



Разработка
клиентских сценариев
с использованием

Javascript

Урок №5

Формы

Содержание

Создание элементов формы.....	3
Схема интернет-ресурса	3
Что такое формы	5
Атрибуты формы	7
Создание формы	11
Элементы формы	15
Домашнее задание.....	28
Программирование элементов формы.....	30
Еще раз о коллекциях элементов.....	30
Добавление элементов.....	40
Подробнее о клонировании и вставке	46
Удаление элементов.....	51
Домашнее задание.....	55

Материалы к уроку прикреплены к данному PDF-файлу.

Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

Создание элементов формы

Схема интернет-ресурса

Чтобы понимать роль и место форм, особенности их атрибутов и элементов, давайте рассмотрим общую схему организации интернет-ресурса.



Обычно в организации ресурса выделяют три уровня: **пользователь, клиент и сервер.**

Уровень пользователя – это визуальный интерфейс ресурса. Именно его, как правило, и называют сайтом. Это то, что мы видим на экране браузера после успешной загрузки ресурса. Для создания интерфейса применяется язык разметки HTML.

Уровень клиента – это программное обеспечение, которое создает и прорисовывает интерфейс. Оно также принимает от пользователя введенные данные, реагирует на нажатие кнопок и ссылок, выбор списков и т. п., и передает их на сервер. Уровень клиента обеспечивается браузером и для дополнительного управления его работой используется язык JavaScript.

Уровень сервера – это «сердце» ресурса. Тут хранятся основные данные, обрабатываются запросы, проверяется авторизация, подготавливаются ответы на различные запросы клиента. На сервере располагается база данных ресурса, хранящая информацию об основных предметах ресурса (товарах, новостях, файлах), о пользователях, их истории посещений и сообщений, и многое другое. Работу сервера обеспечивают так называемые серверные технологии – PHP, ASP.NET и им подобные.

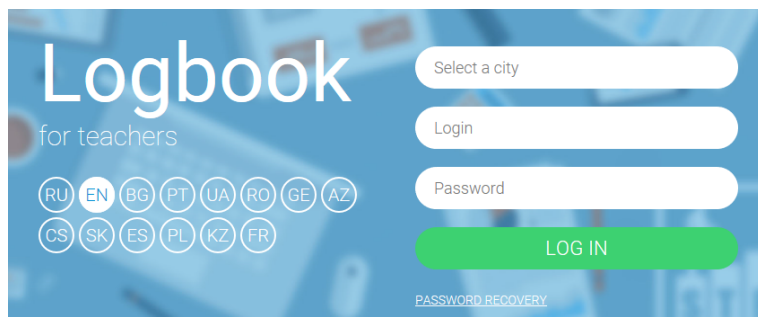
Особую роль в обмене играет канал связи как средство передачи данных от клиента серверу и обратно. Как правило, это обобщается словом «сеть» – кабели, коммутаторы, Wi-Fi-модули и т. д. Однако канал связи накладывает определенные ограничения на взаимодействие клиента и сервера. Например, канал не позволяет передавать произвольные символы, а требует их преобразования в специальный вид – транспортную кодировку. Увидеть действие транспортного кодирования несложно – введите в поиск Гугла *«академия шаг»* и скопируйте адресную строку браузера. Получится что-то в духе:

<https://www.google.com.ua/?q=%D0%B0%D0%BA%D0%B0%D0%B4%D0%B5%D0%BC%D0%B8%D1%8F+%D1%88%D0%B0%D0%B3>

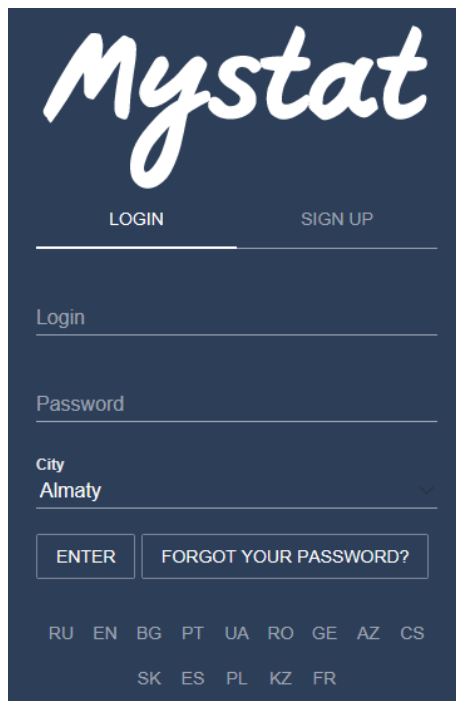
Это и есть транспортное кодирование, необходимое для канала связи. Обычно, кодирование совершает сам браузер и нам дополнительных действий для кодирования совершать не нужно. Тем не менее, помнить об этом надо тогда, когда мы встраиваем ссылки в наш сайт. Их желательно сразу создавать в кодированной форме, чтобы упредить возможные ошибки передачи данных.

Что такое формы

Как мы смогли увидеть в предыдущем разделе, процесс обмена данными в сетевом ресурсе довольно сложен и содержит несколько этапов. Для того чтобы разработчик сайта не вникал в каждый этап со своими особенностями, был создан специальный объект HTML – **форма (form)**. Это механизм автоматизированной отправки данных, введенных пользователем, на серверную сторону сайта. Поскольку слово «форма» имеет много смыслов, в данном случае оно ближе всего по сути к форме обратной связи. Одно из самых распространенных применений формы, присутствующее практически на каждом сайте, – авторизация, форма входа на сайт.



The image shows a login interface for a 'Logbook for teachers'. It features a blue background with a blurred image of a laptop. The title 'Logbook' is in large white font, with 'for teachers' in smaller white font below it. To the right of the title is a dropdown menu labeled 'Select a city'. Below the title is a grid of 16 circular buttons with two-letter language codes: RU, EN (highlighted), BG, PT, UA, RO, GE, AZ in the first row, and CS, SK, ES, PL, KZ, FR in the second row. To the right of the language buttons are three input fields: 'Login', 'Password', and a green 'LOG IN' button. At the bottom right, there is a link for 'PASSWORD RECOVERY'.



The image shows a dark blue login and registration form for 'Mystat'. At the top, the word 'Mystat' is written in a large, white, cursive font. Below it are two links: 'LOGIN' and 'SIGN UP'. A horizontal line separates these from the input fields. There are three input fields: 'Login', 'Password', and 'City'. The 'City' field has a dropdown menu with 'Almaty' selected. Below the input fields are two buttons: 'ENTER' and 'FORGOT YOUR PASSWORD?'. At the bottom, there are two rows of language codes: 'RU EN BG PT UA RO GE AZ CS' and 'SK ES PL KZ FR'.

Группа элементов интерфейса может быть привязана к форме при помощи средств разметки HTML и отправлена на сервер, будучи обработанной и декодированной автоматически, без необходимости вмешательства в эти процессы со стороны разработчика.

Создается форма тегом `<form>`, закрывающий тег `</form>` обязателен. Элементы, описанные между открывающим и закрывающим тегом формы, автоматически привязываются к этой форме. По умолчанию, тег формы не вносит стиливых особенностей, то есть группировка элементов в форму не меняет способ их отображения. Но, как и любой другой объект, форма может быть дополнительно стилизована.

Атрибуты формы

Соответственно своему предназначению, форма должна обеспечивать отправку данных на сервер. Выше мы уже отметили, что сервер – это большой программный комплекс, и отправить данные просто «на сервер» – это как просто «послать письмо» в столицу. Без указания адресата письмо до столицы дойдет, но затеряется где-то в недрах главпочтамта. Так же и с запросами к серверу необходимо указывать «адресата» – имя той из множества программ на сервере, которая примет отправленные формой данные. Имя адресата задается атрибутом формы **action**: `<form action="add_user.php">`.

Вторым важным атрибутом формы является метод (или способ) отправки сообщения серверу. Здесь снова нужно вспомнить, что сервер – это сложный комплекс, и он может обрабатывать запросы разного типа: запросы на чтение, запись, модификацию, удаление. Например, метод **HEAD** запрашивает у сервера только заголовочную часть сайта, без его тела, для быстрого получения метаданных наподобие названия (**title** указывается в заголовочной части). Метод **DELETE** является командой на удаление каких-то данных из ресурса. Весь перечень возможных методов запроса для действующего на сегодня протокола HTTP/1.1 приведен и детально описан в специальном стандарте RFC 2616.

