**):

```
#! C:/Python/python

b = 'Язык программирования '

def func():
  d = b + 'Python'
  print(d)

print('Content-type: text/html; charset=utf-8\n\n')

func()
```

Здесь функция **func** составляет одну строку из двух исходных. Результат ее работы показан на рисунке 3.21.1.

Рис. 3.21.1. Результат работы функции без параметров

А это пример функции с параметрами (**файл 3_21_2.py**):

```
#! C:/Python/python

def func(a, b):
  d = a + b
  return d

print('Content-type: text/html; charset=utf-8\n\n')

print(func(5, 10))
```

Она вычисляет сумму двух исходных чисел. Какой результат мы получили, видно на рисунке 3.21.2.

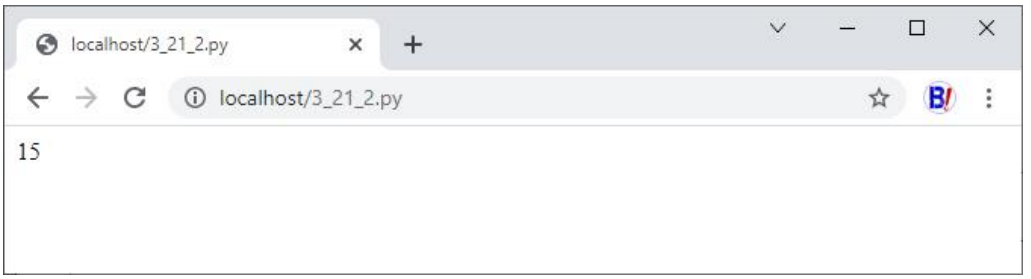

Рис. 3.21.2. Результат работы функции с параметрами

Инструкция **return** не обязательно должна быть последней в функции. Возможны ситуации, когда **return** располагается, например, в середине функции, а ее выполнение зависит от соблюдения определенных условий. Такую ситуацию демонстрирует программа из файла **файл 3_21_3.py**:

```
#! C:/Python/python

def func(a, b):
  if a < b:
    d = b - a
    return d
  else:
    print('a больше, чем b')

  print('<br>В первом случае эта строка не выводится, ',
   'так как <b>return</b> прерывает выполнение функции')

print('Content-type: text/html; charset=utf-8\n\n')

print(func(5, 10))

print('<br><br>')

func(5, 1)
```

Здесь при первом вызове **func** программа прерывается после вычисления разности чисел, переданных в параметрах. Соответственно, строка

```
print('<br>В первом случае эта строка не выводится, ',
 'так как <b>return</b> прерывает выполнение функции')
```

не печатается.

При втором вызове функции ее блок, содержащий инструкцию **return**, пропускается, а значит, последняя строка теперь появляется на экране компьютера.

Результат двух разных вызовов функции **func** показан на рисунке 3.21.3.

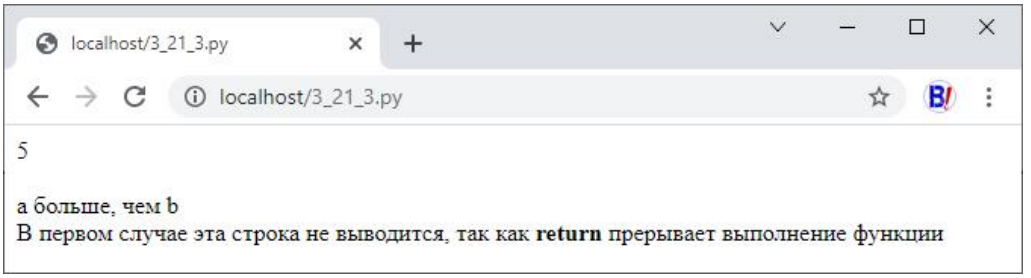

Рис. 3.21.3. Результат работы функции с инструкцией return в середине

У программиста есть возможность сохранить в переменной ссылку на функцию. И потом вызвать функцию по имени этой переменной. Вот пример (**файл 3_21_4.py**):

```
#! C:/Python/python

def func(a, b):
    d = a + b
    return d

s = func

print('Content-type: text/html; charset=utf-8\n\n')

print(s(5, 10))
```

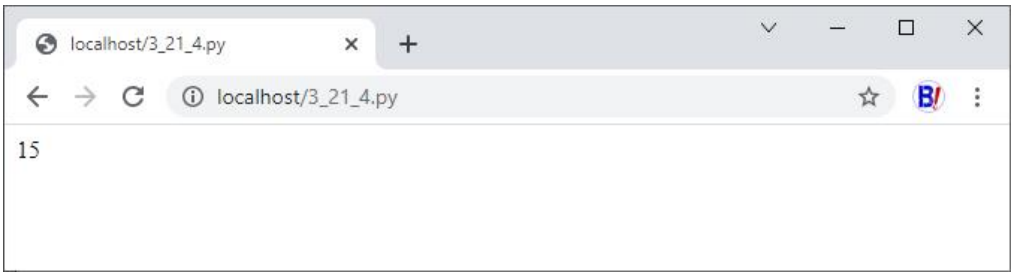

Рис. 3.21.4. Результат работы функции, запущенной через обращение к переменной

Результат суммирования — 15 — будет выведен на страницу (рис. 3.21.4).

Помимо именованных функций, в Python также предусмотрены анонимные функции. Они начинаются с ключевого слова **lambda**, за которым следует: либо двоеточие и инструкция, либо параметры, двоеточие, инструкция. Пример можно посмотреть, запустив программу **файл 3_21_5.py** (рис. 3.21.5):

```
#! C:/Python/python

s1 = lambda: 1 + 2
s2 = lambda a, b: a + b

print('Content-type: text/html; charset=utf-8\n\n')
```

```
print(s1())
print('<br><br>')
print(s2(5, 10))
```

В этой программе мы запускаем две анонимные функции: одну — без параметров, вторую — с параметрами.

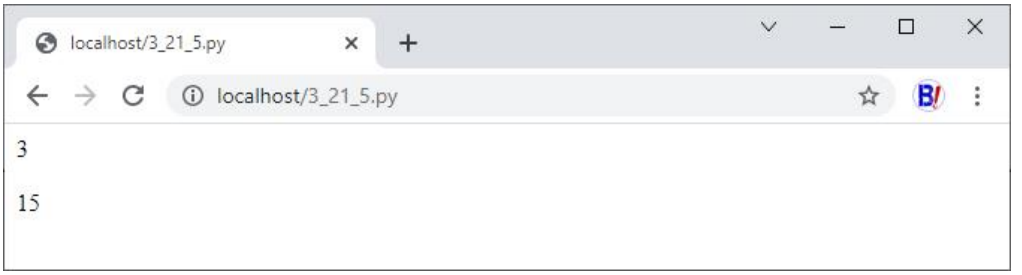

Рис. 3.21.5. Результат работы двух анонимных функций

3.22. Модули

Модулем является любая программа, которую можно подключить к вашему файлу Python или к другому модулю.

Представим, что вы создали какую-то программу, которая подходит для многих ваших разработок. Копировать и вставлять ее в десятки других файлов — крайне непродуктивно. Гораздо проще создать всего один файл и подключать его к другим программам по мере надобности. Особенно данный аспект важен, если вы написали что-то востребованное среди программистов Python. Гораздо проще делиться с ними не кодом, который может состоять из сотен строк, а готовым файлом.

Полезность модулей резюмируем следующим образом: они удобны тем, что их код написан один раз, а используется самыми разными программистами в самых разных проектах.

Процесс подключения модуля принято называть «импортированием». Делается это так: после строки с указанием адреса вы на новой строке пишете **import** и через пробел имя модуля. Указывать расширение и путь к файлу модуля не надо. Пример подключения модуля:

```
import cgi
```

В одну программу можно импортировать сразу несколько модулей. Некоторые авторы книг по программированию на Python предлагают записывать модули в одну строку через запятую. Хотя с точки зрения интерпретатора это корректная запись, однако создатель Python Гвидо ван Россум и стандарт PEP8 настаивают на том, чтобы каждый модуль подключался в новой строке отдельной директивой **import**.

`import cgi, os` — **это неправильно!**

Правильная запись показана ниже:

```
import cgi
import os
```

Имя модуля выполняет в программе, к которой он подключен, роль идентификатора. Через этот идентификатор разработчик получает доступ к переменным, свойствам, классам, функциям модуля. Это очень удобно.

Большинство модулей написаны на Python.

В стандартной поставке Python для Windows очень много разных модулей. Нам для примеров, использованных в главах 4 и 5 этой книги, понадобятся 3:

– **cgi** — необходим для получения данных из форм или значений параметров из строки запроса;

– **os** — предоставляет набор функций для взаимодействия с операционной системой компьютера;

– **re** — необходим для компиляции регулярных выражений.

Если вы хотите использовать лишь какой-то отдельный объект модуля, необходимо применить инструкцию **from**, после которой идет имя модуля, затем инструкция **import**, а за ней идентификатор интересующего вас объекта. Такая запись может выглядеть следующим образом:

```
from math import sqrt
```

Аналогичная система записи применяется при использовании отдельного модуля из пакета:

```
from PIL import Image
```

Это, кстати, реальная инструкция, использованная в одном из примеров. Мы еще увидим ее в разделе 4.12.

Как, наверное, понятно из последнего примера, пакет — это набор модулей, объединенных в один каталог.

А теперь попробуем написать собственный модуль и подключить его к программе. Новый модуль создадим в папке **C:\Apache24\htdocs**, а назовем его, например, **mymod.py** (обратите внимание: имена модулей не могут начинаться с цифры). Внутри модуля напишем функцию **func**, которая возводит в квадрат число, переданное в функцию в качестве параметра из внешней программы:

```
def func(i):
  return i ** 2
```

Теперь напишем основную программу. Она будет передавать в модуль число **7** и печатать возвращенный результат. Назовем этот файл **3_22_1.py**.

В начале программы указываем путь к интерпретатору:

```
#! C:/Python/python
```

а затем подключаем модуль:

```
import mymod
```

Печатаем строку заголовка:

```
print('Content-type: text/html; charset=utf-8\n\n')
```

и предварительный текст:

```
print('Квадрат числа 7 - ')
```

В последней строке вызываем из модуля **mymod** функцию **func** с параметром **7**:

```
mymod.func(7)
```

и печатаем результат:

```
print(mymod.func(7))
```

Его вы можете посмотреть на рисунке 3.22.1. Как видите, наш модуль прекрасно работает.

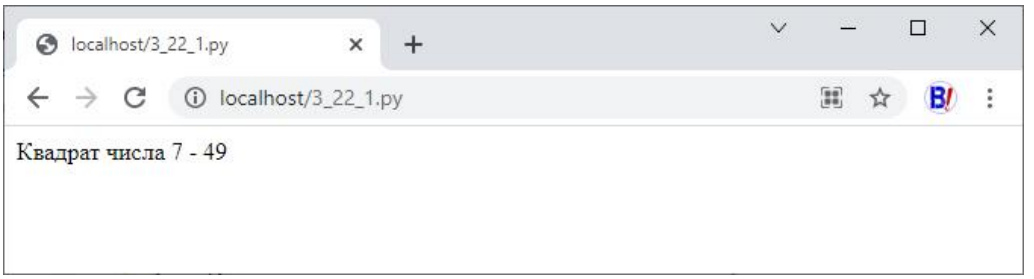

Рис. 3.22.1. Результат вычисления, произведенного функцией из модуля

Теперь еще один важный момент. Вы, наверное, обратили внимание, что после первого запуска нашего модуля в рабочем каталоге **C:\Apache24\htdocs** появилась новая папка **__pycache__** с файлом внутри. Это произошло вот по какой причине. При запуске модуль сначала компилируется и преобразуется в особый байт-код. Для откомпилированных модулей Python создает в каталоге с модулем специальную папку **__pycache__** и размещает в ней откомпилирован-

ный файл с именем **module.cpython-XX.pyc**, где **module —** имя модуля, **cpython-** — стандартная часть, присутствующая в именах всех откомпилированных файлов, **XX —** первые две цифры номера версии Python, а **.pyc —** расширение файла. До тех пор, пока в коде основного модуля не произойдут изменения, при каждом его вызове Python будет запускать именно откомпилированный файл. Делается это для ускорения работы программ.

4. Практика

Настало время применить полученные знания на практике. В этой главе мы рассмотрим несколько простых случаев использования Python в web. Необходимые для демонстрации примеров папки и файлы находятся в папке «Глава 4» zip-архива. Чтобы запускать программы в браузере, выполните следующие действия:

– удалите из папки **C:\Apache24\htdocs** все файлы;

– скопируйте все содержимое папки «Глава4» zip-архива;

– поместите скопированные папки и файлы в папку **C:\Apache24\htdocs**;

– чтобы запустить тот или иной пример в браузере, в строке адреса введите **http://localhost/file_name.py** или **http://localhost/file_name.html**, где **file_name.py** и **file_name.html** — имена файлов.

4.1. Вывод больших объемов HTML-кода

Особенность HTML-страниц состоит в том, что они имеют изначально заданную структуру. Конечно, с помощью сценариев на JavaScript можно внести определенные изменения и дополнения, но делается это только после загрузки документа. Между тем разработчику очень часто бывает необходимо сформировать страницу непосредственно в процессе ее загрузки на основании параметров, переданных в запросе. И тут на помощь приходят специальные языки программирования, например PHP или Python.

PHP удобен тем, что его инструкции встраиваются непосредственно в разметку страницы. У языка программирования, который мы рассматриваем в этой книге, такая возможность не предусмотрена. Поэтому разработчики поступают обратным образом — встраивают HTML-код в файлы Python. Выглядит это примерно так:

```
#! C:/Python/python
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                       |
| Код Python                            |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|

print('Content-type: text/html\n\n')
print('<!DOCTYPE html>')
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                                         |
| HTML-код                                                |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                       |
| Код Python                            |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                                         |
| HTML-код                                                |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                                       |
```

```
| Код Python          |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
```

```
print('</body></html>')
```

Зачем это надо? В большинстве случаев разработчик не пишет для каждой страницы отдельный файл Python, а создает одну программу, которая на основе данных из запроса формирует ту или иную страницу с тем или иным содержанием.

Автору не раз приходилось наблюдать и в литературе по Python, и в реальных разработках подход, при котором каждая строка HTML-кода печаталась отдельно:

```
print('Content-type: text/html\n\n')
print('<!DOCTYPE html>')
print('<html lang="ru">')
print('<head>')
print('<meta charset="utf-8">')
print('<title>Заголовок</title>')
print('</head>')
print('<body>')
print('<div>')
...
...
...
```

На мой взгляд, это крайне непродуктивно, нерационально и неэкономно. Как видите, все эпитеты с приставкой «не».

Между тем есть способ поместить в одну функцию **print** сразу множество строк HTML-кода. Тем самым можно сократить программу, а также избавиться от чрезмерного использования **print**.

Помните, в главе 3 мы обсуждали тему различных кавычек? Один из вариантов их применения состоял в использовании слева и справа от текста тройных одинарных кавычек — для размещения большого объема комментариев:

```
'''Этот
текст
не
выводится
в
браузере'''
```

Но! Если передать некий текст, заключенный в тройные одинарные кавычки, функции **print**, то этот текст будет напечатан. Так почему бы не передавать в качестве такого текста HTML-разметку:

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

print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
```

```
<title>Заголовок</title>
</head>
<body>
<div>
...
...
...''')
```

Как видите, цель достигнута — мы напечатали необходимый фрагмент страницы и при этом сэкономили: вместо 9 раз использовали функцию **print** только 2 раза.

С помощью данного подхода можно выводить на печать произвольно большие объемы HTML-кода. Данный принцип хорошо иллюстрирует файл **4_1_1.py** из папки «Глава 4» zip-архива (рис. 4.1.1). Мы напечатали документ со множеством рисунков (их 33), всего дважды использовав функцию **print**:

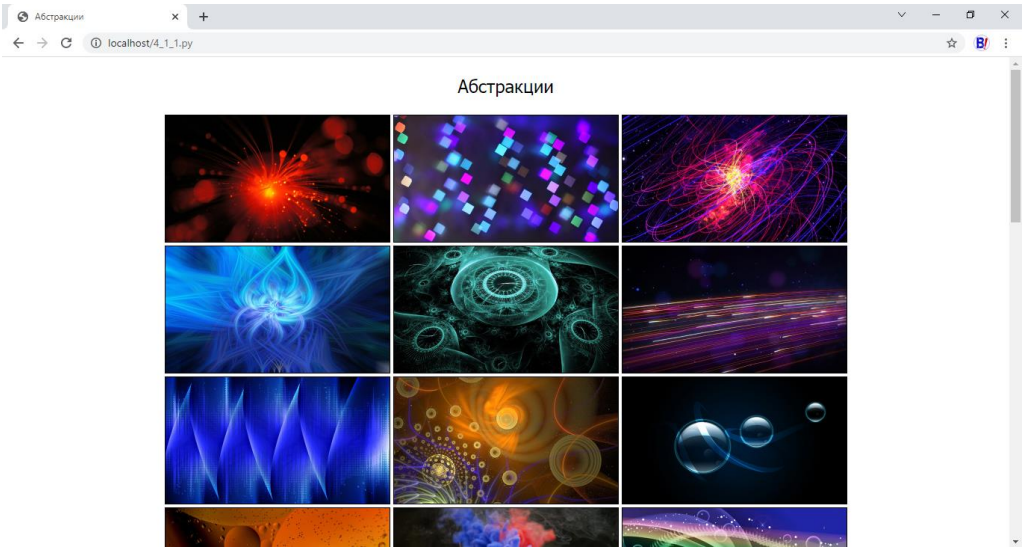

Рис. 4.1.1. Страница с большим количеством HTML-кода, выведенного всего двумя функциями print

```
#! C:/Python/python

# Начинаем печатать страницу.
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Абстракции</title>
<style>
body {margin: 0;}
div {position: relative; width: 1000px; margin: auto;
                  text-align: center;}
p {text-align: center; font-family: Tahoma;
```

```
                    font-size: 24px;}
img {width: 300px; border: 1px solid #000000;}
</style>
</head>

<body>
<div>
<p>Абстракции</p>
<img src="abs/1.jpg" alt="Абстракция">
<img src="abs/2.jpg" alt="Абстракция">
<img src="abs/3.jpg" alt="Абстракция">
<br>
...
<img src="abs/31.jpg" alt="Абстракция">
<img src="abs/32.jpg" alt="Абстракция">
<img src="abs/33.jpg" alt="Абстракция">
</div>
</body>
</html>''')
```

Завершая этот раздел, хочу обратить внимание читателей на еще один важный момент. Если раньше мы писали тип загружаемого документа так:

```
print('Content-type: text/html; charset=utf-8\n\n')
```

то в данном примере мы удалили указание кодировки:

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

Теперь она устанавливается непосредственно в заголовочной части страницы:

```
<meta charset="utf-8">
```

Отныне этого подхода мы будем придерживаться во всех остальных примерах и в программах из главы 5.

4.2. Получение данных из форм

Для получения данных, введенных в полях формы, необходимо использовать модуль **cgi**. В нем есть специальный класс с конструктором **FieldStorage**, который создает объект, включающий в себя данные из форм, сохраненные в виде словаря. Такой словарь состоит из пар «имя поля — значение». Извлечь значения можно с помощью метода **getfirst**, который принимает в качестве параметра имя поля формы.

Получение данных из форм мы проиллюстрируем на примере довольно простого скрипта **4_2_1.py**.

Запустите программу в браузере. Вы увидите форму с двумя полями — «Введите имя» и «Введите фамилию» — и кнопкой «Отправить» (рис. 4.2.1).

Ниже — текст «Вас зовут» без продолжения, так как мы еще ничего не вводили и не отправляли форму.

Заполним поля. Например, введем имя Иван, а фамилию Иванов. Нажмем кнопку «Отправить». Программа получит данные из формы и выведет их рядом с текстом «Вас зовут» (рис. 4.2.2.).

Начинаем писать программу.

Первым делом указываем путь к интерпретатору:

```
#! C:/Python/python
```

Подключаем модуль **cgi**:

```
import cgi
```

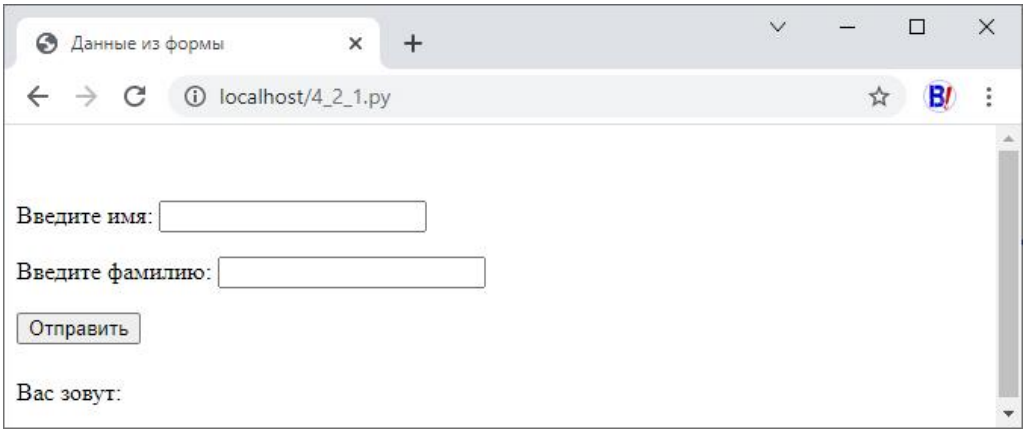

Рис. 4.2.1. Форма

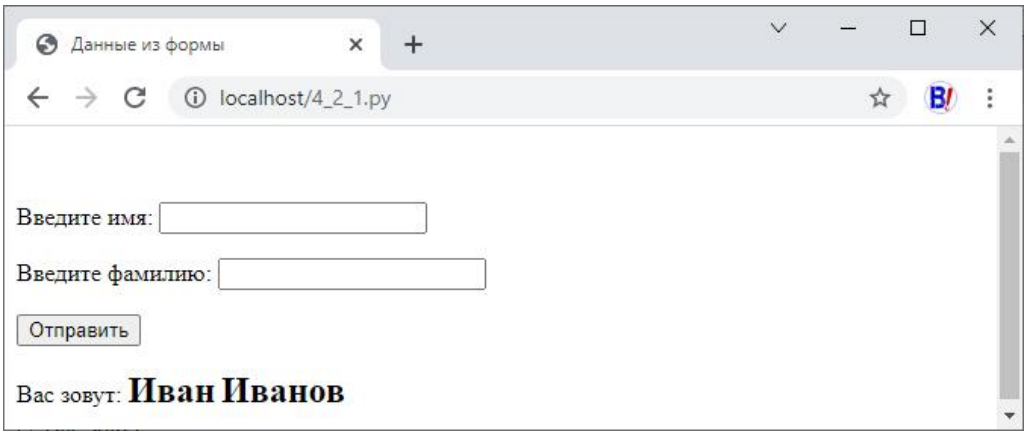

Рис. 4.2.2. Данные из полей формы извлечены

Объявляем две «пустые» переменные — **nam** и **sur**:

```
nam = ''
```

```
sur = ''
```

Объясним данный «ход». Эти переменные нужны для получения имени и фамилии пользователя. На что надо обратить внимание: страница с формой и программа обработки данных — это один и тот же документ, имеющий две стадии исполнения. В исходном состоянии (то есть сразу после запуска программы) данных из формы еще нет, а переменные уже внедрены в тело документа:

```
<p>Вас зовут: <span>''' + nam + '</span> <span>' +
                  sur + '''</span></p>
```

Если их не объявить и не инициализировать в самом начале, это вызовет ошибку в работе скрипта. Значит, на то время, пока переменные не используются по прямому назначению (получение данных из полей формы), мы должны присвоить им пустое значение.

Идем дальше. Создаем переменную **form**, в которую помещаем объект класса **FieldStorage** с данными из формы:

```
form = cgi.FieldStorage()
```

Проверяем, получены ли данные:

```
if form.getfirst('nam'):
  nam = form.getfirst('nam')
if form.getfirst('sur'):
  sur = form.getfirst('sur')
```

Присвоение переменным значений из полей происходит только в том случае, если мы заполнили и отправили форму. В исходном состоянии условия ложны и этот блок кода пропускается.

Печатаем страницу:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Данные из формы</title>
<style>
span {font-weight: bold; font-size: 24px;}
</style>
</head>
<body>
<p> </p>
<form method="POST" action="4_2_1.py">
<p>Введите имя: <input name="nam"></p>
<p>Введите фамилию: <input name="sur"></p>
<p><input type="submit" value="Отправить"></p>
</form>
<p>Вас зовут: <span>''' + nam + '</span> <span>' +
                  sur + '''</span></p>
```

```
</body>
</html>''')
```

Сразу после загрузки документа переменные пустые, значит из строки

```
<p>Вас зовут: <span>''' + nam + '</span> <span>' +
                sur + '''</span></p>
```

напечатается только «Вас зовут:»

После отправки данных в этой строке будут дополнительно напечатаны имя и фамилия.

Как видите, мы успешно отправили данные из формы, благополучно их получили и вывели в окно браузера.

4.3. Проверка данных

Термин «проверка» у многих читателей, наверное, ассоциируется с вопросами безопасности web-ресурса. Этой темы мы коснемся позже, в разделе 5.5, когда станем разбирать наш проект — сайт студии звукозаписи. А пока же констатируем факт: обычно проверка решает две задачи:

– необходимо убедиться, что клиент отправил именно те данные, которые вы ожидаете получить;

– нужно поставить защитные барьеры для недоброжелателей, которые захотят нанести ущерб вашему ресурсу.

Этими вопросами мы займемся в данном разделе. Сразу хочу предупредить: в задачу книги не входит цель продемонстрировать сложную многоуровневую проверку данных из форм. Автор просто хотел показать алгоритм решения этой проблемы. Мы рассмотрим несколько простых приемов тестирования значений из полей формы. В их числе будут:

– проверка наличия данных в полях (чтобы не получить пустую форму);

– определение, не выходят ли данные за рамки минимального количества знаков;

– выяснение, не выходят ли данные за рамки максимального количества знаков (отправка больших объемов информации используется некоторыми «товарищами», чтобы вызвать замедление работы вашего ресурса или даже его «зависание»);

– проверка соответствия данных установленному формату.

Проверку первых трех пунктов мы проиллюстрируем на примере скрипта **4_3_1.py**.

Запустите файл в браузере. Вы увидите форму, аналогичную той, что была в предыдущем разделе (рис. 4.3.1). Ниже — текст «Вы заполнили не все поля!», сигнализирующий о том, что надо ввести имя и фамилию.

