Оглавление

Предисловие....3
Часть 1. Языки HTML и CSS....4
Урок 1. Знакомство с языком HTML....4
Урок 2. Атрибуты тегов....10
Урок 3. Списки....14
Урок 4. Гиперссылки....18
Урок 5. Таблицы....23
Урок 6. Графические изображения....29
Урок 7. Язык CSS....34
Урок 8. Верстка web-страниц....39
Часть 2. Язык JavaScript....46
Урок 9. Знакомство с JavaScript....46
Урок 10. Ветвление....52
Урок 11. Циклы....57
Урок 12. Функции....61
Урок 13. Массивы....64
Часть 3. HTML-формы и JavaScript....71
Урок 14. Введение в объектно-ориентированное программирование (ООП)....71
Урок 15. Формы. Кнопки и текстовые поля....77
Урок 16. Формы. Переключатели и списки....83
Практикум....88
Практическая работа 1. Текстовая web-страница. Форматирование текста....88
Практическая работа 2. Списки в html-документе....90
Практическая работа 3. Гиперссылки....91
Практическая работа 4. Таблицы....93
Практическая работа 5. Графические изображения....94
Практическая работа 6. Верстка web-страниц....96
Практическая работа 7. Линейные программы и ветвление....98
Практическая работа 8. Циклы....100
Практическая работа 9. Массивы....102
Практическая работа 10. Формы. Кнопки и текстовые поля....103
Практическая работа 11. Формы....104

Предисловие

В наше время web-дизайн и web-программирование — модное и востребованное направление. Практически каждая организация имеет сегодня свой собственный web-ресурс, все большее количество программного обеспечения работает через web-интерфейс. Можно с достаточно высокой вероятностью предполагать, что многие выпускники школы, которые выберут профессии в области информационных технологий, в будущем будут работать в этой области или, по крайней мере, непосредственно контактировать с теми, кто там работает. Поэтому чрезвычайно важно показать школьникам основные идеи и направления современного подхода к разработке веб-страниц.

Конечно, многие профессиональные сайты с динамическим содержанием строятся на основе готовых систем управления содержанием (англ. CMS = Content management system), на которые “натягивается” шаблон (стиль оформления). Однако веб-разработчику любого уровня необходимо знать, “что там внутри”, как строится web-страница, которую генерируют скрипты.

Поскольку изучение серверных языков типа PHP и устройства CMS никак не укладывается в программу (даже углубленного) школьного курса информатики, остается использовать отпущенные учебные часы для изучения языка HTML и создания статических web-страниц. В то же время знакомиться только с HTML, оставаясь на уровне, который был еще как-то приемлем 10–15 лет назад, сейчас нельзя. Не говорить о CSS и JavaScript уже невозможно.

Основная задача учебника состоит в том, чтобы дать ученикам основные теоретические сведения и практические навыки для проектирования «живых» гипертекстовых документов, пригодных для публикации как в сети Интернет, так и для локального использования в качестве интерактивных мультимедийных приложений: познакомиться с языком HTML; освоить способы конструирования гипертекстовых страниц в соответствии с современным уровнем развития этой области информационных технологий (HTML5); изучить правила построения каскадных стилевых таблиц (CSS); изучить основы программирования сценариев на языке JavaScript; освоить методику построения программных объектов, интегрируемых в общую среду «браузер-гипертекстовое приложение».

Данный учебник рассчитан на учащихся старших классов, уже знакомых с кодированием и обработкой текстовой и графической информации, а также имеющих некоторые знания и опыт в области процедурного программирования. Материал этого учебника расширяет эти знания, показывает особенности современных языков программирования, а также знакомит с основными принципами объектно-ориентированного программирования.

Учебник может быть использован для проведения занятий внеурочной деятельности, элективных курсов, кружков, а также для преподавания данной темы на уроках информатики.

Часть 1. Языки HTML и CSS

Урок 1. Знакомство с языком HTML

На этом уроке вы познакомитесь со структурой html-документа в соответствии со стандартом HTML5; основными тегами, позволяющими разместить текст на web-странице и тегами, необходимыми для правильной её интерпретации браузером.

Для создания Web-сайтов используется несколько языков. Основным языком является язык HTML. Знание этого языка необходимо любому создателю Web-сайтов от новичка до профессионала, независимо от того используются ли «готовые решения» или сайт пишется с нуля. Данный язык отвечает за разметку Web-документа, определяя его внешний вид.

> **HTML** (*HyperText Markup Language*) - язык гипертекстовой разметки документов. Он определяет внешний вид страниц в интернете.

Текст, написанный на языке HTML не нужно компилировать, переводить в машинный язык, он интерпретируется браузером.

> **Браузер** (*от англ. browse - листать*) – программа, которая интерпретирует и визуализирует гипертекстовые документы.

Вид Web -страницы задаётся тегами. Теги являются ключевыми словами. Они заключены в угловые скобки:

```
< тег >
```

Теги могут быть непарными и парными. Одиночные теги выглядят как показано выше. Парные теги имеют открывающийся и закрывающийся тег. Закрывающийся тег отличается от открывающегося наличием в теге косой черты (слеша):

```
< тег > … </ тег >
```

Web-документ начинается непарным тегом `<!doctype>`, определяющим тип документа. Это необходимо для того, чтобы браузер знал как его обрабатывать. Далее идет открывающийся тег `<html>`, являющийся началом Web-страницы. Заканчивается страница закрывающимся тегом `</html>`. Между этими двумя парными тегами располагается головная часть программы (заголовок программы) и ее тело.

```
<!DOCTYPE html>
<HTML>
  Заголовок документа
  Тело документа
</HTML>
```

Заготовок web-страницы ограничивается тегами <head> … </head>. В заголовке описываются общие правила интерпретации html-документа. Данная информация необходима браузеру и не будет отображена на странице. Тело документа находится между тегами <body> и </body>. Здесь располагаются команды, следуя которым браузер выводит информацию в окно документа.

```
<!DOCTYPE html>
<HTML>
  <HEAD>
      Заголовок документа
  </HEAD>
  <BODY>
      Тело документа
  </BODY>
</HTML>
```

Заголовок содержит метаданные — это информация о самой web-странице. Например тег <meta> содержит информацию о кодировке текста, используемой в данном html-документе:

```
<META charset=utf-8>
```

Между тегами <title> … </title> содержится имя web-страницы, оно будет отображено в строке заголовка браузера:

```
<TITLE> Заголовок </TITLE>
```

Давайте теперь попробуем создать наш первый html-документ. Для этого нам потребуется любой текстовый редактор. Лучше будет, если он обладает возможностью подсветки синтаксиса, распознавая теги языка HTML.

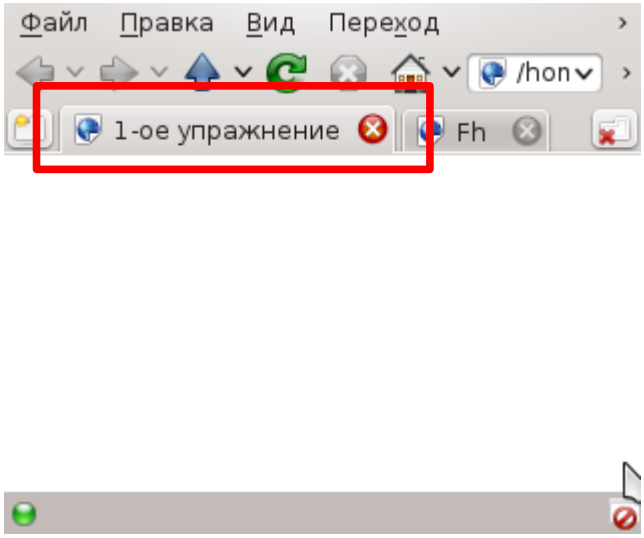
Наберем в текстовом редакторе следующий текст, восстановив полную структуру html-документа. Язык HTML не чувствителен к регистру букв, поэтому теги можно набирать как прописными так и строчными буквами.

```
<!DOCTYPE html>
<HTML>
    <HEAD>
        <META charset=utf-8>
```

```
        <TITLE> 1-ое упражнение </TITLE>
    </HEAD>
    <BODY>

    </BODY>
</HTML>
```

После того как текст набран, его необходимо сохранить. При сохранении, для того чтобы браузер его опознал, документу нужно присвоить расширение html или htm, например **upr_1.html**.



Теперь откроем файловый менеджер, отыщем сохранённый нами файл и сделаем по нему двойной щелчок. Если файл был сохранен правильно, он откроется в браузере.

Что мы видим в окне браузера? Заголовок, вписанный нами между тегами `<title>` … `</title>` отобразился в одной из закладок браузера. Сама страница абсолютно пустая. Это и не мудрено. Ведь между тегами `<body>` … `</body>` у нас ничего не записано. Перейдем к заполнению тела html-документа.

Для этого рассмотрим теги, которые могут располагаться в теле документа. Среди них теги заголовков `<Hn>`, где n — это уровень заголовка. Уровни отличаются друг от друга размером шрифта. Всего в языке HTML шесть уровней заголовков от 1 (самого большого) до 6 (самого маленького). Текст заголовка выделяется полужирным шрифтом.

```
<Hn> текст заголовка </Hn>
```

На странице может присутствовать горизонтальная линия, которая тянется от левой границы страницы до правой, не зависимо от её размера. Иногда используется в качестве разделителя. Создается непарным тегом

```
<HR>
```

Текст абзацев выделяется тегами `<P>`:

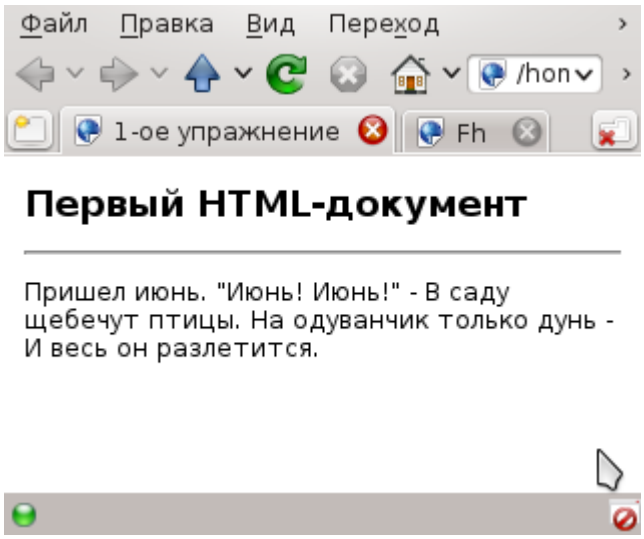
```
<P> текст абзаца </P>
```

Используя рассмотренные выше теги, оформим в виде web-страницы стихотворение Маршака. Внесем изменения в уже имеющийся у нас документ. В теле

документа, расположенного между тегами <body> … </body>, сделаем заголовок документа «Первый HTML-документ» и отделим его от стихотворения вертикальной чертой. Само четверостишие поместим в абзац :

```
<!DOCTYPE html>
<HTML>
    <HEAD>
        <META charset=utf-8>
        <TITLE> 1-ое упражнение </TITLE>
    </HEAD>
    <BODY>
     <H1> Первый HTML-документ </H1>
          <HR>
          <P>
       Пришел июнь. "Июнь! Июнь!" –
       В саду щебечут птицы.
       На одуванчик только дунь –
       И весь он разлетится.
     </P>
    </BODY>
</HTML>
```

Сохраним дополненный html-документ, вернемся к браузеру и обновим страницу. Что мы видим? Заголовок выделен полужирным, он отделен от остальной страницы горизонтальной чертой. А вот четверостишия у нас не получилось. Несмотря на то, что мы при наборе текста в конце каждой строки нажимали клавишу Enter, он напечатан как один абзац.

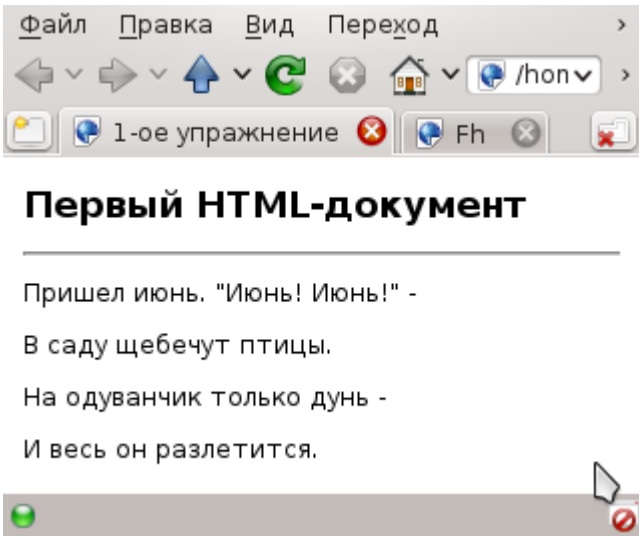


Действительно браузер игнорирует символ конца строки, вставляемый клавишей Enter. Для браузера абзац продолжается от открывающегося до закрывающегося тега . Как выйти из этого положения? Как разместить текст в четыре строки?

Первое что приходит на ум сделать четыре абзаца. Давайте попробуем. Внесем изменения в наш html-документ:

```
<P> Пришел июнь. "Июнь! Июнь!" –
<P> В саду щебечут птицы.
<P> На одуванчик только дунь –
<P> И весь он разлетится.
```

Давайте посмотрим что у нас получилось. Текст стихотворения действительно расположился в четыре строки. Правда он не смотрится как единое четверостишие. Дело в том, что после каждого абзаца браузер автоматически делает отбивку.



Обратите внимание, что в данном примере тег <p> использован как непарный. В таком случае абзац продолжается от одного тега <p> до другого.

Очевидно, для того чтобы внутри четверостишия не было отбивок, оно должно быть единым абзацем. Но при этом строки должны переноситься. Для этого в языке HTML есть тег принудительного переноса строк
. Этот тег непарный. Тогда текст нашего четверостишия будет выглядеть следующим образом:

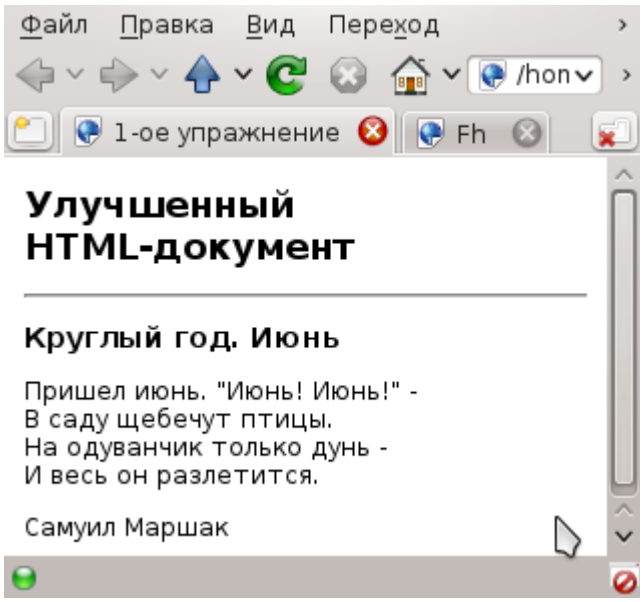
```
<P>
   Пришел июнь. "Июнь! Июнь!" – <BR>
   В саду щебечут птицы.<BR>
   На одуванчик только дунь –<BR>
   И весь он разлетится.
```

Ниже представлен полный текст нашего html-документа:

```
<!DOCTYPE html>
<HTML>
    <HEAD>
        <META charset=utf-8>
        <TITLE> 1-ое упражнение </TITLE>
    </HEAD>
    <body>
       <H1> Улучшенный HTML-документ </H1>
       <HR>
```

```
        <H2> Круглый год. Июнь </H2>
        <P>
           Пришел июнь. "Июнь! Июнь!" – <BR>
           В саду щебечут птицы.<BR>
           На одуванчик только дунь –<BR>
           И весь он разлетится.
        <p> Самуил Маршак
     </BODY>
  </HTML>
```

Так выглядит наш улучшенный html-документ. В нем кроме заголовка 1-го уровня после горизонтальной линии идет заголовок 2-го уровня — название стихотворения. Далее само четверостишие, которое наконец-то выглядит должным образом. Страницу завершает автор данного четверостишия, расположенный в виде отдельного абзаца.



Еще раз обратим внимание на то, что абзацы браузер по умолчанию отделяет отбивками для лучшего визуального восприятия.

Основные теги, рассмотренные на уроке:	
`<!doctype>`	Тип документа
`<html> … </html>`	Начало и конец html-документа
`<head> … </head>`	Заголовок документа
`<meta>`	Метаданные страницы
`<body> … </body>`	Тело документа
$<h_n>$ … $</h_n>$	Заголовок
`<hr>`	Горизонтальная линия
`<p>`	Абзац
` `	Принудительный перенос строки

Урок 2. Атрибуты тегов

На этом уроке вы узнаете, что теги могут содержать атрибуты; познакомитесь с одним из наиболее часто встречающихся атрибутов — атрибутом, отвечающим за стили оформления.

Открывающие теги могут содержать атрибуты, то есть дополнительную информацию о свойствах отображаемого на экране элемента. Атрибуты записываются внутри открывающего тега после его имени в виде:

- отдельного ключевого слова;
- ключевого слова, знака “=” и параметра (значения атрибута).

Порядок следования атрибутов в теге не важен. Действие атрибута продолжается от открывающего тега, в котором он задан, до закрывающего в том случае, если тег парный. Если же тег непарный, то атрибут действует только внутри тега.

Одним из часто используемых атрибутов является атрибут style, позволяющий задавать оформление различным элементам web-страницы. Данный атрибут имеет следующий синтаксис:

```
<тег style= "свойство_1: значение_1;
  свойство_2: значение_2...">
```

Одно свойство атрибута style отделяется от другого точкой с запятой. Порядок следования свойств также не важен.

Вернемся к html-документу, созданному нами на прошлом уроке. Заголовок первого уровня отделяет от остальной части текста горизонтальная линия. Это черная тонкая линий, тянущаяся от левой до правой границы страницы. Давайте изменим ее внешний вид. Для начала сделаем ее толще. Толщину (высоту объекта) задает свойство `height`. Толщина задается в пикселях, например:

```
<HR style="height: 5px"
```

По умолчанию линия имеет толщину 1 пиксель. Изменим также цвет нашей линии и уберем обводку (контур) линии, который становится заметен после изменения цвета. Цвет обводки по умолчанию черный.

```
<HR style="height: 5px; background-color: red;
  border: none">
```

Цвета фона и цвет текста для всего документа можно задать при помощи свойства `background-color` и `color` в теге `<body>`:

```
<BODY style="background-color: cornsilk; color: maroon ">
```

В данном примере у нас получился темно красный текст на бледно желтом фоне, разделенный ярко красной горизонтальной линией. Еще раз посмотрим: если нам нужно задать какое-либо свойство, которое будет распространяться на всю web-страницу, мы его указываем в теге `<body>`. Как быть, если нужно определить цвет шрифта лишь для одной части текста? Конечно же нужно разместить свойство `color` в теге данного элемента.

Изменим цвет заголовка первого уровня на темно фиолетовый:

```
<H2 style="color: indigo"> Улучшенный HTML-документ
</H2>
```

Цвета в html-документах могут задаваться двумя способами. Первый способ — это использование именованных цветов. В приведенных выше примерах используется именно такой подход, при котором указывается название цвета. Браузеры различают около 140 именованных цветов.

Второй способ — указание шестнадцатеричного кода цвета в модели RGB. Т.к. RGB модель трехканальная (красный канал, зеленый канал и синий канал), то цвет задается тройкой двузначных шестнадцатеричных чисел, перед которыми ставится знак «#». Приведенный ниже пример обеспечит абсолютно такой же цвет заголовка, что и в предыдущем примере:

```
<H2 style="color: #4b0082"> Улучшенный HTML-документ
</H2>
```

В Интернете можно найти большое количество ресурсов, на которых размещены таблицы именованных цветов. Воспользовавшись данными ресурсами можно легко узнать названия тех или иных цветов. Так же существуют генераторы цветов, формирующие их шестнадцатеричный код.

Для выравнивания текста используется свойство `text-align`. Это свойство может принимать одно из четырех значений: `left`, `center`, `right`, `justify`.

`<P>`	По умолчанию текст выравнивается по левому краю;
`<P style="text-align: left">`	Выравнивание по левому краю;
`<P style="text-align: center">`	Выравнивание по центру;
`<P style="text-align: right">`	Выравнивание по правому краю;
`<P style="text-align: justify">`	Выравнивание по ширине.

Чтобы задать размер шрифта в тексте используется свойство `font-size`. В html-документах применяется несколько единиц измерения для определения размера шрифта. Размер может определяться в пикселях или пунктах. При этом после числового значения следуют символы `px` или `pt`, которые должны следовать сразу после числа, причем между числом и `px` (`pt`) пробела быть не должно.

```
<P  style="font-size: 16px">
<P  style="font-size: 12pt">
```

Два приведенных примера выводят текст одинакового размера. Размер в шестнадцать пикселей соответствует двенадцати пунктам. Таким размером браузер выводит текст по умолчанию. Существуют еще подходы использования относительного размера шрифта, но мы их сегодня рассматривать не будем.