Поскольку формы являются упрощенным способом автоматизированной доставки данных на сервер, для них доступны лишь два метода запроса – **GET** и **POST**. Для реализации других методов необходимо создавать соб-

ственные функции для отправки сообщений на серверы, взамен стандартным формам. Подобные приемы будут рассмотрены далее в технологии AJAX.

С указанием метода, объявление формы будет выглядеть как `<form action="add_user.php"method="GET">`. Часто можно увидеть в учебниках, справочниках и кодах сайтов, что названия методов пишут то малыми, то большими буквами. С точки зрения HTML, регистр написания названия метода роли не играет (можно писать как большими, так и маленькими буквами). Однако стандарт RFC 2616 к регистру чувствителен и требует только большие буквы. Соблюдая его требования, будем указывать название метода в верхнем регистре. Будем надеяться, что при изучении более сложного материала по серверным запросам эта привычка упредит некоторые ошибки.

Различие между методами **GET** и **POST** заключается в следующем. **Метод GET** включает данные в состав URI (в адресную строку). Именно пример URI с **GET**-параметрами был приведен в предыдущем разделе, демонстрирующий транспортное кодирование. К адресу сайта <https://www.google.com.ua/> прямо в адресной строке браузера дописываются данные, отправляемые на сервер. В данном случае – поисковый текст. Для разделения адреса и данных в **GET**-запросах применяется символ знака вопроса **?**. В общем случае, если на сайт example.itstep.org нужно передать два параметра **A=1** и **B=2**, то **GET**-вариант запроса будет выглядеть как:

```
example.itstep.org?A=1&B=2
```

Символ вопроса, как уже было сказано, разделяет адрес сайта и параметры, знак **&** разделяет параметры между собой, если их несколько.

Чтобы разобраться как работает **метод POST**, необходимо рассмотреть полный состав сообщений, которыми обмениваются клиент и сервер. Не вдаваясь в глубокие подробности, все же скажем, что, кроме самого запроса, сообщение включает в себя дополнительные данные – заголовки. Рассмотрим их на следующем примере некоторого сообщения, отправляемого браузером на сервер:

```
POST /add_user.php HTTP/1.1
Host: 123.45.67.89
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64;
rv:47.0) Gecko/20100101 Firefox/47.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
username=itstep&pass=MyItstep
```

Первая строка – это, собственно, сам запрос. Команда, которую мы передаем на сервер. Вторая указывает IP-адрес. В третьей передаются данные о типе браузера, подготовившего запрос. Далее следует указание типа содержимого запроса (заголовок **Content-Type**). Приведенное значение соответствует отправке данных из формы. Затем указывается длина сообщения (заголовок **Content-Length**) в символах, после чего следует тело сообщения – параметры, которые мы передаем из формы. Формат передачи данных такой же, как в **GET**-запросах: **имя = значение**, разделение при помощи **&**.

Следует отметить, что, во-первых, заголовков в запросе на самом деле значительно больше, здесь лишь приводится простой пример и, во-вторых, **GET**-запросы являются такими же сообщениями, только первая строка у них содержит команду **GET** и данные содержатся в ней, а не в теле сообщения.

Из-за того что передаваемые **POST**-данные включены в тело сообщения, адресная строка в браузере не меняется. Передача параметров пользователю непосредственно не видна. Однако, если страница построена с применением **POST**-данных, то есть страница появилась после отправки формы, то обновление страницы потребует повторной передачи тех же параметров. Пользователь получит предупреждение: **На странице, которую вы ищете, использовалась введенная вами информация. При возврате на эту страницу может потребоваться повторить выполненные ранее действия. Продолжить?**

Confirm Form Resubmission



The page that you're looking for used information that you entered.
Returning to that page might cause any action you took to be repeated.
Do you want to continue?

Continue

Cancel

С **GET**-данными такой проблемы не возникает, т. к. данные автоматически передаются из самой адресной строки.

Краткое сравнение методов отправки формы представим в виде таблицы:

Метод	Преимущества	Недостатки	Область использования
GET	- Позволяет создать ссылку с параметрами - Упрощает контроль передачи	- Данные легко подсмотреть* - Объемные данные загромождают адрес	Передача коротких данных, автономные ссылки
POST	- Позволяет скрыть передаваемые данные - Не имеет ограничений на длину данных	- Ошибки передачи сложно заметить - Сообщение от браузера при обновлении страницы	Передача изображений, файлов, больших данных

* Часто этот недостаток воспринимается как убийственный для **GET**-метода. Ради справедливости, отметим, что **POST**-данные подсмотреть не намного сложнее, открыв HTTP-сообщение полностью.

Создание формы

Для выполнения практических упражнений по работе с формами и их элементами, необходимо отметить несколько особенностей. Во-первых, форма должна отправлять данные на сервер, который передал клиенту страницу сайта, на которой расположена форма. Поскольку разработку серверной части мы еще не изучали, упражнения будем выполнять в виде отдельных html-файлов, размещенных

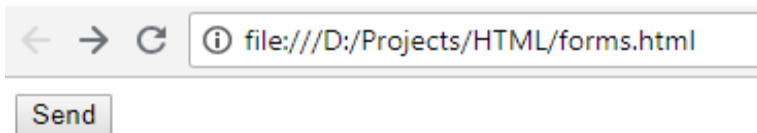
на одном компьютере, не разделенном на *клиент* и *сервер*. Что в таком случае является сервером и куда форма будет отправлять данные?

Для ответа на эти вопросы создадим пустой файл `forms.html` в нашей рабочей папке, например, `D:\Projects\HTML`. Выбор папки произволен, вы можете создать файл в любой папке на любом диске. Но в дальнейшем описании будем отсылаться именно к этой папке и этому диску.

Создадим в файле простейшую форму с единственным элементом, отправляющим форму (детальнее об элементах поговорим в следующем разделе).

```
<form action="/" method='GET'>
<input type='submit' value='Send'>
</form>
```

Как видно, форма обращается к «корню сайта», то есть установлен атрибут `action='/'`. Откроем созданный файл в браузере и увидим примерно следующее:



Анализируя адресную строку, можем сделать выводы:

- для работы с нашей страницей использован файловый протокол (`file:///`). Для сайтов в интернете используются протоколы `http://` и `https://`;
- корнем сайта установлен диск `D:/`.

Убедимся в этом, нажав кнопку отправки формы.

В результате получим следующее:



Index of D:\

	Name	Size	Date Modified
	Book/		7/17/18, 7:58:54 AM
	Document/		7/17/18, 8:04:34 AM
	Download/		7/12/18, 11:13:11 PM
	Music/		7/17/18, 8:02:30 AM
	Photo/		7/17/18, 7:59:09 AM
	Projects/		7/17/18, 7:59:19 AM
	Save/		7/17/18, 7:59:41 AM
	Soft/		7/14/18, 8:54:33 PM
	System Volume Information/		7/13/18, 1:57:32 PM
	Video/		7/17/18, 8:03:52 AM
	Work/		7/17/18, 7:59:30 AM

Появление в конце адресной строки знака вопроса свидетельствует о правильном срабатывании формы. Открытие содержимого диска **D:/** подтверждает, что именно он является корневым элементом нашего «сайта».

Сделаем вывод: не стоит указывать слеш в качестве адресата формы (**action='/'**). Это выражение часто можно встретить в учебниках или справочниках как простейший пример создания формы. Следует помнить, что это применяется именно для сайтов и означает «корень сайта» в котором, обычно, и располагается форма с примером.

При работе с файлами атрибут **action** следует указывать пустым (**action=''**). В таком случае данные будут передаваться по тому же адресу, по которому создана

форма. Как вариант, можно вообще не указывать этот атрибут при объявлении формы. Но, в учебных целях лучше о нем не забывать, все же указывая пустые кавычки.

Замените атрибут `action` нашей формы на пустые кавычки, сохраните файл, обновите страницу в браузере и нажмите **Send**. Убедитесь, что переадресация не происходит и мы попадаем на ту же страницу. Признаком срабатывания формы будет служить появление знака вопроса в конце URI.

Для отправки формы будем всегда использовать метод **GET**, в основном благодаря возможности контроля правильности передачи данных (точнее, подготовки к передаче), так как проверить их получение со стороны сервера мы не сможем (в виду его фактического отсутствия). На данном этапе ограничимся лишь контролем того, что данные формы правильно собраны, подготовлены и включены в состав URI.