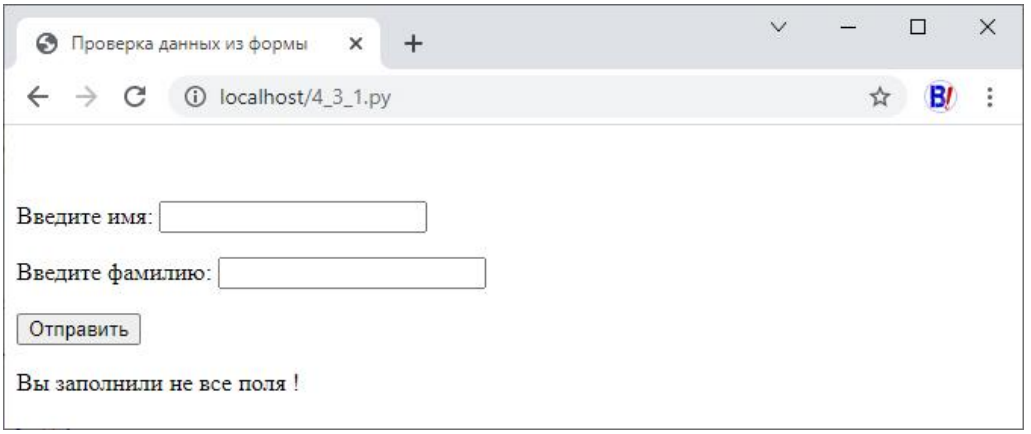

Рис. 4.3.1. Форма

Если нажать кнопку «Отправить», ничего не заполняя или заполнив только одно поле, то надпись ниже формы не изменится — «Вы заполнили не все поля!».

Если ввести в обоих полях по одному знаку, то вы получите предупреждение «Слишком короткое имя или фамилия! Таких не бывает» (рис. 4.3.2).

Если ввести в обоих полях более 15 знаков (число выбрано произвольно), то будет выдано предупреждение «Слишком длинное имя или фамилия! Таких не бывает» (рис. 4.3.3).

И только корректные данные будут приняты скриптом и выведены на страницу под формой (рис. 4.3.4).

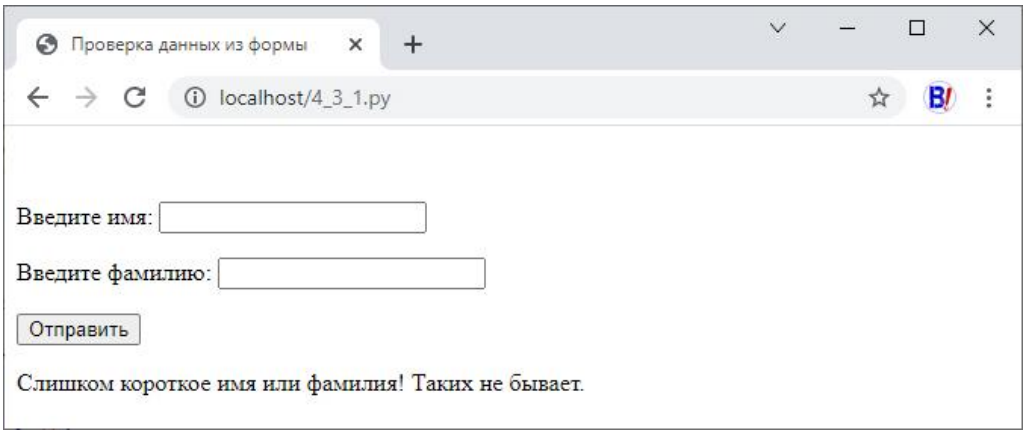

Рис. 4.3.2. Предупреждение, если в полях формы введено всего по одному знаку

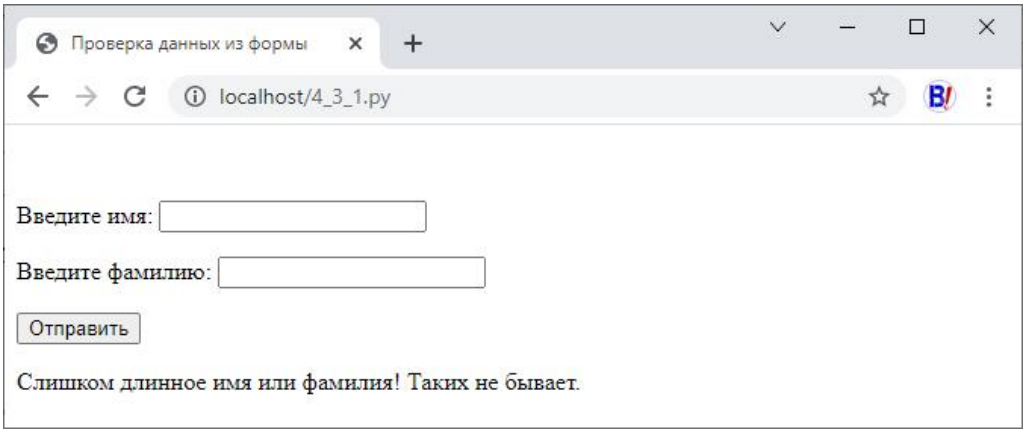

Рис. 4.3.3. Предупреждение, если количество знаков в полях формы превышает допустимые значения

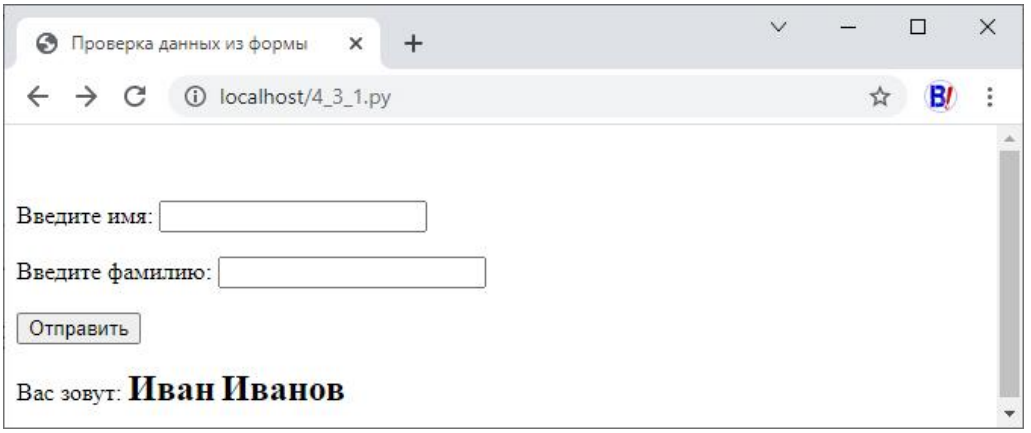

Рис. 4.3.4. Если данные корректны, выводим их на страницу

Разберем код программы.

Указываем путь к интерпретатору и подключаем модуль **cgi**:

```
#! C:/Python/python
import cgi
```

Получаем значения параметров из формы:

```
form = cgi.FieldStorage()
nam = form.getfirst('nam')
sur = form.getfirst('sur')
```

Печатаем часть страницы до закрывающего тега </form>:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
```

```
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Проверка данных из формы</title>
<style>
span {font-weight: bold; font-size: 24px;}
</style>
</head>
<body>
<p> </p>
<form method="POST" action="4_3_1.py">
<p>Введите имя: <input name="nam"></p>
<p>Введите фамилию: <input name="sur"></p>
<p><input type="submit" value="Отправить"></p>
</form>''')
```

Начинаем проверку.

Первым делом определяем, содержатся ли какие-то данные в переменных **nam** и **sur**. Если форма отправлена пустой, то выводим соответствующее предупреждение:

```
if nam is None or sur is None:
  print('<p>Вы заполнили не все поля !</p>')
```

Следующий шаг — проверка, не слишком ли короткие имя и фамилия. Если в них меньше двух знаков, значит, отправка формы — чья-то неуместная шутка. Сообщаем об этом шутнику:

```
elif len(nam) < 2 or len(sur) < 2:
  print('<p>Слишком короткое имя или фамилия!
                Таких не бывает.</p>')
```

Слишком много символов — тоже признак того, что отправитель не является добросовестным посетителем сайта. Сигнализируем о превышении количества знаков в полях:

```
elif len(nam) > 15 or len(sur) > 15:
  print('<p>Слишком длинное имя или фамилия!
                Таких не бывает.</p>')
```

Если форма заполнена правильно, печатаем введенные данные:

```
else:
  print('<p>Вас зовут: <span>' + nam +
       '</span> <span>' + sur + '</span></p>')
```

На последнем этапе завершаем печать страницы:

```
print('''</body>
</html>''')
```

Для иллюстрации проверки на соответствие данных установленному формату мы рассмотрим программу **4_3_2.py**. Она проверяет корректность введенного адреса электронной почты (рис. 4.3.5).

В примере выше мы определяли количество знаков в полях формы. Однако для выяснения корректности e-mail этих параметров недостаточно. Мы не проверили:

– наличие в адресе электронной почты знака **@**;

– наличие обязательной точки, разделяющей доменные имена первого и второго уровней;

– наличие в адресе запрещенных символов.

Решить данную проблему нам поможет регулярное выражение. Попробуем написать его для проверки адреса почты.

Вообще, корректный e-mail может содержать самые разнообразные символы. Но мы будем ориентироваться на требования, которые предъявляют к адресам российские почтовые сервисы. А они заметно снижают перечень таких символов. Например, в Яндекс.Почте установлены следующие требования:

– разрешены буквы латинского алфавита, цифры, точки, дефисы;

– адрес не должен начинаться с цифры, дефиса или точки;

– адрес не может заканчиваться точкой или дефисом.

Есть правила и для доменных имен второго уровня:

– они могут содержать латинские буквы, цифры и дефис;

– имя может начинаться с цифры или буквы;

– имя не может завершаться дефисом.

Введите в поле формы электронный адрес, нарушающий эти правила и нажмите клавишу «Enter» на компьютере. Ниже поля появится короткое сообщение «Ошибка!» (рис. 4.3.6). Теперь введите корректный адрес и снова нажмите «Enter». Вы увидите подтверждение, что «E-mail правильный» (рис. 4.3.7).

Прежде чем разбирать код такой программы, выясним, что должно представлять собой регулярное выражение, созданное по правилам, описанным выше.

Адрес не должен начинаться с цифры, дефиса или точки. Этому требованию удовлетворяет вот такой фрагмент выражения:

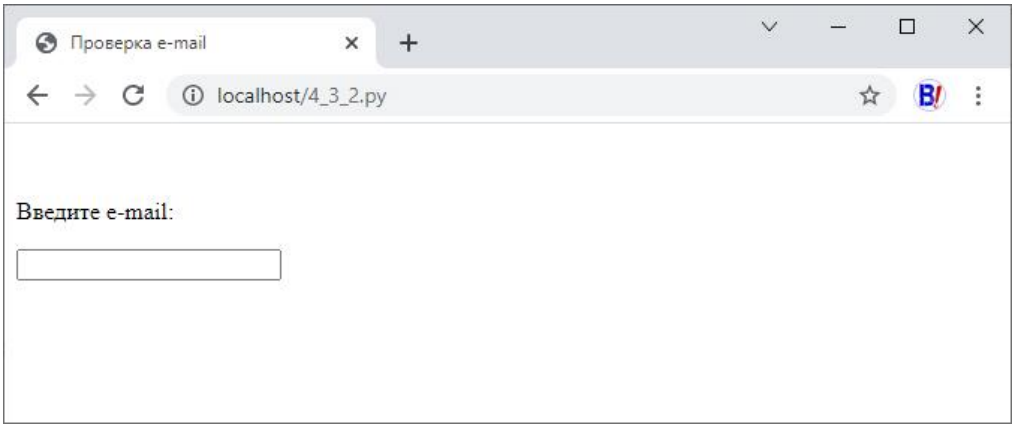

Рис. 4.3.5. Поле для ввода адреса электронной почты

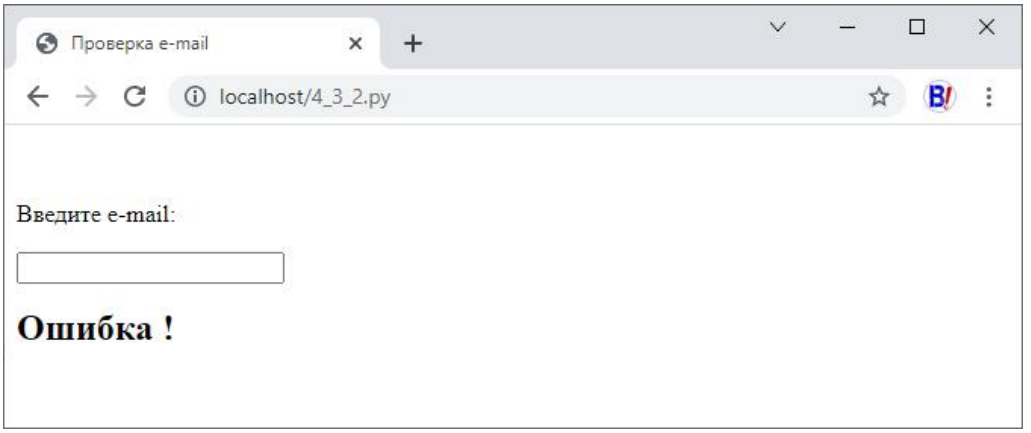

Рис. 4.3.6. Сообщение о некорректном e-mail

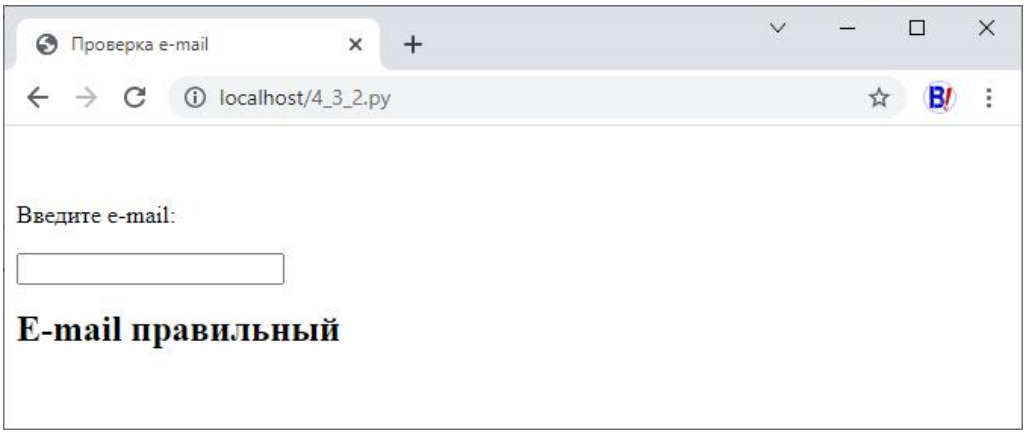

Рис. 4.3.7. Подтверждение корректности адреса электронной почты

```
[a-z]
```

То есть, первый символ — только буква. Затем идет некоторое количество произвольных символов из тех, что разрешены — буквы, цифры, точка, дефис:

```
[a-z\d.-]+
```

Знак «+» означает, что символов может быть более одного. Наконец, завершающий символ — буква или цифра:

```
[a-z\d]
```

Дальше используем почтовый знак **@**.

Теперь проверка доменного имени второго уровня. Оно начинается с буквы или цифры:

```
[a-z\d]
```

Следом могут идти буквы, цифры и дефис:

```
[a-z\d-]+
```

Завершающий символ — буква или цифра:

```
[a-z\d]
```

Имя домена второго уровня отделяется точкой от имени домена первого уровня:

```
\.
```

Самое короткое доменное имя первого уровня состоит из двух символов (например, **ru**). Самое длинное из обнаруженных автором — из 13 (**international**). Поэтому адрес почты должен завершаться после точки минимум двумя, а максимум — тринадцатью буквами:

```
[a-z]{2,13}
```

Напомню, что регулярное выражение заключается в апострофы. Для обозначения начала и конца ввода используются специальные знаки ^ и **$**. А начинать строку лучше с префикса **r**, чтобы не экранировать обратные слеши:

```
r'^...$'
```

В итоге у нас получилось вот такое регулярное выражение:

```
r'^[a-z][a-z\d.-]+[a-z\d]@[a-z\d][a-z\d-]+[a-z\d]\.
                    [a-z]{2,13}$'
```

Теперь перейдем непосредственно к программе.

Типичное начало — указываем путь к интерпретатору и подключаем необходимые модули — **cgi** и **re**:

```
#! C:/Python/python
```

```
import cgi
import re
```

Получаем адрес электронной почты:

```
form = cgi.FieldStorage()
em = form.getfirst('em')
```

Начинаем печать страницы:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
```

```
<head>
<meta charset="utf-8">
<title>Проверка e-mail</title>
<style>
b {font-size: 24px;}
</style>
</head>
<body>
<p> </p>
<form method="POST" action="4_3_2.py">
<p>Введите e-mail:</p>
<p><input name="em"></p>
</form>''')
```

Проверяем, введен ли e-mail:

```
if em:
```

Если нет, просто печатаем завершающие теги документа:

```
print('''</body>
</html>''')
```

Если да, создаем регулярное выражение:

```
rv = r'^[a-z][a-z\d.-]+[a-z\d]@[a-z\d][a-z\d-]+
                [a-z\d]\.[a-z]{2,13}$'
```

и компилируем его, одновременно указывая, что проверка должна выполняться без учета регистра букв:

```
it = re.compile(rv, re.I)
```

Теперь выполняем в адресе почты поиск совпадения с регулярным выражением:

```
res = it.search(em)
```

Если поиск был успешным, печатаем подтверждение «E-mail правильный». Если поиск не увенчался положительным результатом, выводим предупреждение «Ошибка!»:

```
if res:
  print('<p><b>E-mail правильный</b></p>')
else:
  print('<p><b>Ошибка !</b></p>')
```

4.4. Загрузка файлов

Один из важных элементов работы с формами — это загрузка клиентом файлов на сервер. Рассмотрим данный процесс на примере загрузки изображений с компьютера посетителя.

Данную опцию реализует программа **4_4_1.py**.

Запустив ее, мы увидим форму с двумя кнопками: «Выберите файл» и «Отправить». Под ними текст, приглашающий к выбору, — «Выберите файл!» (рис. 4.4.1).

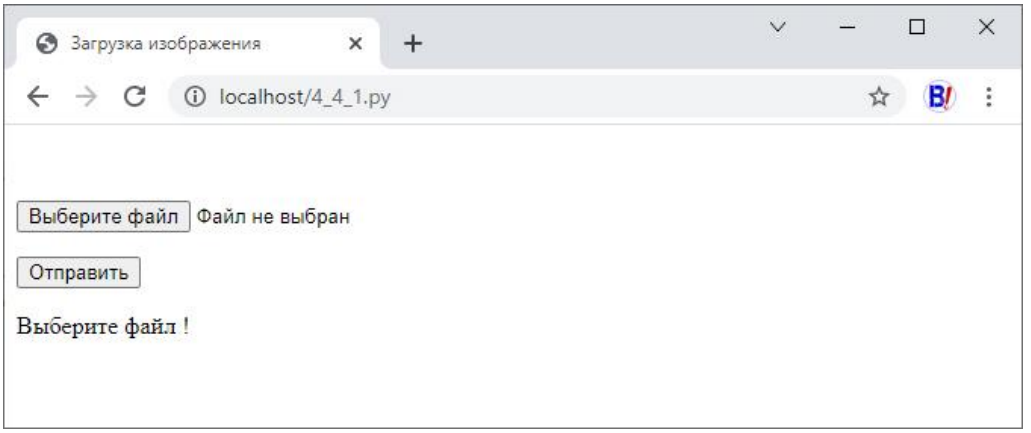

Рис. 4.4.1. Форма для загрузки файлов

Если ничего не выбирать, а просто нажать кнопку «Отправить», то страница перезагрузится — и все.

Если выбрать файл для загрузки (например, из нашей папки **abs**), то его имя с расширением появится на странице, рядом с кнопкой «Выберите файл» (рис. 4.4.2).

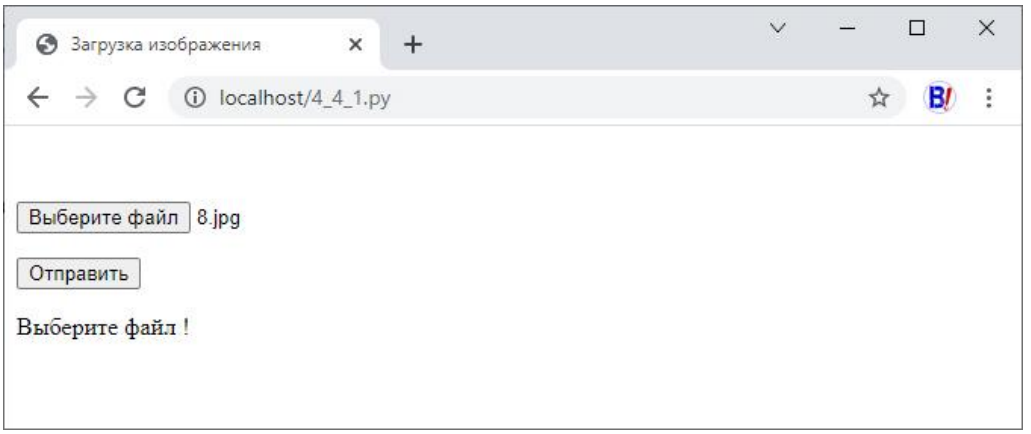

Рис. 4.4.2. Файл выбран и готов к загрузке

Нажав кнопку «Отправить», вы загрузите файл на сервер в папку **pict**, после чего данное изображение появится на странице под формой (рис. 4.4.3).

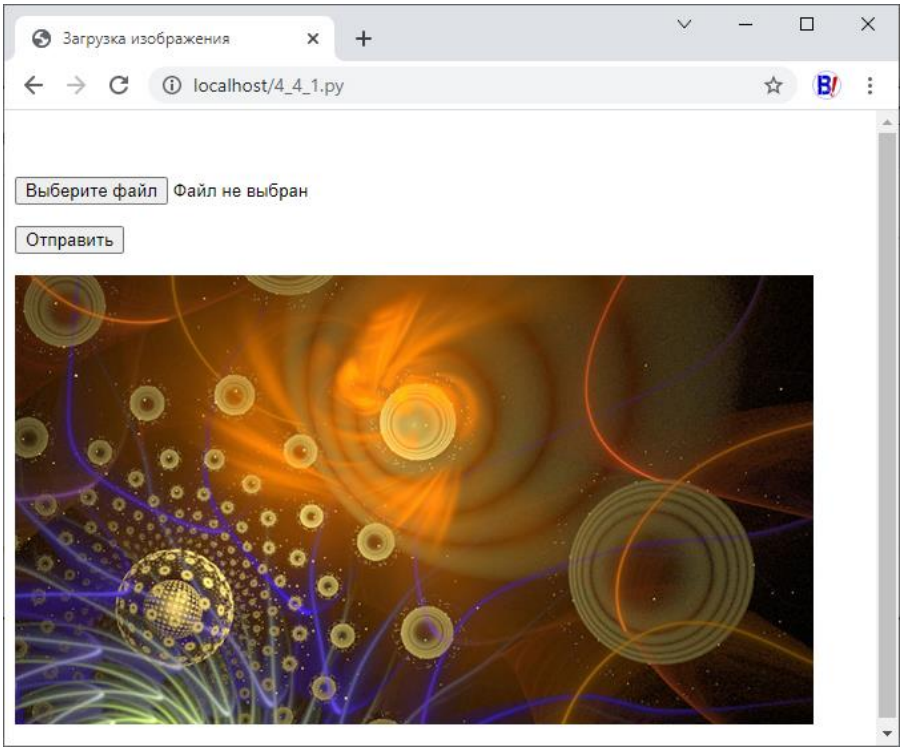

Рис. 4.4.3. Файл загружен и показан на странице

Как видите, процесс выглядит точно так же, как на других ресурсах.

Программа в обязательном порядке начинается с указания пути к интерпретатору. Это действие настолько уже привычно и стандартно, что мы в дальнейшем не будем упоминать о нем. А вот загружаемые модули перечислим, так как для разных скриптов они могут быть разными. В данном случае необходимы два модуля:

```
import cgi
import os
```

Создаем новый объект формы:

```
form = cgi.FieldStorage()
```

и получаем данные:

```
imfo = form.getfirst('pict')
```

Вторая операция нужна только для того, чтобы создать элемент для проверки, была ли выполнена загрузка. Реальное получение файла произойдет позже и другим способом.