Для установления начертания в тексте служат сразу несколько свойств. Свойство `font-weight` отвечает за жирность текста и может принимать одно из двух значений: `normal` — обычный и `bold` — полужирный. Свойство `font-style` отвечает за курсив и принимает значения `normal` — обычный и `italic` — курсив. Подчеркнутым текст делает свойство `text-decoration`, возможные значения которого `none` — без изменений, `underline` — подчеркнутый текст, `overline` — надчеркнутый текст и `line-through` — перечеркнутый текст. В примере, расположенном ниже, приведен текст, имеющий начертание полужирный курсив:

```
<P  style="font-weight: bold; font-style: italic">
```

Если мы хотим, чтобы в тексте работала автоматическая расстановка переносов нужно воспользоваться свойством `hyphens` со значением `auto`. Расстановка переносов будет производиться в соответствии с внутренним словарем браузера. Для того, чтобы браузер правильно определил словарь, в теге `<html>` будем использовать языковой атрибут `lang="ru"`.

Мы рассмотрели как применяются стили к абзацу, заголовкам. А как выделить отдельное слово или букву в тексте? Для этого используется тег `<span>`. Данный тег парный. Части текста, расположенной между открывающимся и закрывающимся тегом `<span>`, можно назначить свой стиль. В следующем примере первая буква абзаца будет красной и большего размера:

```
<P  style="font-size: 12pt; text-align: justify">
<SPAN style="color: red; font-size: 24pt"> П </SPAN>
    ечальная участь ожидает того, кто наделен талантом, но
```

```
    вместо того, чтобы развивать и совершенствовать свои
    способности, чрезмерно возносится и придается праздности и
    самолюбованию.
</P>
```

Теги, рассмотренные на уроке:		
`<span> … </span>`		Часть текста
Свойства атрибута *style*, рассмотренные на уроке:		
`height:`	`значение (px)`	Высота объекта
`background-color:`	`цвет`	Цвет заднего плана (цвет фона)
`color:`	`цвет`	Цвет переднего плана (цвет текста)
`text-align:`	`left, center, right, justify`	Выравнивание текста
`text-decoration:`	`none, underline, overline, line-through`	Подчеркивание текста
`font-size:`	`значение (pt или px)`	Размер шрифта
`font-weight:`	`bold или normal`	Жирность шрифта
`font-style:`	`italic или normal`	Курсивный шрифт

Урок 3. Списки

На этом уроке мы вспомним какие типы списков существуют; вы познакомитесь со способами оформления различных видов списков в html-документах.

Для оформления текста в html-документах часто используют списки. Считается, что они увеличивают читабельность документов за счет краткости пунктов. К тому же они имеют отступы, это выделяет их из основного текста. Что же такое список?

> **Список** — это способ систематизации информации, в основе которого лежит перечень элементов.

В тексте используются маркированные и нумерованные списки. **Маркированный список** используется в том случае, когда порядок элементов списка не важен. Каждый элемент такого списка браузер предваряет меткой в начале строки, а сами строки смещает вправо.

Маркированный список начинается открывающимся тегом `<ul>` и заканчивается закрывающимся тегом `</ul>`. Каждый элемент списка располагается внутри парного тега `<li>`:

```
<UL>
    <LI> первый элемент </LI>
    <LI> второй элемент </LI>
    ...
    <LI> последний элемент </LI>
</UL>
```

Давайте попробуем использовать маркированный список в web-документе. Оформим в виде маркированного списка …

```
                    Маркированный список

Что едят Тигры на завтрак? Все, кроме:

    • меда,
    • желудей,
    • чертополоха,
    • чая с конфетами,
    • морковки.
```

По умолчанию браузер помещает перед каждым элементом списка метку в виде закрашенного кружка. Вид метки, которую использует браузер, настраива-

ется при помощи свойства `list-style-type` и может принимать значение `disc`, `circle` и `square`.

`<UL>`	• диск
`<UL style="list-style-type: disc">`	• диск
`<UL style="list-style-type: circle">`	◦ окружность
`<UL style="list-style-type: squarev">`	▪ квадрат

Нумерованный список используется в тех случаях, когда порядок следования элементов списка важен. Строки нумерованного списка также сдвигаются в право, таким образом браузер выделяет список в тексте. Нумерованный список начинается открывающимся тегом `<ol>` и заканчивается закрывающимся тегом `</ol>`. Каждый элемент, как и в маркированном списке, располагается внутри парного тега `<li>`:

```
<OL>
    <LI>первый элемент</LI>
    <LI>второй элемент</LI>
    ...
    <LI>последний элемент</LI>
</OL>
```

Создадим html-документ, содержащий в себе нумерованный список.

```
                    Нумерованный список

Кто сломал теремок?:

   1. Волк.
   2. Медведь.
   3. Кощей Бессмертный.
   4. Леший.
   5. Баба Яга.
   6. Иван-дурак.
```

Нумерация может быть как цифровой, так и буквенной. По умолчанию браузер использует нумерацию арабскими числами. Тип нумерации настраивается при помощи свойства `list-style-type` и для нумерованного списка может принимать значение `decimal`, `lower-latin` , `lower-roman`, `upper-latin` и `upper-roman`.

`<OL>`	Арабскими цифрами (1, 2, 3,...)
`<OL style="list-style-type:`	Арабскими цифрами (1, 2, 3,...)

`decimal">`	
`<OL style="list-style-type: upper-latin">`	Прописными буквами (A, B, C,...)
`<OL style="list-style-type: lower-latin">`	Строчными буквами (a, b, c,...)
`<OL style="list-style-type: upper-roman">`	Большими римскими цифрами (I, II, III,...)
`<OL style="list-style-type: lower-roman">`	Малыми римскими цифрами (i, ii, iii,...)

Иногда на web-страницах используются вложенные или многоуровневые списки. Многоуровневые списки содержат в себе несколько списков, вложенных друг в друга, образующих иерархическую структуру.

Вложенный список — это список, который является элементом другого списка.

Попробуем создать html-документ следующего вида:

Рекомендации по использованию списков

1. Маркированный список удобен тогда, когда порядок элементов в нем не важен. Примеры:
   - список продуктов для приготовления борща;
   - список команд исполнителя;
   - список школьных предметов.
2. Нумерованный список особенно хорош для описания действий, в которых важен порядок следования. Примеры:
   - рецепт приготовления борща;
   - программа для исполнителя;
   - расписание уроков.

Html-код «тела» документа для данного примера представлен ниже:

```
<H1 style="text-align: center;  color: darkblue">
     Рекомендации по использованию списков
</H1>
    <HR style="height: 2px; background-color: indigo">
    <OL>
      <LI style="hyphens: auto; text-align: justify">
           Маркированный список удобен тогда, когда
           порядок элементов в нем не важен. Примеры:
           <UL style="list-style-type: circle">
```

```
            <LI>список продуктов для приготовления борща;
            <LI>список команд исполнителя;
            <LI>список школьных предметов.
          </UL>
      <LI style="hyphens: auto; text-align: justify">
          Нумерованный список особенно хорош для описания
          действий, в которых важен порядок следования.
          Примеры:
          <UL style="list-style-type: circle">
            <LI>рецепт приготовления борща;
            <LI>программа для исполнителя;
            <LI>расписание уроков.
          </UL>
    </OL>
```

Теги, рассмотренные на уроке:		
`<ul> … </ul>`		Маркированный список
`<ol> … </ol>`		Нумерованный список
`<li> … </li>`		Элемент списка
Свойства атрибута *style*, рассмотренные на уроке:		
`list-style-type:`	`disc, circle, square`	Виды маркеров для маркированного списка
`list-style-type:`	`decimal, lower-latin, lower-roman, upper-latin, upper-roman`	Виды нумерации для нумерованного списка

Урок 4. Гиперссылки

На этом уроке вы узнаете как функционируют гиперссылки в гипертекстовых документах, как они устроены. Познакомитесь с абсолютной и относительной адресацией при оформлении гиперссылок.

Рассмотренные нами ранее примеры использовали одностраничные html-документы. Но на практике гипертекстовые документы являются многостраничными. Связывают страницы гипертекстового документа между собой гиперссылки.

> **Гипертекст** — это документ, содержащий гиперссылки, которые обеспечивают переходы как внутри документа, так и на другие документы, расположенные на локальном компьютере или в сети.

Термин гипертекст был введён Тедом Нельсоном в 1965 году для обозначения «текста ветвящегося или выполняющего действия по запросу». Обычный текст имеет линейную структуру. Предполагается, что текст читается по порядку. Сначала первая страница, потом вторая, затем третья и так далее.

При работе с гипертекстом пользователь может прервать линейное чтение в месте ссылки, посмотреть другую часть читаемого текста или даже совсем другой текст, а затем продолжить чтение с места прерывания.

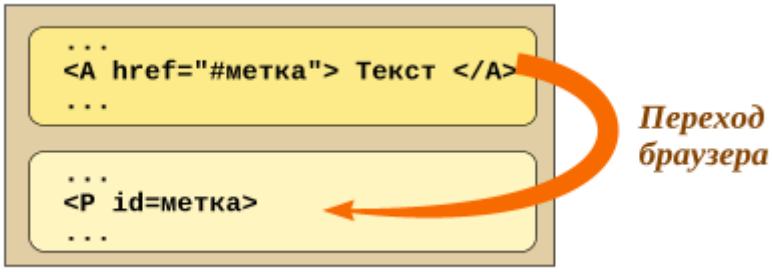
Гиперссылка имеет два конца и направление. Ссылка начинается от указателя гиперссылки, который может указывать на любой Web-ресурс (например изображение, видеоклип, аудиофрагмент, программу, документ HTML, элемент в документе HTML и т. д.), являющийся назначением ссылки. В гиперссылке прописана адресная часть, на которую ссылается указатель. Она скрыта от глаз пользователя.

Как мы определяем в тексте указатель гиперссылки? Эта часть текста выделена цветом и подчеркиванием. По умолчанию браузеры используют синий цвет ссылок, отработанные ссылки становятся фиолетовыми. Также, при наведении курсора на указатель гиперссылки, он превращается в руку с вытянутым указательным пальцем.

Гиперссылка создается с помощью парного тега `<a> … </a>`. Текст, расположенный внутри данной пары тегов, будет являться указателем, т. е. выделен синим цветом и подчеркиванием. Адресная часть указывается в атрибуте `href`:

```
<A href="адресная_часть"> Указатель_гиперссылки </A>
```

Рассмотрим вначале переходы внутри документа. Такие переходы удобны в тех случаях, когда на странице содержится большое количество текстовой информации. Внутренняя навигация делает документ более читабельным. В этом случае в адресной части атрибута `href` в теге `<a>` ставится знак «#». На странице с помощью атрибута `id` расставляются якоря (закладки, метки), на которые будет ссылаться указатель гиперссылки. Для того чтобы ссылка функционировала, имя якоря в атрибуте `id` и в атрибуте `href`, за исключением знака «#», должны как близнецы-братья абсолютно одинаковыми.



Оформим в виде html-документа стихотворение Бориса Заходера «Вредный кот». Для улучшения навигации разделим стихотворение на разделы, разделы пронумеруем. В начале web-страницы сделаем гипертекстовое содержание. Каждый пункт содержания будет переносить нас к соответствующему разделу стихотворения. Оформим содержание в виде нумерованного списка, отделим его от остальной части текста горизонтальными линиями:

Вредный кот
Борис Зоходер

Содержание

1. Петь, здорово...
2. Только было сел за стол...
3. Но меня же не обманешь...
4. Я кота простить готов

1

- Петь, здорово!
- Здравствуй, Вова!
- Как уроки?
- Не готовы...
Понимаешь, вредный кот
Заниматься не дает!

2

Только было сел за стол,
Слышу: «Мяу...» -
«Что пришел?
Он тогда залез на стул,
Притворился, что уснул.
Ну и ловко сделал вид -
Ведь совсем как будто спит! -

3

Но меня же не обманешь...
«А, ты спишь? Сейчас ты встанешь!
Ты умен, и я умен!»
Раз его за хвост!
- А он?
- Он мне руки исцарапал,
Скатерть со стола стянул,
Все чернила пролил на пол,
Все тетрадки мне заляпал
И в окошко улизнул!

4

Я кота простить готов,
Я жалею их, котов.
Но зачем же говорят,
Будто сам я виноват?

Уходи! - кричу коту. -
Мне и так... невмоготу!
Видишь, занят я наукой,
Так что брысь и не мяукай!»

Я сказал открыто маме:
«Это просто клевета!
Вы попробовали б сами
Удержать за хвост кота!»

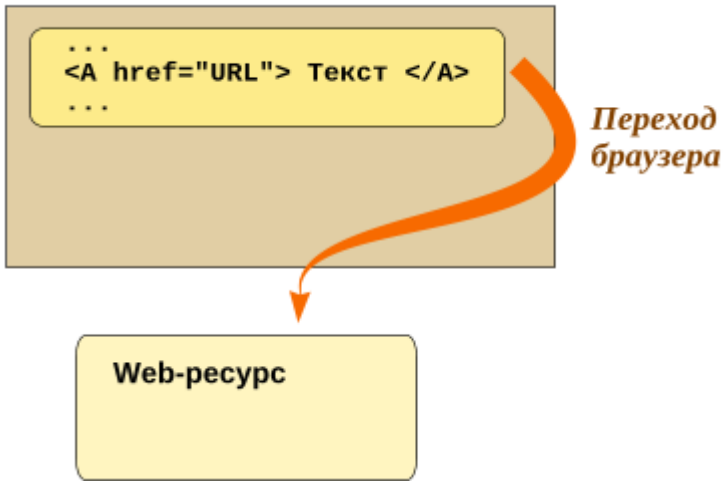
Приведем здесь неполный фрагмент html-кода, который будет формировать нашу web-страницу. Обратим внимание на оформление нумерованного списка. При таком расположении тегов номер элемента не будет иметь отношение к указателю гиперссылки. Если бы мы хотели, чтобы указатель включал в себя и номер элемента, то теги `<li>` необходимо разместить внутри парного тега `<a> … </a>`.

```
<H1> Вредный кот </H1>
<H2> Борис Зоходер </H2>
<HR>
   <H2> Содержание< /H2>
   <OL>
        <LI> <A href="#h1"> Петь, здорово… </A>
        <LI> <A href="#h2"> Только было сел за стол… </A>
        <LI> <A href="#h3"> Но меня же не обманешь… </A>
        <LI> <A href="#h4"> Я кота простить готов </A>
   </OL>
<HR>

<H3 id="h1"> 1 </H3>
<P>
– Петь, здорово! <BR>
– Здравствуй, Вова! <BR>
– Как уроки? <BR>
– Не готовы… <BR>
Понимаешь, вредный кот <BR>
Заниматься не дает!

<H3 id="h2"> 2 </H3>
<P>
Только было сел за стол, <BR>
Слышу: «Мяу...» – <BR>
«Что пришел? <BR>
…
```

Кроме гиперссылок, обеспечивающих переходы внутри документа, используются и ссылки на внешние ресурсы. Такая ссылка в адресной части атрибута `href` содержит адрес файла, к которому необходимо перейти. Адрес может быть указан полностью, тогда он имеет вид `http://имя_сервера/сайт/документ`. Такой адрес называется абсолютным.



Кроме того можно указывать и неполный адрес. Он называется относительным, т.к. путь указывается относительно текущей страницы. Если файл находится в одной папке с текущей страницей, то в адресе достаточно указать лишь имя файла, например `index2.html`. Если файл находится в родительской папке, то адрес будет иметь вид `../документ`, например `../index2.html`. В данном случае две точки означают, что для обнаружения файла необходимо в иерархии каталогов подняться на один уровень вверх.

Используем выше рассмотренный пример для создания внешних ссылок. Разделим имеющийся у нас текст на несколько файлов. В первом файле оставим название стихотворения, автора и гипертекстовое содержание, избавившись от второй горизонтальной черты. В остальных четырех файлах разместим части стихотворения, помеченные цифрами. Назовем эти файлы `index1.html`, `index2.html`, `index3.html` и `index4.html`. Проследим, чтобы все файлы располагались в одной папке.

Ниже представлен html-код тела первого файла:

```
<H1> Вредный кот </H1>
<H2> Борис Зоходер </H2>
<HR>
 <H2> Содержание< /H2>
 <OL>
  <LI> <A href="index1.html"> Петь, здорово… </A>
  <LI> <A href="index2.html"> Только было сел за стол… </A>
  <LI> <A href="index3.html"> Но меня же не обманешь… </A>
  <LI> <A href="index4.html"> Я кота простить готов </A>
```

```
</OL>
```

Из оставшихся файлов приведем содержимое второго файла `index1.html`, его тело содержит код, представленный ниже. По аналогии другие части стихотворения необходимо разделить по оставшимся файлам.

```
<H3> 1 </H3>
<P>
– Петь, здорово! <BR>
– Здравствуй, Вова! <BR>
– Как уроки? <BR>
– Не готовы… <BR>
Понимаешь, вредный кот <BR>
Заниматься не дает!
```

Теги, рассмотренные на уроке:	
`<a> … </a>`	Гиперссылка
Атрибуты тегов, рассмотренные на уроке:	
`href`	Определяет адрес гиперссылки
`id`	Задает идентификатор объекта

Урок 5. Таблицы

На этом уроке вы узнаете об особенностях структуры таблиц в html-документах, научитесь строить простые таблицы и таблицы с объединением ячеек, а также их форматировать.

Для чего используют таблицы в html-документах? Конечно же для представления табличных данных. Хотя иногда таблицы используют также для размещения текстовых и графических данных на странице, но это уже устаревший подход к верстке web-страниц.

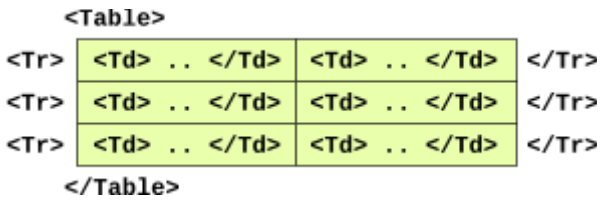


Таблица в html-документе задается парным тегом `<table> … </table>`. Далее таблица разбивается на строки тегами `<tr> … </tr>`. Внутри строк располагаются ячейки, они определяются тегами `<td> … </td>`. Основная структура таблицы показана на рисунке.

В языке Html есть возможность сделать «шапку» таблицы. В таком случае ячейки, являющиеся заголовками столбцов, определяются тегами `<th>...</th>`. Текст в таких ячейках выделяется полужирным шрифтом. Для создания заголовка таблицы используется парный тег `<caption>...</caption>`.

```
<TABLE border="1">
<CAPTION> Простая таблица </CAPTION>
  <TR>          <!-- Первая строка -->
   <TH> 1-ый столбец </TH>          <!-- Первая ячейка -->
   <TH> 2-ой столбец </TH>          <!-- Вторая ячейка -->
  </TR>
  <TR>          <!-- Вторая строка -->
   <TD> 2,1 </TD>                <!-- Первая ячейка -->
   <TD> 2,2 </TD>                <!-- Вторая ячейка -->
  </TR>
  <TR>          <!-- Третья строка -->
   <TD> 3,1 </TD>                <!-- Первая ячейка -->
   <TD> 3,2 </TD>                <!-- Вторая ячейка -->
  </TR>
</TABLE>
```

В рассмотренном примере имеется заголовок, текст которого выравнивается

по центру таблицы, и «шапка». Текст ячеек в шапке указывает нам на номера столбцов. Числа внутри остальных ячеек показывают текущие номера строки и столбца.

Простая таблица

1-ый столбец	2-ой столбец
2,1	2,2
3,1	3,2

В открывающемся теге `<table>` данной таблицы присутствует атрибут `border`, который задает толщину границ таблицы. Если границы не заданы, то в браузере таблица будет нарисована без границ. Часть html-кода, расположенная между знаками `<!--` … `-->` является комментарием и браузером игнорируется.

Пока наша таблица выглядит невзрачно. Займемся ее оформлением. Заголовок по умолчанию выглядит слишком маленьким по сравнению с текстом в «шапке» таблицы. Можно использовать стили, задающие размер шрифта, или сделать заголовок заголовком. Тег заголовка `<h3>` можно использовать как непарный. Т.к. он находится внутри парного тега `<caption>`, зона его действие определена, она распространяется до закрывающегося тега `</caption>`.

```
<CAPTION> <H3> Простая таблица </CAPTION>
```

Конечно же стили могут быть применены для всей таблицы, отдельных ячеек или строк. Но также можно разделить таблицу на разделы и задавать стили для группы строк. Для группировки строк в разделы существуют теги `<thead>`, `<tbody>` и `<tfoot>`, формирующие верхнюю часть («шапку»), центральный и нижний разделы.