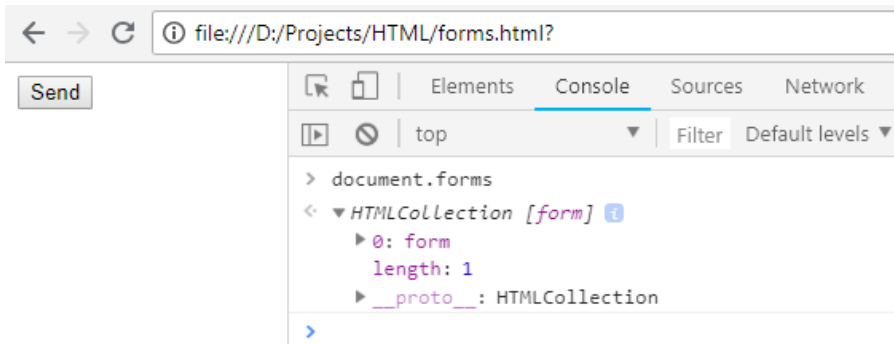
Подытоживая, создание формы при работе с отдельным файлом, с указанием основных атрибутов будет выглядеть как:

```
<formmethod='GET'action=''></form>
```

Форм на одной странице может быть несколько. У каждой может быть свой метод и свой адресат. Для работы с формами при помощи JavaScript, объект `document` имеет специальную коллекцию (массив) `document.forms`. При создании новой формы в эту коллекцию добавляется соответствующий элемент, через который формой можно управлять программным способом. Например, если на данной HTML-странице размещено две формы, то в кол-

лекции `document.forms` будет два элемента (`document.forms[0]` и `document.forms[1]`), отвечающих каждый за «свою» форму. Порядок форм в коллекции соответствует порядку из объявления в HTML-коде.

Откройте в браузере инструменты разработчика (нажав кнопку **F12**), выберите консоль (**Console**), введите в консоли строку `document.forms` и нажмите **Enter**. В полученном ответе можно раскрывать детали, нажимая на треугольные символы.



Скопируйте созданную форму (в файле `forms.html`) и продублируйте ее. Обратите внимание, где появляется вторая кнопка **Send**. Повторите запрос в консоли, убедитесь в расширении коллекции `document.forms`.

Элементы формы

Элементами формы называются объекты общего интерфейса пользователя, значения из которых будут передаваться на сервер при отправке формы. Обычно, это именно те объекты, в которых можно ввести, выбрать или поменять их значение. Исключение, пожалуй, составляет

специальный скрытый элемент, который добавляется к форме программным способом и пользователю для изменения недоступен.

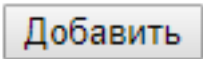
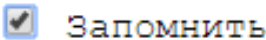
Следует отметить, что не все то, что содержит форма, считается ее элементами. К ним не относятся «статические» объекты HTML – надписи, рисунки, блоки, разрывы строк и прочие средства разметки. Тем не менее, заключенные между тегами открытия и закрытия формы – все объекты являются дочерними для формы.

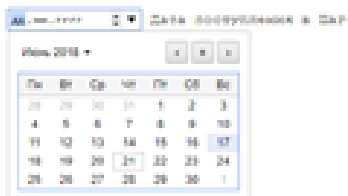
Для того чтобы отличать элементы формы (с изменяемыми значениями) и все дочерние объекты, в объекте `form` предусмотрены две разные коллекции:

- коллекция `elements` содержит только элементы формы;
- коллекция `children` – все дочерние объекты.

Большинство элементов формы создаются тегом `input` и отличаются различными значениями атрибута `type`. Кроме них есть и несколько обособленных элементов.

Основные из них приведены в следующей таблице.

Тег	Описание	Примерный вид
<code><input type=button></code>	Кнопка	
<code><input type='text'></code>	Ввод данных пользователем	
<code><input type='checkbox'></code>	«Флажок выбора»	

Тег	Описание	Примерный вид
<code><input type='radio'></code>	Радиокнопки – один выбор из заданного набора	☒ Да ☐ Нет
<code><input type='file'></code>	Диалог открытия файла	Выберите файл Файл не выбран
<code><input type='submit'></code>	Кнопка отправки формы	Сохранить
<code><input type='password'></code>	Поле ввода пароля (без отображения символов)	
<code><input type='hidden'></code>	Скрытое поле	Не отображается (но передается)
<code><input type='date'></code>	Выбор даты	
<code><input type='number'></code>	Поле для ввода числа	Забронировать 1 1 шт
<code><select></code>	Выпадающий список выбора	Тип Мобильный ▼ Мобильный Домашний Рабочий
<code><textarea></code>	Поле для ввода большого текста	Ваш комментарий

Приведенный список в таблице не является полным справочником, а лишь отображает наиболее популярные элементы. Полный список, доступный в текущем стандарте HTML, можно узнать на сайте [WorldWideWebConsortium \(W3C\)](https://www.w3.org) по ссылке: <https://www.w3.org> или на альтернативных ресурсах справочного или учебного характера.

Элементы соотносятся с формой и попадают в ее коллекции, если они создаются между открывающим (`<form>`) и закрывающим (`</form>`) тегами формы. Если элемент создается в другом месте, он может быть соотнесен с формой при помощи специального атрибута `form`. В таком случае форме необходимо присвоить идентификатор (`id`) и для элемента, созданного вне тега `<form id="telForm">`. Для привязки к форме необходимо указать указанный атрибут `<input form="telForm" ...>`.

Для иллюстрации работы с формами создадим страницу, обеспечивающую добавление новой записи (контакта) в телефонную книгу. Она должна содержать инструменты для ввода имени абонента и номеров телефонов, с указанием их типов.

Изначально предлагается ввод одного номера, но по нажатию кнопки **One more phone** должны добавляться поля для ввода нового номера и его типа. Также предполагается наличие возможности выбора приоритетного номера.

Примерный вид страницы, которую мы хотим получить, представлен на следующем рисунке.

Add new contact

Name

Phone number	123456789	Phone type	Home	Priority	☐
Phone number	987654321	Phone type	Cellular	Priority	☒
Phone number	Enter phone number	Phone type	Work	Priority	☐

Приведем HTML-код страницы и прокомментируем его ниже. В созданном ранее файле [forms.html](#) полностью замените содержимое, скопировав или набрав этот код. Файл также доступен в папке [Sources_5.1](#), архив которой приложен к pdf-файлу данного урока.

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content=
    "text/html; charset=UTF-8">
<title>Phone book</title>
<style>
body{font-family:"Courier New", Courier,
    monospace;}
</style>
</head>
<body>
<h2>Add new contact</h2>
<form method='GET'>
<b>Name</b><input type="text" placeholder=
    "Enter name"/><br/><br/>
<input type="submit" value="Save"/>
<input type="button" value="One more phone"
onclick="add_click()" style="margin-left:50px" />
<br/><br/>
```

```

        Phone number <input type="text"
        name="phone" id="ph" placeholder=
        "Enter phone number" />
        Phone type <select name="type">
<option value="1" selected>Cellular</option>
<option value="2">Home</option>
<option value="3">Work</option>
</select>
        Priority <input type="radio"
        name="main" value="1" checked />
</form>
</body>
</html>

```

Согласно поставленным условиям, изначально в форме создается одна строка для ввода данных. Поэтому в приведенном коде создается только одна группа элементов для номера и его типа. Добавление новых номеров будет реализовано как реакция на нажатие кнопки с вызовом функции `add_click()` в следующей главе.

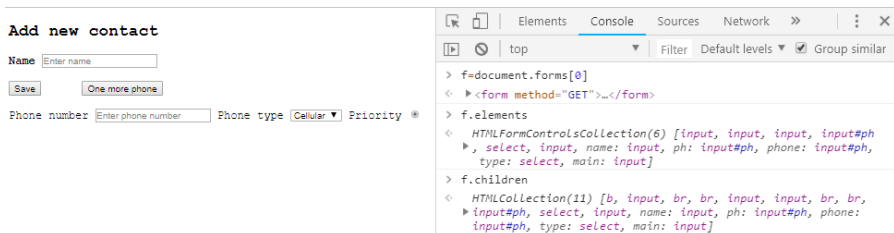
Сохраните изменения в файле `forms.html` и обновите соответствующую страницу в браузере. Убедитесь, что после обновления страницы ее вид соответствует приведенному на рисунке, только с одной строкой для ввода данных.

На данном этапе можно посмотреть на различие коллекций элементов формы и ее дочерних элементов. Вызовем консоль разработчика (нажав **F12**), если она была закрыта после предыдущего упражнения. Запросим первую (нулевую в массиве) форму из коллекции документа

и сохраним ее в переменной `f`, введя в консоль строку:

```
f=document.forms[0]
```

Отообразим коллекции формы. Набираем в консоли `f.elements` (и нажимаем **Enter**), затем `f.children` (и снова **Enter**). Убеждаемся, что в первой коллекции 6 элементов (5 `input` и 1 `select`), тогда как во второй – 11, включая разметочные элементы `b` и `br` (см. рис. далее). Обратите внимание, что элементы формы входят в обе коллекции, то есть доступ до них возможен при помощи любой из них. Детальное содержание коллекций можно посмотреть, нажимая на раскрывающие «треугольники».



Нажатие на кнопку **Добавить номер** пока не приводит ни к каким действиям, при этом кнопка **Save** отправляет форму (если быть точным, то это не кнопка, а элемент `submit`, автоматически отправляющий форму). При нажатии на нее в адресной строке браузера появляются данные, отделенные от адреса знаком вопроса. Заполните поля, нажмите **Сохранить** и убедитесь в переносе введенных данных в состав URI.

Если вы правильно переписали (или скопировали) код страницы и внимательно посмотрели на те данные, которые попали в адресную строку после нажатия **Save**,

то, наверняка, обратили внимание, что в адресную строку не попала информация о введенном имени.

Форма передает данные только из тех элементов, которым дано программное имя (определен атрибут `name`). В нашем случае элементу ввода имени программное имя не присвоено. Поэтому, введенные данные игнорируются. Такая же ситуация с кнопками – они тоже элементы формы (как было видно из коллекций), но их значения не передаются.

Попробуйте задать атрибут `name` полю ввода имени, переписав строку в виде:

```
<b>Name</b><input type="text" placeholder=
"Enter name" name="name"/><br/><br/>
```

и убедитесь, что после этого данные об имени komponуются в состав URI. Также добавьте атрибут `name` кнопке и элементу `submit`, убедитесь, что в таком случае попадают в перечень передачи и их значения, которыми являются надписи на кнопках.

Рассмотрим еще один пример, иллюстрирующий дополнительные возможности управления стилями как самой формы, так и ее элементов. Возьмем в качестве идеи форму авторизации **Logbook** (см. рис. выше) и сделаем ее значительно упрощенную версию. Создайте в вашей рабочей папке новый файл `form2.html`.

Саму форму стилизуем как блочный элемент (`display: block`) с фоновым цветом синей гаммы (`background-color:#5599EE`). Зададим ей ширину (`width:400px`), высоту (`height:160px`) и отступ внутренних элементов от границы формы (`padding:20px`). Сохраним стилевые описания

под идентификатором `#logForm`. Получим следующий HTML-документ.

```
<!doctype html>
<html>
<head>
  <style>
    #logForm{
      background-color:#5599EE;
      display:block;
      height:160px;
      padding:20px;
      width:400px;
    }
  </style>
</head>
<body>
  <form method="GET" action="" id="logForm">

    </form>
</body>
</html>
```

Откройте полученный документ в браузере и убедитесь, что на пустой странице отображается прямоугольник синего оттенка.

Добавим надпись **LogForm** в левую часть формы. В теле документа между открытым и закрытым тегами формы вставим блок `<div id="formText">LogForm</div>`. В стилевых определениях зададим ему белый цвет текста (`color:white`), размер шрифта (`font-size:40px`), полужирное начертание (`font-weight:bold`). Для того чтобы элементы формы могли размещаться правее надписи (об-

текать), зададим ей левое плавание (**float:left**), установим ширину, близкую или равную половине ширины формы (**width:200px**) и высоту, как у формы (**height:200px**). Не забываем, что в расчет высоты также входит внутренний отступ (**padding**), добавляющийся как сверху, так и снизу. Полное стилевое описание блока примет вид:

```
#formText{
    color:white;
    float:left;
    font-size:40px;
    font-weight:bold;
    height:200px;
    width:200px;
}
```

Дальше размещаем сами элементы формы: селектор выбора города, поля ввода логина и пароля, а также кнопка отправки формы. Все они имеют схожий стиль скругленного блока, только кнопка отправки отличается цветом и выравниванием текста. Создадим стилевой класс, отвечающий за скругленную форму элементов. Большинство стиливых определений уже повторялись и не комментируются. Основу округления составляют свойства радиуса границы (**border-radius:15px**) и отключения прорисовки самой границы (**border-width: 0**).

```
.rounded{
    border-radius:15px;
    border-width:0;
    display:block;
    height:30px;
```

```
margin:10px 0;
padding:0;
width:170px;
}
```

Для выделения кнопки отправки зеленым цветом, с выравниванием по центру, создадим стилевое определение следующего содержания.

```
#enter{
    background-color:green;
    color:white;
    text-align: center;
}
```

Теперь создадим сами элементы формы. В теле формы создаем селектор выбора городов. Для примера ограничимся тремя. Следует отметить, что селектор не поддерживает подсказку (**placeholder**), поэтому применяется технология создания дополнительного поля выбора (опции) с атрибутами скрытости (**hidden**) и недоступности для выбора (**disabled**). В то же время, эта опция устанавливается как выбранная изначально (**selected**), а ее значение задается пустым (**value=""**), что позволит в будущем проводить его контроль программными методами. Опция-подсказка размещена в селекторе на последнем месте. Итоговый код для селектора примет вид:

```
<select class="rounded" name="city">
    <option value="Kiev">Kiev</option>
    <option value="Nikolaev">Nikolaev
        </option>
```

```

        <option value="Odessa">Odessa</option>
        <option value="" disabled selected
                                hidden> City</option>
    </select>

```

Создание остальных элементов не содержит хитростей. Для всех указываем класс **rounded**, отвечающий за округлую рамку. Для кнопки отправки формы – дополнительный идентификатор (**enter**), задающий ее особенности. HTML-код элементов будет выглядеть следующим образом:

```

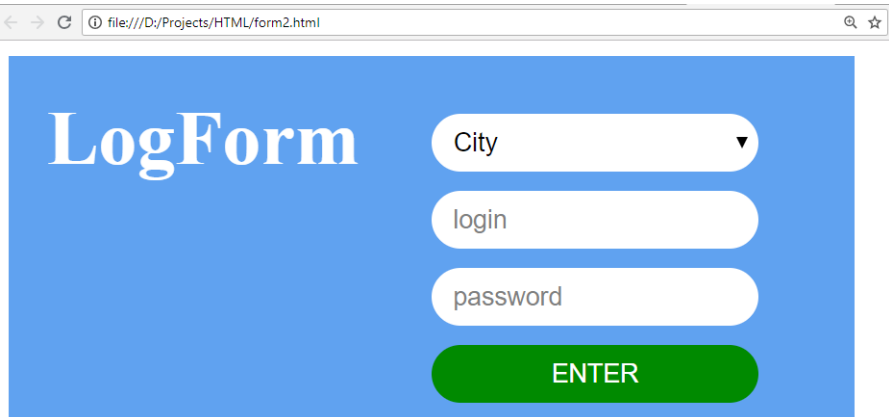
<input type="text" name="login" class="rounded"
placeholder="login"/>
<input type="password" name="password" class=
"rounded" placeholder="password"/>
<input type="submit" id="enter" class="rounded"
value="ENTER"/>

```

Полный код файла **form2.html** доступен в папке **Sources_5.1**, архив которой приложен к pdf-файлу данного урока.

После завершения оформления кода сохраните результат и откройте файл **forms2.html** в браузере. Если он уже открыт – обновите страницу, нажав клавишу **F5** или кнопку на интерфейсе браузера.

В итоге наша форма в браузере будет иметь следующий вид:



The screenshot shows a web browser window with the address bar displaying `file:///D:/Projects/HTML/form2.html`. The main content area has a blue background. On the left, the text "LogForm" is written in a large, white, serif font. To the right of the text, there is a form with three input fields stacked vertically: a dropdown menu with "City" and a downward arrow, a text input field with "login", and another text input field with "password". Below these fields is a green button with the text "ENTER" in white, uppercase letters.

Заполните поля формы, нажмите кнопку **ENTER**. Убедитесь, что введенные данные попадают адресную строку браузера.

Включите консоль разработчика и проведите инспекцию коллекций `document.forms`, `document.forms[0].elements`, `document.forms[0].children`, как было описано выше.

Домашнее задание

1. Дополните созданную форму телефонного справочника возможностью:
 - ввода даты рождения (`input type="date"`);
 - выбора файла с фотографией (`input type="file"`);
 - ввода электронной почты (`input type="email"`);
 - указания сайта (`input type="url"`);
 - ввода «секретного слова» (`input type="password"`);
 - выбора любимого цвета (`input type="color"`);
 - отметки Напоминать о дне рождения (`input type="checkbox"`);
 - сброса введенных данных (`input type="reset"`).
2. Обратите внимание, в каком формате передаются данные от каждого из элементов формы (в адресной строке браузера).
3. Примените стилевые определения для элементов формы на свой вкус.

Вопросы к домашнему заданию

1. Что такое HTML-формы? Для каких целей они нужны?
2. Как получить программный доступ к форме? Ко всем формам?
3. Что такое элементы формы? Каким образом можно получить к ним доступ?

4. Какими способами можно указать, что элемент относится к форме?
5. При каких условиях данные из элемента передаются с отправкой формы?

Программирование элементов формы

Еще раз о коллекциях элементов

Вернемся к форме телефонной книги, созданной нами ранее, и рассмотрим более детально ее структуру с различных точек зрения. Напомним код этой формы и ее внешний вид:

```
<form method='GET'>
  <b>Name</b> <input type="text" placeholder=
    "Enter name" name="name" /><br/><br/>
  <input type="submit" value="Save"/>
  <input type="button" value="One more phone"
    onclick="add_click()" style="margin-left:50px"/>
<br/><br/>
  Phone number <input type="text" name="phone"
    id="ph" placeholder="Enter phone number" />
  Phone type <select name="type">
    <option value="1" selected>Cellular</option>
    <option value="2">Home</option>
    <option value="3">Work</option>
  </select>
  Priority <input type="radio" name="main"
    value="1"
    checked />
</form>
```

Name

Phone number

Phone type

Priority ☒

Как мы уже знаем, существуют две коллекции для хранения информации о структуре формы – **elements** и **children**. Первая хранит только элементы самой формы, вторая – все дочерние элементы. Однако существует еще одна коллекция – **childNodes**, хранящая все дочерние узлы. Чтобы разобраться, в чем заключаются отличия между этими коллекциями, проведем их детальный анализ.

Для анализа коллекций создадим средства для их отображения на странице браузера. Конечно, можно проводить анализ при помощи средств консоли разработчика, как мы делали ранее, но для многократного использования удобнее создать специальные кнопки. После обновления страницы в консоли придется повторять действия по доступу к форме и ее коллекциям, тогда как нажать на кнопку будет гораздо проще.

Создадим после объявления формы дополнительную кнопку, которая будет обеспечивать анализ коллекции **elements**, а также добавим абзац (параграф) для вывода данных на страницу. Добавлять элементы надо именно после формы (после закрытого тега **</form>**), иначе интересные нас коллекции формы будут меняться после добавления новых кнопок в саму форму. Фрагмент кода создания кнопки и абзаца приведен ниже, полный код доступен в папке **Sources_5.2**, архив которой приложен к pdf-файлу данного урока (файл **form5_2-1.html**).

```
<button onclick="showElements()">Elements</button>  
<p id="out"></p>
```

В разделе описания скриптов добавим функцию

`showElements()`, которая будет отвечать на нажатие кнопки.

```
function showElements(){
    var e = document.forms[0].elements,
    p=document.getElementById("out");
    p.innerHTML = "";
    for ( var i=0; i<e.length; i++){
        p.innerHTML += e[i].tagName + " - " +
            e[i].name + " - " + e[i].value + "<br>"
    }
}
```

Функция запрашивает коллекцию элементов первой формы документа (`document.forms[0].elements`), получает доступ к абзацу, предназначенному для вывода информации (`p = document.getElementById("out")`), после чего проходит циклом по коллекции и выводит на страницу данные об имени тега (`tagName`), программном имени (`name`) и значении (`value`) каждого элемента. Для разделения выводимых данных используется знак - с пробелами с двух сторон.

После нажатия на кнопку на странице, появится следующее содержимое:

Elements

```
INPUT - name -
INPUT - - Save
INPUT - - One more phone
INPUT - phone -
SELECT - type - 1
INPUT - main - 1
```

Эти 6 элементов мы уже наблюдали в консоли в предыдущей главе. Здесь видно, что, например, первый эле-

мент `input` имеет имя `name` но не имеет значения (пустое пространство после второго `-`). Это пустая строка ввода для имени абонента. Можете ввести в нее произвольные данные, снова нажать на кнопку **Elements** и убедиться в правильности наших выводов.

Второй элемент, наоборот, не имеет имени (это видно по пустому пространству между знаками `-`), но имеет значение **Save**. Это кнопка (точнее, элемент `submit`) отправки формы. Мы специально не давали имя, чтобы исключить этот элемент из перечня передаваемых на сервер данных.

В то же время, последние элементы имеют и имя, и значение. Это достигнуто применением атрибутов `selected` для списочного элемента (`select`) и `checked` – для радиокнопки (последний `input`).

Аналогичным образом проведем анализ коллекции **children**. Добавим соответствующую кнопку сразу после кнопки **Elements**, абзац для вывода будем использовать тот же, новый создавать не будем.

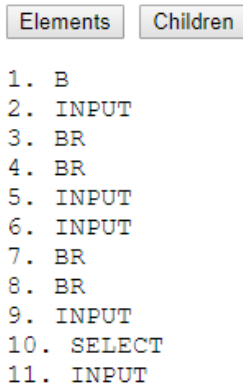
```
<button onclick="showChildren()">Children</button>
```

В секции `<script>` опишем функцию, отвечающую за нажатие кнопки.

```
function showChildren(){
    var c = document.forms[0].children,
        p = document.getElementById("out");
    p.innerHTML = "";
    for ( var i=0; i<c.length; i++){
        p.innerHTML += (i+1) + ". " +
            c[i]. tagName + "<br>"
    }
}
```

Содержание функции практически полностью повторяет предыдущую, но, во-первых, в качестве анализируемой коллекции используется `children`. Во-вторых, выводим только порядковый номер элемента и имя его тега, поскольку значения (`values`) актуальны только для элементов формы и имени (`name`) также даны не всем элементам.

Сохраним измененный файл и обновим страницу в браузере. После нажатия на кнопку `Children` появится следующий текст:



The screenshot shows a web interface with two buttons, 'Elements' and 'Children'. Below the buttons is a list of 11 HTML elements, numbered 1 through 11. The elements are: 1. B, 2. INPUT, 3. BR, 4. BR, 5. INPUT, 6. INPUT, 7. BR, 8. BR, 9. INPUT, 10. SELECT, and 11. INPUT. The element 'B' is bolded in the original image.

Это тоже знакомые нам 11 элементов, соответствующие структуре формы. Первый объект **B** – это выделенная жирным шрифтом надпись `Name` перед полем ввода имени. Объекты `BR` – это разрывы строк, примененные нами для оформления.

Аналогичным образом, после кнопки `Children` добавим третью кнопку, анализирующую наиболее полную коллекцию `childNodes`:

```
<button onclick="showNodes()">Nodes</button>
```

Ее функция нажатия будет иметь вид:

```
function showNodes(){
    var n = document.forms[0].childNodes,
        p = document.getElementById("out");
    for ( var i=0; i<n.length; i++){
        p.innerHTML += (i+1) + ". " + n[i].
                               nodeName + "<br>"
    }
}
```

В данной коллекции не все элементы имеют имя тега, поэтому для вывода использовано поле **nodeName** – имя узла, наиболее общее представление объекта DOM. Напомним, что полный код со всеми функциями доступен в папке **Sources_5.2**, архив которой приложен к pdf-файлу данного урока (файл **form5_2-1.html**).

Сохраняем изменения, обновляем страницу и нажимаем на кнопку **Nodes**. В результате получим следующее:

Elements	Children	Nodes
1. #text		
2. B		
3. #text		
4. INPUT		
5. BR		
6. BR		
7. #text		
8. INPUT		
9. #text		
10. INPUT		
11. #text		
12. BR		
13. BR		
14. #text		
15. INPUT		
16. #text		
17. SELECT		
18. #text		
19. INPUT		
20. #text		

К имеющимся 11-ти элементам коллекции `children` добавлены 9 элементов с именем узла `#text`. Это текстовые элементы, связанные с реальным оформлением страницы, в том числе и не совсем преднамеренным.

Так, первый узел `#text` соответствует разделительному элементу между открывающим тегом формы и началом надписи `Name`. Обратите внимание, что в исходном коде эта надпись (`<b>Name</b>`) формируется с новой строки после тега `<form>` и содержит отступ в начале строки для красоты оформления кода:

```
<form method='GET'> ← разрыв строки
```

```
<b>Name</b>
```

↑ отступ

Вызовем консоль браузера (F12) и запросим детализацию этого элемента. Введем `f=document.forms[0]` для получения самой формы (нажимаем `Enter`). Дальше вводим `t=f.childNodes[0]` для детализации первого (нулевого) элемента коллекции `childNodes` формы. После нажатия `Enter` увидим `#text`. Нажмем на раскрытие детальных данных (треугольник слева от надписи) и получим список полей данного объекта (см. рис. на стр. 37).

Его содержимое дублируется в нескольких полях (`data`, `nodeValue`, `textContent`, `wholeText`) и представляет собой перевод строки (символ ↵), после которого следуют несколько пробелов, соответствующих отступу в HTML-коде формы.

Add new contact

Name

Phone number

Phone type Cellular ▾

Priority ⊙

```

Elements Console Sources
top Filter
> f=document.forms[0]
> <form method="GET">...</form>
> t=f.childNodes[0]
< #text
  assignedSlot: null
  baseURI: "file:///D:/Projects/HTML/"
  childNodes: NodeList []
  data: ""
  firstChild: null
  isConnected: true
  lastChild: null
  length: 9
  nextElementSibling: b
  nextSibling: b
  nodeName: "#text"
  nodeType: 3
  nodeValue: ""
  ownerDocument: document
  parentElement: form
  parentNode: form
  previousElementSibling: null
  previousSibling: null
 .textContent: ""
  wholeText: ""

```

Как вы, наверняка, помните, любую последовательность пробельных и разрывных символов браузер воспринимает как один пробел, который отображается в режиме сбора строчных элементов и игнорируется при сборке блочных или плавающих элементов.

Можем в этом убедиться, исследовав второй элемент **#text** в коллекции **childNodes**. Как видно, он располагается на 3-м месте в коллекции между элементами **B** и **INPUT**. Присмотревшись к HTML-коду формы, обратим внимание на пробел после закрывающего тега **** и открывающего **<input...>** в строке **Name <input type="text">** (для наглядности разрыв увеличен).

Уберите этот пробел в HTML-коде, чтобы теги следовали друг за другом неразрывно (**<input...>**). Сохраните файл, обновите страницу в браузере. Убедитесь, что между надписью **Name** и полем ввода пробел исчез:

Name

А при нажатии кнопки **Nodes** отображаются 19 элементов без исследованного нами элемента **#text** на 3-й позиции.

Можете повторить описанные действия для исключения первого элемента **#text**, удалив пробелы и разрыв строки между тегами формы и надписи **Name**. Убедитесь, что элемент из коллекции исчезает, однако внешний вид страницы никак не изменяется, поскольку браузер еще не перешел в режим сборки строчных элементов и пробелы игнорируются.

Элементы **#text**, кроме разметки HTML-кода, соответствуют также надписям, не оформленным в собственные теги. Например, 14-й узел **#text** (в первоначальной коллекции из 20-ти элементов) соотносится с надписью **Phone number**. Это можно проверить, повторив описанные выше действия в консоли браузера (`f=document.forms[0]`, `t=f.childNodes[13]`).

Кроме собственно надписи, этот элемент содержит также разрыв строки и отступ перед и после текста. То есть элемент собирает в себе все, что находится между закрытым тегом (в данном случае `<br/>`) и следующим открытым (`input` – для ввода телефона). Повторно обратите внимание, что для надписи **Name**, заключенной в тег `<b>`, создается не узел **#text**, а объект **B**.

Подведем некоторые итоги по исследованию структуры формы и ее коллекций.

Все средства оформления HTML-кода в виде разрыва строк, отступов (пробелы или табуляции) между тегами, в конце концов, попадают в DOM-структуру страницы. За счет них увеличиваются коллекции дочерних элементов и,

естественно, количество занятой памяти и трудоемкость обновления страницы или ее части. В нашем простом примере таких элементов у формы почти половина (9 из 20-ти), что не позволяет ими пренебрегать, оправдываясь, что их не так уж и много. Конечно, отказываться от красивого оформления HTML-кода не стоит, по крайней мере, на этапе разработки. Но при выводе готового продукта можно задуматься и об оптимизации.

Да и стиль программирования может содержать оптимизированные конструкции. Например, если переместить пробел в надписи **Name** внутрь тега `<b>`, то есть записать `<b>Name </b><input...` вместо `<b>Name</b> <input...` (разрывы все так же увеличены для наглядности), то лишнего элемента `#text` создано не будет, а отступ между надписью и полем ввода сохранится.

Пробелы, находящиеся внутри тегов (`<b>Name </b>`), а также пробелы, разделяющие их атрибуты (`<form method='GET'>`), не переносятся в структуру DOM в виде отдельных объектов. Это может быть использовано для задач оформления кода и упреждения создания новых элементов.

Надписи, оформленные без собственных тегов, попадают в специальные объекты `#text`. С одной стороны, это может служить рекомендацией все же обрамлять надписи тегами для более предсказуемой структуры документа. С другой стороны, расширение формы путем создания дополнительных надписей в процессе выполнения программных скриптов (как говорится, «на лету») потребует от нас использование специальных узлов `#text`, не соответствующих ни одному стандартному тегу.

Добавление элементов

Разобравшись в особенностях структуры формы и ее элементов, поставим себе задачу реализовать функциональность добавления нового поля для ввода дополнительного номера телефона по нажатию кнопки **Добавить номер**. Изначально для этого мы заложили обработчик события нажатия кнопки **One more phone** в виде функции `add_click()` (`onclick="add_click()"`).

Для того чтобы добавить новый номер, необходимо создать несколько дополнительных элементов и включить их в состав формы. Проанализируем детально состав одной строки ввода номера телефона:

1. Сначала идет разрыв (`<br>`), обеспечивающий переход на новую строку.
2. Затем надпись: **Phone number**.
3. Далее, элемент **INPUT** для ввода номера телефона.
4. Надпись: **Phone type**.
5. Селектор выбора типа номера.
6. Надпись: **Priority**.
7. Радиокнопка для отметки основного номера.

Также следует напомнить, что данные формы отправляются только из тех элементов, которым заданы имена (атрибут **name**). Причем для разных полей имена должны быть разными, чтобы была возможность разделить данные на стороне сервера. Исключение составляют только радиокнопки выбора основного номера, для которых имя

должно быть одинаковым, чтобы обеспечить их зависимость между собой.

Чтобы формировать различные имена для новых элементов ввода, введем глобальную переменную-счетчик `phoneCounter`, значение которой будем приписывать к именам полей. В момент первого запуска функции эта переменная должна быть инициализирована. Используем проверку `if(typeof phoneCounter == 'undefined')` для определения первого запуска функции, при котором данная переменная должна быть установлена в `1`. После этого счетчик можно увеличивать обычным способом (`phoneCounter++`). Соответственно, начало функции будет выглядеть следующим образом (полный код со всеми функциями доступен в папке `Sources_5.2`, архив которой приложен к pdf-файлу данного урока (файл `form5_2-2.html`)).