Печатаем основную часть страницы:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Загрузка изображения</title>
</head>
<body>
<p> </p>
<form method="POST" action="4_4_1.py"
            enctype="multipart/form-data">
<p><input type="file" name="pict"></p>
<p><input type="submit"></p>
</form>''')
```

Обратите внимание: теперь в описание формы мы добавили атрибут **enctype** со значением **multipart/form-data**. Этот атрибут необходимо указывать при отправке файла (или нескольких файлов).

Проверяем, пришли из формы данные или нет:

```
if imfo:
```

Если файл не был отправлен, пишем требование его выбрать:

```
else:
  print('Выберите файл !')
```

и завершаем печать документа:

```
print('''</body>
</html>''')
```

Если файл был загружен, получаем его:

```
up = form['pict']
```

и выясняем имя файла:

```
ima = os.path.basename(up.filename)
```

Второе действие необходимо, чтобы сохранить загруженный файл под его исходным именем.

Следующий шаг — создаем в папке **pict** новый файл и помещаем данные этого объекта в переменную **fim**:

```
with open('pict/' + ima , 'wb') as fim:
```

Обратите внимание, что функция open запускается с параметром **w** и модификатором **b**. Это значит, что файл открывается для записи и в бинарном режиме.

Четвертый шаг — запись данных исходного файла в тот, что был создан на предыдущем этапе:

```
fim.write(up.file.read())
```

Здесь сначала происходит считывание оригинала:

```
up.file.read()
```

а потом запись:

```
fim.write(...)
```

Финальная процедура — вывод на страницу полученного изображения, чем подтверждается факт успешной загрузки:

```
print('<p><img src="pict/' + ima + '" alt="Фото"></p>')
```

Разобранная нами программа носит лишь демонстрационный характер. Понятно, что при реальной загрузке файлов на реальном сайте нельзя обойтись без тщательной проверки. Необходимо выяснить:

– что файл относится именно к тому типу, который вы ожидаете получить;

– что «вес» файла не превышает разрешенного предела.

4.5. Условный и безусловный вывод

Термины «условный» и «безусловный» вывод не являются общепринятыми, поэтому необходимо пояснить их значение.

Начнем с того, что файл Python может запускаться в «чистом» виде, например так: **http://localhost/4_5_1.py**. А может с параметрами, например так: **http://localhost/4_5_1.py?t=value**.

Когда файл запускается без параметров, но некие данные в нем встроены по умолчанию, это называется безусловным выводом. То есть информация уже присутствует в файле, независимо ни от каких условий. Чтобы ее изменить, необходимо выполнить запрос с параметрами, которые имеют новые характеристики.

Условный вывод описывает ситуацию, когда изначальные данные отсутствуют, а та или иная информация появляется на странице только в зависимости от переданных параметров.

Зачем нужны такие способы программирования? Ответ очень простой: для корректировки исходного документа на основании параметров из запроса.

В первую очередь рассмотрим безусловный вывод. Для этого запустите демонстрационную программу **4_5_1.py**. Что вы увидите на странице? Текст «Город, в котором вы живете: **Москва**» (рис. 4.5.1). Значение «Москва» установлено в программе по умолчанию. Но, вполне возможно, вы житель Самары. Чтобы сообщить об этом, добавьте в конце строки запроса следующий

текст: **?t=Самара**. Должно получиться **http://localhost/4_5_1.py?t=Самара**. Вставьте курсор в строку адреса и нажмите «Enter». Страница перезагрузится, а текст изменится. Теперь он читается так: «Город, в котором вы живете: **Самара**» (рис. 4.5.2). Название города изменено в соответствии с переданным значением параметра **t**.

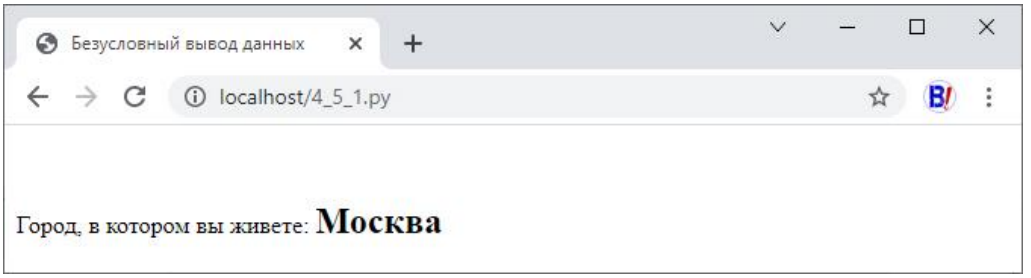

Рис. 4.5.1. Информация, выводимая на страницу по умолчанию

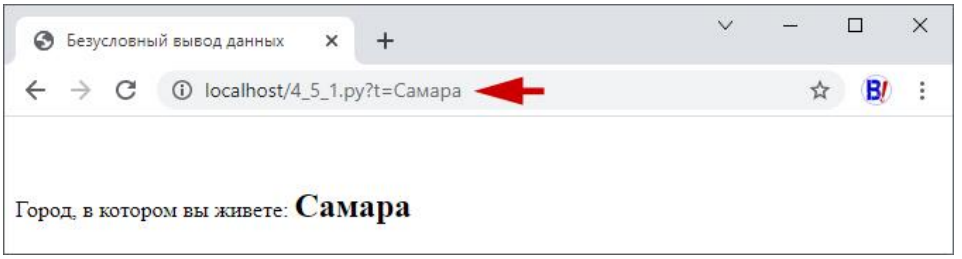

Рис. 4.5.2. Название города изменено в соответствии с переданным значением параметра **t**

Реализация этого алгоритма очень проста.

Создаем файл программы. Добавляем необходимый модуль:

```
import cgi
```

Вводим переменную **town** с изначально установленным значением:

```
town = 'Москва'
```

Получаем данные из строки запроса:

```
form = cgi.FieldStorage()
```

Проверяем, есть ли какое-то значение у параметра **t**:

```
if form:
  town = form.getfirst('t')
```

Печатаем страницу:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Безусловный вывод данных</title>
<style>
span {font-weight: bold; font-size: 24px;}
</style>
</head>
<body>
<p> </p>
<p>Город, в котором вы живете: <span>''' + town +
                    '''</span></p>
</body>
</html>''')
```

Если в запросе параметр **t** отсутствует, то значение переменной **town** не изменится и на странице по-прежнему будет указан город Москва:

```
<p>Город, в котором вы живете: <span>''' + town +
                    '''</span></p>
```

Если вы установили параметру **t** значение «Самара», то название города на странице поменяется. Вот и все.

Теперь перейдем к примеру с условным выводом. Он реализован в файле **4_5_2.py**.

Запустите его. Вы увидите страницу с текстом «Город, в котором вы живете:» без указания населенного пункта (рис. 4.5.3). Его название появится только при условии, что в параметре установлено какое-то значение.

Теперь предположим, что вы — житель подмосковного Сергиева Посада. Вводим это название в качестве значения параметра **t** (рис. 4.5.4) и нажимаем клавишу «Enter». Смотрим, как выбранное название появилось на странице. Заодно обращаем внимание, что браузер заменил пробел в наименовании города на символы **%20** (рис. 4.5.5).

Разберем эту программу.

Как и в предыдущей, импортируем модуль **cgi**:

```
import cgi
```

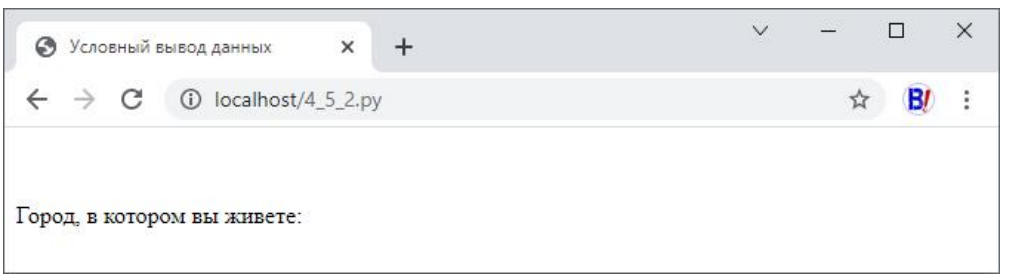

Рис. 4.5.3. Исходный вид страницы с названием населенного пункта в зависимости от значения параметра t

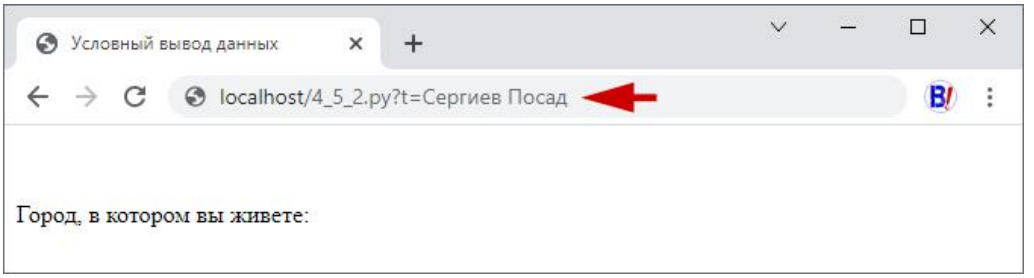

Рис. 4.5.4. Вводим название города

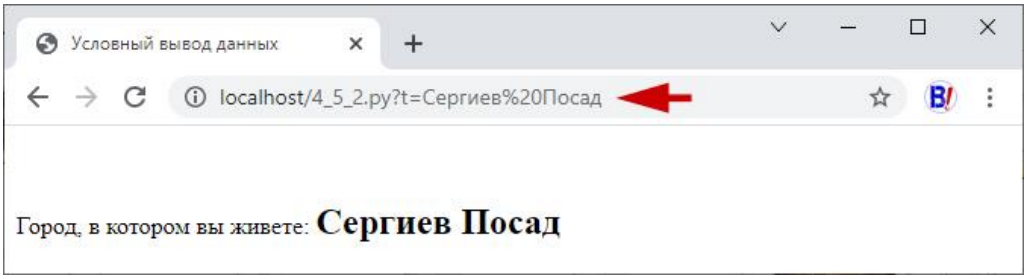

Рис. 4.5.5. Данные появились на странице

Следом формируем страницу:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Условный вывод данных</title>
<style>
span {font-weight: bold; font-size: 24px;}
</style>
</head>
<body>
<p> </p>
<p>Город, в котором вы живете:''')
```

Создаем новый объект:

```
form = cgi.FieldStorage()
```

и выясняем, не пустой ли он:

```
if form:
```

Если населенный пункт указан, получаем значение параметра **t**:

```
town = form.getfirst('t')
```

выводим название города на страницу:

```
print (' <span>' + town + '</span>')
```

после чего завершаем печать:

```
print('''</p>
</body>
</html>''')
```

4.6. Контент по запросу

В продолжение предыдущей темы рассмотрим ситуацию, как в зависимости от значения параметра может меняться не просто какая-то строчка в документе, а содержание всей страницы.

Запустите в браузере файл **4_6_1.py**. Пусть вас не смущает, что вы видите совершенно пустую страницу. Смысл примера не в этом. Мы продемонстрируем, как будет формироваться контент в соответствии со значением, переданным в параметре.

Измените строку адреса следующим образом: **http://localhost/4_6_1.py?a=1** и нажмите клавишу «Enter». Вы увидите на странице небольшую статью про абстрактную живопись (рис. 4.6.1). Она извлечена из файла **1.txt** папки **content**.

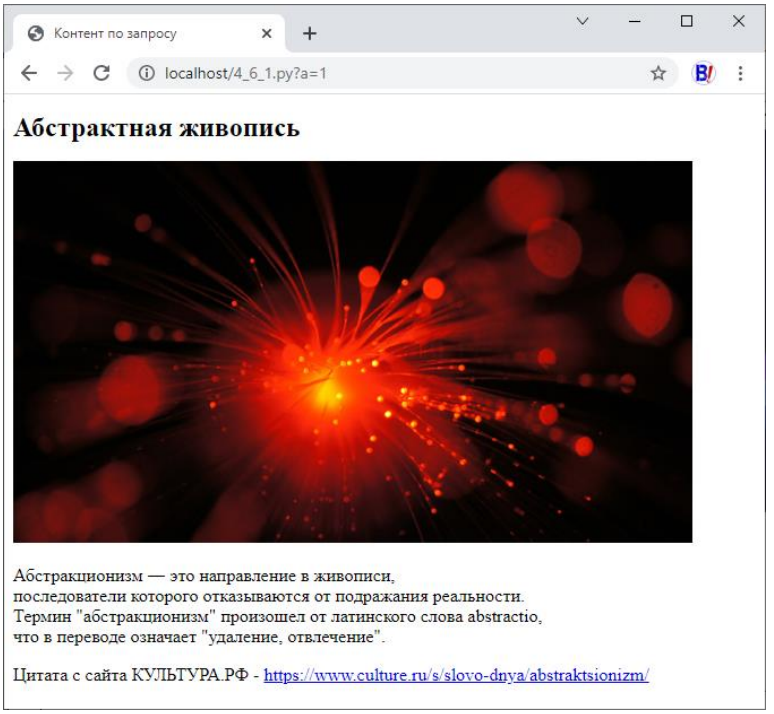

Рис. 4.6.1. В соответствии с запросом загружен файл 1.txt

Теперь в строке запроса измените цифру **1** на **2**: **http://localhost/4_6_1.py?a=2**. Появится другая страница под названием «Абстракционизм» (рис. 4.6.2). Она извлечена из файла **2.txt** папки **content**. (Обе статьи цитируют сайт КУЛЬТУРА.РФ.)

Как видите, указание в строке запроса одного и того же параметра с разными значениями позволяет одному и тому же файлу Python загружать в браузер разные страницы. Мы уже говорили об этом, но автор посчитал нелишним напомнить: в подавляющем большинстве случаев разработчик не пишет для каждой страницы отдельный файл Python, а создает одну программу, которая на основе данных из запроса формирует ту или иную страницу с тем или иным содержанием.

Перейдем к разбору программы. Нам понадобится всего один модуль:

```
import cgi
```

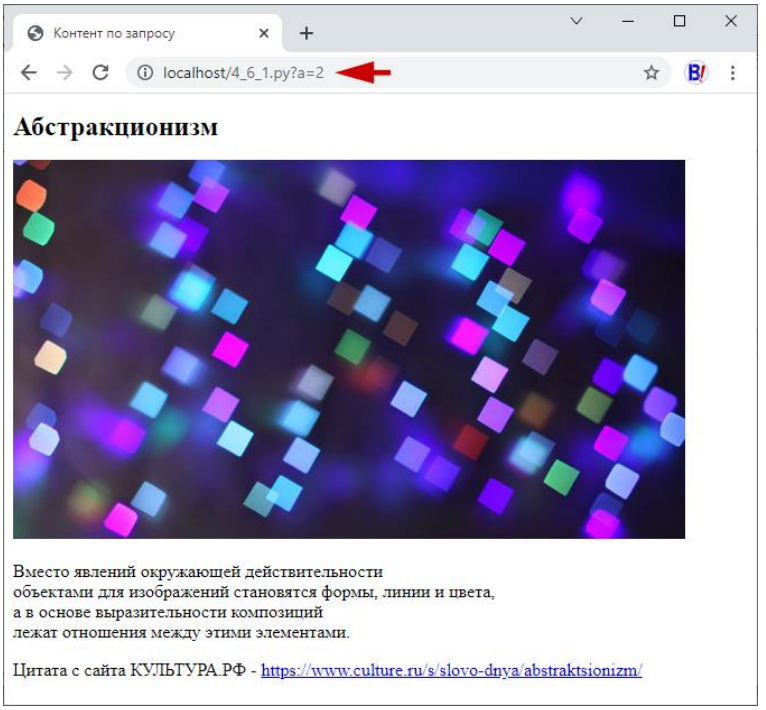

Рис. 4.6.2. Загружен файл 2.txt

Печатаем страницу:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Контент по запросу</title>
<style>
b {font-size: 24px;}
</style>
</head>
<body>''')
```

Получаем данные из формы:

```
form = cgi.FieldStorage()
```

Проверяем, передано ли значение параметра:

```
if form:
```

Если да, извлекаем номер текстового файла:

```
abs = form.getfirst('a')
```

и считываем его, не забыв указать кодировку:

```
fo = open('content/' + abs + '.txt', encoding='utf-8')
ti = fo.read()
fo.close()
```

Печатаем HTML-код и текст из файла:

```
print(ti)
```

и добавляем закрывающие теги HTML-кода страницы:

```
print('''</body>
</html>''')
```

Завершая этот раздел, будет не лишним еще раз напомнить читателям про безопасность сайта и проверку данных из запросов. В нашей программе цифра из параметра **a** соответствовала номеру файла из папки **content**. Это нормальная практика при условии, что в программе Python предусмотрена проверка на существование файла с введенным именем и разрешено открытие файлов только из директории **content**.

4.7. Передача параметров в условия, циклы и функции

Передача через строку запроса параметров в условия, циклы и функции применяется для нескольких целей:

– изменения контента, заданного по умолчанию;

– вывода данных, которые зависят от значений в строке запроса;

– вычислений, оперирующих параметрами из запроса.

Проиллюстрируем эти рассуждения несколькими примерами.

Для начала запустим в браузере программу **4_7_1.py**. Вы увидите страницу, на которой вам предлагается выбрать из выпадающего списка, к какой возрастной категории вы относитесь (рис. 4.7.1). Оставьте изначально выбранный пункт «Меньше 21 года» и нажмите кнопку «Комментировать». Появится сообщение: «Увы. Ваш возраст не позволяет вам стать депутатом» (рис. 4.7.2). Выберите пункт

«Больше 21 года» и снова нажмите кнопку. Теперь программа сообщит вам, что в этом случае вы можете избираться в законодательные органы власти (рис. 4.7.3).

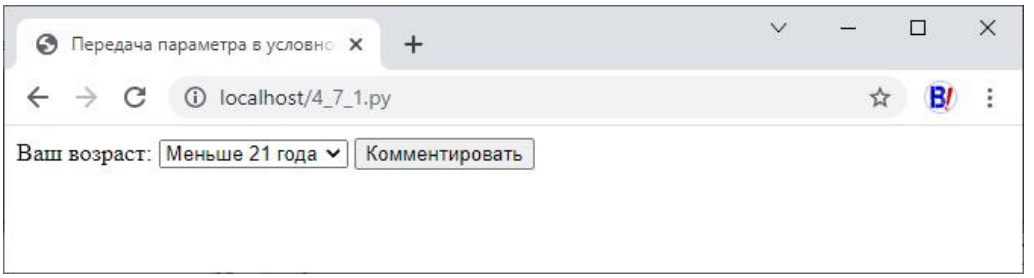

Рис. 4.7.1. Форма проверки возраста

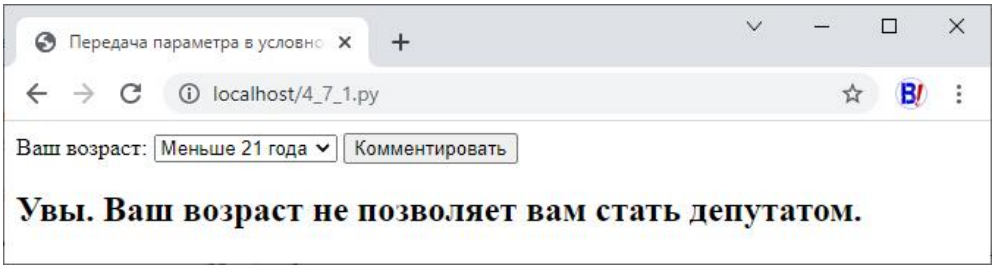

Рис. 4.7.2. Отрицательный ответ

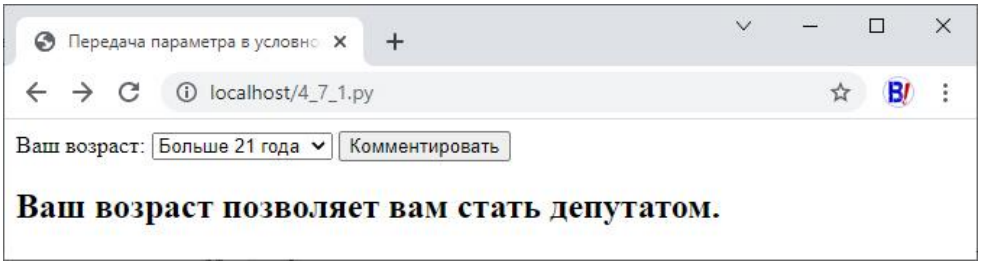

Рис. 4.7.3. Положительный ответ

Понятно, что текст сообщения зависит от значения параметра, который вы передали в условное выражение, записанное в программе. Посмотрим, как обрабатываются эти данные.

Для работы программы нам потребуется один модуль — **cgi**. После его импортирования сразу приступаем к выводу в браузер элементов страницы:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Передача параметра в условное выражение</title>
<style>
b {font-size: 24px;}
</style>
```

```
</head>
<body>
<form method="POST" action="4_7_1.py">
Ваш возраст: <select name="a">
<option selected value="1">Меньше 21 года</option>
<option value="2">Больше 21 года</option>
</select>
<input type="submit" value="Комментировать">
</form>''')
```

Здесь мы прерываем вывод, так как пришло время напечатать ответ посетителю. Для этого сначала получаем данные из формы:

```
form = cgi.FieldStorage()
```

и проверяем, что они не пустые:

```
if form:
```

Получаем значение из выпадающего списка:

```
age = int(form.getfirst('a'))
```

и передаем его в условное выражение:

```
if age == 1:
    print ('<br><b>Увы. Ваш возраст не позволяет
              вам стать депутатом.</b>')
  else:
    print ('<br><b>Ваш возраст позволяет вам
               стать депутатом.</b>')
```

Как вы убедились, здесь в качестве параметров условного выражения выступают значения, полученные из формы.

Следующий пример иллюстрирует использование значения параметра в цикле. Запустим файл **4_7_2.py**. Эта программа позволяет вычислить сумму чисел от 1 до 10 — на выбор посетителя (рис. 4.7.4).

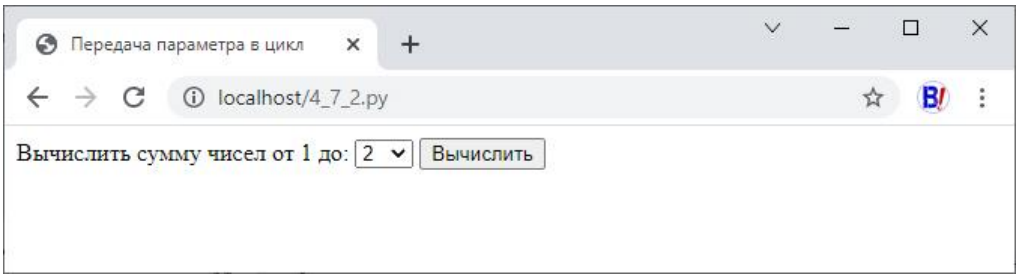

Рис. 4.7.4. Программа вычисления сумм чисел

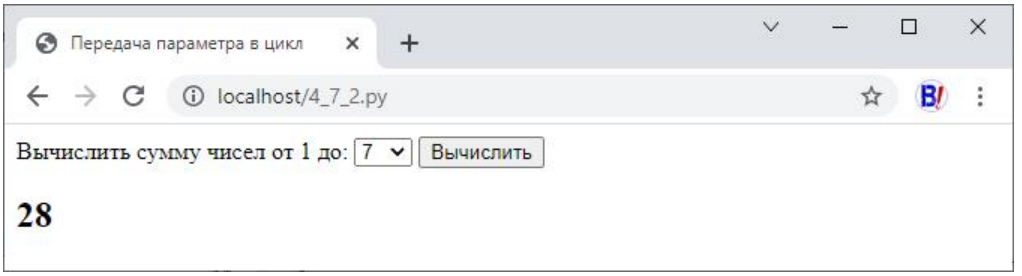

Рис. 4.7.5. Результат вычислений

Допустим, мы решили узнать сумму чисел от 1 до 7. Выберем цифру **7** в списке и нажмем кнопку «Вычислить». Страница перезагрузится и появится результат — 28 (рис. 4.7.5).

Откроем файл в редакторе и посмотрим, как он устроен.

Первым делом мы увидим, что данной программе нужен опять всего один модуль — **cgi**.

Объем HTML-кода здесь немного больше:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Передача параметра в цикл</title>
<style>
b {font-size: 24px;}
</style>
</head>
<body>
<form method="POST" action="4_7_2.py">
Вычислить сумму чисел от 1 до: <select name="s">
<option selected value="2">2</option>
<option value="3">3</option>
<option value="4">4</option>
<option value="5">5</option>
<option value="6">6</option>
<option value="7">7</option>
<option value="8">8</option>
<option value="9">9</option>
<option value="10">10</option>
</select>
<input type="submit" value="Вычислить">
</form>''')
```

Вслед за получением и проверкой данных

```
form = cgi.FieldStorage()

if form:
  sum = int(form.getfirst('s'))
```

создаем две переменные:

```
rs = 1
i = 0
```

Переменная **rs** необходима для сохранения результатов вычислений. Ей присвоено начальное значение **1**, так как суммирование ведется от единицы. **i** — счетчик проходов цикла. Кроме того, **i** имеет особое назначение, которое станет понятно при разборе операций, выполняемых в цикле.

А вот и сам цикл:

```
while i < sum - 1:
  rs += sum - i
  i += 1
```

Условием цикла является выражение

```
while i < sum - 1:
```

Это значит, что при значении параметра (сохраненного в переменной **sum**), равном **2,** в цикле будет один проход, при значении **3** — два прохода, при значении **4** — три прохода и т. д.

На каждом проходе происходит суммирование чисел — но в обратном порядке:

```
rs += sum - i
i += 1
```

Посмотрим этот порядок на примере значения **3** (то есть будет два прохода цикла). На первом проходе к исходному значению результатов вычисления прибавляется число, представляющее верхнюю границу диапазона минус начальное значение счетчика:

```
rs(1) += sum(3) – i(0)
```

Получаем 1 + (3 – 0) = 4. На втором проходе полученная сумма увеличивается на значение верхней границы минус единица (это значение счетчика на втором проходе):

```
rs(4) += sum(3) – i(1)
```

Получаем 4 + (3 – 1) = 6. Это конечный результат.

Если читатель продолжит данный алгоритм, то убедится, что он верен для любого верхнего предела.

После выхода из цикла нам остается напечатать результат:

```
print ('<br><b>' + str(rs) + '</b>')
```

и закрывающие теги страницы:

```
print('''</body>
</html>''')
```

Обратите внимание, что результатом работы цикла является число, а выводим мы на печать строку. Поэтому в процессе вывода происходит преобразование числа в строку:

```
str(rs)
```

Третья программа — **4_7_3.py** — показывает, как можно использовать значение параметра в функции.

В браузере она создает страницу, предназначенную для вычисления квадратов чисел от 1 до 10 (рис. 4.7.6). Как и в предыдущем примере, снова выберем цифру **7**, нажмем «Вычислить» и получим результат — **49** (рис. 4.7.7).

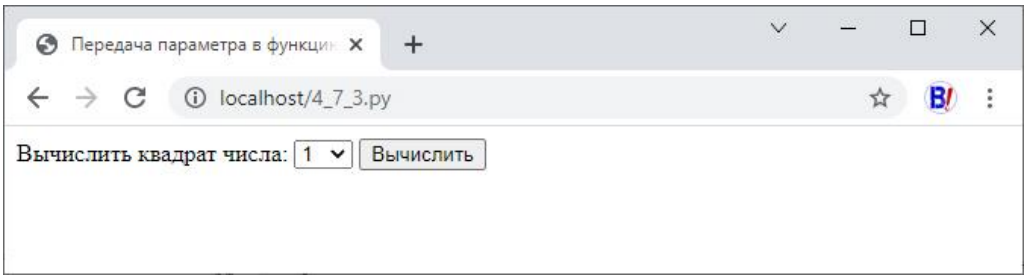

Рис. 4.7.6. Программа вычисления квадратов чисел

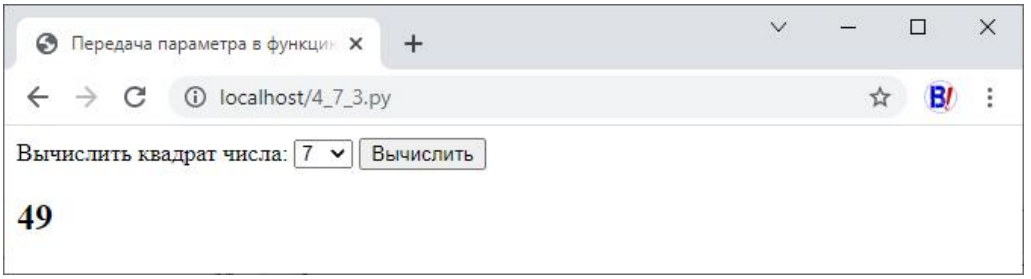

Рис. 4.7.7. Результат вычислений

Программа выглядит следующим образом.