Простая таблица

1-ый столбец	2-ой столбец
2,1	2,2
3,1	3,2

Зальем ячейки верхней части и центральной части таблицы разными цветами. Например шестнадцатеричный код цветов верхней и центральной части может быть `BFCC8E` и `F1F4DE`. Также выровним текст в ячейках центральной части по центру.

```
...
<THEAD style="background-color: #BFCC8E">
  <TR>          <!-- Первая строка -->
      <TH> 1-ый столбец </TH>     <!-- Первая ячейка -->
      <TH> 2-ой столбец </TH>     <!-- Вторая ячейка -->
  </TR>
 </THEAD>
```

```
<TBODY style="background-color: #F1F4DE;
      text-align: center">
  <TR>          <!-- Вторая строка -->
      <TD> 2,1 </TD>              <!-- Первая ячейка -->
...
```

Ячейки можно группировать не только по строкам, но и по столбцам. Для этого в таблице необходимо организовать раздел, который описывается парным тегом <colgroup> … </colgroup>. Этот раздел располагается внутри тега <table> после описания заголовка таблицы в теге <caption>, но перед разделом <thead>, если он присутствует.

Простая таблица

1-ая строка	1,2	1,3	1,4
2-ая строка	2,2	2,3	2,4
3-ая строка	3,2	3,3	3,4

Внутри раздела <colgroup> тег <col> формирует группы столбцов. Атрибут span тега <col> определяет их количество. В рассмотренном примере таблица поделена на два раздела. В первый раздел входит один первый столбец. Тег <col>, отвечающий за этот раздел, не содержит атрибута span. Во втором теге <col> в атрибуте span указано, что раздел объединяет три столбца. Каждый из разделов имеет свой цвет.

```
<TABLE border=1 >
<CAPTION> <H3> Простая таблица </CAPTION>

  <COLGROUP>
      <COL style="background-color: #B5CFE8">
      <COL span="3" style="background-color: #E6F1FF">
  </COLGROUP>

  <TR>          <!-- Первая строка -->
      <TH> 1-ая строка </TH> <!-- Первая ячейка -->
      <TD> 1,2 </TD>              <!-- Вторая ячейка -->
      <TD> 1,3 </TD>              <!-- Третья ячейка -->
      <TD> 1,4 </TD>              <!-- Четвертая ячейка -->
  </TR>

  <TR>          <!-- Вторая строка -->
      <TH> 2-ая строка </TH> <!-- Первая ячейка -->
      <TD> 2,2 </TD>              <!-- Вторая ячейка -->
      <TD> 2,3 </TD>              <!-- Третья ячейка -->
```

```
        <TD> 1,4 </TD>              <!-- Четвертая ячейка -->
    </TR>

    <TR>            <!-- Третья строка -->
        <TH> 3-ая строка </TH> <!-- Первая ячейка -->
        <TD> 3,2 </TD>              <!-- Вторая ячейка -->
        <TD> 3,3 </TD>              <!-- Третья ячейка -->
        <TD> 1,4 </TD>              <!-- Четвертая ячейка -->
    </TR>
</TABLE>
```

В нашем примере структура таблицы проста. Но на практике часто встречаются таблицы с объединением ячеек. Для объединения используются атрибуты `colspan` и `rowspan`. Атрибут `colspan` объединяет ячейки по горизонтали, а `rowspan` — по вертикали.

Таблица с объединениями

1-ый столбец	1,2 и 1,3		1,4
	2,2	2,3	2,4
	3,2	3,3	3,4

В первой строке рассмотренного примера в первой ячейке использован атрибут `rowspan`, объединяющий по вертикали три ячейки в единый столбец. Первые ячейки в остальных строках отсутствуют. Фактически атрибут `rowspan` растягивает ячейку (1, 1) на три строки.

В первой же строке во второй ячейке использован атрибут `colspan`, объединяющий ячейки по горизонтали. Здесь та же самая ситуация: ячейка (1, 2) растянута на два столбца.

```
<TABLE border="1" style="background-color: #FCFAE0">
<CAPTION> <H3>Таблица с объединениями </CAPTION>

    <TR>            <!-- Первая строка -->
        <TH rowspan="3" style="background-color: #F7A54A">
            1-ый <BR>столбец </TH>
        <TD colspan="2" style="background-color: #FFD690">
            1,2 и 1,3</TD>
        <TD> 1,4 </TD>
    </TR>

    <TR>            <!-- Вторая строка -->
        <TD> 2,2 </TD>
        <TD> 2,3 </TD>
```

```
        <TD> 2,4 </TD>
    </TR>

    <TR>          <!-- Третья строка -->
        <TD> 3,2 </TD>
        <TD> 3,3 </TD>
        <TD> 3,4 </TD>
    </TR>
</TABLE>
```

Теперь обратим внимание на размер таблицы и ячеек. Если они не заданы, то размеры определяются содержимым ячеек. Т.е. чем больше содержимое ячейки, тем она шире. Это хорошо видно на предыдущем примере, где первый столбец занимает больше места, чем остальные.

Таблица с объединениями

1-ый столбец	1,2 и 1,3	
	2,2	2,3
	3,2	3,3

Задать ширину и высоту можно с помощью свойств стилей `width` и `height`. Существует два подхода к определению размеров таблицы и ее элементов: абсолютный и относительный. Абсолютный размер задается в пикселях, а относительный — в процентах. В следующем примере показана таблица шириной 250 пикселей, у которой первый столбец занимает 35% ширины всей таблицы. Обратим внимание на то, что при определении ширины столбца при помощи `width` таблицы не нужно задавать ширину для каждой ячейки столбца в отдельности, достаточно это сделать для первой ячейки. Остальные ячейки в столбце данную ширину наследуют. Получается, что ширина столбцов определяется в первой строке таблицы. Фрагмент html-кода таблицы:

```
<TABLE border="1" style="background-color: #FCFAE0;
  width: 250px">
<CAPTION> <H3>Таблица с объединениями </CAPTION>
   <TR>              <!-- Первая строка -->
     <TH  rowspan="3"  style="background-color:  #F7A54A;
        width: 35%"> 1-ый <BR>столбец </TH>
     <TD colspan="2" style="background-color:#FFD690">
           1,2 и 1,3</TD>
   </TR>

   <TR>              <!-- Вторая строка -->
       <TD> 2,2 </TD>
       …
```

Мы уже упомянули про границы в таблице. По умолчанию каждая ячейка в таблице, как и таблица в целом, имеют собственные границы. Из-за это все границы кажутся двойными. Вид границ определяется свойством `border-collapse`, который может иметь значение `separate` (у каждой ячейки своя рамка) или `collapse` (одна общая граница). В следующем примере будет построена таблица с границей в виде тонкой одинарной линии.

```
<TABLE border="1" style="width: 250px;
  border-collapse: collapse">
```

Теги, рассмотренные на уроке:		
`<table> … </table>`		Таблица
`<tr> … </tr>`		Строка таблицы
`<td> … </td>` и `<th> … </th>`		Простая и заголовочная ячейка
`<caption> … </caption>`		Заголовок таблицы
`<thead> … </thead>`		Верхний раздел таблицы
`<tbody> … </tbody>`		Основной раздел таблицы
`<tfoot> … </tfoot>`		Нижний раздел таблицы
`<colgroup> … </colgroup>`, `<col>`		Блок описания разделов таблицы по столбцам
Атрибуты, рассмотренные на уроке:		
`border`		Толщина границ
`colspan` и `rowspan`		Объединение ячеек по горизонтали и вертикали
Свойства атрибута *style*, рассмотренные на уроке:		
`width` и `height`	`значение (px или %)`	Ширина и высота
`border-collapse`	`separate` или `collapse`	Тип границы таблицы

Урок 6. Графические изображения

На этом уроке вы узнаете как разместить графическое изображение в html-документе, как использовать изображение в качестве фона и маркера в списках.

При создании web-страниц в языке HTML5 можно использовать как векторные так и растровые графические изображения. Из растровых используют три графических формата: GIF, PNG и JPG.

> **Растровое изображение** — это изображение, состоящее из пикселей, которые строго упорядочены. Каждый пиксель имеет свой цвет.

Формат JPG был разработан специально для передачи фотографий. Он поддерживает миллионы цветов и позволяет получать изображения высокого качества. Конечно, высокое качество отражается на размерах файла. В графическом редакторе, можно указать степень качества, тем самым достичь нужного компромисса “качество-размер файла”.

Формат GIF поддерживают только 256 цветов. Этого достаточно для получения качественных иллюстраций, но до фотографического качества далеко. Зато данный формат обладает дополнительными возможностями.

Во-первых у него есть возможность простой анимации за счет послойной смены изображений. В HTML нет различия между заданием вывода на экран простого GIF или анимированного.

Во-вторых — это прозрачная графика. Это позволяет избавиться от строго прямоугольных иллюстраций, убрать фон из некоторых изображений и вписывать рисунок в документ более привлекательным образом.

В-третьих данных формат поддерживает чересстрочную развертку. Применяется для больших GIF. Иллюстрация разделяется на четыре части. Первая часть содержит 1, 5, 9,... строки изображения. Вторая — 2, 6, 10,... Третья — 3, 7, 11,... Четвертая — 4, 8, 12,... Браузер строит на экране сначала первую часть картинки, потом вторую, затем третью и четвертую. Получается эффект постепенного проявления изображения.

Формат PNG аналогичен формату GIF, но поддерживает большее количество цветов (до 32 бит/пиксел). Также в формате PNG лучше реализована работа с прозрачностью. Возможно этот формат вскоре вытеснит формат GIF.

В стандарте языка HTML5 появилась возможность наряду с растровыми ис-

пользовать и векторные изображениями в формате SVG. Векторные изображения имеют одно существенное преимущество перед растровыми: возможность масштабирования без потери качества.

> **Векторное изображение** — это изображение, состоящее из геометрических примитивов: линий, прямоугольников, овалов и фигур достаточно сложной формы.

Для вставки графического изображения используется тег `<img>`. Тег не парный, но имеет несколько атрибутов. Атрибут `src` позволяет задать имя и путь графического файла.

Вспомним еще раз об абсолютной и относительной адресации. Абсолютная адресация требует указания полного пути к изображению и удобна в том случае, когда используется изображение, являющееся частью чужого ресурса. Если же изображение лежит на одном локальном компьютере (удаленном сервере) с html-документом, то правильнее использовать относительный адрес. Например, тег `<img src=img.gif>` заставит браузер отобразить на экране графический файл `img.gif` из текущего каталога.

Обычно графические файлы не смешивают с текстовыми html-документами, а помещают в отдельный каталог, называя его например `Pic`, который является подкаталогом для каталога с программами (html-файлами), т. е. Каталог `Pic` лежит в одной папке рядом с html-файлами. Тогда тег вывода графики будет иметь вид:

```
<IMG src="./Pic/img.gif">
```

Если браузер не находит картинки в указанном месте на диске, он вместо нее рисует на экране прямоугольник и вписывает в него надпись, которая задана атрибутом `alt`, например:

```
<IMG src="monkey.jpg"  alt="Обезьянка">
```

Текст, указанный в атрибуте `alt`, выводится на экране также в том случае, если изображение найдено, но не успело еще загрузиться. При необходимости вывести всплывающую подсказку у картинки нужно воспользоваться атрибутом `title`:

```
<IMG src="monkey.jpg"  alt="Обезьянка"
      title="Обезьяна Кристал">
```

Свойства стилей `width` и `height` позволяют задавать ширину и высоту (в

пикселях) графического изображения на web-странице. Если данные свойства отсутствуют, то изображение будет выводится в реальном размере.

При масштабировании изображений необходимо помнить, что только векторные изображения масштабируются без потери качества. Растровые изображения при увеличении дают эффект пикселизации. Каждый пиксель заменяется группой пикселей того же цвета, образуя квадратик. При уменьшении изображения происходит обратный процесс: группа пикселей заменяется одним, что снижает четкость мелких деталей.

Масштабировать растровое изображение можно в очень небольших пределах. Если есть в наличии большое изображение, но в html-документе оно должно быть маленьким, то лучше воспользоваться графическим редактором. Существует ряд программ, в которых используются различные алгоритмы сглаживания. Хорошая практика, когда изображения для web заранее «готовятся».

Изображение можно использовать в качестве фона. Фоновое изображение может быть применено почти к любому элементу: абзацу, ячейке таблицы, таблице в целом или web-странице. Для установки изображения в качестве фона используется свойство `background-image`. Данное свойство имеет следующий синтаксис:

```
background-image: url (адрес);
```

Например

```
<BODY style="background-image:
  url (./Pic/my_background .jpg );">
```

В данном примере изображение, находящееся в каталоге `Pic`, установлено на фон всего html-документа. По умолчанию изображение выкладывается плиткой, т. е. многократно повторяется по горизонтали и вертикали. Свойство `background-repeat` управляет заполнением фона. Возможные значения: `repeat` (значение по умолчанию), `no-repeat` — фоновое изображение не повторяется, `repeat-x` — изображение повторяется только по горизонтали, `repeat-y` — изображение повторяется только по вертикали. Например:

```
<BODY style="background-image:
  url (./Pic/my_background .jpg );
  background-repeat: repeat-y">
```

Обычно фон прокручивается вместе с содержимым web-страницы, при этом текст и фон оказываются сцепленными, представляют как бы единое целое. Но фон также можно и зафиксировать, тогда содержимое будет прокручиваться не

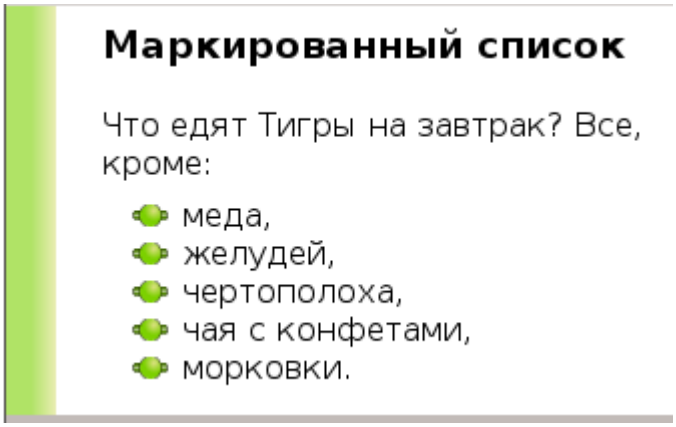
зависимо от фона. За фиксацию фона отвечает свойство `background-attachment`. Это свойство может принимать одно из двух значений: `scroll` (значение по умолчанию) и `fixed` — запрещает прокрутку фона, фиксируя его.

```
<BODY style="background-image: url (./Pic/my_background.jpg
);
        background-attachment: fixed">
```

Достаточно много стилевых свойств описывают состояние фона. Это и `background-color`, и `background-image`, и `background-repeat`, и `background-attachment`... Все эти свойства можно объединить вместе в свойстве `background`, при этом значения различных свойств, описывающих состояние фона элемента, разделены пробелом. Например:

```
<BODY style="background: url (fon.jpg)  repeat-y  fixed ">
```

Как мы помним, в маркированном списке ограниченное количество маркеров: круг, диск и квадрат. Изображение также можно использовать в качестве маркера списка. Для этого в открывающийся тег маркированного списка `<ul>` нужно добавить свойство `list-style-image`. Удобнее использовать маркеры в формате GIF с прозрачным фоном. Например:




```
<UL style="list-style-image: url (./Pic/marker.gif)">
```

Давайте вспомним с вами и о гиперссылках. Пока что в качестве указателя гиперссылки мы использовали текст. Для того, чтобы из картинки сделать указатель гиперссылки, необходимо тег `<img>` вложить внутрь парного тега `<a>`... `</a>`.

```
<A href=переход> <IMG src=файл> </A>
```

Например:

```
<A href="401.htm"> <IMG src="./pic/auto.gif "> </A>
```

Теги, рассмотренные на уроке:		
`<img>`		Графическое изображение
Атрибуты, рассмотренные на уроке:		
`src`		Задает путь к графическому файлу
`alt`		Задает альтернативный текст
`title`		Задает текст всплывающей подсказки
Свойства атрибута *style*, рассмотренные на уроке:		
`width` и `height`	`значение (px)`	Ширина и высота изображения
`background-image`	`URL`	Указывает путь к фоновому изображению
`background-repeat`	`repeat, no-repeat, repeat-x, repeat-y`	Управляет заполнением фона изображением
`background-attachment`	`scroll, fixed`	Управляет фиксацией изображения
`list-style-image`	`URL`	Устанавливает изображение в качестве маркера маркированного списка

Урок 7. Язык CSS

На этом уроке вы познакомитесь с различными способами использования стилей, узнаете как благодаря этому сократить код и сделать его более читабельным.

На протяжении большинства предыдущих уроков мы с вами использовали стили для оформления различных элементов в web-страницы. Мы их использовали непосредственно в тегах html-кода. Никто не запрещает их так использовать. Это даже оправдано в тех случаях, когда элемент встречается на странице один раз, например присутствует только один заголовок 1-го уровня или одна горизонтальная черта.

Но такой подход не удобен в том случае, если у нас встречается большое количество однотипных элементов. Например на странице много абзацев, которые должны выглядеть одинаково. В каждом теге `<p>` размещать повторяющиеся описания одних и тех же стилей не рационально.

Вспомните какая удобная возможность была в таблицах, позволяющая нам группировать столбцы или строки, и задавать единожды оформление для всей группы. Такая возможность есть и в CSS.

> **CSS** (*Cascading Style Sheets)* — каскадные таблицы стилей, служащие для описания внешнего вида документа, созданного при помощи языка разметки.

Стили, встроенные непосредственно в теги элементов html-кода, так и называются: «встроенные». Они имеют самый высокий приоритет. CSS предлагает и другие способы использования стилей. Стили можно оформить в виде отдельной таблицы стилей. Идея использования CSS заключается в том, чтобы отделить оформление web-страницы от ее структуры, описываемой языком HTML.

Таблиц стилей может быть несколько. Браузер последовательно проходит по таблицам, соблюдая иерархию, считывает и применяет стили согласно их приоритету. Такой процесс называется каскадированием. Это действительно напоминает каскад водопадов.

Для подключения таблицы стилей внутри страницы необходимо использовать не атрибут `style`, а тег `<style>`. Данный тег парный и располагается в головной части страницы, ограниченной парой тегов `<head>` … `</head>` после `<title>`. Стили внутри тега `<style>` … `</style>` описываются следу-

ющим образом: указывается элемент html-документа, оформление которого необходимо задать, затем в фигурных скобках располагаются стили. Как и в первом случае стиль состоит из свойства и значения, разделенных двоеточием. Сами стили отделяются друг от друга точкой с запятой, их желательно располагать на разных строках.

```
Элемент {
   свойство_1: значение_1;
   свойство_2: значение_2;
   ...
        }
```

Например:

```
<STYLE>
BODY {
   background-color: black
        }
H1 {
   text-align: center;
   color: gold
        }
P {
   text-align: justify;
   color: yellow
        }
</STYLE>
```

Цвет фона web-страницы в описанном выше примере черный, заголовок первого уровня имеет золотой цвет и выровнен по центру, а текст всех абзацев желтый и выровнен по ширине. Абзацев в тексте может быть много. Везде, где браузер встретит тег `<p>`, к этой части текста будут применены выше описанные стили.

Стили, описанные внутри тега `<style>` … `</style>` называются «глобальными». Их действие распространяется на все элементы html-документа. Использование «глобальных» стилей вместо «встроенных» улучшают html-документ. Во-первых размер кода уменьшается из-за отсутствия дублирования одних и тех же стилей у одинаковых элементов. Во-вторых при таком подходе есть гарантия, что одинаковые элементы будут действительно выглядеть одинаково. В-третьих облегчается редактирование документа. Например нам нужно изменить размер шрифта в тексте для всех абзацев. Для этого нам нужно изме-

нить значение только в одном месте — в таблице стилей.

А теперь представим себе, что мы создает не одну web-страницу, а целый сайт, состоящий из множества web-страниц. Для того, чтобы сайт был выдержан в «одном стиле» необходимо, чтобы страницы сайта выглядели похоже. Неужели нам придется копировать одну и ту же таблицу стилей в разные html-документы?

Для этих целей существует третий способ использования стилей — это «связные» стили. В этом случай таблица стилей выносится в отдельный css-файл. Этот файл текстовый, имеет расширение css. Структура таблицы стилей точно также, как и при глобальном их использовании в теге `<style>`. Пример содержимого css-файла:

```
BODY {
   background-image: url (./Pic/my_background.jpg );
   background-attachment: fixed"
        }
H1 {
   text-align: center;
        }
p {
   text-align: justify;
   color: yellow
        }
```

Для того, чтобы связать файл, содержащий стили, с web-страницей в головном разделе `head` размещается тег `<link>`, содержащий ссылку на css-файл:

```
<LINK rel="stylesheet" href="./css/style.css">
```

Периодически на web-странице возникает необходимость различного оформления однотипных элементов. Рассмотрим пример: имеется текст, содержащий цитаты. Цитаты, как и абзацы основного текста помечены тегом `<p>`. Цитаты должны внешне отличаться от основного текста. Получается, что нам необходимо иметь два типа абзацев. Как быть в случае использования единой таблицы стилей?