```
<script>
function add_click(){
    if (typeof phoneCounter == 'undefined')
        phoneCounter = 1;
    phoneCounter++;
    var f = document.forms[0];
```

В переменной `f` сохранен первый элемент коллекции форм документа, то есть наша основная форма, к которой мы будем добавлять элементы.

Для создания новых элементов в JavaScript предусмотрено несколько функций. Функция `document.createElement` создает элемент по имени тега, например, разрыв строки можно создать вызовом `document.createElement('br')`,

а элемент ввода – `document.createElement('input')`. Другая функция `document.createTextNode` служит для создания узлов типа `#text`, для которых имя тега не задано. В качестве аргумента функция принимает текст, который будет отображаться. Например, `document.createTextNode('Phone number')`.

Следует отметить, что описанные функции только создают объекты заданного типа, но не включают их в структуру документа. Для этих целей необходимо воспользоваться методом `appendChild` того объекта, к которому элементы добавляются. В нашем случае – формы, сохраненной в переменной `f`.

Таким образом, добавление разрыва строки в форму будет выглядеть следующим образом:

```
var b = document.createElement('br');  
f.appendChild(b);
```

А добавление надписи **Phone number**:

```
var t = document.createTextNode('Phone number ');  
f.appendChild(t);
```

Добавление поля ввода нового номера потребует дополнительных действий. Новому созданному элементу типа `input` необходимо явно указать тип `'text'` (в предыдущей главе было показано, что в зависимости от типа элемент может вести себя совершенно по-разному). Также нужно сформировать имя с учетом переменной-счетчика и, по аналогии с уже существующим полем, добавить подсказку (`placeholder`) **Enter phone number**.

В итоге, создание поля и его добавление к форме будет иметь следующий код:

```
var phoneInput = document.createElement('input');
phoneInput.type = 'text';
phoneInput.name = 'phone' + phoneCounter;
phoneInput.placeholder = 'Enter phone number';
f.appendChild(phoneInput);
```

Добавление надписей **Phone type** и **Priority**, а также радиокнопки (разновидность **input**) полностью аналогичны описанным выше примерам.

Остановимся подробнее на добавлении списка выбора типа телефонного номера. Напомним его структуру:

```
<select name="type">
  <option value="1" selected>Cellular</option>
  <option value="2">Home</option>
  <option value="3">Work</option>
</select>
```

Для того чтобы повторить такой же элемент, необходимо создать элемент типа **select**, три элемента типа **option**, заполнить их данными и добавить к родительскому **select**. Однако, в данном случае можно воспользоваться еще одной функцией, позволяющей создавать копии (клоны) элементов – **cloneNode**. Поскольку новый элемент будет отличаться только именем, этот путь будет более простой – создать клон и поменять имя, вместо пошагового воссоздания нового элемента с абсолютно одинаковой внутренней структурой.

Для того чтобы создать клон элемента, нужно получить к нему доступ. Сделать это можно несколькими способами, воспользуемся самой короткой коллекцией формы `elements`. Так как список имеет имя (`name='type'`), доступ к нему можно получить, обратившись к коллекции по этому имени `f.elements['type']`. Это значение можно сохранить в программную переменную (`selector`), после чего вызвать метод клонирования для создания новой переменной `var newSelector = selector.cloneNode(true)`. Аргумент `true` в вызове функции указывает, что нужно скопировать все содержимое элемента полностью. Остается поменять имя новому объекту и добавить его к форме.

Полный код функции приведен ниже. Напомним, что полный код со всеми функциями доступен в папке `Sources_5.2`, архив которой приложен к pdf-файлу данного урока (файл `form5_2-2.html`).

```
function add_click(){
    if (typeof phoneCounter == 'undefined')
        phoneCounter = 1;
    phoneCounter++;
    var f = document.forms[0];

    var b = document.createElement('br');
    f.appendChild(b);

    var t = document.createTextNode
        ('Phone number ');
    f.appendChild(t);

    var phoneInput = document.
        createElement('input');
    phoneInput.type = 'text';
```

```

phoneInput.name = 'phone' + phoneCounter;
phoneInput.placeholder =
    'Enter phone number';
f.appendChild(phoneInput);

var t2 = document.createTextNode
    (' Phone type ');
f.appendChild(t2);

var selector = f.elements['type'];
var newSelector = selector.cloneNode(true);
    console.log(newSelector);
newSelector.name = 'type' + phoneCounter;
f.appendChild(newSelector);

var t3 = document.createTextNode
    (' Priority ');
f.appendChild(t3);

var mainRadio = document.
    createElement('input');
mainRadio.type = 'radio';
mainRadio.name = 'main';
mainRadio.value = phoneCounter;
f.appendChild(mainRadio);
}

```

Сохраните полученный код и обновите страницу браузера. Теперь при нажатии на кнопку **One more phone** появляются дополнительные «строки» ввода. Эти строки появляются перед кнопками исследования коллекций, т. к. они вынесены за форму, а строки добавляются к самой форме. Убедитесь, что при нажатии на эти кнопки новые элементы (**phone2, type2, phone3, type3,...**) отобража-

ются в составе коллекций. Проверьте, что радиокнопки работают корректно и списки выбора не зависят друг от друга.

Add new contact

Name

Phone number	Enter phone number	Phone type	Cellular	Priority	☐
Phone number	Enter phone number	Phone type	Home	Priority	☒
Phone number	Enter phone number	Phone type	Work	Priority	☐

```

INPUT - name -
INPUT - - Save
INPUT - - One more phone
INPUT - phone -
SELECT - type - 1
INPUT - main - 1
INPUT - phone2 -
SELECT - type2 - 1
INPUT - main - 2
INPUT - phone3 -
SELECT - type3 - 1
INPUT - main - 3

```

Введите тестовые данные и нажмите **Save**. Проверьте правильность и полноту переноса введенных данных в адресную строку браузера.

Подробнее о клонировании и вставке

Как мы увидели, создание комплексных элементов со сложной внутренней структурой можно упростить, применяя способ клонирования. Следует помнить, что клон создает полную копию объекта, в том числе имена

и значения всех составных блоков. Если в новом элементе необходимо поменять все эти имена и значения, то механизм клонирования получится ничем не проще или короче, чем воссоздание объекта шаг за шагом по одному элементу. Если же изменений в новом комплексе немного, то клонирование действительно упростит код.

В нашем случае, новая «строка» для ввода номера полностью повторяет предыдущую, за исключением новых имен полей для номера телефона и его типа. Можно ожидать, что при помощи клонирования добавление элементов упростится.

Чтобы воспользоваться клонированием, элементы, которые добавляются, надо поместить в общий контейнер (`<div>`). Поменяем определение формы:

```
<form method='GET'>
  <b>Name </b><input type="text" placeholder=
    "Enter name" name="name" />
  <div style="margin:10px 0;">
    Phone number <input type="text"
      name="phone" id="ph" placeholder=
        "Enter phone number" />
    Phone type <select name="type">
      <option value="1" selected>Cellular
        </option>
      <option value="2">Home</option>
      <option value="3">Work</option>
    </select>
    Priority <input type="radio" name="main"
      value="1" checked />
  </div>
  <input type="button" value="One more phone"
    onclick="add_click()" />
```

```
<input type="submit" value="Save" style=
    "margin-left:50px" />
</form>
```

Кнопки **One more phone** и **Save** перенесены вниз, «строка» с номером, типом и радиокнопкой помещена в блок **<div>**. Так как блочный элемент сам по себе создается с новой строки, разрывы **
** в новой форме не используются, а отступ между строками задается стилем блока-оболочки (**style="margin:10px 0;"**).

В новой форме еще больше проявляется разница между коллекциями **elements** и **children**. Запросите состав этих коллекций в консоли. В коллекции **elements** видны все те же 6 элементов – без группировок в отдельные блоки. Тогда как в коллекции **children** всего 5 элементов, третьим из которых идет блок **div**, в котором сгруппированы несколько элементов формы. Об этом следует помнить при работе с формами: коллекция **elements** не хранит информацию о группировке элементов в блоки, **children** – хранит.

Поменяем также функцию **add_click()**. Полный код со всеми функциями доступен в папке **Sources_5.2**, архив которой приложен к pdf-файлу данного урока (файл **form5_2-3.html**).