Добавляем необходимый модуль

```
import cgi
```

и печатаем страницу:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
```

```
<head>
<meta charset="utf-8">
<title>Передача параметра в функцию</title>
<style>
b {font-size: 24px;}
</style>
</head>
<body>
<form method="POST" action="4_7_3.py">
Вычислить квадрат числа: <select name="s">
<option selected value="1">1</option>
<option value="2">2</option>
<option value="3">3</option>
<option value="4">4</option>
<option value="5">5</option>
<option value="6">6</option>
<option value="7">7</option>
<option value="8">8</option>
<option value="9">9</option>
<option value="10">10</option>
</select>
<input type="submit" value="Вычислить">
</form>''')
```

Получаем и проверяем значение из формы:

```
form = cgi.FieldStorage()
if form:
  sum = int(form.getfirst('s'))
```

Пишем функцию, которая вычисляет квадрат числа и возвращает строку с результатом:

```
def func(i):
  t = i ** 2
  return '<br><b>' + str(t) + '</b>'
```

Вызываем функцию **func**, передавая ей в качестве параметра значение из выпадающего (**sum**) списка, и печатаем результат:

```
print(func(sum) +
'''</body>
</html>''')
```

4.8. Установка cookie

Cookie — это текстовые файлы, которые тот или иной сайт создает на вашем компьютере. Cookie устанавливаются для конкретного ресурса и через конкретный браузер (то есть если на вашем компьютере несколько web-обозревателей, то cookie будут создаваться отдельно для каждого из них, в зависимости от того, через какой браузер вы станете посещать тот или иной ресурс). В таких файлах может храниться самая разнообразная информация:

– о прохождении регистрации или аутентификации пользователем;

– о его интересах и предпочтениях;
– об избранных страницах;
– о времени предыдущих посещений ресурса;
– о сделанных заказах в Интернет-магазинах;
– множество иных данных.

В Python имеется довольно простой «механизм» установки и чтения cookie. Рассмотрим его на конкретном примере. Для этого необходимо запустить файл **4_8_1.py**. Вы увидите страницу с выпадающим списком, в котором вам предлагается сделать выбор из двух операций: «Установить» и «Посмотреть» cookie (рис. 4.8.1). Если сразу выбрать пункт «Посмотреть», то на странице появится текст «Вы еще не установили cookie» (рис. 4.8.2).

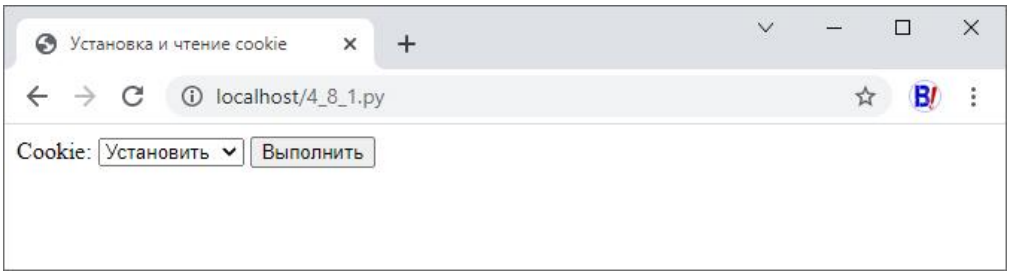

Рис. 4.8.1. Страница установки cookie

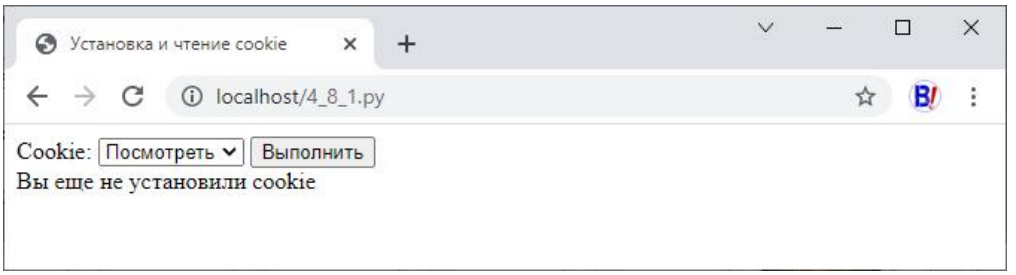

Рис. 4.8.2. Сообщение о том, что вы еще не установили cookie

Щелкнем кнопкой «Выполнить» при выбранном пункте «Установить». Ниже под формой появится сообщение «Установлено» (рис. 4.8.3). Снова жмем «Выполнить» и смотрим результат — «**Cookie: testcook=12345**» (рис. 4.8.4). Имя cookie **testcook** и значение **12345** выбраны произвольно.

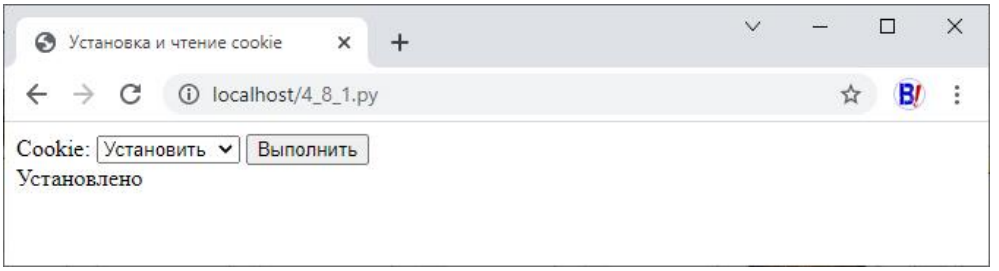

Рис. 4.8.3. Cookie установлен

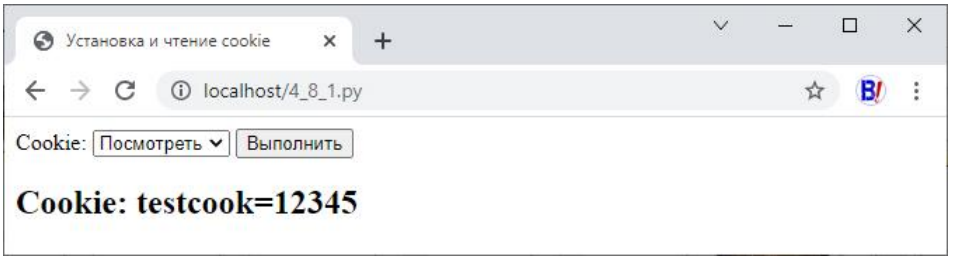

Рис. 4.8.4. Смотрим, какой cookie мы установили

Углубимся в код программы.

Нам потребуются два модуля:

```
import cgi
import os
```

Добавляем переменную **sc**, в которую станем помещать выводимую на страницу информацию. Так как в исходном состоянии никакого текста нет, то переменная пустая:

```
sc = ''
```

Выясняем, переданы ли данные из формы:

```
form = cgi.FieldStorage()
if form:
```

Получаем значение параметра — **1** (режим установки cookie) или **2** (режим чтения cookie), одновременно преобразуя строковый тип значения в числовой:

```
co = int(form.getfirst('c'))
```

Начинаем проверять значение. Если оно равно **1:**

```
if co == 1:
```

то производим установку cookie:

```
print('Set-cookie: testcook=12345')
```

и помещаем в переменную **sc** сообщение «Установлено»:

```
sc = 'Установлено'
```

Если передано 2, то получаем cookie, используя объект сопоставления **environ**, который нам предоставляет модуль **os**. В этом объекте данные состоят из пар «ключ — значение» (именно поэтому при чтении cookie мы получаем запись вида **testcook=12345**). При необходимости разделить ключ и значение —

сделать это не составит труда (будет либо операция получения значения по ключу, либо разбиения строки на две части по знаку равенства).

Итак, получаем cookie:

```
if co == 2:
  htc = os.environ.get("HTTP_COOKIE")
```

и проверяем, содержит ли переменная **htc** какие-либо данные или нет. Если переменная не пустая, значит, cookie уже установлено, и можно показать его имя и значение:

```
if htc:
  sc = '<br><b>Cookie: ' + htc + '</b>'
```

Если cookie не установлено, то переменная **htc** будет пустой, а значит, необходимо сообщить пользователю о том, что установка еще не была произведена:

```
else:
  sc = 'Вы еще не установили cookie'
```

В результате всех этих действий перед началом печати страницы переменная **sc** у нас либо пустая, либо содержит одно из трех сообщений:

– «Вы еще не установили cookie»;

– «Установлено»;

– «**Cookie: testcook=12345**».

Печатаем документ, внедрив в него значение переменной **sc**:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Установка и чтение cookie</title>
<style>
b {font-size: 24px;}
</style>
</head>
<body>
<form method="POST" action="4_8_1.py">
Cookie: <select name="c">
<option selected value="1">Установить</option>
<option value="2">Посмотреть</option>
</select>
<input type="submit" value="Выполнить">
</form>''' + sc +
'''</body>
</html>''')
```

Последний комментарий к нашей программе. Мы выбрали самый простой вариант установки cookie. Он будет существовать до тех пор, пока открыт браузер. После его закрытия этот cookie будет удален. Поэтому при повторном запуске web-обозревателя всю процедуру установки необходимо провести заново.

4.9. Бесконечная лента

Бесконечную ленту новостей или событий вы могли наблюдать в российских или зарубежных социальных сетях. В главной ленте новостей, в группах, в личных и корпоративных аккаунтах сообщения и фотографии добавляются постепенно, по мере прокрутки страницы вниз. Причем, количество дозагрузок ограничено только объемом базы данных, из которой черпается информация. В результате, постоянно смещая вниз движок полосы прокрутки, вы можете загрузить на страницу и 100, и 1000, и много больше сообщений.

В основе данного метода лежит технология Ajax. Она позволяет в фоновом режиме загружать на страницу данные с сервера. Для реализации такого подхода необходим JavaScript-сценарий, который формирует запросы на стороне клиента, и серверная программа, которая отправляет обратно требуемые данные.

Мы с вами напишем два простых файла — HTML-страницу с лентой рисунков и программу на Python, которая будет порциями отправлять очередные картинки на страницу.

Запустите в браузере файл ленты — **4_9_1.html**. В исходном состоянии на странице «прописаны» пять изображений (рис. 4.9.1). Если у вас очень большой экран, уменьшите размер окна браузера, чтобы получить требуемый эффект.

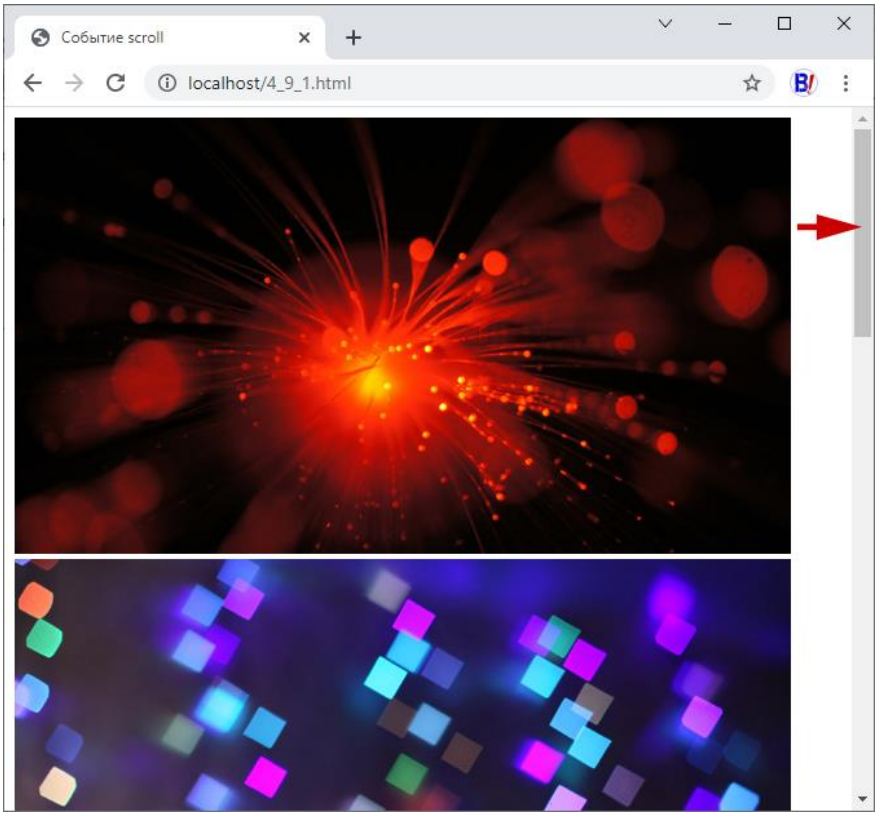

Рис. 4.9.1. Исходный вид страницы и ползунка на полосе прокрутки

Поэкспериментируем. Прокрутите страницу вниз до упора. Ползунок неожиданно укоротится и немного сместится вверх, а на странице появится несколько новых рисунков. Попробуйте опять прокрутить страницу вниз до упора. Эффект повторится: ползунок еще укоротится и вновь сместится вверх, а на странице появится очередная порция картинок (рис. 4.9.2). Так будет происходить до тех пор, пока вы не решите, что экспериментов достаточно. Это и есть наша версия бесконечной ленты новостей.

Тут необходимо сделать одну оговорку. Естественно, что автор не стал собирать несколько сотен изображений для максимально натуральной демонстрации эффекта бесконечной ленты. В папке **abs**, из которой «черпаются» эти изображения, всего 33 файла. Поэтому снимки довольно быстро закончатся. Но, согласитесь, этого вполне достаточно для наглядного примера. Тем более что непрерывно уменьшающийся размер ползунка подтверждает — картинки загружаются именно в процессе прокрутки страницы.

Кстати, на этом работа бесконечной ленты не прекратится. Вместо фотографий на странице начнут появляться иконки, замещающие отсутствующие изображения. То есть процесс действительно будет бесконечным.

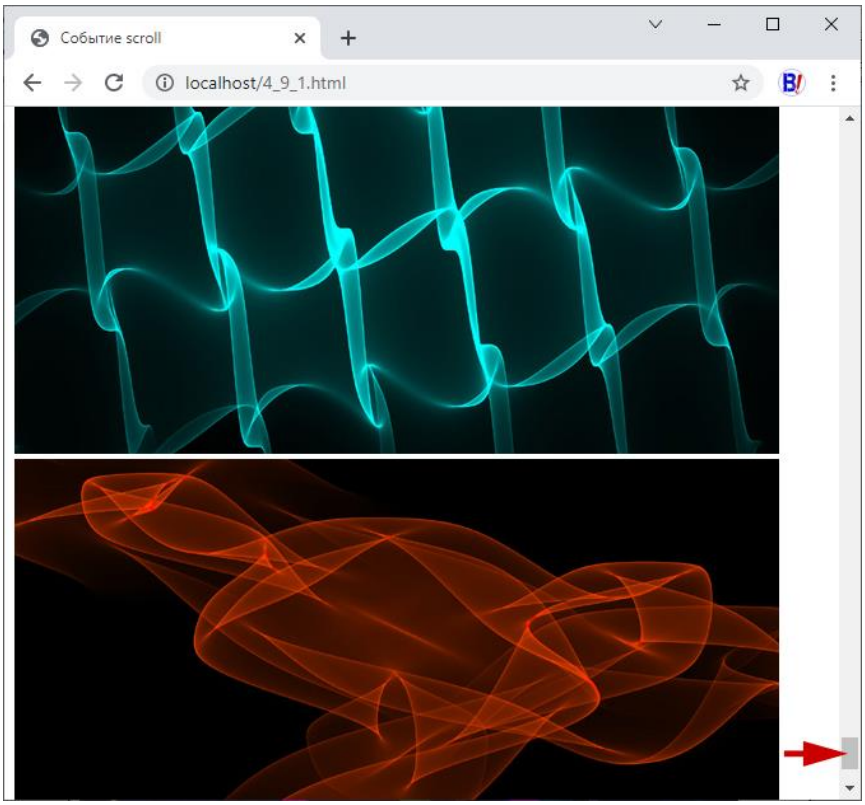

Рис. 4.9.2. Загружено уже больше 30 рисунков

Сценарий, который отправляет запросы серверной программе, мы не станем разбирать. При желании вы можете посмотреть JavaScript-код позже, открыв страницу **4_9_1.html** в текстовом редакторе Notepad++. Там все очень просто.

А вот программу отправки изображений с сервера — **4_9_1.py** — посмотрим сейчас. Она очень короткая и состоит всего из шести строк.

Начинаем традиционно:

```
#! C:/Python/python
import cgi
```

Затем получаем запрос от сценария:

```
form = cgi.FieldStorage()
st = form.getfirst('st')
```

Здесь в переменную **st** пишется порядковый номер затребованного фото. Это удобный подход, так как в качестве имен рисунков у нас применяются числа от **1** до **33**.

Сообщаем браузеру тип данных, передаваемых в ответе:

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

В последней инструкции программы формируем строку ответа:

```
print('<img src="abs/' + st + '.jpg" alt="Фото"><br>')
```

Именно этот код будет добавлен к разметке в конце HTML-страницы.

4.10. Поиск по файлам

Поиск по сайту — функция, применяющаяся на многих ресурсах. Она позволяет, например, искать те или иные термины в статьях или находить нужные товары в Интернет-магазинах.

Мы тоже напишем несложную систему поиска, которая для облегчения задачи будет искать слова по ограниченному количеству файлов — всего по двум. Здесь акцент сделан именно на простоте, чтобы программа легко читалась. Поэтому она будет иметь ограниченную функциональность и находить только стопроцентные совпадения. Например, если вы захотите найти в тексте слово «живопись», то результаты будут положительными лишь в том случае, когда в одном из файлов есть слово «живопись» в именительном падеже. Если в тексте есть слово «живописи» (то есть в родительном падеже), то поиск не даст результатов. Также надо отметить, что учитывается регистр символов. А это значит, что буква «а» не даст совпадения с «А».

В качестве страниц, содержащих тексты, у нас выступят уже знакомые вам файлы с рассказами об абстракционизме, находящиеся в папке **content**. Сама программа записана в файле **4_10_1.py**.

Запустим скрипт. Вы увидите приглашение ввести слово, поле для него и кнопку «Искать» (рис. 4.10.1). В обоих файлах из папки **content** почти нет одинаковых слов, а хочется показать читателям, что поиск может быть успешен одновременно и в одном, и в другом текстах. Поэтому, чтобы наш эксперимент был наглядным, автор выбрал в качестве искомого элемента часть слова — **тся**. Введем три этих буквы и нажмем кнопку.

Результат поиска показан на рисунке 4.10.2. Первым указан адрес файла, в котором нашлось совпадение, а дальше — строка, где есть искомое сочетание букв. Совпадающие фрагменты выделены более крупным жирным шрифтом.

Смотрим файл **4_10_1.py**.

В этот раз нам нужны два модуля:

```
import cgi
import os
```

После их импортирования начинаем печатать страницу:

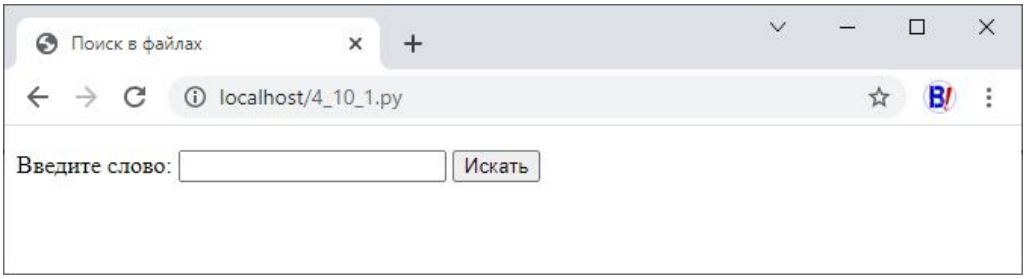

Рис. 4.10.1. Поисковая форма

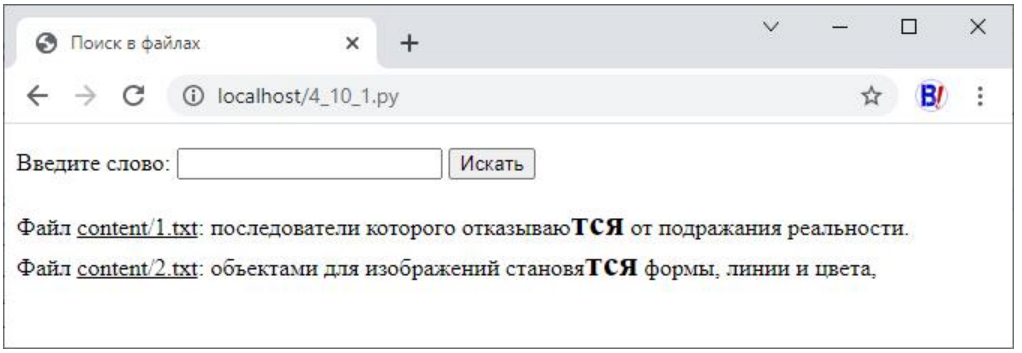

Рис. 4.10.2. Результат поиска

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Поиск в файлах</title>
<style>
b {font-size: 24px;}
</style>
</head>
```

```
<body>
<form method="POST" action="4_10_1.py">
<p>Введите слово: <input name="wo">
<input type="submit" value="Искать"></p>
</form>''')
```

Далее идет хорошо знакомый нам «механизм» получения данных из формы:

```
form = cgi.FieldStorage()
if form:
  wo = form.getfirst('wo')
```

Следом получаем список файлов в папке **content**:

```
opdir = os.listdir('content')
```

Запускаем цикл для последовательного просмотра файлов:

```
for x in range(1, len(opdir) + 1):
```

Так как имена файлов состоят из цифр и начинаются с единицы, то в диапазоне начальное значение отсчета — **1**.

Определяем количество элементов списка и добавляем к нему единицу:

```
len(opdir) + 1
```

из-за того, что конечное значение не входит в диапазон.

Открываем очередной файл:

```
fo = open('content/' + str(x) + '.txt',
                  encoding='utf-8')
```

одновременно преобразуя тип переменной **x** из числового в строковый:

```
str(x)
```

С помощью функции **readlines** считываем файл построчно:

```
ti = fo.readlines()
```

Закрываем файл:

```
fo.close()
```

Итак, у нас есть список **ti** со строками, расположенными по порядку их следования. Теперь запускаем следующий цикл — уже для просмотра каждой строки в отдельности:

```
for i in range(0, len(ti)):
```

Здесь в диапазоне начальная точка отсчета **0**, а конечная (напоминаю, не входящая в диапазон) равна количеству строк в списке:

```
len(ti)
```

С помощью метода **find** проверяем, не содержится ли в строке искомое слово:

```
if ti[i].find(wo) != -1:
```

Если оно обнаружено, создаем переменную **newwo**, в которую помещаем заменяющий фрагмент — искомое слово (оно сохранено в переменной **wo**), обрамленное тегами **b**:

```
newwo = '<b>'+ wo +'</b>'
```

Подчеркиваю: это мы создали именно замещающий объект, который вставим вместо найденного с помощью метода **replace**:

```
newti = ti[i].replace(wo, newwo)
```

Завершающий шаг вложенного цикла — печать результатов поиска:

```
print('Файл <u>content/' + str(x) + '.txt</u>: ' +
                          newti)
```

Последние строки программы выводят закрывающие теги HTML-страницы:

```
print('''</body>
</html>''')
```

4.11. Подсветка ссылок

Если вам приходилось пользоваться социальными сетями, то вы помните, как там в сообщение добавляется ссылка на какой-либо ресурс. Вы вставляете ее в текст, но до тех пор, пока новость не отправлена в ленту, ссылка представляет собой просто набор символов. Зато когда новость опубликована, ссылка становится рабочей и вы видите ее как выделенную цветом, подчеркиванием или еще как-то. Такая операция на жаргоне программистов называется подсвечиванием ссылки.

Вообще, подсветить ссылку — задача не совсем простая. И вот по какой причине. Хорошо, если ссылка отделена пробелами от других слов или знаков загружаемого текста. Тогда «подсветить» ее довольно просто. Но она может быть расположена в конце предложения, допустим, вот так: «**Вы были на сайте https://mysite.ru?**». Как узнать, что знак вопроса уже не является элементом ссылки? Или так: «**Есть много ресурсов с примерами кода (один из них сайт https://mysite.ru).**». Как определить, что скобка и точка не относятся к ссылке? Более того, по ошибке ссылка может вообще не иметь пробелов между сосед-

ними словами и выглядеть так: «**В тексте есть вот такаяhttps://mysite.ruссылка**». Подобное нередко случается при наборе текста со смартфона. Как в таком случае отделить ссылку?

Автор предлагает вам рассмотреть небольшую программу, рассчитанную на выявление ссылок вида **https://mysite.ru**, как бы они ни были замаскированы в тексте. Обращаю ваше внимание: программа корректно работает только с простыми ссылками! Вот такую строку **https://mysite.ru/content/cont.py?arg=no&reg=yes&time=1** она уже не распознает. Для ее подсвечивания нужны более сложные программы.

Итак, у нас есть программа, которая находит ссылки, начинающиеся с **https://** и заканчивающиеся доменным именем первого уровня **.ru**. Она содержится в файле **4_11_1.py**. Запустим его. Мы увидим поле с текстом, в который без пробелов встроена ссылка на поисковый сервис Yandex (рис. 4.11.1). Ниже поля — кнопка «Проверить». Нажмем ее. Результат показан на рисунке 4.11.2. Как видите, программа распознала ссылку. Если навести на нее указатель мыши, в нижней левой части браузера мы увидим ее адрес — он правильный. А если кликнуть на ссылке, то мы попадем на главную страницу ресурса **yandex.ru**. Кстати, в порядке эксперимента вы можете самостоятельно добавить еще текст и ссылки для проверки скрипта.

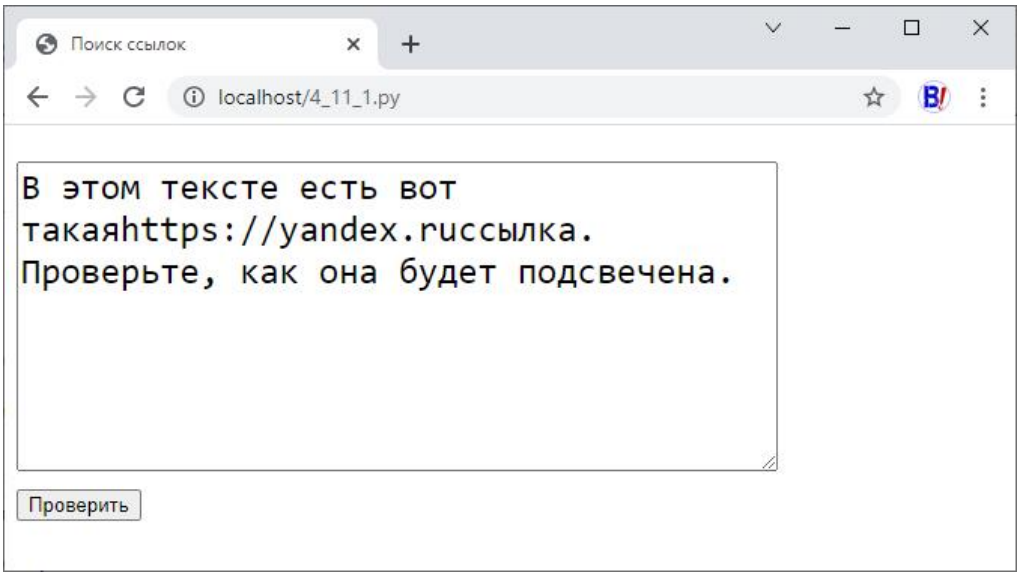

Рис. 4.11.1. Исходный текст в поле формы

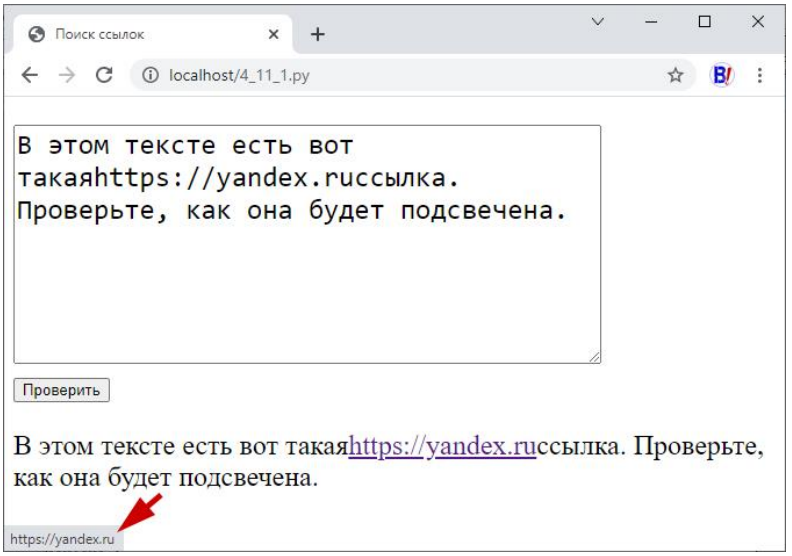

Рис. 4.11.2. Подсвеченная ссылка

Перейдем к разбору кода.

Программе для работы необходим один модуль **cgi**.

На первом этапе мы выводим в браузер часть страницы до закрывающего тега **</form>**:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Поиск ссылок</title>
<style>
textarea {width: 500px; height: 200px;
                  font-size: 24px;}
p {font-size: 24px;}
</style>
</head>
<body>
<form method="POST" action="4_11_1.py">
<p><textarea name="ta">В этом тексте есть вот
            такаяhttps://yandex.ruссылка.
Проверьте, как она будет подсвечена.</textarea><br>
<input type="submit" value="Проверить"></p>
</form>''')
```