В этом случае необходимо разделить абзацы на классы. Каждому классу присваивается имя, которое отделяется от названия элемента точкой. В теле html-документа у элементов страницы указывается к какому классу они относятся.

```
<STYLE>
```

```
  P {
       text-align: justify;
       hyphens: auto;
   }
  P.quote {
       font-weight: bold;
       font-style: italic;
       }
</STYLE>
…
<P> … Информационный взрыв, свидетелями  которого
  мы являемся, прозорливо предвидел еще в 1844 г.
  молодой Энгельс.
<P class="quote"> «Наука, движется вперед пропорционально
  массе знаний, унаследованных ею от предшествующих
  поколений».
```

В таблице стилей мы видим описание обычного абзаца `P`, текст которого выровнен по ширине и содержит автоматические переносы, а также описание абзаца класса `quote`. В таблице стилей он значится как `P.quote`. В теле html-документа у абзацев-цитат будет указана принадлежность к данному классу: `<P class="quote">`. В стилях указано, что цитаты будут выделяться полужирным курсивом.

Здесь нужно сказать о наследовании. Абзацы `P.quote` являются разновидностью абзацев `P`, Наблюдается иерархия элементов: обычный абзац является родительским элементом, а абзац класса `quote` — дочерним. Дочерние элементы наследуют оформление родительских элементов. Это означают что выравнивание по ширине и автоматическая расстановка переносов распространяются на все абзацы, включая и абзацы цитат. В таблице стилей у элемента `P.quote` описаны лишь свойства, которыми он отличается от своего родителя.

Один элемент web-страницы может относиться к нескольким классам одновременно. В таком случае они указываются в кавычках через пробел:

```
<STYLE>
 P.ital {
       font-style: italic
       }
 P.bol {
       font-weight: bold
```

```
        }
</STYLE>
…
<P class="ital bol">
```

В данном примере элемент `P.ital` предписывает делать текст абзаца курсивом, а элемент `P.bol` — делать текст абзаца полужирным. Абзацы описанные как `<P class="ital bol">` будут иметь полужирный курсив.

Теги, рассмотренные на уроке:	
`<style> … </style>`	Раздет описания стилей
`<linc>`	Ссылка

Урок 8. Верстка web-страниц

На этом уроке вы познакомитесь с современными подходами к многоколоночной верстке html-документов, которая предложена стандартом HTML5.

Создание web-сайта состоит из нескольких этапов. Вначале рисуется дизайн-макет страниц сайта, отражающий его внешний вид. Этим занимаются дизайнеры. Дизайн-макет представляет из себя изображение, созданное в одном из графических редакторов. Затем верстальщик переносит этот дизайн-макет в html-документ, создавая структуру будущего сайта. Третий этап — это наполнение сайта содержимым. Как правило этим занимаются совершенно другие люди. Они размещают текстовую, графическую и иную информацию в специально отведенных для этого верстальщиком местах.

Верстка — это взаимное расположение графических и текстовых блоков, заголовков и элементов навигации на web-странице.

Как уже было сказано выше, одним из способов верстки является табличная верстка. Таблица, разбитая на ячейки, позволяет разделить страницу на составные части. Но такой способ верстки на сегодняшний день считается архаичным. Страницы, созданные с помощью табличной верстки, очень часто некорректно отображаются на устройствах с небольшими экранами.

В стандарте HTML5 предлагается другой подход. Для удобства верстки вся web-страница делится на составные части. Их может быть разное количество. В каких-то html-документах одни части могут присутствовать, в то время как в других они могут отсутствовать. Существует ряд тегов для определения этих составных частей.

Первая составная часть страницы — это «шапка». Обычно это самая верхняя часть страницы. Она определяется парным тегом `<header>` и может содержать заголовок страницы, логотип сайта, иногда панель навигации в виде горизонтального меню. Также в этом разделе могут располагаться средства для поиска по сайту, авторизации и т.д..

Вторая часть — это «подвал», который находится в нижней части страницы. Данный раздел формируется парным тегом `<footer>`. Нижний колонтитул мо-

жет содержать информацию о создателях и авторских правах, контактную информацию, ссылки на какие-либо документы и т. д.

Между «шапкой» и «подвалом» располагается раздел, содержащий основной контент. Определяется данный раздел парным тегом `<section>`. Этот раздел может быть единым, а может быть разделен на составные части: статьи, записи в блоге и т. д. В таком случае каждый самостоятельный элемент оборачивается парой тегов `<article>` … `</article>`.

Очень часто в сайтах используется структура, когда между «шапкой» и «подвалом» кроме раздела с основным содержимым есть еще разделы. Их может быть один или два, они могут располагаться слева или справа об блока с основным контентом. Такая верстка называется двух или трехколоночной. Обычно дополнительные колонки создаются парным тегом `<aside>`. Эти колонки содержат информацию, не имеющую прямого отношения к основному контенту. Это могут быть новостные блоки, блоки с рекламой. Также дополнительная колонка может содержать панель навигации, содержащую гипертекстовое меню. Панель навигации размечаются парным тегом `<nav>`.

Создадим html-документ, содержащий детские стихи Валентина Берестова. Используем для этой цели двухколоночную структуру. Правая большая по размеру колонка будет содержать сами стихотворения, а в левой расположим меню, благодаря которому можно будет осуществлять навигацию по контенту.

Только где же этот кто-то,
И куда он мог залезть?

Мяч

Бьют его, а он не злится,
Он поёт и веселится,
Потому что без битья
Нет для мячика житья!

Стихотворения взяты с сайта Дети Онлайн

Верхняя часть («шапка») по ширине занимает всю страницу и содержит лишь заголовок нашей web-страницы:

```
    <HEADER>
        <H2>Детские стихи Валентина Берестова </H2>
    </HEADER>
```

Теперь сделаем самую нижнюю часть нашей страницы («подвал»). В этой части укажем название сайта, откуда взяты стихотворения и сделаем на него ссылку.

```
<FOOTER>
   Стихотворения взяты с сайта
   <A href="https://deti-online.com/"> Дети Онлайн </A>
</FOOTER>
```

Между «шапкой» и «подвалом» нам необходимо разместить две колонки. Правая колонка является разделом `<section>`. В ней каждое стихотворение представим как отдельный элемент, использовав для этого парный тег `<article>`. Название стихотворения — заголовок третьего уровня. Он же будет являться якорем для перехода по гиперссылке, поэтому содержит атрибут `id`. Здесь фрагмент этого раздела:

```
<SECTION>
   <ARTICLE>
       <H3 id="st1">Заиньки </H3>
        <P>Маленькие заиньки <BR>
        Захотели баиньки,<BR>
        Захотели баиньки,<BR>
        Потому что маленьки.
   </ARTICLE>
```

```
    <ARTICLE>
        <H3 id="st2"> Котофей </H3>
        <P>В гости едет котофей,<BR>
        …
```

Теперь сформируем левую колонку, для этого под разделом <section> создадим раздел <aside>. В левой колонке присутствует панель навигации <nav>, представленная в виде маркированного списка. Под панелью навигации краткая текстовая информация о Берестове и его фотография:

```
<ASIDE>
  <NAV>
      <Ul>
          <LI> <A href="#st1"> Заиньки </A>
          <LI> <A href="#st2"> Котофей </A>
          <LI> <A href="#st3"> Искалочка </A>
          <LI> <A href="#st4"> Мяч </A>
      </Ul>
  </NAV>
      Берестов Валентин Дмитриевич (1928–1998) – русский
      детский поэт
  <IMG src="berestov.jpg">
</ASIDE>
```

Посмотрим на наш html-документ в браузере. К сожалению мы не видим пока двух колонок. Все блоки располагаются последовательно друг за другом в том порядке, в котором мы их расположили в html-коде: «шапка», блок основного контента, блок, содержащий панель навигации и дополнительную информацию, «подвал».

Для того, чтобы разделы <section> и <aside> расположились в две колонки между разделами <header> и <footer>, воспользуемся свойством float. Данное свойство определяет по какому краю должен быть выровнен элемент, заставляя остальные элементы обтекать его с другой стороны. В нашем случае раздел <section> должен выравниваться по правому краю, а <aside> — по левому. Зададим размеры каждого раздела. Добавляем таблицу стилей:

```
<TITLE> Стихи Берестова </TITLE>
    <STYLE>
       section {
           float: right;
```

```
            width: 65%;
        }
        aside {
            float: left;
            width: 33%;
        }
   </STYLE>
     </HEAD>
```

Вот теперь у нас получилась структура, которую мы пытались создать. Теперь действительно разделы `<section>` и `<aside>` расположены в две колонки. Две колонки занимают все пространство страницы, под правую отводится 2/3 этого пространства, а под левую — 1/3. Между колонками сохраняется пространство, составляющее 2% от ширины страницы.

Правда «подвал» наслоился на два предыдущих раздела и находится по центру страницы. Но проблема с «подвалом» решается просто. Необходимо запретить у раздела `<footer>` обтекание какими-либо элементами. Для этого воспользуемся свойством `clear`, которое отменяет заданные ранее обтекания. Значение `both` указывает на то, что обтекание отменяется со всех сторон. После добавления данного свойства «подвал» занимает нижнюю часть страницы.

```
footer {
   clear: both;
   }
```

Добавим в таблицу стилей цвет фона для «шапки», «подвала», панели навигации и названий стихотворений в колонке с основным контентом.

```
header {
   background-color: #AFFEFB;
   }
nav {
   background-color: #FFB866;
   }
footer {
   background-color: #AFFEFB;
   clear: both;
   }
H3 {
   background-color: #FCEBC3;
   }
```

Сделаем окантовку вокруг каждого стихотворения как это показано в образце выше. Т.к. все стихотворения являются подразделами раздела `<section>` и обернуты парными тегами `<article>`, то в таблице стилей для элемента `article` добавим свойство `border`, описывающее границы данного элемента. У данного свойства указываются три параметра: первый — толщина границы в пикселях, второй определяет стиль линии (`solid` — плоская одинарная линия), третий задает ее цвет.

```
article {
  border: 1px solid #FDD194;
  }
```

Результат установления свойства `border` — появление границ у элементов `article`. Теперь каждое стихотворение имеет свою рамку. Но рамки эти слиплись между собой и тест прилип к их левым границам. Для корректного отображения необходимо добавить поля и отступы у элемента `article`. Отступы — это расстояние между видимыми или невидимыми границами объектов. За их организацию в CSS отвечает свойство `margin`. Поля — это расстояние между границей и внутренним содержимым элемента, задаются свойством `padding`.

Поля и отступы у объектов задаются очень похоже. Они могут быть заданы одним числом, тогда со всех сторон отступы (поля) будут одинаковы. Если существует необходимость иметь с разных сторон разные отступы или разные поля, тогда они задаются четырьмя числами. В таком случае первое число показывает отступ сверху, второй — отступ справа, третий — с низу и четвертый — слева. Т.е. очередность назначение отступов или полей происходит по часовой стрелке.

Добавим поля и отступы для элемента `article`, при этом разъединив рамки и обечив пространство между рамкой и текстом стихотворения.

```
article {
  border: 1px solid #FDD194;
  margin: 8px;
  padding: 5px;
  }
```

Теги, рассмотренные на уроке:		
`<header> … </header>`		«Шапка» страницы
`<footer> … </footer>`		«Подвал» страницы
`<section> … </section>`		Раздел, содержащий основной контент
`<article> … </article>`		Самостоятельный элемент контента (статья)
`<aside> … </aside>`		Раздел, содержащий информацию, не относящуюся к основному контенту
`<nav> … </nav>`		Панель навигации
Свойства атрибута *style*, рассмотренные на уроке:		
`width` и `height`	`Значение (%)`	Ширина и высота раздела
`float`	`left, right, none`	Задает параметры обтекания элемента
`clear`	`Left, right, both`	Запрещает обтекание элемента
`margin`	`Значение (px)`	Задает отступы вокруг элемента
`padding`	`Значение (px)`	Задает поля внутри элемента

Часть 2. Язык JavaScript

Урок 9. Знакомство с JavaScript

На этом уроке вы познакомитесь с синтаксисом одного из языков, используемых для web-программирования; научитесь составлять линейные программы, организуя ввод, вывод и обработку данных.

На предыдущих уроках мы с вами научились создавать web-страницы, но эти страницы были статическими. Дело в том, что язык HTML не обладает большими возможностями для создания интерактивных страниц. Конечно же язык CSS несколько расширяет эти возможности, но для создания динамических страниц необходимо использовать языки программирования.

Статические web-страницы — это страницы, содержимое и внешний вид которых неизменны. Такие страницы для всех пользователей выглядят одинаково.

Динамические web-страницы — это страницы, содержимое которых постоянно меняется в процессе работы пользователя.

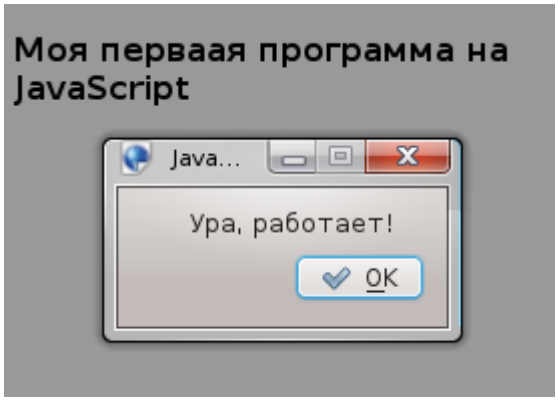
Все языки, используемые для web-программирования можно разделить на две группы: языки, работающие на стороне сервера, и языки, работающие на стороне клиента. У языков первой группы программа обрабатывается сервером, браузеры клиентов получают лишь результат работы этой программы. К таким языкам относится PHP. У языков второй группы программа обрабатывается на клиентских компьютерах и никак не влияет на работы других клиентов. К таким языкам относится JavaScript.

JavaScript – язык программирования, интерпретируемый браузером. Программа, написанная на JavaScript называется скриптом или сценарием. Она представляет из себя текст, не требующий компиляции, может помещаться внутри HTML-документа или располагаться в отдельном файле. Коды JavaScript не заменяют, а лишь дополняют коды HTML.

При размещении кода внутри html-документа, он размещается в теле документа между парными тегами `<script>`, например:

```
<!DOCTYPE html>
<HTML>
  <HEAD>
      <META charset="utf-8" />
      <TITLE> JavaScript </TITLE>
  </HEAD>
  <BODY>
      <H3> Моя первая программа на JavaScript </H3>
      <SCRIPT>
          alert (' Ура, работает! ');
      </SCRIPT>
      <P> Программа закончена
  </BODY>
</HTML>
```

Браузер считывает и выполняет html-код последовательно сверху вниз. Сначала выводится заголовок «Моя первая программа на JavaScript», затем выполняется скриптовая часть. Ее результат показан на картинке: всплывающее окно с надписью «Ура, работает!». После нажатия на кнопку «Ok» всплывающее окно закрывается и выводится надпись «Программа закончена».



Пример, рассмотренный выше, содержит скрипт, в котором использована стандартная функция `alert`. Она используется для вывода информации на экран. Ее синтаксис:

```
alert ("текстовая строка");
```

При ее вызове на экране появляется окно с текстовым сообщением и одной единственной кнопкой «Ok». Данное окно является модальным, т. е. до тех пор, пока пользователь не закроет окно, он не сможет взаимодействовать с web-страницей. Сообщение в окне выводится в одну строку. Если необходимо вывести текст в две строки, для переноса строк используется комбинация символов «\n».

Для вывода информации непосредственно на страницу html-документа мы будем использовать метод `document.write`. Текст, выводимый с помощью `alert` или `document.write` располагается в кавычках или апострофах:

```
document.write (“надпись”);
```

```
document.write ('надпись');
```

Для ввода информации используется функция `prompt`. Данная функция формирует всплывающее модальное окно, содержащее текстовую строку, под которой располагается поле для ввода текста. Внизу окна кнопки «Ok» и «Отмена». Функция `prompt` содержит два текстовых параметра, разделенных запятой: надпись, выводимая в окне текстовая строка и значение по умолчанию в поле ввода. Если значение по умолчанию не нужно и поле для ввода текста должно остаться пустым, то второй параметр — пустые кавычки (апострофы)

```
prompt ("надпись", "значение ввода по умолчанию");
prompt ('надпись', ' ');
```

Для хранения информации, как и в других языках программирования, используются переменные. При объявлении переменных используется ключевое слово `var`, после которого указывается имя переменной:

```
var x;              // Переменная с именем x
```

В отличие от большинства других языков программирования, при объявлении переменной не указывается ее тип. Язык JavaScript является слабо типизированным. Одна и та же переменная может хранить информацию разных типов. В процессе выполнения программы браузер сам пытается определить какого типа данные находятся сейчас в переменной.

Переменные необязательно объявлять вначале, она может быть описана в любом месте программы до ее использования. При объявлении переменной может происходить ее одновременная инициализация, переменной может быть сразу присвоено значение:

```
var y = 10;               // Описание с присвоением числа
var name = "Есенин";      // Описание с присвоением строки
```

Для арифметических операций в языке JavaScript используются следующие знаки:

- `+` сложение (например `x+y`);
- `-` вычитание (например `x-10`);
- `*` умножение (например `2*x`);
- `/` деление (например `y/7`);
- `%` остаток от деления (например `x%3`);

Для строковых выражений используется операция соединения, обозначаемая знаком “+”. В результате сцепления несколько строк соединяются в одну.

Например в следующем примере переменная `poet` принимает значение «Демьян Бедный» :

```
poet = "Демьян " + "Бедный";
```

Когда операция “+” связывает строку с другим типом, то браузер расценивает данную операция как сцепление. При этом результат преобразуется к строковому типу данных. В следующем примере переменная z получит значение “45”

```
z = 4 + "5";
```

Как мы видели в предыдущих примерах, операция присваивания в языке JavaScript обозначается знаком “=”. Если рассматривать переменную как некий контейнер, то присваивание есть процесс помещения в данный контейнер некого содержимого. Содержимым переменной может быть число, строка или иной тип данных. Операция присваивания может сочетаться с другими операциями, например:

```
z += "синий кит";
```

В данном примере переменной `z` присваивается результат сцепления строки "синий кит" с предыдущим значением переменной `z`, т. е. вышеописанная операция — более короткий способ записи операции `z = z + "синий кит"`.

```
P -= 4;
```

В данном примере переменной `P` присваивается результат вычитания числа 4 из предыдущего значение переменной `P`, т. е. это выражение соответствует выражению `P = P – 4`.

```
P *= 12;
```

В данном примере переменной `P` присваивается результат умножения предыдущего значения переменной на число 12, т. е. предыдущее значение увеличивается в 12 раз. Это соответствует выражению `P = P * 12`.

```
P /= y-x;
```

В данном примере переменной `P` присваивается результат деления предыдущего значения переменной на выражение, справа от знака присваивания. Это соответствует выражению `P = P / (y-x)`.

```
P %= 5;
```

В данном примере переменной `P` присваивается остаток от деления предыдущего значения переменной на пять, что соответствует выражению `P = P % 5`.

В языка JavaScript возможна и такая запись:

```
z++;
++z;
z--;
--z;
```

В первых двух случаях значение переменной z увеличивается на единицу, а в третьем и четвертом — уменьшается на единицу. Получается, что запись в первой и второй строке равнозначна записи `z = z+1`, а запись в третьей и четвертой строке равнозначна записи `z = z-1`.

В чем же разница между первой и второй, и третьей и четвертой записью? Разница в приоритете операций, который необходимо учитывать при взаимодействии с другими операторами. Рассмотрим пример.

```
a = 7;
b = ++a;
```

После выполнения данного фрагмента программы обе переменных будут иметь значение 8. В первой строке переменной `a` присваивается значение 7. Во второй строке вначале значение переменной `a` увеличивается на 1, а затем новое значение переменной `a` присваивается переменной `b`.

```
a = 7;
b = a++;
```

Во втором примере после выполнения данного фрагмента программы переменные будут иметь разные значения. В первой строке переменная `a` принимает значение 7. Во второй строке сначала переменной `b` присваивается значение переменной `a`, которое равно 7, а после этого происходит увеличение значения переменной `a` на 1. После операции `b=7`, `a=8`.

Давайте попробуем составить простейшую вычислительную программу. Программа будет запрашивать у пользователя число и увеличивать его в пять раз. Ниже представлен полный html-код страницы, содержащий программу:

```
<!DOCTYPE html>
<HTML>
   <HEAD>
      <META charset="utf-8">
   <TITLE> Умножение числа </TITLE>
   </HEAD>
   <BODY>
      <H1> Умножение числа </H1>
```

```
        <HR>
        <SCRIPT>
         var a = prompt ('Введите число', '1');
         document.write ('Вы ввели число ', a );
         a*=5;
         document.write ('<br> Мое число в 5 раз больше – ',
a);
        </SCRIPT>
     </BODY>
  </HTML>
```

Поясним некоторые моменты. В первой строке скрипта объявляется переменная `a` и ей сразу же присваивается значение, полученное от функции `prompt`, т. е. введенное пользователем в поле ввода всплывающего окна. Если пользователь сразу нажмет кнопку «Ok», то переменной `a` будет передано значение по умолчанию, в нашей программе оно равно 1.

В последней строке скрипта при выводе результата присутствует тег `<br>`. Он используется для разметки выводимого в html-документе текста. В противном случае весь текст, выводимый `document.write` будет размещен в одну строку. Причем теги html-разметки являются строковыми данными, поэтому помещены вместе с выводимой информацией внутрь апострофов (кавычек).