```
function add_click(){
    if (typeof phoneCounter == 'undefined')
        phoneCounter = 1;
    phoneCounter++;
    var f = document.forms[0];
```

```
var line = f.children[2];
var newLine = line.cloneNode(true);
newLine.children[0].name = 'phone' +
                           phoneCounter;
newLine.children[0].value = '';
newLine.children[1].name = 'type' +
                           phoneCounter;
newLine.children[2].checked = false;
f.insertBefore(newLine,f.
                children[phoneCounter+1]);
}
```

Первые строки совпадают с предыдущей редакцией функции и были прокомментированы ранее.

Получаем доступ к блоку `<div>`, отвечающему за «строку» ввода номера. Как уже было проверено в консоли браузера, этот блок находится на третьей позиции коллекции `children` формы. Соответственно, получить к нему доступ можно при помощи строки `var line = f.children[2]`.

Затем создается клон этого блока (`var newLine = line.cloneNode(true)`).

После клонирования заменяем имена полей ввода с учетом счетчика, а также сбрасываем значения поля ввода имени (`newLine.children[0].value = ''`) и радиокнопки (`newLine.children[2].checked = false`), чтобы при клонировании создавались пустые поля, и радиокнопка была неотмеченной.

Последней строкой кода новый элемент добавляется к форме. Только вместо рассмотренного ранее метода `appendChild` использован метод `insertBefore`. Метод

`appendChild` добавляет элемент в конец коллекции, тогда как `insertBefore` может вставить в любое место. Поскольку мы перенесли кнопки добавления и отправки вниз формы, применение `appendChild` будет добавлять строки после кнопок, что не совсем правильно с точки зрения дизайна.

Для того чтобы использовать метод `insertBefore`, нужно указать два аргумента. Первым аргументом этого метода выступает новый элемент, который добавляется к форме. Вторым – элемент, перед которым нужно вставить новый.

Определить «место», в которое необходимо вставлять новую строку помогает введенный для имен полей счетчик `phoneCounter`. Его значение позволяет нам определить количество «строк» и вставить новую строку в конце остальных, но перед кнопками добавления и отправки формы.

Сохраните изменения, обновите страницу браузера, убедитесь в работоспособности кнопок.

Add new contact

Name

Phone number Phone type Priority ☒

Phone number Phone type Priority ☐

Phone number Phone type Priority ☐

Отказ от разрывов строк (`<br>`) и применение отступов (`margin`) позволило плавно регулировать расстояние между элементами. Визуально, элементы кажутся не так

плотно сверстаны, как в предыдущих примерах.

Итак, механизм клонирования действительно позволил значительно упростить создание группы новых элементов. Метод `insertBefore`, в свою очередь, предоставляет более гибкий способ добавления элементов формы.

Удаление элементов

В завершение, рассмотрим процесс удаления элементов программным способом.

Для исключения элемента `elem` из дочерних узлов объекта `obj` применяется метод `obj.removeChild(elem)`. То есть для удаления элемента необходимо сначала получить программный доступ к родительскому объекту, после – к дочернему элементу, который нужно удалить и, затем, вызвать метод `removeChild` родительского объекта с передачей в качестве параметра дочернего.

Элементы можно удалять как по одному, так и сразу блоки элементов. В последнем случае, элементы должны быть сгруппированы каким-либо образом (например, в блок `<div>`) и в метод `removeChild` должна быть передана ссылка на группу элементов.

Расширим функциональность нашей формы добавлением возможности удаления записей, которые были добавлены в процессе работы (то есть все, кроме первой). Группировка элементов формы в блоки, сделанная для задач клонирования, упростит нам и процесс удаления.

Во-первых, нужно модифицировать функцию `add_click()` в которой происходит создание новых «строк» для ввода телефонов. Добавим в состав новых строк

кнопку удаления данной строки. Перед последней строкой функции `add_click()` добавим следующий код (полный код функции доступен в папке **Sources_5.2**, архив которой приложен к pdf-файлу данного урока, файл **form5_2-4.html**).

```
var removeButton = document.createElement('input');
removeButton.type = 'button';
removeButton.value = 'Remove';
removeButton.addEventListener('click',
                             rem_click);
newLine.appendChild(removeButton);
```

Большинство операций уже были описаны выше: создается элемент типа **input**, устанавливается его тип **button** и значение (надпись на кнопке) **Remove**. Далее устанавливаем обработчик события нажатия кнопки (**click**), связывая его с функцией **rem_click** при помощи метода **addEventListener**. В завершение добавляем кнопку в клонированный блок **newLine**.

Во-вторых, необходимо создать функцию **rem_click**, которая будет отвечать за нажатие кнопки и, собственно, удаление строки. Так как для всех кнопок используется одна и та же функция, различать их будем при помощи источника сообщения, то есть прототип функции-обработчика должен предусматривать аргумент: **rem_click(event)**.

При появлении события нажатия кнопки будет вызван его обработчик и передан объект события **event**. В этом событии источник полученного сообщения хранится в поле **event.target**. Именно эта конструкция даст нам возможность определить кнопку, которая была нажата.

Однако нас интересует не столько сама кнопка, сколько блок `<div>`, в котором она находится. Обратиться к этому блоку можно несколькими способами: при помощи метода `parentNode` или метода `closest`. Метод `parentNode` возвращает родительский элемент, который для кнопки и будет блоком `<div>`. Метод `closest` позволяет найти ближайший родительский элемент заданного типа. Второй метод предпочтительнее, если кнопка будет вложена в какие-либо дополнительные элементы. Соответственно, определить блок-«строку» в котором была нажата кнопка можно при помощи следующего кода:

```
var line = event.target.closest('div')
```

Родительским объектом для блока является форма. Доступ к форме мы уже неоднократно получали через коллекцию форм документа. Итоговый код функции `rem_click` примет вид:

```
function rem_click(event){  
    var line = event.target.closest('div');  
    var f = document.forms[0];  
    f.removeChild(line);  
    phoneCounter--;  
}
```

В конце функции надо не забыть уменьшить счетчик наших блоков, т. к. это значение используется для поиска последней строки.

После составления всех функций проверяем работоспособность всех функций формы экспериментальным путем.

* За счет реализации удаления блоков в программе появилась потенциальная ошибка. Допустим, пользователь дважды нажал кнопку добавления номера. При этом сформируются поля с именами `phone2` и `phone3` в новых строках. После этого пользователь удаляет вторую (среднюю) строку (с `phone2`), а последняя (с `phone3`) остается. Переменная-счетчик `phoneCounter` в функции удаления (`rem_click`) будет уменьшена (с 3 на 2). Если снова нажать кнопку добавления, счетчик увеличится (с 2 на 3) и снова сформирует имя `phone3`. В результате на форме будут две строки с одинаковыми именами полей (`phone3`). Отказаться от уменьшения счетчика нельзя, т. к. его значение используется для определения последней строки на форме.

Решить эту проблему можно путем разделение счетчиков для формирования имен и для оставшегося количества строк. Попробуйте сделать это самостоятельно.

Для контроля готовое решение доступно папке [Sources_5.2](#), архив которой приложен к pdf-файлу данного урока, файл [form5_2-5.html](#).

Домашнее задание

1. Реализуйте возможность добавлять к контакту несколько дат с выбором типа: день рождения, день профессии, день первой встречи (можете дополнить или поменять перечень). Используйте для выбора типа даты элемент `<select>`.
2. Реализуйте возможность загружать несколько файлов с подписями каждого из них. Содержание подписи не ограничивается (`<textarea>`).
3. Реализуйте возможность удалять добавленные элементы (пп.1-2) каждый по отдельности.
4. Проверьте функциональность формы, убедитесь, что все данные, в т. ч. от добавленных элементов, передаются при отправке формы (попадают в адресную строку браузера).

Вопросы к домашнему заданию

1. Раскройте последовательность действий для добавления нового элемента в форму?
2. Чем отличаются методы `appendChild` и `insertBefore`?
3. Для чего нужен механизм клонирования элементов? Какой порядок создания клона?
4. Как задать или поменять имя нового созданного элемента? Как установить ему обработчик события?
5. Как исключить (удалить) элемент из формы?

© Компьютерная Академия TOP, 2022

top-academy.ru

Все права на защищенные изображения, аудио и видео принадлежат их авторам или законным владельцам. Фрагменты произведений используются исключительно в иллюстративных целях в объеме, оправданном целью, в рамках образовательного процесса и в образовательных целях в соответствии с пунктом 4 статьи 1273 Гражданского кодекса Российской Федерации "Об авторском праве и смежных правах". Объем и метод цитируемых работ соответствуют стандартам, не противоречат нормальному использованию работы, и не ущемляет законные интересы авторов и правообладателей. Процитированные фрагменты работ могут быть заменены альтернативными, незащищенными аналогами и, как таковые, соответствуют критериям добросовестного использования.

Все права защищены. Любое воспроизведение, полностью или частично, запрещено. Согласование использования произведений и их фрагментов осуществляется с авторами и другими правообладателями. Материалы из этого документа могут быть использованы только со ссылкой на ресурс.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется в соответствии с действующим законодательством Российской Федерации.