Затем получаем и проверяем данные из формы:

```
form = cgi.FieldStorage()
if form:
  ta = form.getfirst('ta')
```

С помощью метода **split** разделяем полученный текст на отдельные слова (разделителем при отсутствии символов в скобках служит пробел):

```
wo=ta.split()
```

Запускаем цикл, который перебирает по очереди все получившиеся слова:

```
for zz in range(0, len(wo)):
```

Используя метод **find**, смотрим, есть ли в очередном слове символы **https://**:

```
if wo[zz].find('https://') >= 0:
```

Если они нашлись, разбиваем это слово на два фрагмента, используя в качестве разделителя символы протокола:

```
spwo=wo[zz].split('https://')
```

Дальше ищем во втором фрагменте доменное имя **.ru**:

```
if spwo[1].find('.ru') >= 0:
```

Если оно есть, в свою очередь делим второй фрагмент на два по разделителю **.ru**:

```
newspwo = spwo[1].split('.ru')
```

Первый фрагмент, получившийся в результате последней операции, будет являться частью ссылки без имени протокола и без доменного имени. Теперь мы может собрать ссылку в ее «натуральном» виде:

```
lin = 'https://' + newspwo[0] + '.ru'
```

Следующий шаг — создание ссылки, замещающей ту, что встроена в текст:

```
newlin = '<a href="' + lin + '" target="_blank">' +
                       lin + '</a>'
```

Теперь можно произвести замену текста ссылки на саму ссылку:

```
ta = ta.replace(lin, newlin)
```

Перебрав все слова и выявив все ссылки, можно напечатать получившийся текст:

```
print('<p>' + ta + '</p>')
```

и допечатать «остатки» страницы:

```
print('''</body>
</html>''')
```

Как видите, программа у нас получилась простой и короткой.

4.12. Работа с изображениями

Разные языки программирования имеют специальные функции и библиотеки для работы с изображениями. Не остался в стороне от этих тенденций и Python. Разработчикам доступна замечательная библиотека (или пакет — кому как удобнее называть) Pillow или PIL (Python Imaging Library). Мы познакомимся с ней на примере определения размеров картинок, а также научимся менять эти размеры, что является весьма распространенной практикой в web-программировании. Типичный пример — клиент закачивает на сервер слишком большое изображение. Задача разработчика — предусмотреть такую ситуацию и программными методами уменьшить рисунок до приемлемого размера.

В стандартный дистрибутив Python для Windows библиотека Pillow не входит. Ее необходимо установить отдельно. Чем мы сейчас и займемся.

Делается все очень просто. Откройте приложение «Командная строка» от имени администратора. Мы уже пользовались приложением в разделе 2.3, когда устанавливали сервер Apache. Напомню: для этого нажмите кнопку «Пуск», в меню выберите «Служебные — Windows», найдите пункт «Командная строка» и щелкните на нем правой (правой!) кнопкой мыши. В выпадающем списке выберите «Дополнительно», а затем «Запуск от имени администратора» (рис. 4.12.1). Откроется окно программы.

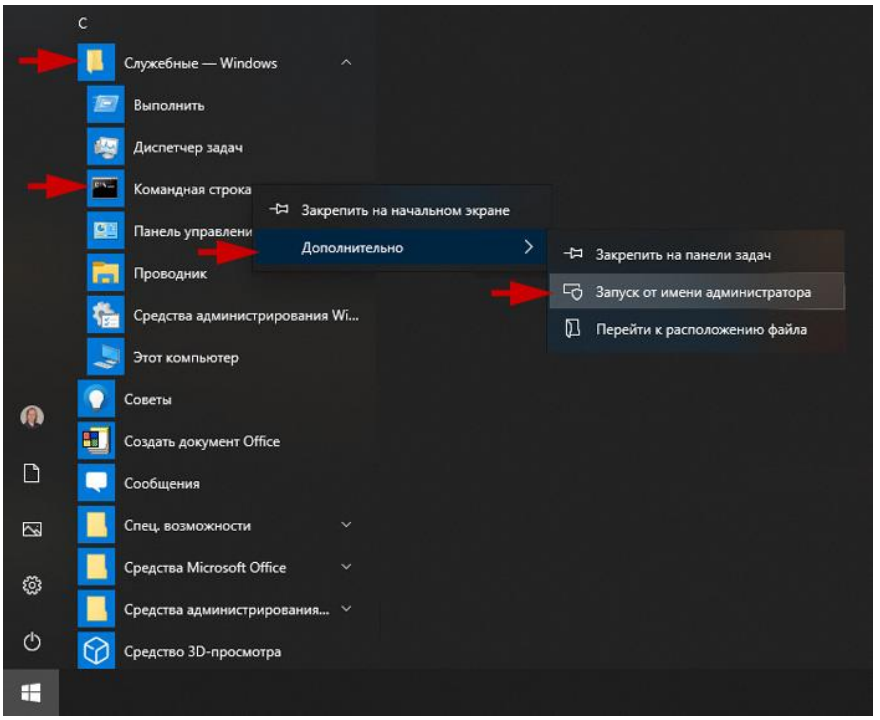

Рис. 4.12.1. Запуск командной строки

Теперь надо инсталлировать пакет. Для этого в окне командной строки сразу после

```
C:\WINDOWS\System32>
```

наберите

```
py -m pip install pillow
```

Должно получиться (рис. 4.12.2)

```
C:\WINDOWS\System32>py -m pip install pillow
```

Нажмите «Enter». Нужно будет немного подождать. Когда процесс инсталляции закончится (рис. 4.12.3), в окне программы вновь появится строка

```
C:\WINDOWS\System32>
```

Установка завершена. Закройте окно командной строки и перезагрузите компьютер.

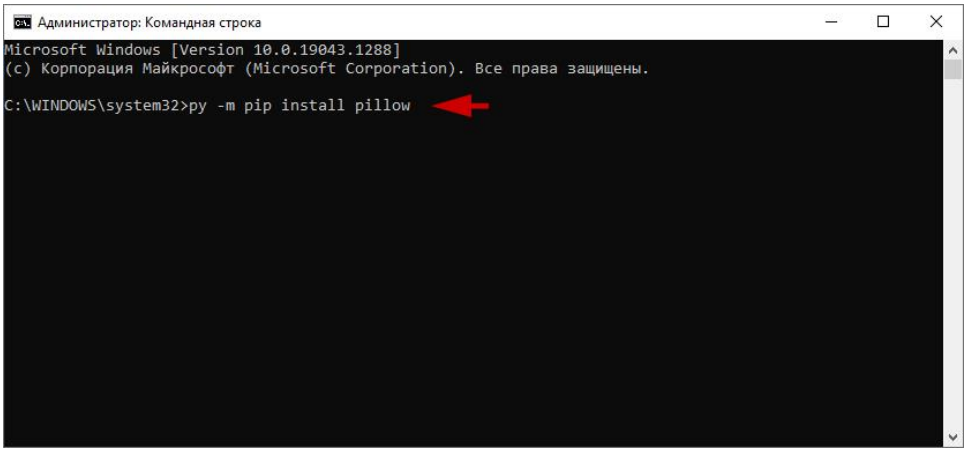

Рис. 4.12.2. Установка библиотеки Pillow

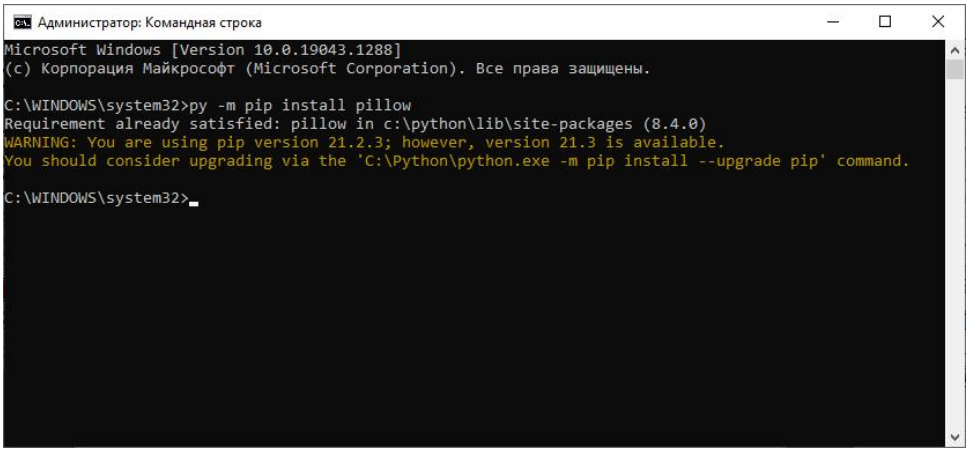

Рис. 4.12.3. Установка библиотеки Pillow завершена

Теперь можно приступать к экспериментам с библиотекой.

Первая программа, которую мы напишем, будет проверять размеры изображений, находящихся в папке **abs**.

Запустите в браузере файл **4_12_1.py**. Вы увидите страницу, на которой в левом верхнем углу есть выпадающий список рисунков из папки **abs** и кнопка «Узнать размер» (рис. 4.12.4). Выберите из списка какую-то картинку и нажмите кнопку. Страница перезагрузится, и на ней появится искомый рисунок с указанием его размера в пикселях (рис. 4.12.5).

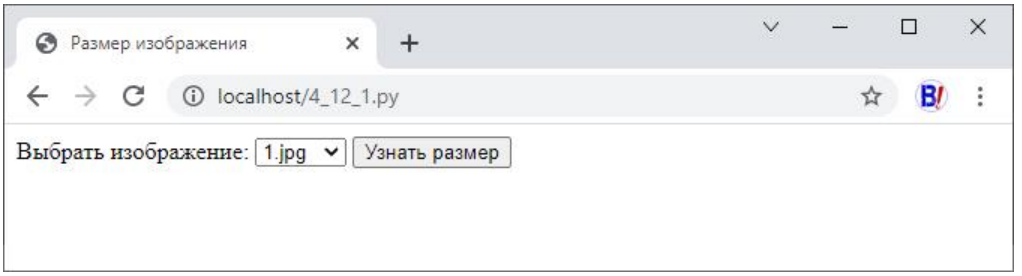

Рис. 4.12.4. Страница выбора изображения

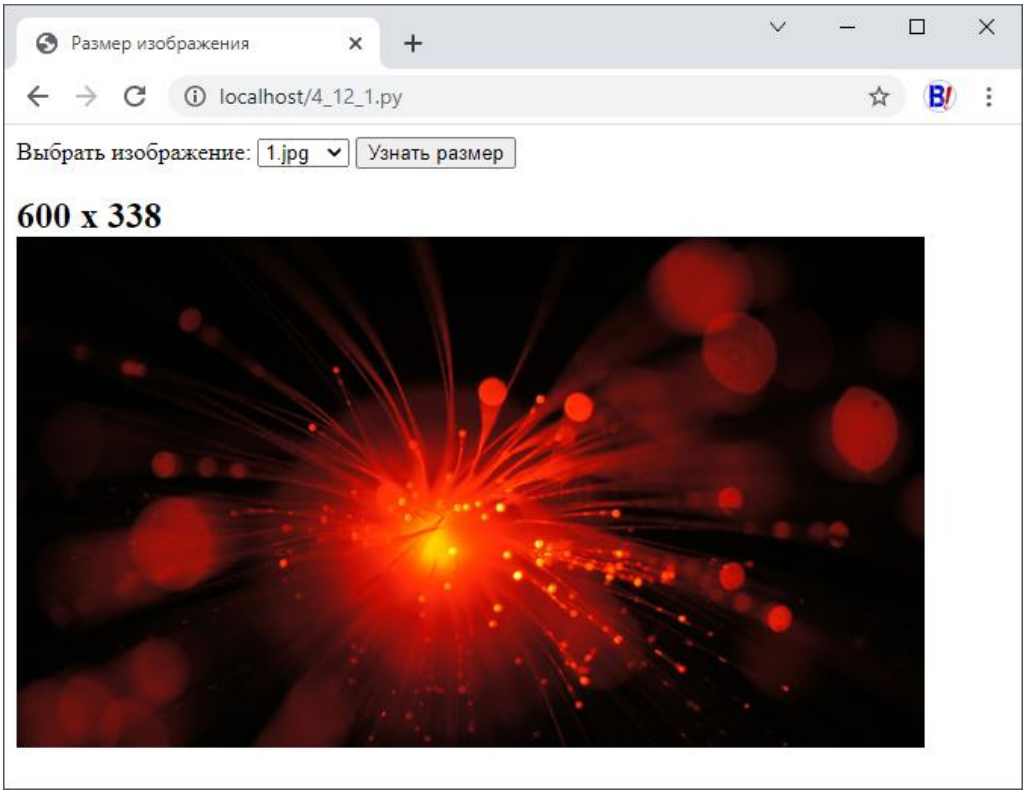

Рис. 4.12.5. Рисунок и его размер

Теперь посмотрим, как устроена программа.

Для ее работы нужны три модуля — **cgi**, **os** и модуль **Image** пакета Pillow:

```
import cgi
import os
from PIL import Image
```

Начинаем печатать страницу:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Размер изображения</title>
<style>
b {font-size: 24px;}
</style>
</head>
<body>
<form method="POST" action="4_12_1.py">
Выбрать изображение: <select name="im">''')
```

Чтобы заполнить список **select** именами рисунков, воспользуемся функцией **listdir** из модуля **os**.

Прочитаем содержимое папки **abs**:

```
opdir = os.listdir('abs')
```

и запустим цикл печати тегов **option**. Для этого введем счетчик проходов:

```
i = 0
```

Цикл выполняет работу, пока истинно условие:

```
while i < len(opdir):
```

Самый первый файл из списка должен попасть в разряд выбранных по умолчанию. Поэтому вводим условие

```
if i == 0:
```

при выполнении которого тег **option** печатается с атрибутом **selected**:

```
print('<option selected value="' + opdir[i] + '">' +
               opdir[i] + '</option>')
```

Во всех остальных проходах печатается «обычный» тег **option**:

```
else:
  print('<option value="' + opdir[i] + '">' +
               opdir[i] + '</option>')
```

В конце каждого прохода увеличиваем счетчик на **1**:

```
i += 1
```

Сформировав выпадающий список, завершаем печать формы:

```
print('''</select>
<input type="submit" value="Узнать размер">
</form>''')
```

После нажатия кнопки «Узнать размер» получаем данные из формы:

```
form = cgi.FieldStorage()
```

и проверяем, существуют ли они:

```
if form:
  adr = form.getfirst('im')
```

Теперь приступаем к самому главному — определению размера выбранного изображения. Получаем его имя:

```
ima = Image.open('abs/' + adr)
```

и считываем значения свойства **size**:

```
wi, he = ima.size
```

Так как полученные значения относятся к числовому типу, преобразуем их в строковые (в процессе создания строки с указанием размеров картинки):

```
res = '<br><b>' + str(wi) + ' x ' + str(he) +
                        '</b><br>'
```

Теперь можно выдать на страницу размеры и сам рисунок:

```
print(res + '<img src="abs/' + adr + '" alt="Фото">')
```

Нам осталось только завершить формирование страницы:

```
print('''</body>
</html>''')
```

Вторая программа, которую мы напишем, будет изменять размеры изображений, находящихся в папке **abs** и помещать уменьшенные рисунки в папку **pict**.

Файл второй программы — **4_12_2.py**. Запустите его в браузере. Вы увидите страницу, очень похожую на ту, что была в предыдущем случае. Отличается

только кнопка — на ней надпись «Изменить размер» (рис. 4.12.6). Выберите из списка какой-то рисунок и нажмите кнопку. Страница перезагрузится, и на ней появится уменьшенная в четыре раза картинка с указанием ее нового размера в пикселях (рис. 4.12.7). Вы также можете убедиться, что новое изображение оказалось в папке **pict**, открыв данный каталог.

Для этой программы снова нужны три модуля:

```
import cgi
import os
from PIL import Image
```

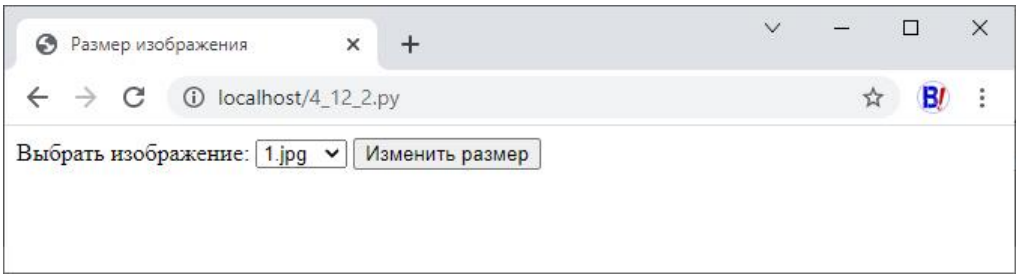

Рис. 4.12.6. Страница выбора изображения

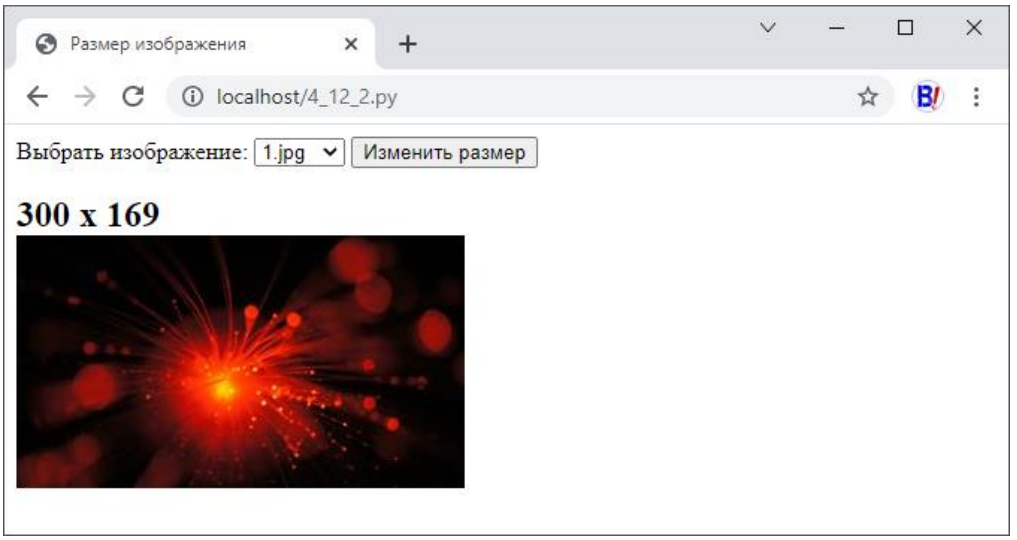

Рис. 4.12.7. Новый рисунок и его размер

Следующий блок кода полностью совпадает с аналогичным блоком из первого примера — вплоть до момента завершения формирования выпадающего списка. Пропустим эту часть.

Теперь добавляем новую кнопку:

```
print('''</select>
<input type="submit" value="Узнать размер">
</form>''')
```

Получаем данные с именем запрошенного изображения:

```
form = cgi.FieldStorage()
if form:
  adr = form.getfirst('im')
```

Узнаем размер этого рисунка:

```
ima = Image.open('abs/' + adr)
wi, he = ima.size
```

Вычисляем, какие нужно задать размеры у картинки, чтобы ее ширина и высота уменьшилась в два раза:

```
newwi = int(wi/2)
newhe = int(he/2)
```

Меняем размер изображения, используя метод **resize**:

```
newima = ima.resize((newwi, newhe))
```

Сохраняем новое изображение в папке **pict**:

```
newima.save('pict/' + adr)
```

Приводим новые размеры к строковому типу данных:

```
res = '<br><b>' + str(newwi) + ' x ' + str(newhe) +
                      '</b><br>'
```

Выводим на страницу уменьшенный рисунок и его размеры:

```
print(res + '<img src="pict/' + adr + '" alt="Фото">')
```

Завершаем печать страницы:

```
print('''</body>
</html>''')
```

Как видите, получение и уменьшение размеров изображений с помощью библиотеки Pillow — довольно простая задача.

5. Пишем сайт

В этой главе мы напишем простой web-сайт. Его компоненты — папки и файлы — собраны в папке «Глава 5» zip-архива. Чтобы привести сайт в рабочее состояние, выполните следующие действия:

– удалите из папки **C:\Apache24\htdocs** все файлы;

– скопируйте все содержимое папки «Глава 5» zip-архива;

– поместите скопированные папки и файлы в папку **C:\Apache24\htdocs**;

– чтобы запустить сайт в браузере, в строке адреса введите **http://localhost/** (откроется главная страница сайта).

5.1. Структура сайта

Думаю, не имеет смысла долго объяснять, что задача проекта — не впечатлить читателей навороченным дизайном и сложным контентом, а рассказать, как можно написать какой-либо ресурс на Python. Поэтому автор решил сделать максимально простой сайт с максимально простой структурой и максимально простым дизайном. Каждая страница ресурса, опять же для простоты, имеет фиксированную ширину (чтобы не усложнять дело адаптивной верстkой) и включает четыре компонента (тоже фиксированной ширины):

– заставку с эмблемой (имеет не только фиксированную ширину, но и высоту);

– блок меню (имеет фиксированную ширину и высоту) с элементами навигации по страницам ресурса;

– блок новостей — имитирует список последних событий;

– контейнер для основного содержания.

Рис. 5.1.1. Структура сайта

Также необходимо напомнить, что тема ресурса — реклама студии звукозаписи (не существующей в реальности).

Всего сделано 6 страниц. Пять из них посвящены музыкальным инструментам, которыми оснащена студия. Шестая содержит форму для отправки заявок на услуги звукозаписи. Страница «Синтезаторы» является одновременно главной.

Компоновка страницы сайта показана на рисунке 5.1.1.

Главную страницу сайта можно видеть на рисунке 5.1.2.

Рис. 5.1.2. Главная страница сайта

Нескольких комментариев к рисунку 5.1.2.

1. Все картинки и фотографии, использованные в качестве элементов дизайна, а также в качестве контента, взяты с сайта бесплатных изображений. Данный сайт разрешает использовать эти изображения без указания авторства — в любых целях, в том числе коммерческих.

2. Текст страниц написан автором на основе данных из нескольких источников и носит произвольный характер. Все марки производителей музыкальных инструментов — выдуманные.

3. Блок новостей носит чисто изобразительную нагрузку. Страницы анонсированных событий, как и ссылки на них, не существуют.

На каждой странице, кроме «Контакты», в области контента есть по два изображения. Посетитель может увеличить любое из них, щелкнув мышью по картинке. При этом страница будет закрыта белым полупрозрачным слоем с увеличенным изображением. Если теперь кликнуть на нем или прокрутить страницу, увеличенная картинка и полупрозрачный слой исчезнут. Управляет данным процессом сценарий, написанный на JavaScript. Его мы разберем позже, в главе 6.

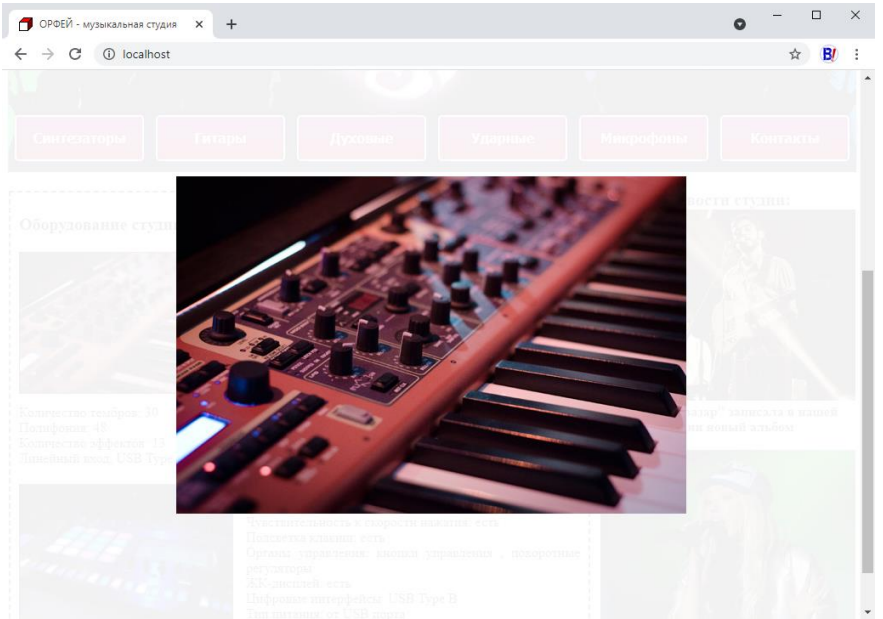

Рис. 5.1.3. Увеличенное изображение на странице

5.2. Компоненты

Зададимся вопросом: что хочет получить студия звукозаписи от сайта? Две вещи:

– рекламу своих услуг и возможностей;

– заявки от потенциальных клиентов.

Исходя из этого, выходит, что сайт должен иметь страницы, представляющие техническое оборудование студии, и страницу, где клиент может оформить и отправить заявку. В свою очередь, наличие второго пункта подразумевает, что администратор должен иметь возможность получать и фиксировать заявки.

Автор выбрал систему сбора заявок, основанную не на отправке их по электронной почте, а на записи в файл, который можно легко прочитать через браузер и перенести данные из файла на свой компьютер. Так у читателя появляется больше возможностей наблюдать за работой Python. К тому же это вполне равноценная замена. Судите сами. В обоих случаях необходим запуск на компьютере соответствующей программы — почтового клиента или web-обозревателя, а также копирование данных из письма или со страницы с заявками. Вариант с файлом даже удобнее: не надо по очереди открывать каждое письмо. Открыл файл и сразу перенес на свой ПК все данные. Раз у нас будет файл, значит, администратор должен иметь возможность удалять из него уже скопированные заявки.

Чтобы реализовать все запланированные цели, нам потребуется следующий набор файлов Python:

– шаблон сайта, в который будут загружаться данные из разных страниц;

– программа записи заявок;

– страница просмотра заявок, доступная только администратору;

– программа удаления заявок, запустить которую может только администратор.

Кроме того, понадобится еще целый ряд компонентов, которые будут содержать все остальные элементы, необходимые для функционирования сайта.

Посмотрим, что есть в zip-архиве для решения поставленных задач:

– файл **index.py** — шаблон сайта;

– файл **rec.py** — программа записи заявок;

– файл **admin.py** — страница просмотра заявок;

– файл **del.py** — программа удаления заявок

– 4 папки — **img**, **studio**, **content** и **set**.

Папка **img** содержит два файла:

– **muz.jpg** — заставка сайта;

– **net.jpg** — его назначение выяснится в разделе 6.1.

В папке **studio** собраны фотографии, использованные в оформлении страниц.

В папке **content** 6 текстовых файлов. Пять из них содержат контент основных страниц, а в файле **contact.txt** записан HTML-код формы для заявок.

При создании проекта не использовалась база данных. Как видите, вся информация, размещенная на сайте, хранится в простых текстовых файлах. Сделано это по двум причинам:

– сайту нечего скрывать, так как никакой секретной информации он не содержит;

– так проект получается проще и доступнее для повторения.

Естественно, при желании вы можете переписать соответствующие фрагменты кода для взаимодействия с базами данных.

В папке **set** находятся:

– **index.css** — таблица стилей сайта;

– **admin.css** — таблица стилей страницы просмотра заявок;

– **index.js** — программа для просмотра на страницах сайта увеличенных фото;

– **cont.js** — программа проверки данных из формы и передачи на запись;

– **del.js** — программа передачи команды на удаление заявок;

– **title.txt** — файл с названием сайта, выводящимся в закладке страницы;

– **nav.txt** — HTML-код меню;

– **news.txt** — HTML-код блока новостей;

– **cont.py** — роль этого файла разъяснится в разделе 5.4.

Итак, часть файлов из архива образуют структуру сайта, а другая часть — страницу администратора. Сначала выясним, как взаимодействуют файлы, необходимые для работы сайта.

«Скелет» любой страницы — HTML-код с разметкой (рис. 5.2.1). Непосредственно в теле документа прописаны все его элементы, кроме:

– основного содержания;

– названия сайта, которое выводится на вкладке страницы браузера;

– меню;

– блока новостей;

– таблицы стилей;

– сценариев на JavaScript.

Созданием разметки на странице управляет «оболочка», написанная на Python.

Таблица стилей и сценарий просмотра фото добавляются в HTML-код в процессе загрузки документа, минуя «посредников». Название, меню, новостной блок и основное содержание вставляется с помощью Python-оболочки из соответствующих txt-файлов. В Python-коде предусмотрен механизм определения, какой именно контент должен быть загружен для той или иной страницы.

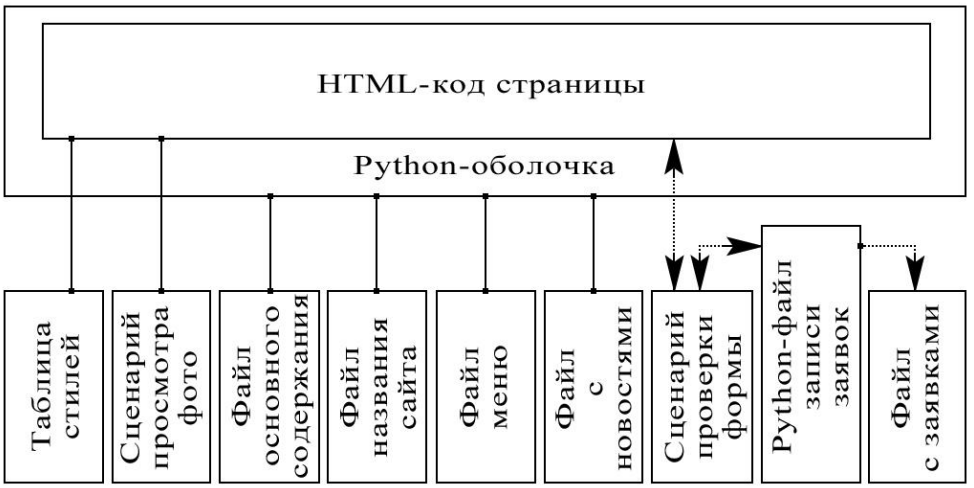

Рис. 5.2.1. Взаимодействие файлов сайта

У оболочки есть еще одна функция: проверять, соответствует ли адрес в строке запроса адресу реально существующей страницы. Если искомая страница отсутствует, то никаких предупреждений не выводится, а просто загружается главная страница.

Если загружена страница «Контакты», то к ней подключается файл **cont.js**. После отправки формы данные проверяются этим скриптом и отправляются на запись файлу **rec.py**. По ее завершении сценарию **cont.js** возвращается соответствующий сигнал, который преобразуется в текст подтверждения успешной записи, выводящийся на странице.

Схема взаимодействия файлов страницы заявок показана на рисунке 5.2.2.

Таблица стилей вставляется в HTML-код в процессе загрузки документа. Текст в тег **title** и заявки добавляются с помощью Python-оболочки из соответствующих файлов.

Если администратор запустил процесс удаления заявок, данные отправляются сценарию **del.js**, а затем программе **del.py**. Заявки удаляются, сценарий получает сигнал об этом и выводит на страницу необходимое сообщение.

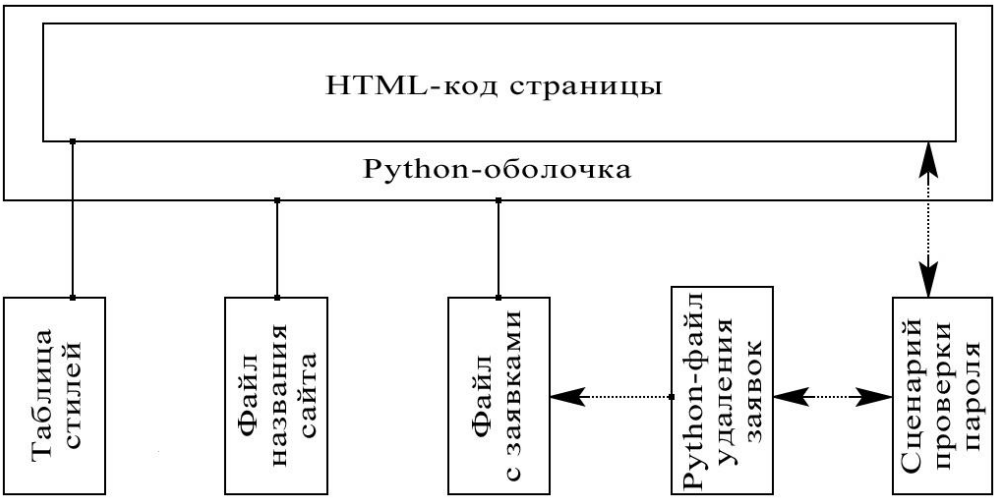

Рис. 5.2.2. Взаимодействие файлов страницы просмотра заявок

5.3. Файл index.py

Самый главный файл, собирающий сайт из отдельных компонентов — **index.py**.

Начинается он традиционно:

```
#! C:/Python/python
```

Следующий шаг — подключение модулей. Нам понадобятся модуль **cgi** — для получения параметра с адресом затребованной страницы и **os** — для запуска функции чтения каталога со страницами:

```
import cgi
import os
```

Если в строке адреса набрать **http://localhost/**, то должна открыться главная страница («Синтезаторы»). Кроме того, мы помним, что эта же страница загружается в случае ввода неверного адреса. Поскольку адрес главной страницы так важен, создадим для него отдельную переменную:

```
a = 'index'
```

Чтобы узнать, какую страницу надо выдать браузеру, получаем значение параметра **a** (не путать с одноименной переменной!):

```
form = cgi.FieldStorage()
b = form.getfirst('a')
```

Как видите, мы использовали класс **FieldStorage**, экземпляр которого содержит имена параметров (в качестве ключей) и значения из формы. Роль формы у нас выполняет параметр **a** со значением, переданным методом GET в запросе. Значение параметра мы считываем методом **getfirst** и присваиваем его переменной **b**. Эта переменная нужна для сокращения кода, так как дальше мы еще дважды будем использовать значение, полученное из параметра.

Теперь надо проверить, запросил ли посетитель существующую страницу. Если да — загрузить ее в браузер. Если нет — показать главную страницу.

Может возникнуть вопрос: зачем такая проверка, если у нас в меню — только ссылки на существующие страницы? Ответ: сделано это для любителей забраться куда не следует. Они могут ввести в качестве параметра запрос на открытие страницы, не предназначенной для посторонних. Но у нас на этот случай существует четкий ответ: загружается либо существующая страница, либо главная — вместо результатов некорректного запроса.

Для проверки напишем вот такой код:

```
if b:
  opdir = os.listdir('content')
  i = 0
  while i < len(opdir):
    if b + '.txt' == opdir[i]:
      a = b
      break
    i += 1
```

Он выполняется, если параметр **a** и, соответственно, переменная **b** существуют и не являются пустыми:

```
if b:
```

Начинаем процесс с выяснения имен файлов из папки **content**:

```
opdir = os.listdir('content')
```
затем создаем счетчик:

```
i = 0
```

и запускаем цикл, количество проходов которого равно количеству файлов в папке **content**:

```
while i < len(opdir):
  ...
  i += 1
```

Во время каждого прохода мы сравниваем имена файлов со значением из списка:

```
if b + ".txt" == opdir[i]:
```

Если значение параметра с добавленным к нему расширением **".txt"** совпадает с одним из имен файлов, переменной **a** присваивается новое значение — адрес загружаемой страницы:

```
a = b
```

Например, если запрос выглядит так — **http://localhost/index.py?a=guitar,** то загрузится файл **guitar.txt** и откроется страница «Гитары» (фрагмент открытия файла появится в нашем коде позже).

Естественно, что, найдя совпадение, мы можем завершить цикл досрочно:

```
break
```

Если в запросе параметр **a** не существует или пустой, изначально установленное значение переменной **a**

```
a = 'index'
```

не изменится и будет загружена главная страница (например, когда в строке адреса набрано **http://localhost/**). Если значение параметра **a** не совпадает ни с одним именем файла, произойдет то же самое.

Открытием и загрузкой файлов у нас «распоряжается» функция **opfi**, которая в качестве параметра получает адрес файла:

```
def opfi(adr):
  fo = open(adr, encoding='utf-8')
```

```
  ti = fo.read()
  fo.close()
  return ti
```

Работает она максимально просто. Открывает файл:

```
fo = open(adr, encoding='utf-8')
```

считывает его:

```
ti = fo.read()
```

закрывает:

```
fo.close()
```

и возвращает полученный текст:

```
return ti
```

Вывод страницы в браузер начинается с привычной строки:

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

Затем печатается первая часть документа:

```
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>''' + opfi('set/title.txt') + '''</title>
<link rel="stylesheet" href="set/index.css">
<script src="set/index.js"></script>''')
```

Как видите, в процессе вывода заголовочной части документа мы обращаемся к функции **opfi**, чтобы получить надпись дескриптора **title**:

```
...
<title>''' + opfi('set/title.txt') + '''</title>
...
```

Далее загружаем таблицу стилей:

```
<link rel="stylesheet" href="set/index.css">
```

и сценарий просмотра фото:

```
<script src="set/index.js"></script>
```

Если запрошена страница «Контакты», то в головную часть документа еще добавляется JavaScript-файл проверки данных:

```
if a == 'contact':
  print('<script src="set/cont.js"></script>')
```

Последний этап — печать оставшейся (отметим — большей) части страницы:

```
print('''</head>

<body>
<div id="bas"></div>
<div id="view">
<img id="pict" src="img/net.jpg" alt="Photo">
</div>

<div id="basis">

<table id="tab">
<tr><td id="menu" colspan="2">
<nav id="sect">''' + opfi('set/nav.txt') + '''</nav>
</td></tr>
<tr><td id="main">
<div id="cont">''' + opfi('content/'+a+'.txt') +
                    '''</div></td>
<td id="rec">''' + opfi('set/news.txt') +
                     '''</td></tr>
</table>

</div>
</body>
</html>''')
```

Блок

```
<body>
<div id="bas"></div>
<div id="view">
<img id="pict" src="img/net.jpg" alt="Photo">
</div>
```

необходим для демонстрации увеличенных изображений. Его мы разберем в разделе 6.1.

Для «прочности» структуры все элементы видимой части документа помещаются в ячейках таблицы:

```
<table id="tab">
<tr><td id="menu" colspan="2">
<nav id="sect">
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                    |
| Меню                 |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
</nav>
</td></tr>
```

```
<tr><td id="main">
<div id="cont">
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                     |
| Основное содержание         |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
</div></td>
<td id="rec">
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
|                     |
| Новости               |
|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|
</td></tr>
</table>
```

Верхняя строка таблицы имеет одну ячейку, нижняя разбита на 2:

– основное содержание (ширина — 700px);

– блок новостей (ширина — 300px).

Также обращаю внимание, что ячейка контента (**id="main"**) имеет внутри контейнер

```
<div id="cont">
...
</div>
```

Он нужен для размещения основного содержания.

Ссылки, которые необходимы для навигации по страницам, размещены в секции **nav**. HTML-код меню «выдает» функция **opfi**:

```
<nav id="sect">''' + opfi('set/nav.txt') + '''</nav>
```

Эта же функция загружает и основное содержимое:

```
<div id="cont">''' + opfi('content/'+a+'.txt') +
                     '''</div>
```

Здесь адрес файла, который необходимо открыть, формируется следующим образом:

```
'content/'+a+'.txt'
```

где **a** — переменная с именем страницы.

Наконец, блок новостей вставляется тоже благодаря функции **opfi**:

```
<td id="rec">''' + opfi('set/news.txt') + '''</td>
```

Затем печатаются закрывающие теги и все — страница готова.

5.4. Страница «Контакты»

Эта страница отличается только тем, что в нее загружается не текст с фотографиями, а форма для отправки заявок. И еще файл **cont.js**, о чем мы упоминали в предыдущем разделе.

Наполнение страницы «Контакты» представлено на рисунке 5.4.1. Форма имеет три поля — «Ваше имя», «Телефон или e-mail» и «Сообщение», а также кнопку «Отправить». Если какое-то поле оставить незаполненным, то появляется соответствующее предупреждение. Причем выводится оно креативным способом — прямо на кнопку «Отправить», заменяя ее изначальный текст (рис. 5.4.2).

Отправка заявки выполняется с помощью технологии Ajax. То есть в браузере остается та же страница. В это время сценарий из файла **cont.js** сначала проверяет данные, а потом отправляет запрос программе **rec.py**. Там данные проверяются еще раз, после чего происходит их запись в файл **cont.py** (объяснения этому факту — в разделе 5.5). На следующем этапе программа **rec.py** возвращает файлу **cont.js** отчет об успешной записи. На последнем шаге сценарий из **cont.js** выводит на страницу подтверждение о получении заявки (и опять на кнопку отправки данных — рис. 5.4.3).

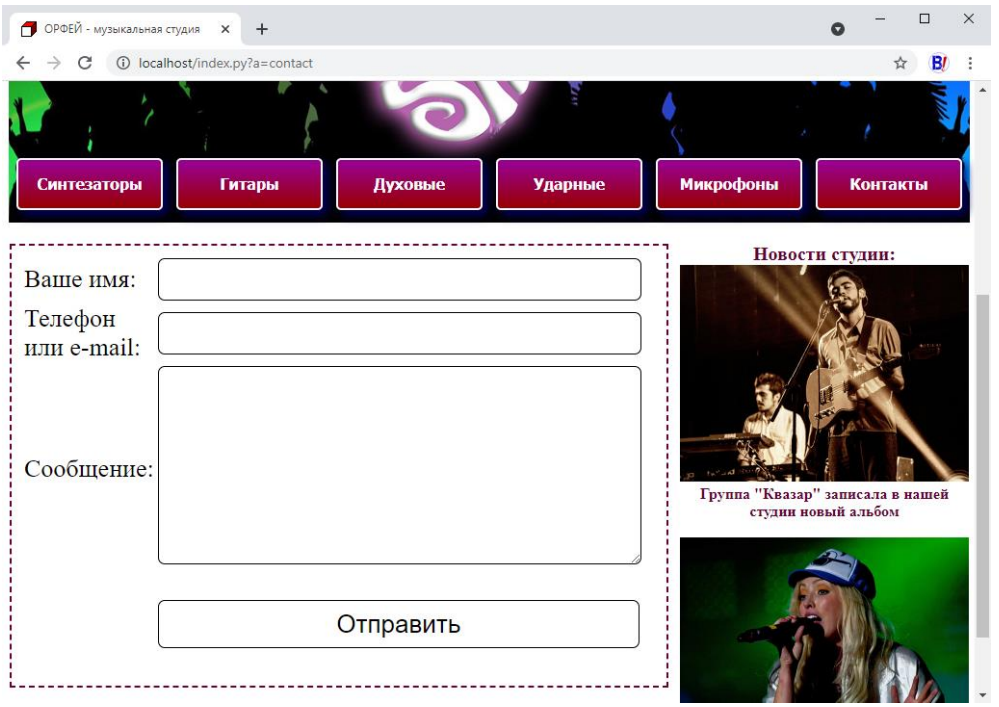

Рис. 5.4.1. Страница «Контакты»

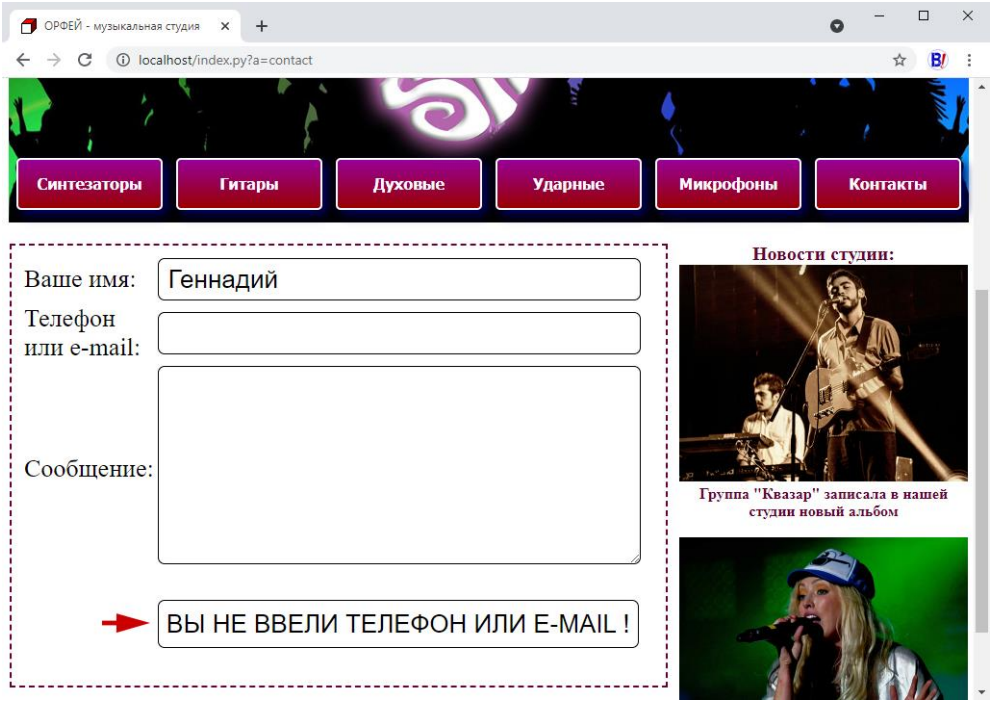

Рис. 5.4.2. Предупреждение, если заполнены не все поля

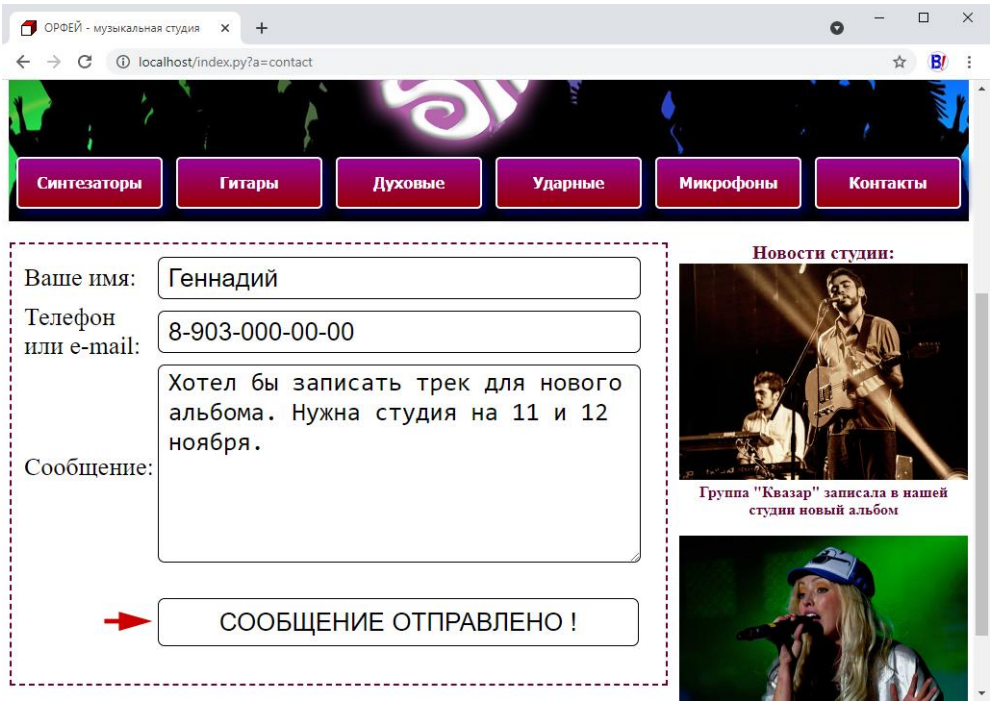

Рис. 5.4.3. Сообщение об успешной отправке заявки

5.5. Файл rec.py

Проверка данных на стороне клиента — это, конечно, хорошо, но совершенно недостаточно для безопасности вашего ресурса. Есть такая категория несознательных граждан, которые очень любят тестировать разные сайты на наличие в них брешей и уязвимостей.

Классический пример, описанный уже неоднократно: некто создает на своем компьютере страницу с такой же формой, как на вашем сайте, но без проверки данных. Затем помещает в форму вредоносное содержимое и отправляет его. Если серверная программа обработки данных не имеет защиты от такого содержимого — ждите проблем.

Поэтому в программу записи заявок добавлена повторная проверка данных — теперь на стороне сервера. Но впрочем, все по порядку.

Файл **rec.py** начинается стандартно:

```
#! C:/Python/python
```

Подключаем модуль **cgi**:

```
import cgi
```

Уже знакомым нам способом получаем значения параметров из формы:

```
form = cgi.FieldStorage()

nam = form.getfirst('nam')
con = form.getfirst('con')
tex = form.getfirst('tex')
```

Отправляем браузеру сообщение о типе загружаемого кода:

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

Теперь начинаем проверку данных.

Сначала смотрим, все ли поля заполнены:

```
if nam is None or con is None or tex is None:
  print(2)
  exit()
```

Если хотя бы одно поле оставлено пустым:

```
if nam is None or con is None or tex is None:
```

возвращаем сценарию в качестве ответа цифру **2:**

```
print(2)
```

и прерываем выполнение программы:

```
exit()
```

Теперь проверяем, соблюдены ли требования к минимальному количеству знаков в полях формы:

```
if len(nam) < 2 or len(con) < 8 or len(tex) < 10:
  print(2)
  exit()
```

Если есть «недовес» символов, то, как и в предыдущем случае, возвращаем сценарию в качестве ответа цифру **2** и останавливаем программу.

Третья проверка — на выполнение требований к максимальному количеству символов в полях:

```
if len(nam) > 50 or len(con) > 50 or len(tex) > 1000:
  print(3)
  exit()
```

Этот блок кода в случае нарушений возвращает цифру **3**.

Цифровые коды ответов будут расшифрованы в разделе 6.2.

А что получит недоброжелатель в ответ на попытку передать в программу некорректные данные, например, слишком маленького или слишком большого объема? Страницу с ничего ему не объясняющей цифрой **2** или **3**.

Нужно сказать, что и рассмотренная нами проверка на стороне сервера тоже не является исчерпывающей. Мы выяснили только заполнение полей и количество символов в них. Но не проверили, использовал ли посетитель разрешенные символы, не попытался ли загрузить вредоносный код. Не убедились, что клиент действительно указал адрес почты или телефона, а не ввел произвольный набор знаков. Впрочем, автор и не стремился загружать программу «обильной» проверкой. Моя задача была — продемонстрировать подход: файлы записи данных обязательно должны проверять входящую информацию.

Идем дальше. Если все проверки были успешными, пишем заявку в файл:

```
fi = open('set/cont.py', 'a', encoding='utf-8')
fi.write(nam + '\n' + con + '\n' + tex + '\n
-----------------------------------------------
                        ---\n\n')
fi.close()
print(1)
```

Мы используем режим «дописывания» в конец файла, чтобы не удалить заявки, пришедшие раньше. Длинный набор дефисов нужен, чтобы визуально отделить одну заявку от другой. В качестве ответа возвращаем сценарию цифру **1**.

Теперь пояснения к тому факту, что запись происходит в файл **cont.py**. Мы договорились не использовать базу данных для упрощения проекта и храним всю информацию в текстовых файлах, так как у нас нет секретов. Ведь эта ин-

формация загружается в ту или иную страницу и видна любому посетителю. Но есть одно исключение — файл для хранения заявок. К нему доступ может быть только у администратора. Значит, текстовый файл не подойдет, ведь посторонний человек может открыть его в браузере, введя соответствующий путь в строке адреса (выяснить его не составляет труда). Что же делать? Выход очень простой — писать заявки в файл с расширением **.py**. Так как в таком файле не будет обязательной строки с путем к интерпретатору, при попытке его загрузить сервер выдаст сообщение об ошибке. А значит «снаружи» прочитать этот файл не удастся. Что нам и требуется.

5.6. Файл admin.py

Файл **admin.py** создает страницу просмотра заявок.

Нам следует продумать, как сделать, чтобы заявки были доступны только администратору. Ответ очевиден и прост: для этого надо ввести систему входа по паролю. Что и реализовано в начале программы.

Обычные первые строки для наших файлов Python:

```
#! C:/Python/python
import cgi
```

Для упрощения запишем пароль в теле программы:

```
pas = 'admin'
```

Понятно, что **admin** — демонстрационный пароль, который в любом реальном проекте должен быть заменен на более сложный, по возможности состоящий из разных символов: букв английского алфавита, цифр, знаков препинания.

Для входа в файле **admin.py** предусмотрен механизм создания страницы с полем ввода пароля (рис. 5.6.1). Необходимо набрать его и нажать клавишу «Enter». Как это реализовано, мы рассмотрим чуть позже.

Рис. 5.6.1. Страница ввода пароля

Получаем данные из поля:

```
form = cgi.FieldStorage()
b = form.getfirst('pr')
```

Пишем функцию, которая будет нужна для открытия файла с названием сайта и файла заявок:

```
def opfi(adr):
  fo = open(adr, encoding='utf-8')
  ti = fo.read()
  fo.close()
  return ti
```

Как видите, она совершенно аналогична одноименной функции из файла **index.py**.

Начинаем печать страницы:

```
print('Content-type: text/html\n\n')
print('''<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>''' + opfi('set/title.txt') + '''</title>
<link rel="stylesheet" href="set/admin.css">''')
```

по ходу дела вставляя название сайта из файла title.txt:

```
...
<title>''' + opfi('set/title.txt') + '''</title>
...
```

Печатая начало документа, мы дошли до строки, после которой необходимо определиться, что мы выводим в браузер дальше: страницу с полем для пароля или страницу с заявками. Для этого проверяем условие соответствия введенного пароля настоящему:

```
if b != pas:
```

Когда это условие выполняется? Например, после того, как мы набрали в браузере адрес страницы заявок: **http://localhost/admin.py**. В этом адресе параметр **pr** отсутствует, поэтому условие истинно, а значит, печатается продолжение страницы с полем **type="password"**:

```
print('''</head>
<body>
<p> </p>
<form method="POST" action="admin.py">
<p><input type="password" name="pr" maxlength="20">
                          </p>''')
```

То же самое произойдет, если введен неверный пароль.

Если условие

```
if b != pas:
```

ложно, это означает, что пароль введен и он правильный. Тогда мы печатаем страницу с заявками (рис. 5.6.2):

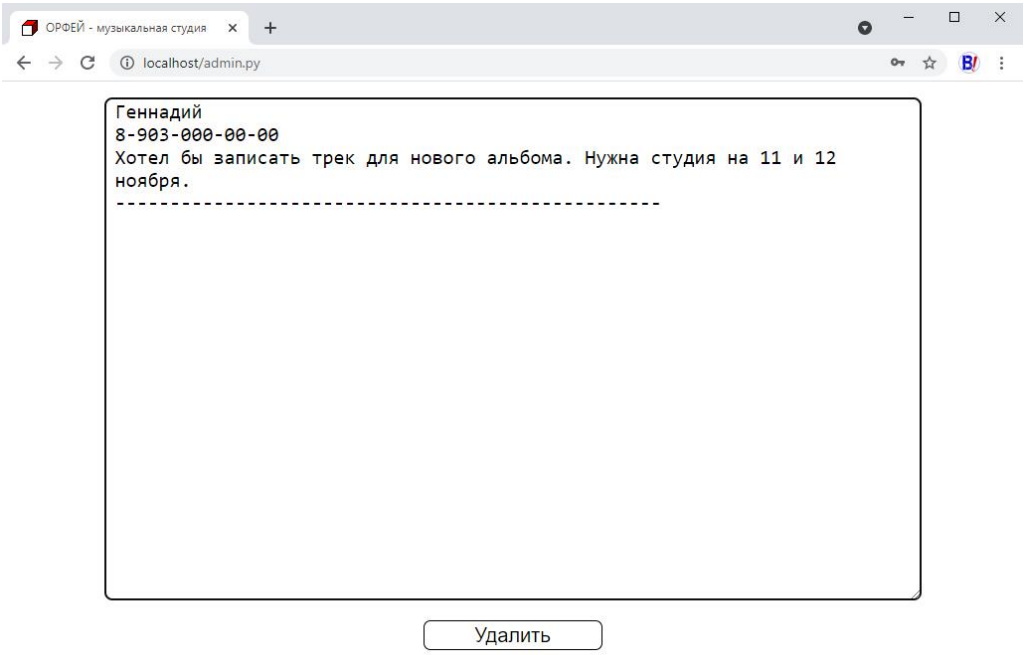

Рис. 5.6.2. Страница с заявками

```
print('''<script src="set/del.js"></script>
</head>
<body>
<p><textarea id="tex">''' + opfi('set/cont.py') +
                    '''</textarea></p>
<form name="dat"><input type="hidden" value="''' +
                  pas + '''" name="hid">
<p><input type="button" value="Удалить" id="but">
                         </p>''')
```

Заявки получаем, передавая в функцию **opfi** адрес необходимого файла, и печатаем их — для удобства копирования — в поле **textarea**:

```
...
<textarea id="tex">''' + opfi('set/cont.py') +
                    '''</textarea>
...
```

На завершающем этапе печатаются закрывающие теги, общие для любой из двух страниц:

```
print('''</form>
</body>
</html>''')
```

5.7. Файл del.py

Итак, администратор получил заявки, скопировал их на свой компьютер, и теперь можно очистить файл **cont.py**. Для этого необходимо нажать кнопку «Удалить».

Что произойдет дальше? Будет запущен сценарий из файла **set/del.js**, который в фоновом режиме передаст управление программе **del.py**.

У нее традиционное начало:

```
#! C:/Python/python

import cgi
```

Данные из скрытого поля (в файле **admin.py**)

```
<input type="hidden" value="''' +pas +
                    '''" name="hid">
```

будут получены уже привычным нам способом:

```
form = cgi.FieldStorage()
hid = form.getfirst('hid')
```

В качестве данных выступает пароль. Он необходим, чтобы программа знала — она получает информацию законным образом от администратора, а не от какого-то недоброжелателя.

Сообщаем браузеру тип выводимого документа:

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

Данная строка нужна независимо от того, был ли отправлен верный пароль или нет. Сейчас вы убедитесь в этом.

Если пароль в запросе отсутствует или он неверный,

```
if hid is None or hid != 'admin':
```

значит, в нашу программу хочет проникнуть взломщик. Отправляем в его браузер страницу с сообщением о некорректном запросе:

```
print('''<!DOCTYPE html>
<html lang="ru">
<head><meta charset="utf-8">
<title>Некорректный запрос !</title>
<link rel="stylesheet" href="set/admin.css">
</head>
<body>
<p> </p><p>Некорректный запрос !</p>
</body>
</html>''')
```

и прерываем выполнение программы:

```
exit()
```

Если пароль верный, производим удаление всех записей из файла **cont.py**:

```
fi = open('set/cont.py', 'w', encoding='utf-8')
fi.write("")
fi.close()
```

и отправляем сценарию **del.js** цифру **1**,

```
print(1)
```

которая будет воспринята сценарием как отчет об успешной очистке файла с заявками. В этом случае все записи из текстовой области на странице заявок исчезнут, а вместо них появится сообщение «ЗАЯВКИ УДАЛЕНЫ!» (рис. 5.7.1). Подробнее о работе сценариев на JavaScript вы прочтете в разделе 6.

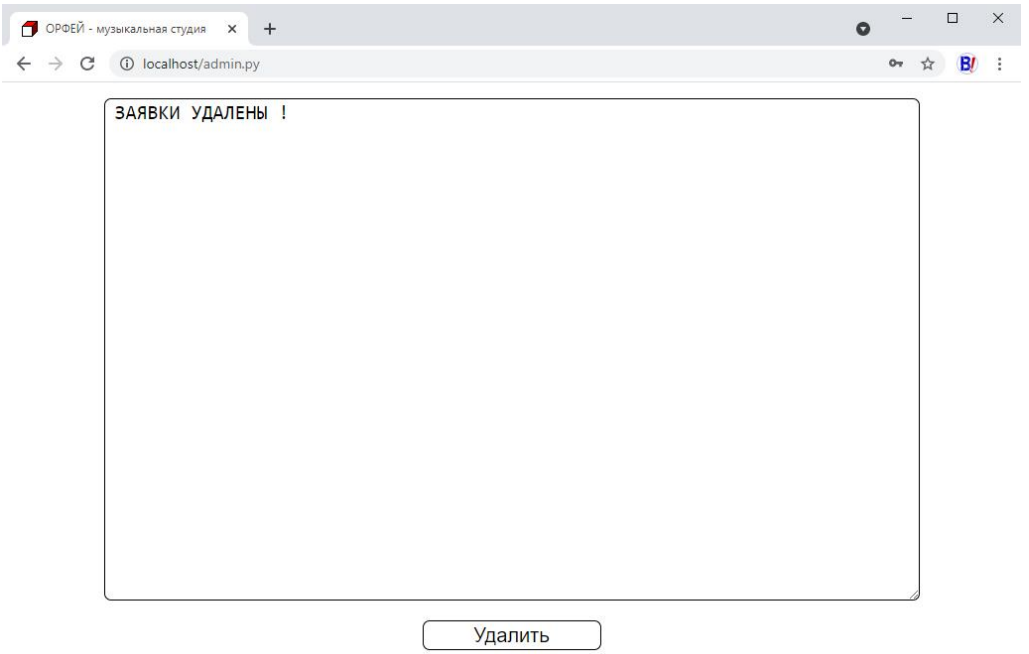

Рис. 5.7.1. Страница с сообщением об удалении заявок

Система просмотра заявок устроена так, что при каждом входе нужно вводить пароль. Если вы считаете, что это неудобно, можно предусмотреть какой-то механизм запоминания администратора после первого входа. Например, использовать cookie.

Завершая разговор про демонстрационный сайт музыкальной студии, необходимо подчеркнуть: хотя наш ресурс полностью работоспособен, это всего лишь наглядная иллюстрация к рассказу о том, как можно использовать Python в web-программировании. Понятно, что настоящие, полноценные сайты устроены сложнее, имеют более обширный функционал и работают с базами данных, а не с текстовыми файлами.

5.8. Перенос проекта на удаленный хостинг

До сих пор мы с вами писали демонстрационные программы и демонстрационный сайт. Однако наступит время, когда вы захотите создать какой-то реальный проект и перенести его с локального на реальный хостинг. Как это сделать — будет рассказано в этом разделе.

Итак, предположим, что у вас есть сайт, полностью сделанный на локальном хостинге персонального компьютера. Все страницы сверстаны, все файлы написаны, все функции протестированы. Пора выложить свое творение в Интернет. Как это сделать, каков порядок действий? Схема должна быть примерно следующая:

– регистрация доменного имени для сайта;
– приобретение виртуального хостинга;
– выяснение особенностей панели управления на хостинге;
– подготовка проекта к переносу;
– перенос проекта на хостинг;
– редактирование файла .htaccess и установка прав на файлы Python;
– указание DNS-серверов на сайте регистратора домена;
– тестирование и устранение проблем;
– продвижение ресурса.

Пойдем по порядку.

Доменное имя лучше всего регистрировать в компании «RU-CENTER» (АО «Региональный Сетевой Информационный Центр»). Их сайт — **https://www.nic.ru/**. Зарегистрируйтесь, заполните анкету, выберите свободное имя для вашего сайта и произведите оплату. Сообщение об успешной регистрации домена придет вам на почту. Также эту информацию можно будет увидеть на сайте компании в разделе «Для клиентов».

Следующий этап — приобретение виртуального хостинга у одного из провайдеров. Сейчас много компаний, предоставляющих серверы для размещения сайтов за весьма небольшую плату. Не торопитесь с выбором, сравните цены и предоставляемые услуги.

Особенно важно, чтобы хостинг удовлетворял следующим требованиям.

1. Наличие интерпретатора Python версии 3 (желательно, чтобы цифра после точки в номере версии была повыше, например 3.8). Обратите внимание, версия 2 (например, 2.7) — не подойдет!

2. Возможность размещать файлы Python в корневой директории сайта.

3. Возможность устанавливать загружаемым по умолчанию файлом — index.py.

Некоторые провайдеры предоставляют тестовый период для проверки хостинга. За тестовый период вносить оплату не надо. Что является хорошей возможностью изучить за это время панель управления, ознакомиться с ее функционалом и понять, все ли вас устраивает. Ну и конечно, перенести ваш проект и проверить его работоспособность на данном хостинге.

Но прежде чем перемещать файлы и папки на хостинг, необходимо подготовить ваш проект к этому событию. Этот процесс состоит из двух этапов:

– замена во всех файлах Python пути к интерпретатору на тот, что вы получили у провайдера;

– переформатирование всех файлов — **py**, **txt**, **css**, **js** — из Windows в Unix представление.

Опишем эти шаги подробнее. Сначала создайте на рабочем столе папку, в которую скопируйте ваш сайт из папки **htdocs** сервера Apache. Загрузите по очереди файлы Python в редактор Notepad++ и замените в них строку с адресом интерпретатора (рис. 5.8.1). Сохраните результаты. Для всех файлов: на нижней панели редактора найдите текст «Windows(CR LF)» и сделайте на нем двойной щелчок. В появившемся меню выберите пункт «Преобразовать в Unix(LF)» и нажмите его (рис. 5.8.1). Файл будет преобразован. Сохраните изменения.

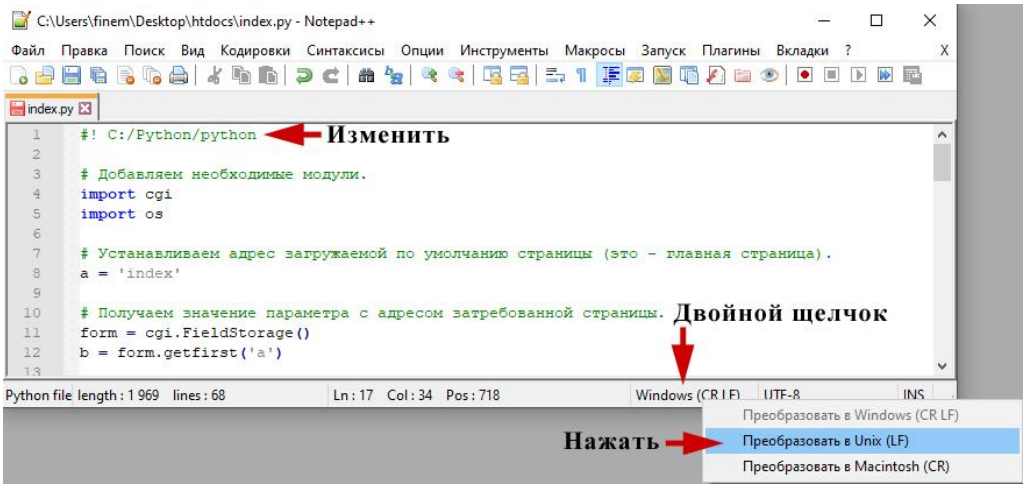

Рис. 5.8.1. Процесс подготовки файла к переносу на хостинг

Перенести файлы и папки на хостинг можно разными способами. Например, воспользовавшись механизмами, которые предоставляет панель управления. Или загрузив проект по протоколу ftp. На второй случай провайдер должен дать вам логин и пароль для входа.

Сейчас автор познакомит вас со способом, который используют крайне редко — и напрасно. Способ настолько прост, что дальше некуда. Для этого мы воспользуемся стандартной программой «Проводник», которую предоставляет операционная система Windows.

1. Откройте любую папку на вашем компьютере и в левой верхней строке введите адрес вашего сайта по ftp-протоколу (рис. 5.8.2).

2. Появится окно для входа на ftp-сервер. Введите в нем имя пользователя (логин) и пароль (рис. 5.8.3).

3. Откроется папка, предназначенная для файлов вашего сайта. Скопируйте все компоненты проекта и переместите их на хостинг (рис. 5.8.4).

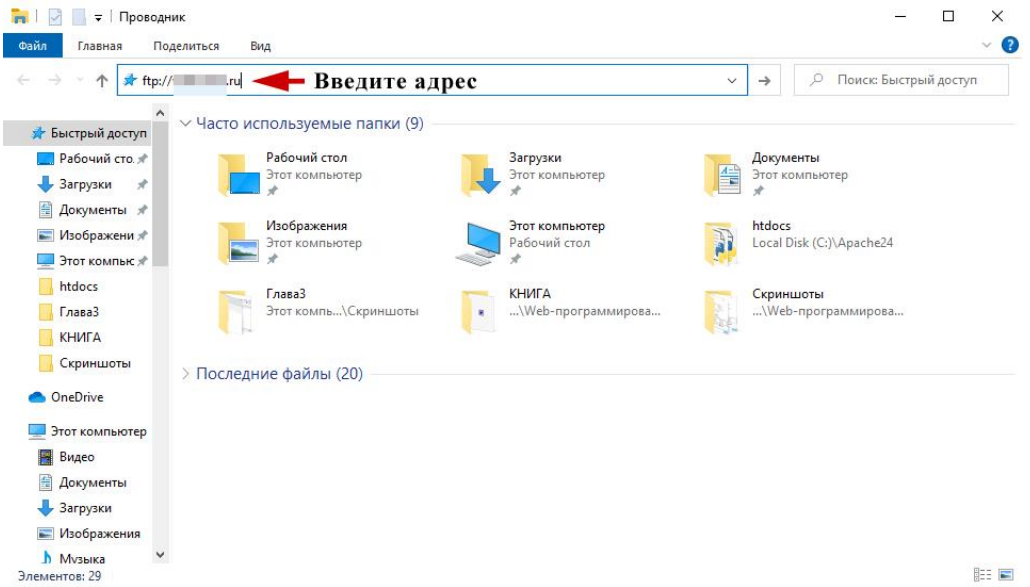

Рис. 5.8.2. Начинаем процесс входа на хостинг по протоколу ftp

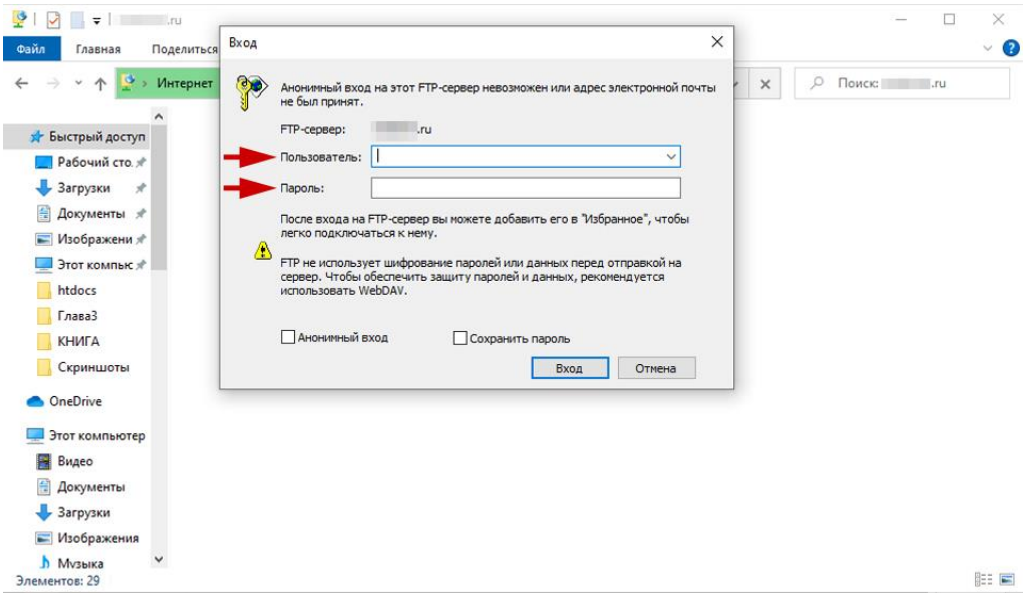

Рис. 5.8.3. Ввод имени пользователя и пароля

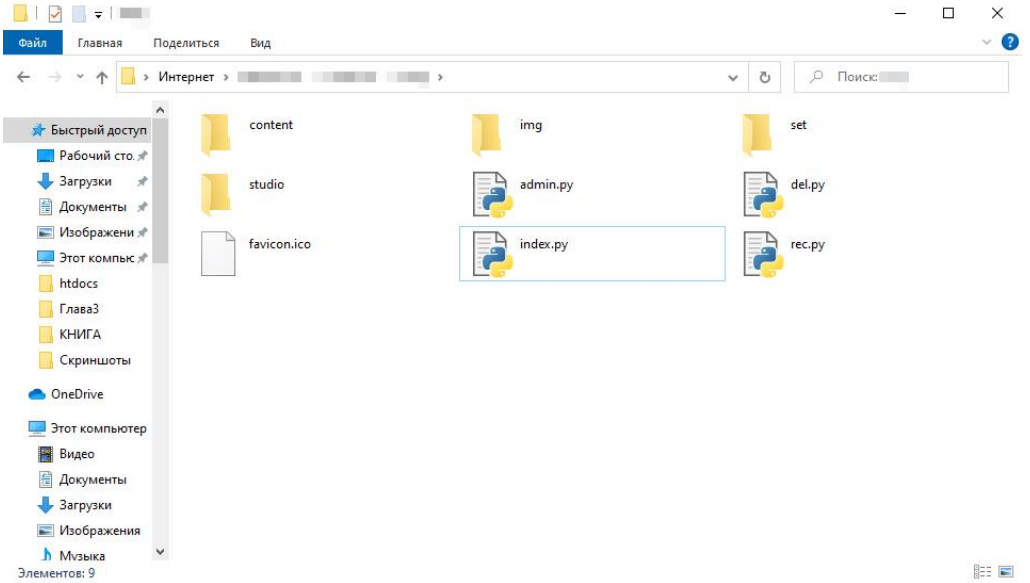

Рис. 5.8.4. Загружаем файлы проекта в папку вашего сайта

Следующий этап работ — редактирование файла .htaccess и установка прав на файлы Python.

Файл .htaccess чаще всего изначально находится в папке, созданной провайдером для вашего сайта. Этот файл необходим, чтобы указать индивидуальные настройки сервера. Обычно администрация хостинга предоставляет необходимую информацию по таким настройкам.

Что касается установки прав на файлы Python, мы рассмотрим два варианта таких действий.

Если вы загружали проект через Проводник, то можете сразу после перемещения файлов установить необходимые права. Для этого щелкните на пиктограмме файла правой (правой!) кнопкой мыши. В открывшемся меню выберите пункт «Свойства» (рис. 5.8.5) и кликните на нем. Появится окно настроек (рис. 5.8.6). Здесь можно установить права на чтение, запись и выполнение для разных пользователей. Нужно установить следующие разрешения:

– владельцу — на чтение, запись и выполнение;

– группе — на чтение и выполнение;

– всем пользователям — на чтение и выполнение.

Завершайте процесс нажатием кнопки «ОК».

Способ второй — выполнить настройки прав в панели управления. Один из вариантов может выглядеть так (на примере хостинга SpaceWeb):

– щелкаете правой кнопкой мыши на пиктограмме файла;

– в открывшемся меню выбираете пункт «Изменить права доступа»;

– устанавливаете их для разных пользователей согласно описанию, приведенному выше (рис. 5.8.7) — в результате в окне «Разрешение» должны появиться цифры 755.

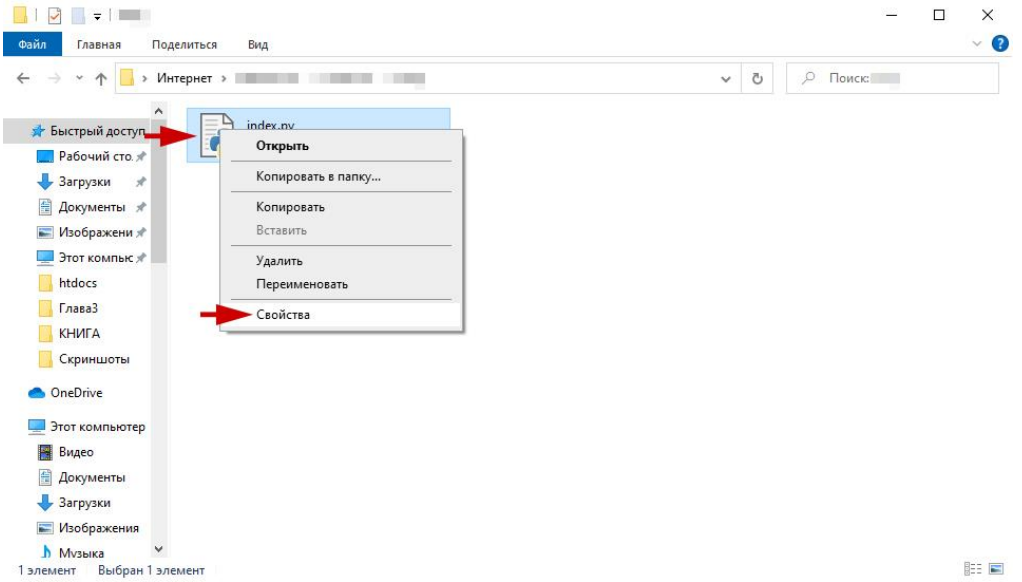

Рис. 5.8.5. Выбор пункта «Свойства» из контекстного меню

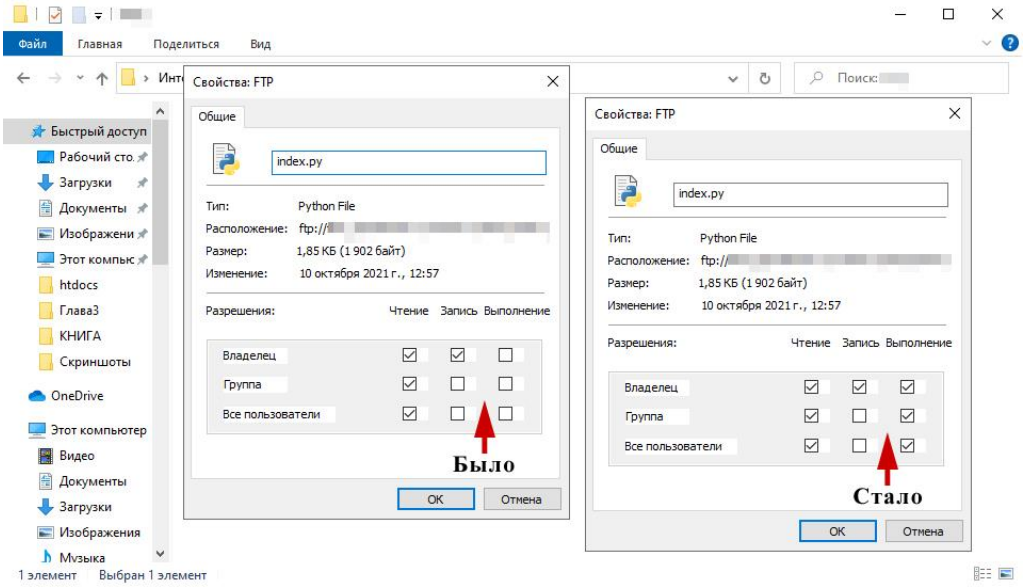

Рис. 5.8.6. Установка прав доступа к файлу в «Проводнике»

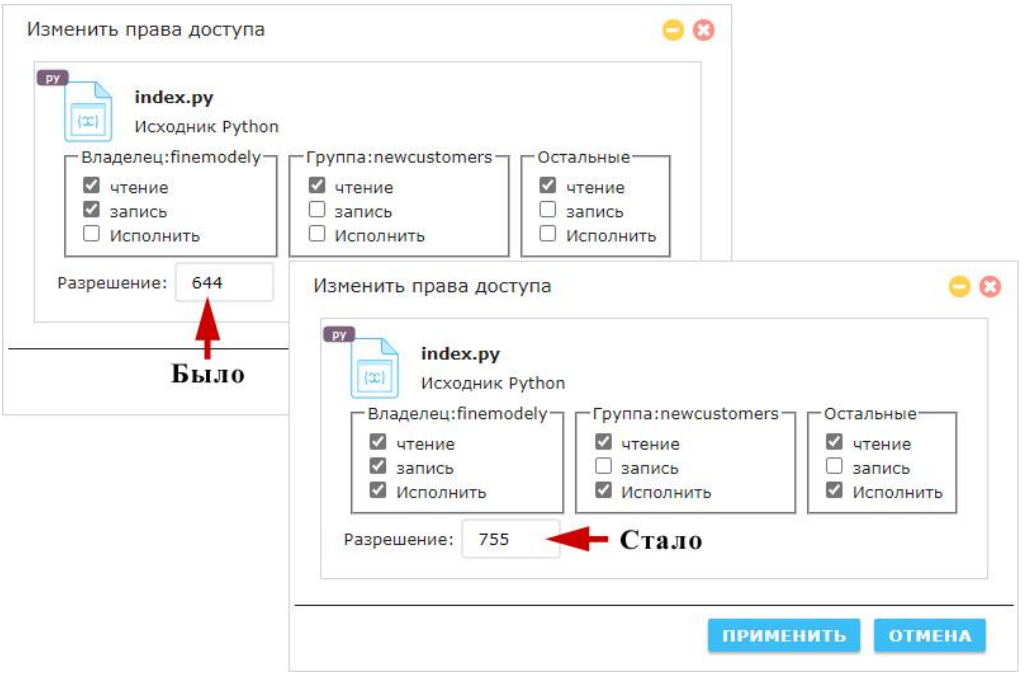

Рис. 5.8.7. Установка прав доступа в панели настроек хостинга

Теперь узнайте у своего провайдера номера DNS-серверов (обычно их количество — от 2 до 4). На сайте регистратора зайдите на свою страницу, откройте вкладку с перечнем ваших доменов и кликните ссылку «Делегировать» напротив имени нужного домена (рис. 5.8.8 — показано на примере компании «RU-CENTER»). Откроется страница управления DNS-серверами (рис. 5.8.9). Заполните поля под строкой «DNS-сервер» и нажмите кнопку «Сохранить изменения». Будьте готовы к тому, что придется подождать несколько часов, пока ваш сайт не станет виден в Интернете.

Этап под названием «тестирование и устранение проблем». Обычно ошибки в работе сайта возникают по следующим причинам:

– вы забыли изменить в одном из файлов Python путь к интерпретатору или указали его неправильно;

– забыли перекодировать какой-то файл из Windows в Unix формат;

– забыли изменить права доступа какого-нибудь исполняемого файла;

– указали не все необходимые настройки в файле .htaccess.

Обязательно проверьте все функции вашего сайта, чтобы убедиться в отсутствии проблем!

Ваш проект заработал? Пора заняться его «раскруткой». Дело это настолько непростое, что автор советует прочитать специальные книги с рекомендациями по продвижению и аналогичные статьи в Интернете.

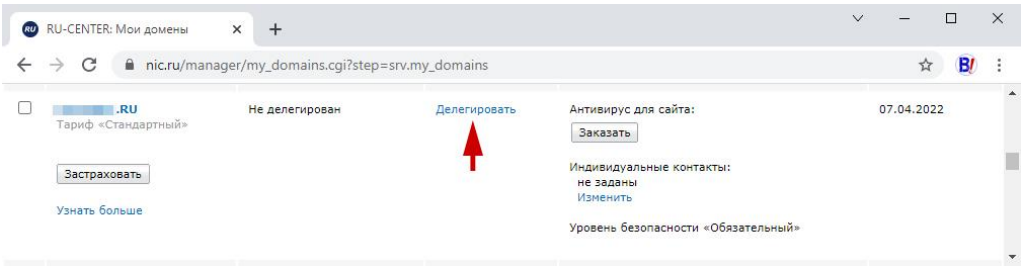

Рис. 5.8.8. Страница с перечнем ваших доменов

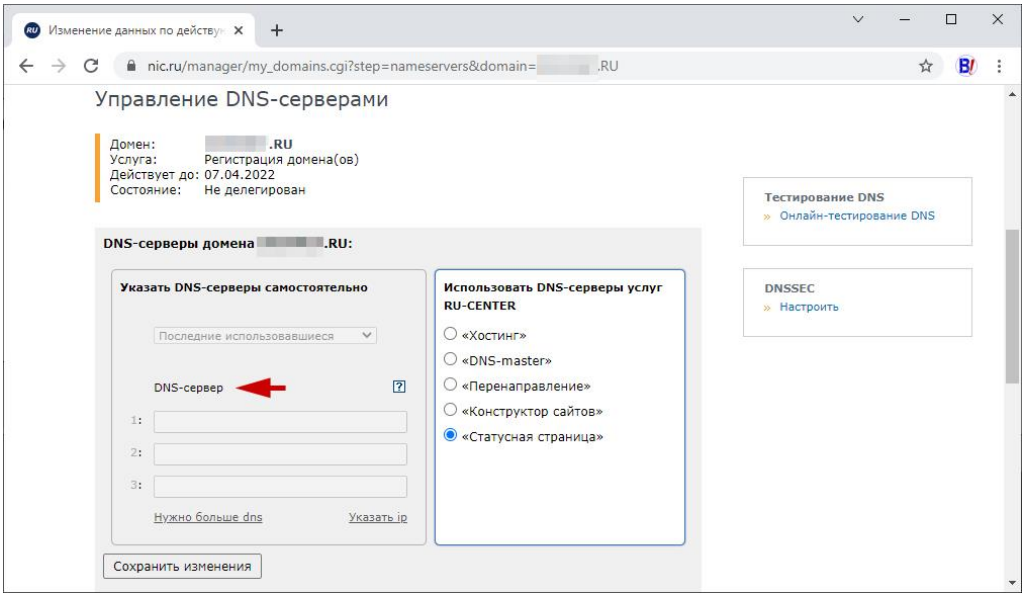

Рис. 5.8.9. Страница управления DNS-серверами

6. Приложения — сценарии на JavaScript

Поскольку в работе сайта сценарии на JavaScript играют важную роль, автор посчитал нелишним объяснить принцип их действия.

Сценариев всего 3:

– управляющий просмотром увеличенных фото на страницах сайта;

– выполняющий проверку формы и отправку данных в файл записи;

– удаляющий все заявки по решению администратора ресурса (естественно, после того, как он перенес эти заявки на свой компьютер).

Все сценарии расположены на локальном хостинге в папке **set**.

Как и основные программы на Python, сценарии на JavaScript тестировались в 5 браузерах, а также в валидаторе кода, расположенном по адресу **https://beautifytools.com/javascript-validator.php**.

6.1. Сценарий просмотра фото

Этот сценарий обеспечивает переход в режим просмотра увеличенного фото и возврат к просмотру страницы. Его файл — **index.js** из папки **set**.

Если на странице сайта в области основного контента есть какое-либо изображение, посетитель может увеличить его, щелкнув мышью по картинке. При этом страница будет закрыта белым полупрозрачным слоем с увеличенным изображением. Если теперь кликнуть на нем или прокрутить страницу, увеличенная картинка и полупрозрачный слой исчезнут. Управляет данным процессом сценарий, написанный на JavaScript. Его мы разберем позже, а пока выясним, что необходимо добавить в HTML-код страницы для просмотра увеличенных фотографий.

Автор добавил в разметку файла **index.py** вот такой фрагмент:

```
<div id="bas"></div>
<div id="view">
<img id="pict" src="img/net.jpg" alt="Фото">
</div>
```

Как видите, здесь 2 слоя. Нижний

```
<div id="bas"></div>
```

имеет белый фон с уровнем непрозрачности **0.95**. Верхний

```
<div id="view">
...
</div>
```

служит контейнером для увеличенного изображения:

```
<img id="pict" src="img/net.jpg" alt="Фото">
```

В исходном состоянии на данный слой загружено изображение **net.jpg** белого цвета размером 1x1 пиксель. Сделано это, чтобы угодить требованиям стандартов Консорциума Всемирной паутины, которые не допускают в разметке изображений с пустым атрибутом **src** (ведь в исходном состоянии никакое фото со страницы в контейнер **div id="view"** не загружено).

При загрузке увеличенного рисунка белый фон занимает все пространство окна браузера, а картинка располагается строго по середине (рис. 6.1.1).

Для вставки данного фрагмента кода автор выбрал место сразу после открывающего тега **body**:

```
...
<body>

<div id="bas"></div>
<div id="view">
<img id="pict" src="img/net.jpg" alt="Фото">
</div>

<div id="basis">
...
```

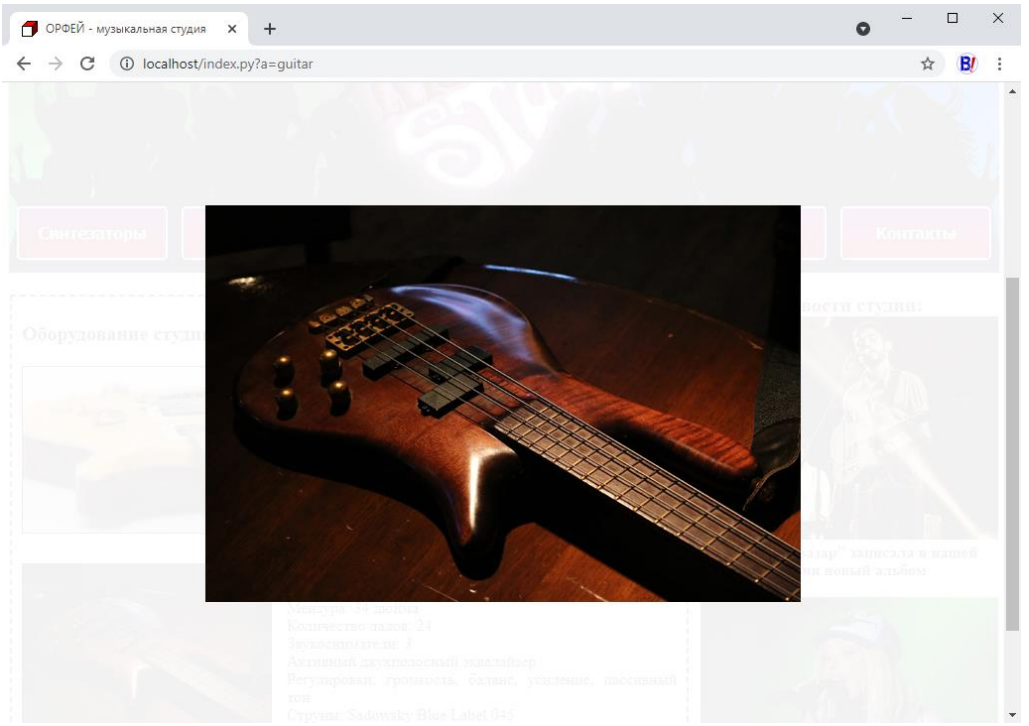

Рис. 6.1.1. Увеличенное изображение на странице

Сценарий подключаем в заголовочной части документа (файл **index.py**) следующим образом:

```
<script src="set/index.js"></script>
```

Что мы найдем в файле сценария? Две функции. Первая необходима для демонстрации увеличенного изображения, вторая — для выхода из режима просмотра.

Так как мы загружаем сценарий в заголовочной части документа, то регистрация необходимых обработчиков событий выполняется только после загрузки страницы:

```
addEventListener("load", function()
{
...
});
```

Для сокращения кода, в котором несколько раз происходят обращения к одним и тем же компонентам, создадим три переменные:

```
let view=document.getElementById("view");
let bas=document.getElementById("bas").style;
let pict=document.getElementById("pict");
```

Теперь зарегистрируем обработчик для клика в области основного содержания. Его роль у нас будет выполнять анонимная функция, получающая в качестве аргумента объект события:

```
document.getElementById("cont").
    addEventListener("click", function(ev)
  {
  ...
  });
```

Обратите внимание: функция просмотра фото увеличивает снимки только из области основного содержания. Эта область представляет собой контейнер **div** с **id="cont"** (от *англ.* content — содержание, наполнение). После щелчка, произошедшего на слое **cont**, надо выяснить, был ли этот щелчок на изображении или нет:

```
if(ev.target.tagName=="IMG")
  {
  ...
  }
```

Если да, то нужно продемонстрировать увеличенную картинку, расположив ее по центру страницы, насколько бы она ни была прокручена по вертикали. Для этого первым делом определяем величину прокрутки:

```
let sc=window.pageYOffset;
```

а затем «передвигаем» слой фона и слой с изображением на эту величину:

```
bas.top=sc+"px";
```

```
view.style.top=sc+"px";
```

Вычисляем адрес «кликнутого» фото и присваиваем этот адрес атрибуту **src** картинки со слоя:

```
pict.src=ev.target.src;
```

Завершающий этап — включение режима просмотра:

```
view.style.visibility="visible";
bas.visibility="visible";
```

Чтобы выйти из режима просмотра, зарегистрируем один обработчик для двух разных событий:

```
view.addEventListener("click", del);
addEventListener("scroll", del);
```

Первый запускает функцию **del**, когда мы щелкаем по увеличенному изображению или фону, а второй — при прокрутке страницы.

В самой функции всего три инструкции:

```
function del()
  {
  view.style.visibility="hidden";
  bas.visibility="hidden";
  pict.src="img/net.jpg";
  }
```

Первая и вторая скрывают фон и картинку, а третья присваивает рисунку со слоя адрес исходного изображения. Этого момента коснемся подробнее.

В принципе, рисунку со слоя можно было бы и не присваивать исходный адрес точечного изображения. Но! В таком случае при низкой скорости соединения с сервером можно столкнуться с неприятным эффектом: допустим, вы посмотрели одно фото, завершили просмотр и щелкнули на второй фотографии. Вновь запустится режим просмотра, но из-за медленной загрузки второго снимка вы некоторое время будете видеть первый. А это, согласитесь, не очень хорошо. Присваивая **src** значение **img/net.jpg**, мы избавляемся от подобных неприятностей, ведь маленькая белая точка загружается очень быстро и совершенно не будет видна в момент медленной загрузки второго снимка. То есть мы просто будем видеть белый полупрозрачный фон до тех пор, пока не появится следующее изображение.

6.2. Сценарий проверки формы

Форму для заявок мы уже рассмотрели в пятой главе, поэтому перейдем к разбору сценария, выполняющего отправку сообщений. Его адрес на локальном хостинге — **set/cont.js**.

Этот сценарий подключается в головной части документа (файл **index.py**) исключительно при загрузке страницы «Контакты»:

```
if a == 'contact':
  print('<script src="set/cont.js"></script>')
```

Схема процесса выглядит следующим образом. После загрузки страницы регистрируется обработчик для нажатия кнопки «Отправить» — функция **cont**:

```
addEventListener("load", function()
{
document.getElementById("but").
        addEventListener("click", cont);
});
```

Представим, что мы не заполнили форму и пытаемся отправить «ничего» на сервер. На этот случай функция **cont** выполняет три проверки. Первая — заполнено ли поле «Ваше имя»:

```
if(document.getElementById("nam").value.length<2)
  {
  document.getElementById("but").value=
              "ВЫ НЕ ВВЕЛИ ИМЯ !";
  return false;
  }
```

Обнаружив, что данное поле пустует, выводим предупреждение «ВЫ НЕ ВВЕЛИ ИМЯ!» и прерываем отправку данных. Напомню, что здесь мы использовали креативное решение — все предупреждения выводятся непосредственно на кнопку **but** вместо ее первоначальной надписи «Отправить»:

```
document.getElementById("but").value=
              "ВЫ НЕ ВВЕЛИ ИМЯ !";
```

Самые короткие имена состоят из двух букв, например: Ян, Ия. Поэтому в проверочном условии

```
if(document.getElementById("nam").value.length<2)
```

2 — это минимальное количество символов, необходимое для успешного завершения проверки этого поля.

Итак, введем какое-то имя и вновь попробуем отправить данные из формы. Однако у нас опять ничего не получится, так как функция теперь проверит заполнение поля «Телефон или e-mail»:

```
else
  {
  if(document.getElementById("con").value.length<8)
   {
   document.getElementById("but").value=
```

```
        "ВЫ НЕ ВВЕЛИ ТЕЛЕФОН ИЛИ E-MAIL !";
   return false;
   }
  ...
  }
```

Самый короткий номер телефона содержит минимум 10 символов: 9990001155. Самый короткий адрес электронной почты — 8: два в начале + @ + два символа доменного имени второго уровня + точка + два символа доменного имени первого уровня (теоретически возможны и более короткие имена, но они настолько уникальное явление, что учитывать их существование не имеет смысла). Поэтому выбираем количество символов для успешной проверки — не менее 8.

Заполним второе поле. Но и сейчас отправить форму не получится, так как третье осталось пустым, что обнаружит следующий фрагмент кода:

```
else
  {
  if(document.getElementById("tex").value.length<10)
   {
   document.getElementById("but").value=
              "ВЫ НЕ ВВЕЛИ СООБЩЕНИЕ !";
   return false;
   }
  ...
  }
```

Введем текст сообщения — минимум 10 символов — и только теперь у нас появится возможность отправить данные.

На этом этапе автор хотел бы заострить особое внимание. Как вы убедились, мы проверили только количество символов в полях формы. На самом деле это была упрощенная проверка, подходящая для нашего учебного сайта, но недостаточная для настоящего, серьезного ресурса. Смотрите: мы не проверили, использовал ли посетитель разрешенные символы, например в имени (получив имя #&_\, мы вряд ли обрадуемся). Не убедились, что клиент действительно указал адрес почты или телефона, а не ввел произвольный набор знаков. Не определили, что в сообщении он написал реальный текст, а не всякую галиматью. Конечно, такая проверка для настоящего клиента не имеет смысла: он не станет засорять форму не пойми чем. Но зато усложнит задачу интернет-хулиганам, которые любят отправлять формы с разной абракадаброй.

Почему же автор не сделал такую строгую проверку? Ответ прост: чтобы не перегружать книгу о Python кодом из JavaScript. Для демонстрационного сайта нам достаточно показать, что перед отправкой обязательно должна существовать проверка данных.

Напомню также, что при желании особо рьяные недоброжелатели могут обойти проверку на стороне клиента. Поэтому основной заслон необходимо ставить на стороне сервера, в программах, которые выполняют запись данных или их отправку по электронной почте.

Идем дальше.

Отправлять заявки мы будем, используя технологию Ajax. Для этого создадим новый объект с данными из формы:

```
let dt=new FormData(document.forms.dat);
```

и новый объект интерфейса **XMLHttpRequest**:

```
let re=new XMLHttpRequest();
```

Методом **open** формируем запрос:

```
re.open("POST", "rec.py", true);
```

и отправляем его серверной программе **rec.py**, которая выполняет запись данных в файл **cont.py**:

```
re.send(dt);
```

Также устанавливаем, что после загрузки ответа должна запускаться функция **show**:

```
re.addEventListener("load", show);
```

Ее задача — на основе полученного ответа вывести то или иное сообщение на кнопке отправки данных:

```
function show()
{
let re=document.getElementById("but");
let ans=this.responseText;
if(ans==1)
  re.value="СООБЩЕНИЕ ОТПРАВЛЕНО !";
if(ans==2)
  re.value="ЗАПОЛНЕНЫ НЕ ВСЕ ПОЛЯ !";
if(ans==3)
  re.value="НЕКОРРЕКТНЫЕ ДАННЫЕ !";

setTimeout(sto, 3000);
}
```

Последняя инструкция указывает программе, что через 3 секунды должна быть запущена функция **sto**. У нее очень простое назначение — вернуть кнопке исходную надпись:

```
function sto()
{
document.getElementById("but").value="Отправить";
}
```

6.3. Сценарий запроса на удаление заявок

Этот сценарий во многом напоминает предыдущий. Только его задача — удалить все записи из файла с заявками. На локальном хостинге он находится по адресу **set/del.js**.

Сценарий, как всегда, подключаем в заголовочной части документа (файл **index.py**):

```
<script src="set/del.js"></script>
```

После загрузки страницы регистрируется обработчик для нажатия кнопки «Удалить» — функция **cont**:

```
addEventListener("load", function()
{
document.getElementById("but").
         addEventListener("click", cont);
});
```

Допустим, администратор сайта перенес все данные потенциальных клиентов на свой компьютер и решил, что пора очистить список. Находясь на странице просмотра заявок, он нажимает кнопку «Удалить». Что происходит после этого?

Запускается функция **cont**, почти аналогичная одноименной функции из предыдущего раздела. Только теперь передаются не набор данных, а пароль из скрытого поля **hid**. И адресатом является файл **del.py**:

```
function cont()
{
let dt=new FormData(document.forms.dat);
let re=new XMLHttpRequest();
re.open("POST", "del.py", true);
re.send(dt);
re.addEventListener("load", show);
}
```

Так как передача выполняется методом **POST**, данные скрыты от посторонних глаз. Получив пароль и убедившись, что он верный, программа **del.py** удаляет все записи из файла **cont.py** и оправляет обратно в качестве ответа цифру **1**. После чего функция **show** очищает текстовую область и выводит в ней сообщение «ЗАЯВКИ УДАЛЕНЫ !».

```
function show()
{
let ans=this.responseText;
if(ans==1)
  document.getElementById("tex").value=
                "ЗАЯВКИ УДАЛЕНЫ !";
}
```

7. Заключение

Итак, мы создали полноценную среду разработки на вашем компьютере. Изучили синтаксис Python. Освоили некоторые приемы написания программ для Интернета. Наконец, создали простой, но вполне работоспособный сайт. Настало время подвести итоги.

Думаю, на примере материала из этой книги вы убедились, что для web-программирования на Python совсем не обязательно использовать фреймворки. Более того, в случае написания простого сайта вполне может оказаться, что ресурс на «чистом» Python создавать быстрее, удобнее, рациональнее. В нашем проекте нет никаких лишних файлов. Но этот аскетизм оправдан — ведь мы получили именно тот результат, на который рассчитывали.

Приведу пример из другого языка программирования, который, на мой взгляд, отлично подходит и для Python. Есть такой ресурс — http://vanilla-js.com/ — шуточный сайт кроссплатформенного фреймворка Vanilla JS, под которым подразумевается чистый JavaScript. Его автор ставит целью напомнить разработчикам, что довольно часто при создании кода можно обойтись без библиотек и фреймворков. В очень многих случаях писать программы на чистом JavaScript гораздо продуктивнее и быстрее. Отсюда, кстати, пошло выражение «ванильный JavaScript».

Мне кажется, есть смысл говорить и «ванильном» Python. Ведь нередко применение фреймворков неоправданно усложняет те или иные проекты, которые вполне могли бы быть гораздо проще. Умение писать на «чистом» языке программирования еще не помешало ни одному разработчику, а только обогатило его опыт и профессиональный уровень. Чего и желает вам автор!

Валерий Викторович ЯНЦЕВ

WEB-ПРОГРАММИРОВАНИЕ НА PYTHON

Учебное пособие

Зав. редакцией литературы
по информационным технологиям
и системам связи *О. Е. Гайнутдинова*
Ответственный редактор *Н. А. Кривилёва*
Корректор *Н. Ю. Наумкина*
Выпускающий *Е. А. Романова*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.10.953.П.1028
от 14.04.2016 г., выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lanbook.ru; www.lanbook.com
196105, Санкт-Петербург, пр. Юрия Гагарина, д.1, лит. А.
Тел.: (812) 336-25-09, 412-92-72.
Бесплатный звонок по России: 8-800-700-40-71

Подписано в печать 03.03.22.
Бумага офсетная. Гарнитура Школьная. Формат 70×100 $^{1}/_{16}$.
Печать офсетная/цифровая. Усл. п. л. 14,63. Тираж 30 экз.

Заказ № 367-22.

Отпечатано в полном соответствии
с качеством предоставленного оригинал-макета
в АО «Т8 Издательские технологии»
109316, г. Москва, Волгоградский пр., д. 42, к. 5.