Задачи для самостоятельного решения

- Дана сторона квадрата `a`. Найти его периметр. Для нахождения периметра квадрата используйте формулу `P = 4·a`.
- Даны стороны прямоугольника `a` и `b`. Найти площадь и периметр прямоугольника. Для нахождения площади используйте формулу `S = a·b`, а для нахождения периметра `P = 2·(a+b)`.
- Даны длины ребер прямоугольного параллелепипеда `a`, `b` и `c`. Найти его объем и площадь поверхности. Для нахождения объема используйте формулу `V = a·b·c`, для нахождения площади `S = 2· (a·b + b·c + a·c)`.

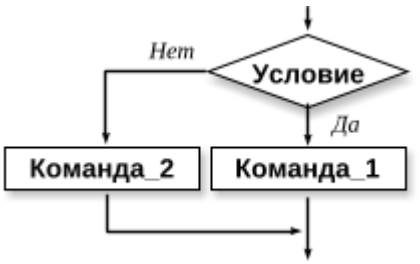
Урок 10. Ветвление

На этом уроке вы познакомитесь с записью конструкции ветвления в полной и неполной форме на языке JavaScript, рассмотрите вложенные ветвления, а также операторы выбора.

На предыдущем уроке мы рассмотрели линейные программы. Они содержали команды ввода и вывода, а также арифметические операции. Все они выполняются последовательно друг за другом. Если построить к этим программам блок-схему, то она будет выглядеть как цепочка блоков, расположенных в одну линию.

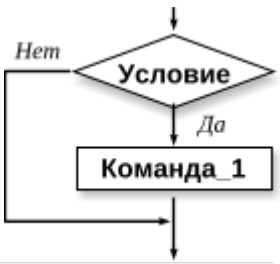
Сегодня мы рассмотрим программы, содержащие ветвление. Ветвящиеся программы нелинейны. Данная конструкция позволяет выбрать каким путем пойдет выполнение программы. Выбор того или иного пути будет зависеть от выполнения условий.

> **Ветвление** – это такая форма организации действий, при которой в зависимости от выполнения некоторого условия совершается либо одна, либо другая последовательность действий.



Существует две разновидности ветвления: в полной и неполной форме. Ветвление в полной форме содержит проверку условия и две альтернативные команды. В данном случае имеет место два варианта выбора. Если условие выполняется, т. е. является истинным, то выполняется одна команда, иначе если условие не выполняется, т. е. является ложным, то выполняется другая команда. Условие в JavaScript записывается в скобках. Первая строка кода отражает ситуацию, когда условие выполняется, вторая — когда условие не выполняется.

```
If (условие) команда_1;
else команда_2;
```



При неполной форме ветвления отсутствует вторая альтернатива. Если условие выполняется, т.е. является истинным, то команда выполняется, в противном случае она просто пропускается. В случае использования неполной формы выбор заключается в том, выполнять данную команду или нет.

```
If (условие) команда_1;
```

Для обозначения операций сравнения в условиях могут присутствовать сле-

дующие знаки:

==	Равно
!=	Не равно
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно

При построении сложных условий простые условия соединяются между собой логическими связками. При этом простые условия в скобки брать не нужно, т. к. в языке JavaScript операции сравнения имеют более высокий приоритет, а логические операции — более низкий. Они обозначаются следующими знаками:

&&	Логическое «и»
||	Логическое «или»
!	Логическое отрицание

Попробуем составить простейшую скриптовую программу, содержащую ветвление.

```
var z = prompt ('Сколько будет 5+5?', '11');
if (z==10) alert ('Все верно');
else alert ('Ты шутишь?');
```

В данном примере функция `prompt` выводит на экран всплывающее окно, содержащее вопрос «Сколько будет 5+5?» и поле ввода, в котором по умолчанию стоит значение 11. Расчет здесь на то, что пользователь может, не изменив значения по умолчанию, нажать кнопку «Ok». При нажатии на эту кнопку, переменной `z` присваивается значение, находящееся в поле ввода всплывающего окна. Далее следует проверка значения переменной `z`, и если оно верное (равно 10), то выводится сообщение «Все верно!». В противном случае выводится сообщение «Ты шутишь?».

В следующем примере ветвление содержит сложное условие, объединяющее в себе два простых с помощью логической операции «или». Условие будет истинным в том случае, когда хотя бы одно из двух простых условий будет истинным. Программа задает пользователю вопрос по поводу количества выходных на этой неделе. И в случае одного выходного, и в случае двух программа выдаст ответ «И у меня тоже!». Во всех остальных случаях появится модальное окно с надписью «У тебя каникулы?».

```
var a = prompt ('Сколько выходных на этой неделе?', '2');
if (a==1 || a==2) alert ('И у меня тоже!');
else alert ('У тебя каникулы?');
```

Одно ветвление может находится внутри другого ветвления, являясь его составной частью. Такие ветвления называются вложенными. Степень вложенности лучше показывать отступами.

```
if (условие1) команда_1;
else if (условие2) команда_2;
     else if (условие3) команда_3;
        …
        else команда_N;
```

Вернемся к примеру со сложением чисел. В первой строке программы функция `prompt` вызывает всплывающее окно для ввода данных. Как мы помним окно содержит две кнопки «Ok» и «Отмена». При нажатии на кнопку «Ok» переменной `z` присваивается значение, находящееся на этот момент в поле ввода. А что произойдет, если пользователь нажмет кнопку «Отмена»? Значение переменной не присваивается. Какое же значение она будет иметь? Никакого (`null`), переменная остается пустой. Вот этой ситуации мы не предусмотрели. Усовершенствуем программу:

```
var z= prompt ('Сколько будет 5+5?', '11');
if (z==null) alert ('Можно было и посчитать...');
else if (z==10) alert ('Все верно');
     else alert ('Ты шутишь?');
```

После ключевых слов `if` и `else` в ветвлении может находиться только один оператор, в том числе и другой оператор ветвления в случае вложенных ветвлений. При необходимости использования нескольких операторов после слов `if` и `else`, необходимо образовать из них составной оператор или блок. Для этого используются фигурные скобки. Цепочка команд, заключенная в фигурные скобки рассматривается как один оператор. Символ “;” за скобкой “}”, закрывающей блок, не ставится.

```
{
   команда_1;
   команда_2;
   …
}
```

Если переменная проверяется на ограниченный набор значений, то удобнее

использовать оператор выбора. В скобках после ключевого слова `switch` указывается переменная, после слов `case` — возможные ее значения, `default` отражает ситуацию, когда ни одно из предложенных значений не подошло.

```
switch (переменная)
{
  case значение_1:
    команды;
    break;
  case значение_2:
    команды;
    break;
  …
  default:
    команды;
}
```

По своей сути запись «`case значение_1:`» соответствует записи «`if переменная == значение_1`». Рассмотрим пример программы, использующей оператор выбора. У пользователя запрашивается номер месяца и выводится количество дней в этом месяце для невисокосного года.

```
Var month = prompt ('Введите номер месяца', '1');
switch (month)
{
  case 1:
    alert ('В месяце 31 день');
    break;
  case 2:
    alert ('В месяце 28 дней');
    break;
  case 3:
    alert ('В месяце 31 день');
    break;
  …
  case 12:
    alert ('В месяце 31 день');
    break;
  default:
    alert ('Неверный номер месяца');
}
```

Здесь показан лишь фрагмент оператора выбора. У переменной `month` может быть двеннадцать возможных значений. Запись оператора `switch` несомненно короче двенадцати вложенных ветвлений, но в данном случае все равно достаточно длинная. Его запись можно оптимизировать:

```
Var month = prompt ('Введите номер месяца', '1');
switch (month)
{
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12:
    alert ('В месяце 31 день');
    break;
  case 2:
    alert ('В месяце 28 дней');
    break;
  case 4:
  case 6:
  case 9:
  case 11:
    alert ('В месяце 30 дней');
    break;
  default:
    alert ('Неверный номер месяца');
}
```

Задачи для самостоятельного решения

- Запросив у пользователя число, вывести сообщение о том, является ли оно четным или нечетным.
- Запросив у пользователя три числа, вывести сумму наибольшего и наименьшего чисел.
- Запросив у пользователя оценку, вывести ее описание: 1 – «плохо», 2 – «неудовлетворительно», 3 – «удовлетворительно», 4 – «хорошо», 5 – «отлично». Если введенное значение не лежит в диапазоне от 1 до 5, то вывести строку «Такой оценки нет».

Урок 11. Циклы

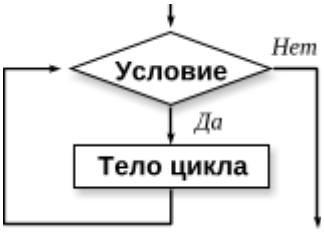
На этом уроке вы рассмотрите различные типы циклов: циклы с предусловием, с постусловием и со счетчиком; познакомитесь с особенностями записи циклов в языке программирования JavaScript; рассмотрите команды, позволяющие управлять работой циклов.

Нередко в программах возникает ситуация, когда какие-то операции повторяются несколько раз. Если повторение происходит в одном месте программы, то для такой организации действий удобно использовать циклы.

> **Цикл** – это конструкция, позволяющая многократно выполнять серию команд.

Однократное повторение в цикле называется итерацией, а повторяющаяся часть — телом цикла. Циклы в языках программирования делятся на две группы: циклы, использующиеся в ситуациях, когда количество итераций заранее известно, либо оно легко рассчитывается, и циклы, использующиеся в ситуациях, когда количество итераций к моменту начала выполнения программы остается неизвестным.

Ко второй группе циклов относятся циклы с условием. Количество итераций в данном случае будет зависеть от выполнения некоего условия. Среди циклов с условием выделяют циклы с предусловием и циклы с постусловием.



В циклах с предусловием условие проверяется в начале. Тело цикла выполняется до тех пор, пока условие выполнятся. При использовании такого типа циклов возможна ситуация, когда условие изначально ложно. В таком случае тело цикла не будет выполнено ни разу.

```
while (условие)
  команда;
```

В циклах с постусловием сначала выполняется итерация цикла, а только затем проверяется условие. Это гарантирует, что тело цикла будет выполнено как минимум один раз. В отличие от циклов с предусловием, в которых выполнение условия являлось условием вхождения в цикл, в циклах с постусловием при выполнении условия происходит его завершение.

```
do команда
while (условие);
```

В теле цикла может находиться только одна команда. При необходимости использования нескольких команд в теле цикла, необходимо образовать из них составной оператор (блок), объединив операторы фигурными скобками. Цепочка команд, заключенная в фигурные скобки рассматривается как один оператор. Напомним еще раз, что символ “;” за скобкой “}”, закрывающей блок, не ставится.

Рассмотрим пример использования цикла с условием. Выведем ряд натуральных чисел от 1 до n. Количество чисел определяет пользователь и до начала программы оно неизвестно. Для вывода ряда сформируем строку S, к которой будем прицеплять числа, располагая их через запятую. С каждой итерацией число увеличивается на единицу.

```
var S = '1';
var i = 1;
var n = prompt ('Введите конечное число', '10');
while (i < n)
{
   i++;
   S += ', ' + i;
}
alert (S);
```

Если количество итераций заранее известно, то удобнее воспользоваться циклом со счетчиком. В качестве счетчика выступает переменная, значение которой увеличивается с каждой итерацией. В языке JavaScript счетчик состоит из трех частей: первоначальной инициализации переменной счетчика, условия выполнения цикла и приращения переменной счетчика.

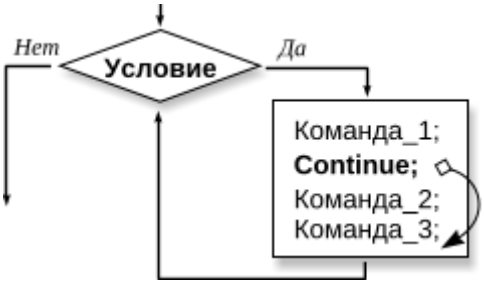
```
for (начало; условие; приращение)
  команда;
```

Для иллюстрации программы, содержащей цикл со счетчиком, составим программу нахождения суммы натуральных чисел в интервале от 1 до 20. В счетчике переменной `i` присваивается первоначальное значение 1, которое с каждой итерацией увеличивается на единицу. Цикл будет работать до тех пор, пока значение переменной счетчика не достигнет 20.

```
var i;
var sum = 0;
for (i=1; i<=20; i++)
       sum += i;
```

```
alert ("Сумма 1+2+...+20="+sum);
```

Для управления циклом используются команды `break` и `continue`. Команда `continue` перебрасывает выполнение на конец тела цикла, обрывая итерацию. После чего цикл переходит к следующей итерации.

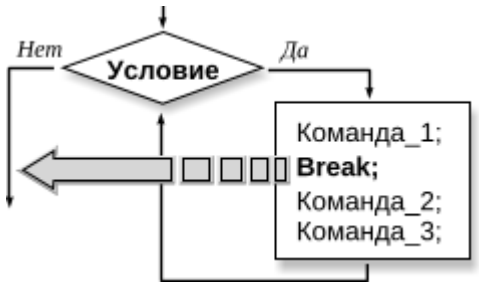


Составим программу, выводящую все четные натуральные числа в диапазоне от 1 до 30. В цикле со счетчиком перебираем числа, проверяем каждое из них на четность. Из четных чисел формируем строку, которая будет выведена по завершении цикла, нечетные числа пропускаем, используя команду `continue`. В строке выводимые числа будут располагаться через пробел.

Как проверить является ли число четным? У четных чисел остаток от деления на 2 равен 0. Если остаток от деления на 2 не равен 0, то значит число нечетное.

```
ar i;
var str = "";
for (i=1; i<=30; i++)
{
  if (i%2 != 0) continue;
  str += i + '  ';
}
alert (str);
```

Иногда во время выполнения программы возникает ситуация, когда цель, стоявшая перед циклом, достигнута еще до его завершения и продолжение цикла нежелательно. В таком случае используется `break`. В отличие от `continue`, команда `break` разрывает цикл, досрочно его завершая. В ситуации когда циклы вложенные, разрывается только один цикл, в теле которого находится команда `break`.



Составим скриптовую программу, выводящую в html-документе ряд чисел, случайным образом сгенерированных браузером в интервале от 1 до 10. Ряд не должен содержать числа 6 и 8. При их выпадении, браузер должен формировать ряд заново с новой строки.

Для генерации случайных чисел нам потребуется функция

`Math.random()`. Данная функция возвращает вещественные числа в диапазоне от 0 до 1. Полученное число будем умножать на 9, прибавлять 1 и округлять до целого значения функцией `Math.floor()`. При умножении на 9 мы получаем числа в диапазоне от 0, если функция `Math.random()` генерирует 0, до 9 в том случае, если `Math.random()` генерирует 1. Прибавление 1 к конечному числу сдвигает диапазон на 1. Полностью конструкция получения случайных чисел выглядит следующим образом:

```
ch = Math.floor(1 + 9*Math.random());
```

В программе объявляем две переменные: `ch` — для хранения случайных чисел, генерируемых браузером, и `i` — счетчик чисел, хранящий количество чисел в ряду. Используем два вложенных цикла. Внутренний цикл со счетчиком формирует ряд чисел, его максимальная длина 10. При выпадении 8 и 6 ряд обрывается. Внешний цикл — цикл с предусловием, он перебирает попытки. Выход из цикла происходит при удачной попытке, когда в ряду действительно оказывается десять чисел.

```
var i=0;
var ch;
while (i<10)
{
   for (i=1; i<10; i++)
   {
        ch = Math.floor(1 + 9*Math.random());
        document.write ( ch, ' ');
        if (ch==8 || ch==6) break;
   }
   document.write ('<br> ');
}
```

Задачи для самостоятельного решения

- Запросив у пользователя число, вывести на страницу html-документа все числа от 1 до введенного числа.
- Запросив у пользователя 10 целых чисел, вывести на страницу html-документа все эти числа, а также количество четных из них.
- Составить программу, перебирающую все натуральные числа в диапазоне от 20 до 50, и выводящую те из них, которые делятся на 3, но не делятся на 5.

Урок 12. Функции

На том уроке мы поговорим о том как создавать и как использовать пользовательские функции, возвращающие или не возвращающие значение.

В программировании часто бывает нужно одну и ту же группу команд повторять несколько раз. Если повторение происходит в одном месте, то удобно используют циклы, рассмотренные ранее. Но если код нужно повторять в разных местах программы — его оформляют в виде функции. Функции применяются также для того, чтобы разделить программу на составные части, являющиеся отдельными смысловыми блоками. Особенно такой подход актуален при составлении больших программ.

> **Функция** – это часть компьютерной программы, содержащая описание определённого набора действий. Функция может быть многократно вызвана из разных частей программы.

Как видно из определения, функция состоит из двух частей: описания функции и вызова функции. Мы уже сталкивались с функциями ранее. На предыдущем уроке мы использовали функции `Math.random()` и `Math.floor()`. Это стандартные функции, их описание включено в библиотеки языка JavaScript, поэтому они еще называются библиотечными. Нам нужно было лишь вызвать эти функции, подставив в них определенные аргументы.

Список библиотечных функций достаточно велик. Но предусмотреть все ситуации в библиотеке невозможно. Поэтому при составлении программ приходится создавать свои собственные, пользовательские функции. Пользовательскую функцию необходимо описать самостоятельно. Описание функции начинается словом `function`, после которого идет название функции и в скобках через запятую список аргументов. Тело функции располагается между фигурными скобками и заканчивается командой `return`, возвращающей результат.

```
function ИмяФункции (список аргументов)
{
  команда_1;
  команда_2;
  …
  return значение;
}
```

Для вызова функции в программе словом `function` уже не пишется, указывается лишь ее имя со списком передаваемых параметров. Причем название

функции в описании и ее вызове одинаковы как близнецы-братья.

```
ИмяФункции (список аргументов);
```

Рассмотрим использование функций в скриптовой программе, находящей максимальное число в паре двузначных чисел. Опишем функцию `MaxCh()`, в качестве аргументов принимающую значения `a` и `b`. Внутри функции происходит сравнение этих значений и максимальное присваивается переменной `max`. В основной части программы функция `MaxCh()` вызывается несколько раз для сравнения пары введенных чисел, а также сравнения каждого из этих чисел с числами 33 и 73.

```
function MaxCh (a,b)
{
  var max;
  if (a>b) max=a;
  else max=b;
  return max;
}

var a = prompt ('Введите первое двузначное число', '');
var b = prompt ('Введите второе двузначное число', '');
document.write ('Вы ввели числа ', a, ' и ', b,
'<br> Из двух введенных наибольшее число ', MaxCh(a,b),
'<br> Сравним 1-е число с числом 33, наибольшее ', MaxCh(a, 33),
'<br> Сравним 2-е число с числом 33, наибольшее ', MaxCh(33, b),
'<br> Сравним 1-е число с числом 73, наибольшее ', MaxCh(a, 73),
'<br> Сравним 2-е число с числом 73, наибольшее ', MaxCh(73, b)
 );
```

Обратим внимание внимание на переменную `max`, которая объявлена внутри функции `MaxCh` и предназначенная для временного хранения максимального числа. Такая переменная называется локальной. С этой переменной могут работать только операторы данной функции, операторам остальной части программы про данную переменную ничего не известно.

> **Локальная переменная** - это переменная, объявленная внутри функции и доступная только в ее пределах.

Команда `return` в функции может отсутствовать. В таком случае случае функция не возвращает никакого значения, а лишь выполняет группу действий.

Проиллюстрируем примером программы, выводящей на экран таблицу сложения. Выведем таблицы сложения для чисел 3, 4, 5 и 6.

```
function SumTabl (x)
{
  var i, j;
  for (i=1; i<x; i++)
  {
      for (j=1; j<x; j++)
      {
          document.write(i,'+',j,'=',i+j, '; ')
      }
      document.write('<br>')
  }
  document.write('<br>')
}

SumTabl (3);
SumTabl (4);
SumTabl (5);
SumTabl (6);
```

Для вывода результатов по строкам и столбцам, имитирующим таблицу, нам потребовалось два цикла со счетчиком. Внутренний цикл перебирает суммы внутри строки, после его окончания происходит переход на новую строку. Внешний цикл перебирает строки.

Задачи для самостоятельного решения

- Запросив у пользователя три числа, вывести на страницу html-документа среднее арифметическое трех пар чисел. Для расчета среднего арифметического опишите и используйте функцию `Aver(x,y)`.
- Запросив у пользователя радиусы трех кругов, вывести на страницу html-документа длины окружностей этих кругов. Для расчета длины окружности опишите и используйте функцию `Circumference(r)`.
- Запросив у пользователя длины сторон пяти квадратов, вывести суммарную площадь этих фигур. Для рассчета площади квадрата опишите и используйте функцию `Square(a)`.

Урок 13. Массивы

На том уроке вы узнеате о том, что такое массив, как он заполняется и выводится; научитесь определять и изменять размер массива, сортировать его, выполнять операции с элементами массива.

Переменная в программе, как уже нам известно, хранит лишь одно значение. Это значение может меняться в ходе работы программы, но у переменной оно всегда будет одно. Нередки ситуации, когда в программе нужно хранить набор значений. Например программа, определяющая динамику цен на сотовые телефоны, должна хранить большое количество цен из разных магазинов города. Использование большого количества переменных катастрофически снижает эффективность программы, усложнаяя ввод, вывол и обработку данных. Для хранения большого количества значений хорошо подходят массивы.

> **Массив** – это множество нумерованных значений, объединенных общим именем.

Массив, в отличие от переменной, может хранить достаточно большую (насколько это позволяет оперативная память компьютера) группу значений. Каждый элемент массива имеет свой индекс.

В языке JavaScript массив оформляется как заключенный в скобки набор элементов, разделённых запятыми.

```
[1, 2, 4, 3, 8, 5]
```

В одном массиве могут храниться элементы разных типов:

```
[1, 'один', 1.5, 2, 'два', 2.2]
```

Массив может не содержать ни одного элемента. Такой массив называется пустым:

```
[]
```

Для объявления массива используется ключевое слово `var`. В программе массив может быть объявлен пустым или проинициализирован, т. е. элементам массива могут быть сразу присвоены значения:

```
var A = [];
var B = [3, 4, 'пять', 6];
```

Если значение нужно присвоить не всем элементам и какие-то элементы необходимо пропустить , то между запятыми просто оставляется пустое место.

В этом случае пропущенные элементы оказываются неопределенными и имеют значение `undefined`.

```
var B = [3, , 'пять', 6];
```

Обращение к элементам массива происходит по их индексам. Индекс указывается в квадратных скобках после имени массива. Индексация элементов в массиве начинается с нуля, т. е. первый элемент массива имеет индекс 0, второй элемент — индекс 1, третий — 2 и т. д.

```
var Ar = [1, 2, 3, 4];
alert (Ar[0]);  // Выведет «1»
alert (Ar[2]);  // Выведет «3»
```

В языке JavaScript можно обратиться к элементу массива с любым индексом, даже если данный элемент отсутствует при инициализации массива. В этом случае не возникает распространенной в других языках ошибки «Выход за границы массива». Считается, что если пользователь обратился к какому-либо элементу, то значит он ему нужен. Он будет создан для пользователя и иметь значение `undefined`.

```
var Ar = [1, 2, 3, 4];
alert (Ar[10]); // Выведет «undefined»
```

Еще одно отличие языка JavaScript от большинства других языков заключается в том, что для вывода всех элементов массива достаточно указать имя массива. Здесь не требуется вывода отдельных элементов в цикле.

```
var Ar = [1, 2, 3, 4];
alert (Ar);     // Выведет «1,2,3,4»
```

Элементы массива являются индексированными переменными, их значения могут меняться в процессе выполнения программы. У элемента массива может меняться даже тип хранимых данных:

```
var Ar = [1, 2, 3, 4];
Ar[1] = 'Два';
alert (Ar);     // Выведет «1,Два,3,4»
```

В процессе работы программы неопределенные и несуществующие элементы также могут быть проинициализированы. В примере, рассмотренном ниже, объявлен пустой массив. Инициализация первых пяти элементов от 0-го до 4-го осуществляется в цикле, элемент с индексом «7» инициализируется отдельно. В результате заполнения два элемента `Ar[5]` и `Ar[6]` оказались незаполненными, они будут иметь значение `undefined` и при выводе массива на месте этих элементов будут пропуски.

```
var Ar = [ ];
var i
for (i = 0; i < 5; i++)
{
  Ar[i] = i;
}
Ar[7] = 7;
alert (Ar);      // Выведет «0,1,2,3,4,,,7»
```

Для получения длины массива используется свойство `length`. Его значение правильнее воспринимать не как количество элементов массива, а как индекс максимального элемента, увеличенный на единицу. Прибавляемая единица корректирует ситуацию, когда нумерация начинается с нуля, а не с единицы.

```
var Ar = [1, 2, 3, 4];
alert (Ar.length);        // Выведет «4»
Ar[7] = 'восемь';
alert (Ar.length);        // Выведет «8»
alert (Ar);           // Выведет «1,2,3,4,,,,восемь»
```

Значение свойства `length` можно изменять вручную, переприсваивать. Если присвоить этому свойству значение большее, чем количество элементов в массиве, то произойдет добавление элементов. Добавленные элементы будут неопределенными, т. е. иметь значение `undefined`.

```
var Ar = [3, 4, 5];
Ar.length = 6;
alert (Ar);           // Выведет «3,4,5,,,»
```

Если присвоить свойству `length` значение меньшее, чем количество находящихся в данный момент в массиве элементов, то последние элементы будут удалены.

```
var Ar = [5, 4, 3, 2, 1, 0, 1, 2];
Ar.length = 5;
alert (Ar);           // Выведет «5,4,3,2,1»
```

Есть и другие способы изменить размер массива. Метод `pop` отрезает последний элемент массива, при этом длина массива уменьшается на единицу. Отрезанный элемент может быть присвоен другой переменной или выведен.

```
var Vegets = ["Морковь", "Свекла", "Капуста", "Картофель"];
var v = Vegets.pop ( );
alert (v);            // Выведет «Картофель»
alert (Vegets);       // Выведет «Морковь,Свекла,Капуста»
```

Метод push добавляет в конец массива любое количество элементов. Добавляемые элементы указываются в круглых скобках, при этом push возвращает новую увеличенную длину массива. Это значение также может быть присвоено какой-либо переменной или выведено.

```
var Poets = ["Пушкин", "Лермонтов"];
var dl = Poets.push ("Блок", "Есенин" );
alert (dl);           // Выведет «4»
alert (Poets);  // Выведет «Пушкин,Лермонтов,Блок,Есенин»
```

Методы pop и push изменяют размер массива, добавляя и удаляя элементы с конца. Есть методы, которые позволяют добавлять и удалять элементы в начале массива. Метод shift извлекает первый элемент массива, возвращая его. Благодаря этому методу массив сокращается на один элемент.

```
var Vegets = ["Морковь", "Свекла", "Капуста", "Картофель"];
var v = Vegets.shift ( );
alert (v);            // Выведет «Морковь»
alert (Vegets);       // Выведет «Свекла,Капуста,Картофель»
```

Метод unshift по аналогии с методом push будет добавлять в начало массива произвольное количество элементов и возвращать новую длину массива. Добавляемые элементы также указываются в круглых скобках данного метода.

```
var Poets = ["Пушкин", "Лермонтов", "Есенин"];
var dl = Poets.unshift ("Блок");
alert (dl);           // Выведет «4»
alert (Poets);  // Выведет «Блок,Пушкин,Лермонтов,Есенин»
```

Используя данные методы изменения массива необходимо помнить, что методы push и pop выполняются браузером быстро, а shift и unshift значительно медленнее. Это связано с тем, что работая с концом массива никаких лишних действий кроме добавления и удаления элементов производить не нужно, тогда как работа с началом массива неизбежно связана с перемещением элементов.

Для того, чтобы вставить какой-либо элемент в начало массива, необходимо сначала освободить для него место. Вставляемый элемент станет первым, но у нас уже есть первый элемент. Переместим первый элемент на место второго, второй — на место третьего и т. д. Таким образом вставка элементов в начало сопровождается циклическим сдвигом всех элементов. При удалении первого элемента происходит сдвиг в обратном направлении.

Для обработки массива, как и с других языках программирования, использу-

ется цикл со счетчиком. В цикле перебираются все элементы и совершаются некоторые действия. Ниже приведен пример программы нахождения суммы элементов массива.

В начале программы объявляем пустой массив `Ar`, переменную счетчик `i` и переменную `sum` для накопления суммы. До того, как мы начали подсчитывать сумму, сумма должна быть равна 0. Запускаем цикл на пять итераций. В цикле каждый элемент заполняем случайными числами в диапазоне от 5 до 15, выводим его на страницу html-документа через пробел и увеличиваем сумму на значение элемента. По окончании цикла выводим получившуюся сумму.

```
var Ar = [];
var i;
var Sum = 0;
for (i=1; i<6; i++)
   {
        Ar[i] = Math.floor(5 + 10*Math.random());
        document.write(Ar[i], " ");
        Sum +=Ar[i];
   }
document.write ("<br> Сумма элементов равна ", Sum);
```

Рассмотрим еще один пример обработки массива — нахождение максимального элемента. Оформим данный алгоритм в виде функции, чтобы можно было ее использовать для работы с различными массивами.

Объявляем внутри функции локальную переменную `max`, которую будем использовать для хранения максимального значения. Постольку-поскольку диапазон значений элементов нам не известен, предположим, что наибольшее значение имеет нулевой элемент. Далее перебираем все элементы от первого до последнего, сравниваем каждый с переменной `max`. Если у какого-то элемента значение больше, чем то, которое записано в переменной `max`, то это значит, что у нас нашелся новый максимальный элемент. Записываем в переменную `max` значение нового максимального элемента.

Выше было сказано, что элементы перебираются все от первого до последнего. Первый элемент всегда нулевой. А вот как найти индекс последнего элемента? Мы с вами уже знаем, что свойство `length` равно индексу последнего элемента плюс единица. Получается, что если мы вычтем из `length` единицу, то получим нужный нам индекс последнего элемента.

```
var Ar = [10, 28, 3, 4, 5];
var Arr = [3, 6, 2, 15, 8, 11];
function MaxElem (array)
      {
       var max= array[0];
       var i;
        for (i = 1; i <= array.length-1; i++)
          {
           if (array[i] > max) max = array[i];
          }
        return max
      }
alert (MaxElem(Ar));      // Выведет «28»
alert (MaxElem(Arr));          // Выведет «15»
```

А вот для сортировки массива в JavaScript перебор элементов не требуется. Для массива есть встроенный метод `sort`, который позволяет отсортировать массив. Таким образом сортировка массива в программе занимает одну строчку:

```
var Ar = ["яблоко","слива","груша"];
Ar.sort();
alert (Ar);      // Выведет «груша,слива,яблоко»
```

К сожалению это хорошо работает только со строками. При попытке сортировать числа он их будет упорядочивать не по значению, а также как и строки — в алфавитном порядке отдельных символов. Сначала будут сравниваться первые цифры, затем вторые цифры и т. д.

```
var Ar = [4, 20, 23, 51, 100];
Ar.sort();
alert (Ar);      // Выведет «100,20,23,4,51»
```

В принципе метод `sort` может сортировать в различном порядке. Ему надо лишь задать этот порядок. Для определения порядка сортировки описывается функция, результат работы которой передается в качестве аргумента методу `sort`.

Сама функция имеет два аргумента. В качестве аргументов передаются два сравниваемых элемента массива. При сортировке по возрастанию функция должна возвращать 1 в том случае, если первый элемент больше второго, -1 — если первый элемент меньше второго, 0 — если они равны.

```
var Ar = [100, 20, 23, 4, 51];

function SortFunct (a,b)
```

```
  {
    if (a>b) return 1;
    else if (a<b) return -1;
    else return 0
  }
  Ar.sort(SortFunct);
  alert (Ar);      // Выведет «4,20,23,51,100»
```

Элементами массива могут быть другие массивы, тогда они оказываются вложенными друг в друга.

```
  var Ar = [["город", "Тула", "Уфа"], ["река", "Амур", "Дон"]];
```

Выше приведен пример двумерного массива. Двумерный массив можно представить в виде простой таблицы. В таком случае каждый одномерный массив представляет из себя отдельную строку. Для большей наглядности его можно записать иначе.

```
  var Ar = [
    ["город", "Тула", "Уфа"],
    ["река", "Амур", "Дон"]
        ];
```

Первый массив (строка) имеет индекс 0, второй массив (строка) имеет индекс 1. Чтобы обратиться к элементу двумерного массива необходимо указать два индекса: индекс массива и индекс элемента внутри массива.

```
  alert (Ar [0][1]);        //Выведет «Тула»
```

Задачи для самостоятельного решения

- Найти наименьшие элементы в трех одномерных массивах разной длины, инициализированных двузначными числами. Для поиска наименьшего элемента опишите и используйте функцию `MinElem(Ar)`.
- Найти сумму элементов с нечетными номерами в массиве из десяти элементов. Все элементы — двузначные целые числа.
- Подсчитать количество элементов, значение которых равно нулю. Длина массива равна 15. Массив заполнен случайными числами в интервале от 0 до 7.

Часть 3. HTML-формы и JavaScript

Урок 14. Введение в бъектно-ориентированное программирование (ООП).

На этом уроке вы познакомитесь с концепцией объектно-ориентированного программирования, являющегося одним из современных подходов в области программирования; рассмотрите основные понятия, которыми оперируют программисты данного направления.

Оглянувшись вокруг мы увидим, что нас окружают различные объекты. Среди них и другие люди, и домашние животные, и предметы быта, и различные электронные устройства… Список можно продолжать очень долго. Если короче — весь мир состоит из объектов.

> **Объект** — некоторая сущность в пространстве, обладающая определённым состоянием и поведением, имеет заданные значения свойств и операций над ними.

Все объекты можно разделить на материальные и виртуальные. Материальные объекты — это объекты имеющие реальное воплощение. Они состоят из определенной материи. Многие объекты из тех, которые нас окружают, можно потрогать. Виртуальные объекты не имеют реального воплощения, они не состоят из материи и реализуются лишь с помощью технических устройств.

Например монитор компьютера — это материальный объект. Корпус монитора сделан из пластика, его можно потрогать. На экране монитора мы видим множество виртуальных объектов, это различные кнопки, панели, меню… У этих объектов нет реального воплощения, они не существуют вне виртуального пространства компьютера.

В программировании последнее время прочно занял место объектно-ориентированный подход. Согласно этому подходу современные программы состоят из отдельных виртуальных информационных объектов, взаимодействующих друг с другом. Программы стали слишком громоздкими, для оптимизации их разделяют на несколько файлов, различные части программы пишутся разными программистами т. к. одному это уже не под силу.

Несмотря на многообразие и материальных, и виртуальных объектов, среди них есть чем-то схожие. Схожие объекты объединяются в классы. Класс определяет признаки объектов, в него входящих. Возьмем для примера одну из моде-

лей сотовых телефонов. Модель сотового телефона и будет классом. Один из телефонов данной модели — это объект, экземпляр класса. Все телефоны одной модели очень похожи между собой и это не удивительно, ведь они выполнены по одному проекту. Но, несмотря ни на что, каждый телефон уникален. Ни один пользователь не спутает свой телефон с чужим.

В объектно-ориентированном программировании получается, что класс — это некий шаблон, по которому «изготавливаются» объекты, сам объект — это рабочая копия, наделенная признаками класса.

Объектно-ориентированное программирование базируется на трех китах: инкапсуляция, полиморфизм и наследование.

Инкапсуляция есть скрытие внутреннего устройства объекта, при этом объект рассматривается как “черный ящик”. Понятие инкапсуляции неразрывно связано с понятием интерфейса. Интерфейс определяет каким образом один объект будет взаимодействовать с другим объектом.

Возьмем к примеру систему полива огорода. Система полива представляет собой емкость-накопитель с одной стороны, распылитель с другой. В центре этой системы находится объект насос, обеспечивающий подачу воды из емкости в распылитель. Для большинства огородников насос — это «черный ящик». Мало кто разбирал насос, чтобы посмотреть как тот устроен. Очень возможна ситуация, при которой после разбора и «изучения» устройства насоса тот перестанет функционировать. Пользователь должен быть знаком не с его устройством, а с его интерфейсом. Интерфейс насоса представлен впускным и выпускным отверстиями, а также ручкой для ручной накачки воды. К впускному отверстию крепится шланг, соединяющий насос с накопителем, к выпускному — шланг, соединяющий насос с распылителем. Пользователю достаточно правильно подсоединить шланги и тянуть ручку насоса для того, что бы система в целом работала.

В повседневной жизни нас сплошь и рядом окружают «черные ящики». Многие из нас даже приблизительно не знают как устроен телевизор, компьютер, стиральная машина или двигатель внутреннего сгорания. Но это не мешает водить автомобиль и пользоваться различными устройствами. На компьютере мы работаем с необходимыми нам программами. Пользователь не имеет представление о том какой программный код был написан программистами для их создания, но они позаботились о пользовательском интерфейсе этих программ.

Полиморфизм позволяет использовать при объектно-ориентированном подходе различные объекты, с одинаковыми интерфейсами. При этом не требуется

видоизменять систему в целом. Вернемся к примеру с насосом. Ручной насос низкопроизводителен. Давайте поможем огороднику, заменим ручной насос электрическим, а еще установим перед насосом фильтр для очистки воды. Электрический насос как и фильтр также имеют входные и выходные отверстия того же самого диаметра, что и у ручного насоса. Замена одного объекта группой других не повлечет за собой перестройку остальной части системы.

Различные компьютерные программы, имея различный круг решаемых задач, используют одинаковые элементы пользовательского интерфейса. Строка заголовка содержит управляющие кнопки закрытия, сворачивания и разворачивания окна программы. Стандартная панель содержит элементы, позволяющие создавать, сохранять, вырезать, копировать вставлять и выполнять другие стандартные операции с объектами программы.

Наследование опирается на иерархию классов. В ООП существует возможность создания новых классов на основе уже существующих. У дочерних классов появляются новые признаки, но в то же время какие-то признаки наследуются от родителей.

Рассмотрим пример. Предположим, что мы — разработчики сотовых телефонов. Ранее выпущенная нами модель имела название X405. Но прогресс не стоит на месте. И мы шагаем в ногу с прогрессом. Разработка новой модели телефона с нуля — это очень дорогостоящая и долговременная процедура, поэтому мы модифицируем уже имеющуюся модель, назвав ее X409. Во многом новая модель будет напоминать предыдущую, хотя некоторые элементы будут заменены на более современные.

Каждый объект в объектно-ориентированном программировании имеет три характеристики: свойства, события и методы.

Свойства являются атрибутами объектов, определяющими какой он этот объект. Свойства характеризуют внешний вид объекта, его положение или состояние. Какими, например, свойствами обладает сотовый телефон? Свойствами телефона будут его цвет, вес, диагональ экрана, тактовая частота процессора, емкость аккумулятора. А какие свойства у электрического насоса. Возможно какие-то свойства будут совпадать со свойствами сотового телефона: цвет, размер… Хотя значения этих свойств будут различаться. Но для человека, выбирающего насос, более важными будут другие свойства, такие как количество потребляемой электроэнергии и объем воды, перекачиваемый в единицу времени.

У материальных объектов свойства неизменны. Покупая сотовый телефон с диагональю экрана 5 дюймов мы уверены, что телефон не вырастет и экран не

увеличится в размерах. Мы можем лишь учитывать эти свойства. У виртуальных объектов их свойства можно легко менять, поэтому по сути своей они являются переменными.

Событие является внешним воздействием на объект. Объекты реагируют на события во время работы приложения. Программирование в ООП во многом является событийным.

Методы — это действия или задачи, которые выполняет объект. Это то, что может делать объект, или можно делать с объектом.

На какие события должен реагировать электрический насос? Он должен реагировать на нажатие кнопок «Вкл» и «Выкл», а также на прекращение подачи воды в случае, когда вода в резервуаре-накопителе закончилась. Какие методы обработки событий должны применяться в этих случаях? При нажатии кнопки «Вкл» насос должен включаться, при нажатии «Выкл», или когда вода закончилась в накопителе, насос должен выключаться.

Методы и свойства имеют непосредственное отношение к объекту, поэтому они всегда указываются только после имени объекта через точку. Пробелы отсутствуют. Таким образом программа интерпретатор понимает о свойстве или методе какого объекта идет речь. В общем виде это выглядит так:

```
объект.свойство
```

или

```
объект.метод
```

Мы с вами в предыдущем уроке говорили о массивах. Для работы с длиной массива использовали `length`. Наверное вы обратили внимание на то, что `length` писалось через точку после имени массива. Вы не задавались вопросом почему? Да потому, что длина массива — это его свойство, одна из его характеристик.

```
var Ar = [1, 2, 3, 4];
var dl = Ar.length;
```

А еще для изменения размера массива мы использовали `push`, `pop`, `shift` и `unshift`. Они тоже писались через точку после имени массива. Все правильно, это методы данного объекта, т. е. действия, которые с объектом массив можно совершить для добавления или удаления элементов. Через точку после указания имени массива пишется и `sort`, необходимый для сортировки массива, потому, что он также является методом данного объекта.

```
var Ar = ["яблоко","слива","груша"];
Ar.push ("вишня");
Ar.sort();
```

Проговорим еще раз. Свойство не может существовать само по себе, оно является характеристикой какого-либо объекта, неразрывно с ним связано. Поэтому свойство и указывается только вместе с объектом. Запись `Ar.length` с точки зрения русского наверное правильнее читать справа налево, получится «длина (свойство) массива `Ar` (объекта)». То же правило распространяется и на методы.

Объекты в html-документе могут располагаться внутри друг друга, образуя иерархию. Для того, чтобы было понятно, что данный объект является составной частью другого объекта, их также записывают через точку. Такое отношение называют предок-потомок. Сначала указываются родительские объекты, затем дочерние.

На вершине иерархии находится объект `window`. Это окно браузера. Все остальные объекты находятся внутри этого окна. У этого окна есть, например, метод `alert`, позволяющий создавать модальное окно с надписью. Постольку-поскольку `alert` — это метод объекта `window`, то правильнее его было бы записывать следующим образом:

```
window.alert ('надпись');
```

Но мы этого ранее не делали. Дело в том, что у главного объекта `window` свойства и методы разрешается записывать не указывая принадлежности.

Обычно страницы сайта открываются в одном и том же окне. Но в некоторых случаях страница открывается в новой вкладке. Чаще это используется, если ссылка ведет на сторонний сайт. Но иногда и один сайт генерирует несколько окон.

Внутри окна браузера располагается страница html-документа. Это дочерний объект `document`. У страницы есть метод, позволяющий выводить текстовую и числовую информацию — это `write`. Как мы помним, методы выводятся через точку после указания объекта:

```
document.write ('выводимая_информация');
```

Если текст выводится на страницу главного окна, то, как мы уже с вами знаем, принадлежность к главному окну указывать не нужно. Если же вывод необходимо сделать в другое окно, то необходимо точно указать на страницу какого окна будет осуществлять вывод метод `write`. Для этого используем перемен-

ную, которая будет выступать в качестве указателя на окно.

```
Переменная.document.write ('выводимая_информация');
```

Чтобы создать новое окно используется метод `open` объекта `window`. В процессе создания новое окно ассоциируется с переменной, которая будет служить указателем на данное окно браузера.

```
Переменная = window.open (url, name, params);
```

Метод `open` имеет три аргумента, которые записываются в кавычках или апострофах. Если какой-то аргумент отсутствует, то ставятся пустые кавычки (апострофы). Этими аргументами являются:

- url — адрес файла, используется в том случае, если имеется готовый файл для вывода в новое окно;
- name — имя окна, которое может потребоваться для указания иерархии других объектов, расположенных в данном окне;
- params — параметры создаваемого окна, такие как его размер, наличие или отсутствие строки состояния, полос прокрутки и т. д.

Попробуем своими руками сгенерировать окно и вывести в него текстовую информацию.

```
<script>
  var myWin= window.open("", "", "width=400,height=200");
  myWin.document.write("<title> Пробная страница </title>",
                "<h2 >Ура, получилось! </h2>",
                "<hr>",
                "<p> JavaScript помог создать новое окно");
</script>
```

Что происходит в данном примере? Генерируем новое окно размером 400x200 пикселей, ассоциируем данное окно с ранее объявленной переменной `myWin`. Далее используем метод `document.write` для вывода текстовой информации в окно `myWin`. Обратим внимание на то, что текст выводится вместе html-разметкой.

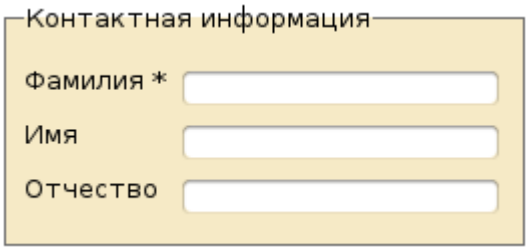
Объекты, рассмотренные на уроке:	
`window`	Окно браузера
`document`	Страница в окне браузера
Методы, рассмотренные на уроке:	
`window.open`	Создание окна

Урок 15. Формы. Кнопки и текстовые поля.

На этом уроке вы научитесь создавать на странице html-документа формы, содержащие поля для ввода текстовой информации, и экранные кнопки, а также обрабатывать их средствами языка JavaScript.

Кроме текстовой и графической информации динамический html-документ может содержать активные элементы, служащие для управления контентом, ввода информации на страницу или обмена данными с сервером. Такие элементы содержат почтовые сервисы для ввода текста писем, поисковые системы для ввода запросов и многие другие сайты. Все активные элементы располагаются в формах.

> **Форма** – это область страницы, являющаяся контейнером для различных элементов управления: кнопок, полей ввода, флажков и меню .



Форма задается парными тегами `<form>` … `</form>`. Если внутри формы элементы нужно разбить на группы, то для группировки элементов используется пара тегов `<fieldset>` … `</fieldset>`. При этом группа будет обведена рамкой. Ей можно назначить цвет фона, цвет рамки и т. д. Если группе необходимо дать название, то название оборачивается парой тегов `<legend>` … `</legend>` и располагается сразу за открывающимся тегом `<fieldset>`.

```
<form>
  <fieldset>
    <legend> Контактная информация </legend>
...
  </fieldset>
</form>
```

В форме активные элементы могут сопровождаться надписями. Надпись задается парным тегом `<label>` … `</label>`. Для того, что бы надписи располагались на разных строках, оформим их как отдельные абзацы.

```
<form>
  <fieldset>
    <legend> Контактная информация </legend>
    <p> <label> Фамилия * </label>
```

```
    <p> <label> Имя </label>
    <p> <label> Отчество </label>
  </fieldset>
</form>
```

Надписи не просто сопровождают активные элементы, они с ними неразрывно связаны. Это позволяет выбирать активные элементы не только щелкая непосредственно на активном элементе, но и на надписи рядом с ним. Чтобы установить связь с элементом в открывающемся теге `<label>` используется атрибут `for`, которому присваивается идентификатор. У активного элемента, связанного с данной надписью указывается аналогичный идентификатор с помощью атрибута id.

```
<form>
  <fieldset>
    <legend> Контактная информация </legend>
    <p> <label for="surname"> Фамилия * </label>
    <p> <label for="name"> Имя </label>
    <p> <label for="patronymic"> Отчество </label>
  </fieldset>
</form>
```

Теперь создадим в форме активные элементы. Для их создания используется тег `<input>`. Данный тег не парный. Внутри данного тега указывается тип создаваемого элемента с помощью атрибута `tipe`. Наша форма содержит поля для ввода текста, они имеют тип `text`. Добавим текстовые поля с помощью тега `<input>`, не забыв указать идентификаторы для связи с надписями.

```
<form>
  <fieldset>
    <legend> Контактная информация </legend>
    <p> <label for="surname"> Фамилия * </label>
       <input type="text" id="surname">
    <p> <label for="name"> Имя </label>
       <input type="text" id="name">
    <p> <label for="patronymic"> Отчество </label>
       <input type="text" id="patronymic">
  </fieldset>
</form>
```

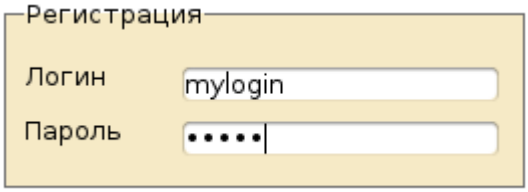
В текстовом поле формы, как и в текстовом поле модального окна, создаваемого функцией `prompt`, может быть значение по умолчанию. Оно будет является подсказкой пользователю о типе вводимой информации. Для создания по

умолчанию используется атрибут `value`, например:

```
<input type="text" id="surname" value="Иванов">
```

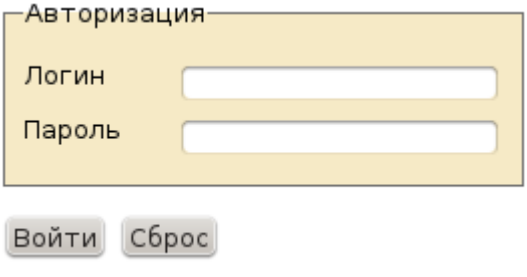
У текстового поля также может быть ограничено количество вводимых символов. Для этого используется атрибут `maxlength`. Ограничение длины применяют например при вводе номера сотового телефона, инвентарных номеров, т. е. в тех случаях, когда длина строки должна быть одинакова.

В некоторых случаях текстовые поля используют не для ввода информации, а для ее вывода. В таких случаях текстовое поле может быть сделано недоступным для редактирования. За это отвечает атрибут `readonly`, присваивающий полю значение «только для чтения».



Кроме текстового поля `text` среди активных элементов присутствует текстовое поле для ввода секретного пароля. Оно имеет тип `password` и отличается от обычного текстового поля тем, что вводимые символы отображаются в виде звездочек или точек. Символы будут зависеть от используемого браузера. Это поле предназначено для того, чтобы другие пользователи не видели вводимую секретную информацию.

```
<form>
  <fieldset>
    <legend> Регистрация </legend>
    <p> <label for="log"> Логин </label>
       <input type="text" id="log">
    <p> <label for="pas"> Пароль </label>
       <input type="password" id="pas">
  </fieldset>
</form>
```



Для того, чтобы произошел процесс авторизации, данная форма должна быть отправлена на сервер. Процесс отправки активируется кнопкой, имеющий тип `submit`. В формах используются также кнопки, возвращающие форму в исходное состояние, имеющие тип `reset`. При нажатии на такую кнопку текстовые поля очищаются или возвращаются значения по умолчанию, определенные атрибутом `value`. Атрибут `value` имеется и у кнопок, он задает текст надписи на кнопке.

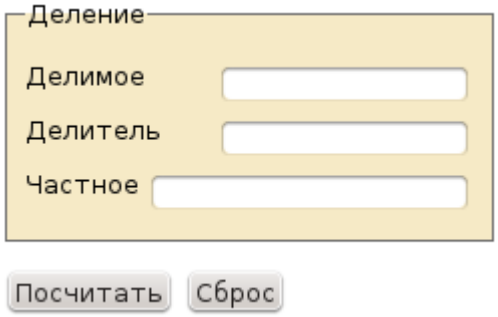
Используя тег `<input>` добавим к нашей форме для авторизации две экранные кнопки. Первая кнопка типа `submit` с надписью «Войти» будет отправлять введенные данные на сервер. Вторая кнопка типа `reset` с надписью «Сброс» будет возвращать форму в исходное состояние в случае неправильно введенных данных. Особо отметим: для того, чтобы эти две кнопки выполняли свои функции правильно, они должны находиться внутри формы.

```
<p><input type="submit" value="Войти">
  <input type="reset" value="Сброс">
```

Конечно же для того, чтобы форма действительно отправилась, должен быть указан адрес куда ее необходимо отправить мощью атрибута `action`. Также необходимо указать некоторые дополнительные параметры отправки. Мы на них не будем сейчас останавливаться. Полный html-код формы может выглядеть так:

```
<form action="server.php" enctype="multipart/form-data"
method="post">
  <fieldset>
    <legend> Регистрация </legend>
    <p> <label for="log"> Логин </label>
       <input type="text" id="log">
    <p> <label for="pas"> Пароль </label>
       <input type="password" id="pas">
  </fieldset>
   <p><input type="submit" value="Войти">
       <input type="reset" value="Сброс">
</form>
```

Не всегда формы обрабатываются на сервере, иногда формы обрабатываются локально браузером. Создадим такую форму, которая будет выполнять деление двух чисел. Используем три текстовых поля: одно для ввода делимого, другое для ввода делителя, третье для вывода частного. Поле для вывода частного сделаем недоступным для записи, применив к нему атрибут `readonly`.



Также нам потребуются две кнопки. Одна из них кнопка «Сброс», которая будет очищать поля формы. Мы с вами уже знаем, что для этих целей используется кнопка типа `reset`. При нажатии на вторую кнопку «Посчитать» должны осуществляться расчеты. Для этих целей не подходит ни кнопка типа `reset`,

ни кнопка типа `submit`. Но в HTML есть еще третий тип кнопок — кнопки типа `button`.

Объекты кнопки реагируют на событие `onclic`, которое происходит при одинарном нажатии на нее левой кнопкой мыши. Методы обработки данного события для кнопок `submit` и `reset` четко определены, одна из них, как мы помним, отправляет форму на сервер, другая возвращает ее в исходное состояние. Для кнопки типа `button`, обработчик события `onclic` не определен. Пользователь сам может задать действия, которые будут происходить при нажатии на токую кнопку. Это как раз то, что нам нужно.

```
<form name="del">
  <fieldset>
    <legend>Деление</legend>
    <p> <label for="del_1"> Делимое </label>
       <input name="Delim" type="text" id="del_1">
    <p> <label for="del_2"> Делитель </label>
       <input name="Delit" type="text" id="del_2">
    <p> <label for="ch"> Частное </label>
       <input name="Chast" type="text" id="ch" readonly>
  </fieldset>
  <p> <input type="button"
           value="Посчитать" onclick="Divisin()">
     <input type="reset" value="Сброс">
</form>
```

Как мы видим, у кнопки «Посчитать» появился обработчик события `onclick`. При нажатии на эту кнопку будет вызываться функция `Divisin()`, совершающая расчеты, но она у нас пока не написана. Давайте подумаем что должна делать данная функция. Она должна забирать делимое из первого текстового поля, забирать делитель из второго текстового поля, проверяя, не является ли данное число нулем, и выводить результат деления в третье текстовое поле.

Встает вопрос, как программе обратиться к тому или иному текстовому полю? Обратиться к любому объекту формы можно по имени. Для этого у всех элементов есть атрибут `name`. Дадим имена необходимым нам объектам. Эти имена уже присутствуют в представленном html-коде. У формы `name="MyForm"`, у первого текстового поля `name="Delim"`, у второго текстового поля `name="Delit"`, у третьего текстового поля `name="Chast"`. За текст у текстового поля отвечает свойство `value`. Мы помним, что свойства объектов являются переменными, а значит мы можем как считывать их значе-

ния, так и присваивать новые. В общем виде обращение с свойству записывается следующим образом.

```
Имя_формы.Имя_объекта.Свойство_объекта
```

Обращение к свойству «текст» первого текстового поля `Delim`, расположенного в форме `MyForm`:

```
MyForm.Delim.value
```

Теперь настало время написать функцию `Division()`, вызываемую при нажатии на кнопку «Посчитать». Функция проверяет, не находится ли во втором текстовом поле значение 0. Если делитель не 0, то делит два числа и выводит результат в третье текстовое поле, в противном случае выводит «Деление невозможно».

```
  <script>
  function Divisin ()
  {
    if (MyForm.Delit.value != 0)
         MyForm.Chast.value     =     MyForm.Delim.value     /
MyForm.Delit.value;
    else MyForm.Chast.value = "Деление невозможно";
  }
  </script>
```

Теги, рассмотренные на уроке:	
`<form> … </form>`	Форма
`<fieldset> … </fieldset>`	Группировка элементов формы
`<legend> … </legend>`	Надпись в группе элементов
`<input>`	Создание активного элемента
`<label> … </label>`	Надпись
Объекты, рассмотренные на уроке:	
`text`	Поле для ввода текста
`password`	Поле для ввода пароля
`reset`	Кнопка очистки формы
`submit`	Кнопка отправки формы на сервер
`button`	Обычная кнопка
Свойства объектов, рассмотренные на уроке:	
`name`	Имя объекта
`value`	Текст

Урок 16. Формы. Переключатели и списки.

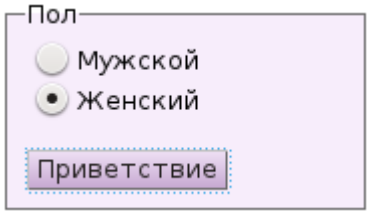
На этом уроке вы научитесь создавать на странице html-документа формы, содержащие радиокнопки, флажки и раскрывающиеся списки., а также обрабатывать их средствами языка JavaScript

Сегодня мы поговорим о других активных элементах формы, таких как переключатели и списки. Среди переключателей различают переключатели с зависимой и независимой фиксацией. К переключателям с зависимой фиксацией относятся радиокнопки.

Свое название они получили в честь кнопок на старых радиоприемниках. Такие радиоприемники имели несколько круглых кнопок, расположенных в ряд. При нажатии на одну из них, остальные отщелкивались. Кнопок могло быть несколько групп, в каждой группе могла быть нажата только одна кнопка.

Радиокнопки очень напоминают своих предшественниц. Они круглой формы, в группе активной может быть только одна радиокнопка. Чтобы группа радиокнопок была зависимой, всем переключателям в группе дается одинаковое имя. В таком случае они рассматриваются как массив, и обращение к каждой из них происходит по индексам.

В массиве радиокнопок одна всегда должна быть активной. За активность кнопок отвечает свойство `checked`.



Создадим форму, содержащую радиокнопки, которые позволяют выбирать пол. При нажатии на кнопку «Приветствие» появляется всплывающее окно с приветствием, соответствующее выбранному полу. Обратим внимание на то, что в данной форме в строке сначала располагаются активные элементы, а затем уже надписи, их сопровождающие. Также как и другие активные элементы, радиокнопки в форме создаются тегом `<input>` , при этом указывается тип `radio`.

```
<form>
  <fieldset>
    <legend> Пол </legend>
    <input name="Pol" type="radio" id="r_1" checked>
       <label for="r_1"> Мужской </label> <br>
   <input name="Pol" type="radio" id="r_2">
       <label for="r_2"> Женский</label> <br> <br>
   <input type="button" value="Приветствие">
```

```
</fieldset>
</form>
```

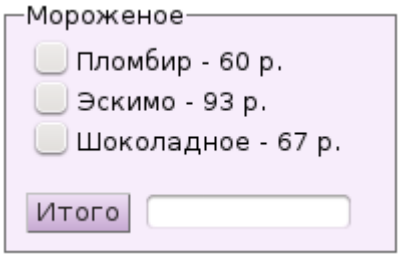
Мы видим, что обе радиокнопки названы одинаково `name="Pol"` и первая радиокнопка по умолчанию активна, т. к. имеет атрибут `checked`. Имеется экранная кнопка `button`, для которой отсутствует пока обработчик события `onclick`, т. е. при нажатии на кнопку «Приветствие» пока ничего не происходит. Добавим метод обработки данного события.

Обработчик не займет много места, поэтому расположим его непосредственно в форме и не будем использовать тег `<script>`. Итак, если активна первая радиокнопка (а на самом деле нулевая, т. к. немерация в массиве начинается с 0), должно появляться всплывающее окно с надписью «Доброе утро, сэр!», в противном случае «Доброе утро, мадам!». Ниже представлен код кнопки с обработчиком события `onclick`.

```
<input type="button" value="Приветствие"
onclick="if (this.form.Pol[0].checked)
       alert('Доброе утро, сэр!');
       else alert('Доброе утро, мадам!')">
```

В данном примере использован еще один способ обращение к форме — `this.form`, что означает «текущая форма», при этом имя форме можно и не давать. В таком случае запись `if (this.form.Pol[0].checked)` следует читать как «если элемент `Pol[0]`, расположенный в текущей форме, активен…».

Кроме радиокнопок к переключателям относятся также и флажки. Флажок представляет собой квадратик, внутри которого можно поставить галочку. Флажки позволяют выбрать несколько пунктов одновременно, т. е. являются переключателями с независимой фиксацией, при этом они имеют различные имена. Чтобы создать флажок, необходимо использовать тег `<input>`, указав тип `checkbox`.



Создадим форму подсчитывающую суммарную стоимость мороженого. Форма будет содержать три флажка типа `checkbox` для выбора трех видов мороженого, кнопка, при нажатии которой происходит подсчет, и текстовое поле для вывода ответа.

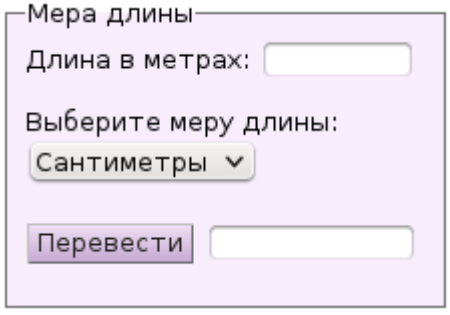
```
<form name="IceCream">
```

```
    <fieldset>
      <legend> Мороженое </legend>
      <input name="Pl" type="checkbox" id="ch1">
         <label for="ch1"> Пломбир - 60 p.</label> <br>
     <input name="Esc" type="checkbox" id="ch2">
         <label for="ch2"> Эскимо - 93 p.</label> <br>
     <input name="Shok" type="checkbox" id="ch3">
         <label for="ch3"> Шоколадное - 67 p.</label> <br>
<br>
     <input type="button" value="Итого" onclick="Sum()">
     <input name="Itog" type="text" size="12">
   </fieldset>
  </form>
```

При нажатии на кнопку «Итого» возникает событие `onclick`. Для обработки этого события вызывается функция `Sum()`. Понятно, что данная функция должна подсчитывать общую сумму, но как это сделать? Здесь могут быть разные подходы к решению. Один из подходов следующий. Описываем локальную переменную `s`, в которой будет накапливаться сумма. Помним, что до начала подсчетов сумма равна нулю. Далее проверяем каждый из флажков. Если флажок установлен, то увеличиваем сумму на стоимость соответствующего мороженного. Конечную сумму выводим вместе со значком «р.» в текстовое поле.

```
  <script>
  function Sum ()
  {
    var s=0;
    if (IceCream.Pl.checked) s += 60;
    if (IceCream.Esc.checked) s += 93;
    if (IceCream.Shok.checked) s += 67;
    IceCream.Itog.value = s + " p."
  }
  </script>
```

Для выбора в формах используются не только переключатели, но и раскрывающиеся списки. Для создания такого списка в форме применяется пара тегов `<select>` … `</select>`. Внутри этой пары располагаются парные теги `<option>` … `</option>`, которые формируют элементы списка. Каждый элемент списка имеет атрибут `value`, значение которого



не обязательно должно совпадать с текстом элемента списка.

В качестве примера создадим форму, благодаря которой можно переводить длину, указанную в метрах, в другие единицы длины (сантиметры, миллиметры и дюймы). Форма будет содержать два текстовых поля, раскрывающийся список для выбора единиц измерения, и кнопка, при нажатии на которую будет осуществляться перевод. Html-код списка будет выглядеть следующим образом:

```
  <select>
    <option value="1"> Сантиметры </option>
    <option value="2"> Миллиметры </option>
    <option value="3"> Дюймы </option>
  </select>
```

Дадим списку имя `List`. Ниже представлен полный html-код, формирующий форму на web-странице:

```
  <form name="myForm">
    <fieldset>
      <legend> Мера длины </legend>
      <label for="ch1"> Длина в метрах:   </label>
         <input  name="Met"  type="text"  id="ch1"  size="8">
<br>
      <p><label for="ch2"> Выберите меру длины:</label><br>
      <select name="List" id="ch2">
         <option value="1">Сантиметры</option>
         <option value="2">Миллиметры</option>
         <option value="3">Дюймы</option>
    </select>
    <br> <br>
    <input type="button" value="Перевести" onclick="Dl()">
    <input name="Itog" type="text" size="12">
    </fieldset>
  </form>
```

Как будет выглядеть функция `Dl()`, которая вызывается при нажатии на кнопку «Перевести»? Т.к. список позволяет выбирать единицы измерения, в функции имеет смысл также использовать оператор выбора, который будет проверять свойство `value` у выбранного элемента списка.

```
  <script>
  function Dl ()
  {
```

```
    switch(myForm.List.value)
    {
      case "1":
        myForm.Itog.value = myForm.Met.value *100 +" см";
        break;
      case "2":
        myForm.Itog.value = myForm.Met.value *1000 +" мм";
        break;
      case "3":
        myForm.Itog.value = myForm.Met.value /0.0254 +" дм";
        break;
    }
  }
  </script>
```

Теги, рассмотренные на уроке:	
`<select> … </select>`	Раскрывающийся список
`<option> … </option>`	Элемент раскрывающегося списка
Объекты, рассмотренные на уроке:	
`radio`	Радиокнопка
`checkbox`	Флажок
Свойства объектов, рассмотренные на уроке:	
`checked`	Активный элемент

Практикум

Практическая работа 1. Текстовая web-страница. Форматирование текста

Представить текст в рамке в виде web-страницы, сохранив его форматирование.

Мертвые души

Поэма

Том первый

К читателю от сочинителя

Кто бы ты ни был, мой читатель, на каком бы месте ни стоял, в каком бы звании ни находился, почтен ли ты *высшим чином* или человек простого сословия, но если тебя **вразумил Бог грамоте** и попалась уже тебе в руки моя книга, я прошу тебя помочь мне.

В книге, которая перед тобой, которую, вероятно, ты уже прочел в ее первом издании, изображен **человек**, взятый из нашего же государства. Ездит он по нашей ***Русской земле***, встречается с людьми всяких сословий, от благородных до простых. Взят он больше затем, чтобы показать *недостатки и пороки русского человека*, а не его достоинства и добродетели, и все люди, которые окружают его, взяты также затем, чтобы показать н*аши слабости и недостатки*; лучшие люди и характеры будут в других частях. В книге этой многое **описано неверно**, не так, как есть и как действительно происходит в Русской земле, потому что я не мог узнать всего: мало жизни человека на то, чтобы узнать одному и сотую часть того, что делается в нашей земле. Притом от моей собственной оплошности, незрелости и поспешности произошло множество всяких ошибок и промахов, так что на всякой странице есть что поправить: ***я прошу тебя, читатель, поправить меня***. Не пренебреги таким делом.

Какого бы ни был ты сам **высокого образования** и **жизни высокой**, и какою бы ничтожною ни показалась в глазах твоих моя книга, и каким бы ни показалось тебе мелким делом ее исправлять и писать на нее замечания, — я прошу тебя это сделать.

А ты, читатель **невысокого образования** и **простого звания**, не считай себя таким невежею, чтобы ты не мог меня чему-нибудь поучить. Всякий человек, кто жил и видел свет и встречался с людьми, заметил что-нибудь такое, чего другой не заметил, и узнал что-нибудь такое, чего другие не знают. А потому не лиши меня твоих замечаний: не может быть, чтобы ты не нашелся чего-нибудь сказать на какое-нибудь место во всей книге, если только внимательно прочтешь ее.

Как бы, например, хорошо было, если бы хотя один из тех, которые **богаты опытом** и **познанием жизни** и знают круг тех людей, которые мною описаны, сделал свои заметки сплошь на всю книгу, не пропуская ни одного листа ее, и принялся бы читать ее не иначе, как взявши в руки перо и положивши перед собою лист почтовой бумаги, и после прочтенья нескольких страниц припомнил бы себе всю жизнь свою и всех людей, с которыми встре-

чался, и все происшествия, случившиеся перед его глазами, и все, что видел сам или что слышал от других подобного тому, что изображено в моей книге, или же противоположного тому, все бы это описал в таком точно виде, в каком оно предстало его памяти, и посылал бы ко мне всякий лист по мере того, как он испишется, покуда таким образом не прочтется им вся книга. Какую бы кровную он оказал мне услугу!

О слоге или красоте выражений здесь нечего заботиться; дело в *деле* и в *правде* дела, а не в слоге. Нечего ему также передо мною чиниться, если бы захотелось меня попрекнуть, или побранить, или указать мне вред, какой я произвел наместо пользы необдуманным и неверным изображением чего бы то ни было. За все буду ему благодарен.

Николай Гоголь

Практическая работа 2. Списки в html-документе

Представить текст в рамке в виде web-страницы, сохранив его форматирование.

Классификация ПО

I. Системное программное обеспечение:
   1. Базовое
      a. Операционные системы
         - системы пакетной обработки;
         - системы разделения времени;
         - системы реального времени;

      b. Операционные оболочки
   2. Сервисное
      a. Драйверы
      b. Утилиты:
         - файловые менеджеры;
         - архиваторы;
         - средства обеспечения компьютерной безопасности;
         - программы обслуживания дисков;

II. Прикладное программное обеспечение:
   1. Программы общего назначения:
      a. Офисные программы:
         - текстовые редакторы
         - табличные процессоры
         - мастер презентаций

      b. Графические редакторы
         - векторные
         - растровые

      c. Системы управления базами данных
      d. Мультимедийные программы
      e. Коммуникационные (сетевые) программы
      f. Компьютерные игры
   2. Программы специального назначения:
      a. Бухгалтерские пакеты
      b. Системы автоматизированного проектирования
      c. экспертные системы
      d. Программы для проведения сложных математических расчетов

III. Инструментальное программное обеспечение:
   1. Компиляторы,
   2. Редакторы связей,
   3. Отладчики, интегрированные системы разработки ПО

Практическая работа 3. Гиперссылки

Создайте игру в загадки в виде многофайлового HTML-документа. Основной файл должен содержать загадки, меню с переходами внутри документа к загадкам, а также указатели гиперссылок на разгадки (ответы). Ответы находятся во вспомогательных файлах, которые также имеют указатели гиперссылок, благодаря которым можно вернуться к основному файлу.

Загадки

1. Свечи
2. Солнечная погода
3. Имена
4. Окружность
5. Плоскость
6. Скорость
7. Спутники
8. Курицы

Свечи

В комнате горело 50 свечей, 20 из них задули. Сколько останется?

Ответ: *Останется 20, а 30 сгорят полностью.*

Солнечная погода

Если в 12 часов ночи идет дождь, то можно ли ожидать, что через 72 часа будет солнечная погода?

Ответ: *Нет, - через 72 часа снова будет полночь.*

Имена

Как известно, все исконно русские женские имена оканчиваются либо на "а", либо на "я": Анна, Мария, Ольга и т.д. Однако есть одно-единственное женское имя, которое не оканчивается ни на "а", ни на "я". Назовите его.

Ответ: *Любовь*

Окружность

На столе лежат линейка, карандаш, циркуль и резинка. На листе бумаги нужно начертить окружность. С чего начать?

Ответ: *Для начала надо найти лист бумаги.*

Плоскость

Из гнезда вылетели три ласточки. Какова вероятность того, что через 15 секунд

они будут находиться в одной плоскости?

Ответ: *Три точки всегда образуют одну плоскость.*

Скорость

С какой скоростью должна бежать собака, чтобы не слышать звона сковородки, привязанной к ее хвосту?

Ответ: *Собаке достаточно стоять на месте.*

Спутники

Один оборот вокруг Земли спутник делает за 1 ч 40 минут, а другой - за 100 минут. Как это может быть?

Ответ: *1 ч 40 мин = 100мин*

Курицы

Если полторы курицы несут полтора яйца за полтора дня, сколько две курицы снесут за два дня?

Ответ: *4 яйца*

Практическая работа 4. Таблицы

Используя таблицы, создать по образцу html-документ, содержащий календарь на текущий месяц. При создании таблицы обратить внимание на выравнивание чисел и текста в ячейках. Столбцы, содержащие дни недели должны быть одинакового размера. Для задания цвета границ ячеек использовать свойство border-color.

Календарь на месяц

ноябрь							
неделя	Понедельник	Вторник	Среда	Четверг	Пятница	Суббота	Воскресенье
1							1
2	2	3	4	5	6	7	8
3	9	10	11	12	13	14	15
4	16	17	18	19	20	21	22
5	23	24	25	26	27	28	29
6	30						

Практическая работа 5. Графические изображения

Представить текст в рамке в виде html-документа, используя табличную верстку. Для этого в документе вставить таблицу без границ, отцентрировав ее по горизонтали. Размер таблицы должен составлять 60-70% от ширины страницы. Для цвета фона страницы использовать графическое изображение, цвет фона таблицы — белый.

В каждом абзаце текста вставить графические изображения, расположив их в шахматном порядке по левому и правому краю от текста (смотри образец). Вставить гипертекстовое меню, обеспечивающее переходы внутри документа. Меню оформить в виде маркированного списка. Маркеры списка — графические элементы.

Устройство компьютера

Системный блок, основная часть компьютера, где происходят все вычислительные процессы. Системный блок достаточно сложен и состоит из различных компонентов. Эти компоненты мы рассмотрим позже.

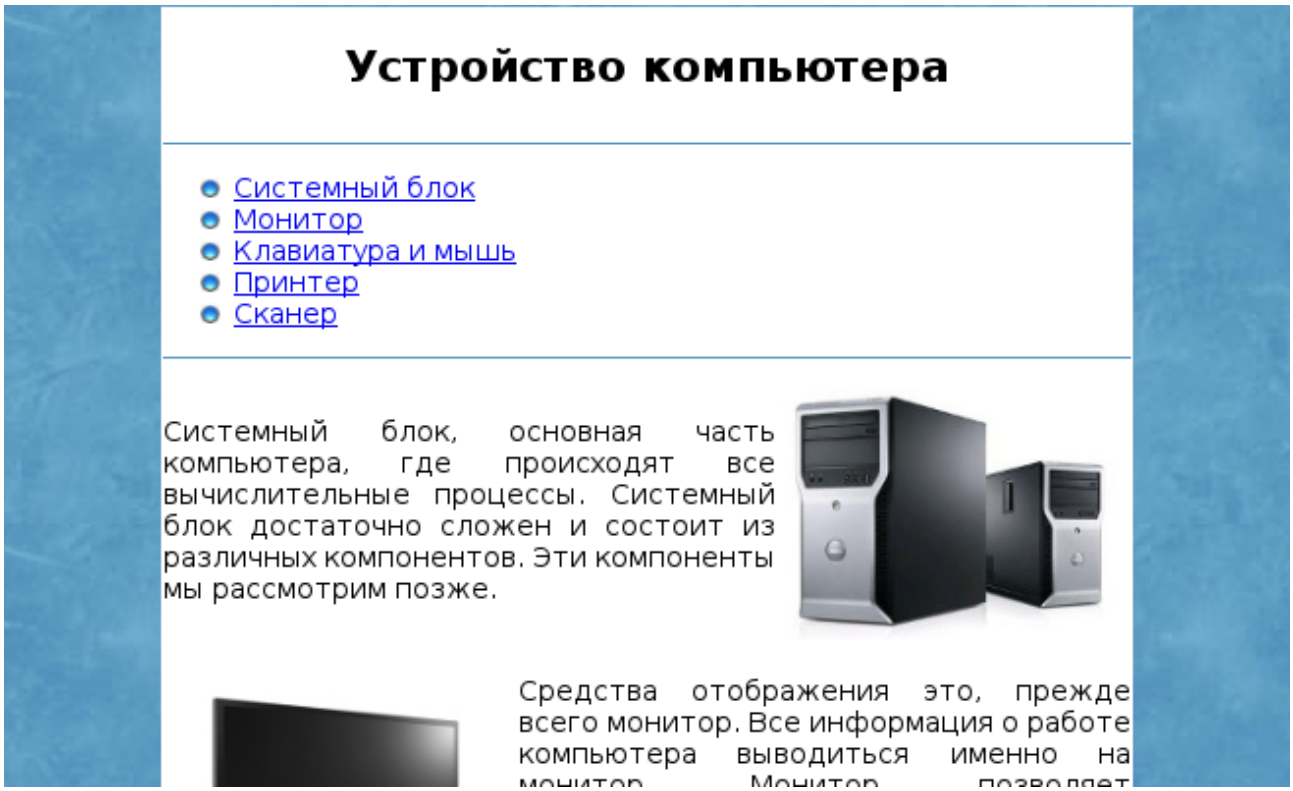
Средства отображения это, прежде всего монитор. Все информация о работе компьютера выводиться именно на монитор. Монитор позволяет отслеживать, что происходит в компьютере в данное время, каким вычислительным процессом занят компьютер.

Клавиатура и мышь, вот основные средства манипулирования, управления компьютером. Также к средствам манипулирования можно отнести различные джойстики, рули с педалями, штурвалы, но они предназначены в основном для управления игровым процессом. Здесь можно отметить, что не все выпускаемые игры могут корректно использовать или вообще использовать тот или иной игровой манипулятор.

Лазерные принтеры бывают цветными и черно-белыми. Лазерные принтеры печатают с помощью лазерного луча. Лазерный луч запекает на бумаге тонер, который попадает из картриджа на бумагу. Лазерные принтеры отличаются скоростью печати, числом печати листов в минуту. Как правило, лазерные принтеры стоят в офисах, т.к. имеют высокую скорость печати и не дорогой по себестоимости отпечатанный лист. Как и струйные принтеры, лазерные принтеры имею картриджи. Эти картриджи заправлены тонером (порошком).

Сканер – устройство для сканирования документов, фотографий и даже фото-негативов. Самый распространенный вид сканеров – планшетный. Разные сканеры имеют различную скорость сканирования. Также сканеры можно поделить по тому расширению, которое они поддерживают при сканировании. В некоторые сканеры устанавливается специальное устройство для сканирования негативов. Сканер обычно подключен к компьютеру через порт USB.

Образец:


Устройство компьютера
Системный блок
Монитор
Клавиатура и мышь
Принтер
Сканер
Системный блок, основная часть компьютера, где происходят все вычислительные процессы. Системный блок достаточно сложен и состоит из различных компонентов. Эти компоненты мы рассмотрим позже.
Средства отображения это, прежде всего монитор. Все информация о работе компьютера выводиться именно на

Практическая работа 6. Верстка web-страниц

Представить текст в таблице в виде html-документа, используя нетабличную верстку. Средствами CSS сформировать на странице «шапку» и «подвал», остальной текст разместить в две колонки.

В таблице стилей создать стили различных частей страницы, а также стили заголовков и абзацев. Для обозначения разных типов абзацев, расположенных в левой и правой колонках, использовать классы.

Важные рецепты	
Кровавая Мери Влить в бокал хорошую 100% кровь по лезвию ножа. Влить водку «Смирновскую». Пить залпом. **Царская водка** Смешать 2 части соляной кислоты и 1 часть азотной со льдом. Слить охлажденную смесь в фужер. Пить залпом. **Ворона де ля фурше** Необходимо наловить десяток ворон и пропустить через мясорубку. Полученный фарш можно пустить на котлеты. Приправляется просеянным сквозь мелкую сетку табачным пеплом и (по желанию) болтами на десять. **Кулебяка** На 100 г. сельди берется: 200 г. свежих огурцов, 200 г. недозрелых яблок, 50 г. лука и 20 г. горчицы. Все тщательно перемешивается. Употреблять, запивая холодным молоком... Через полчаса кулебяка готова.	КОНЧАЙ УЧИТЬСЯ! Сингапурские тридцатиклассники вышли на демонстрацию. Они требуют сокращения обязательного среднего образования. Через два года абсолютно все школы города Аничкино подключат к Интернету. С чем связано нынешнее отключение школ на два года от Интернета, власти города не уточнили.

Образец:

Важные рецепты

Кровавая Мери

Влить в бокал хорошую 100% кровь по лезвию ножа. Влить водку «Смирновскую». Пить залпом.

Царская водка

Смешать 2 части соляной кислоты и 1 часть азотной со льдом. Слить охлажденную смесь в фужер. Пить залпом.

Ворона де ля фурше

Необходимо наловить десяток ворон и пропустить через мясорубку. Полученный фарш можно пустить на котлеты. Приправляется просеянным сквозь мелкую сетку табачным пеплом и (по желанию) болтами на десять.

Кулебяка

На 100 г. сельди берется: 200 г. свежих огурцов, 200 г. недозрелых яблок, 50 г. лука и 20 г. горчицы. Все тщательно перемешивается. Употреблять, запивая холодным молоком... Через полчаса кулебяка готова.

КОНЧАЙ УЧИТЬСЯ! Сингапурские тридцатиклассники вышли на демонстрацию. Они требуют сокращения обязательного среднего образования.

Через два года абсолютно все школы города Аничкино подключат к Интернету. С чем связано нынешнее отключение школ на два года от Интернета, власти города не уточнили.

© Автор страницы

Практическая работа 7. Линейные программы и ветвление.

Вариант 1.

1. Написать скриптовую программу, запрашивающую у пользователя диаметр окружности d и выводящую ее длину. Для расчета длины окружности использовать формулу $L=\pi d$. Значение π взять как 3,14.
2. Написать скриптовую программу, запрашивающую у пользователя длины ребер a, b и c прямоугольного параллелепипеда. Вывести его объем и площадь поверхности. Для расчета объема использовать формулу $V=a\cdot b\cdot c$, для расчета площади поверхности $S=2(a\cdot b+b\cdot c+a\cdot c)$.
3. Написать скриптовую программу, запрашивающую у пользователя два числа. Если их произведение отрицательно, умножить его на -2 и вывести на экран, в противном случае увеличить его в 1,5 раза и вывести на экран.
4. Написать скриптовую программу, запрашивающую у пользователя рост трех человек и выводящую, есть ли среди них два человека одинакового роста.
5. Написать скриптовую программу, запрашивающую у пользователя три натуральных числа и выводящую количество чисел, меньших 7.

Вариант 2.

1. Написать скриптовую программу, запрашивающую у пользователя сторону равностороннего треугольника a и выводящую его периметр. Для расчета периметра равностороннего треугольника использовать формулу $P=3a$.
2. Написать скриптовую программу, запрашивающую у пользователя стороны прямоугольника a и b. Вывести периметр прямоугольника, используя для расчета формулу $P=2(a+b)$, и длину его диагонали. Для расчета диагонали воспользуемся теоремой Пифагора $c=\sqrt{a^2+b^2}$. Для нахождения квадратного корня необходимо использовать `Math.sqrt(подкоренное_выражение).`
3. Написать скриптовую программу, запрашивающую у пользователя число и выводящую новое число по следующему правилу: если введенное число неотрицательно, то новое число получается прибавлением к введенному 20, в противном случае — вычитанием из введенного 30.
4. Написать скриптовую программу, запрашивающую у пользователя целое число и выводящую слово «Принадлежит» в том случае, если число принадлежит интервалу (-3, 8), и «Не принадлежит» в противном случае.
5. Написать скриптовую программу, запрашивающую у пользователя три числа и

выводящую сумму тех из них, которые являются отрицательными.

Вариант 3.

1. Написать скриптовую программу, запрашивающую у пользователя боковую сторону равнобедренной трапеции c и два основания a и b. Вывести периметр равнобедренной трапеции, используя для расчета формулу $P = a + b + 2 \cdot c$.
2. Написать скриптовую программу, запрашивающую у пользователя два ненулевых числа. Вывести сумму, разность, произведение и частное этих чисел.
3. Написать скриптовую программу, запрашивающую у пользователя число. Если оно четное, разделить его на 4, если нечетное — умножить на 5. Вывести полученный результат на экран.
4. Написать скриптовую программу, запрашивающую у пользователя три стороны треугольника a, b и c и выводящую, является ли данный треугольник равносторонним.
5. Написать скриптовую программу, запрашивающую у пользователя три числа и выводящую количество чисел принадлежавших интервалу (1, 5).

Практическая работа 8. Циклы.

Вариант 1.

1. Написать скриптовую программу, выводящую на экран таблицу переводов расстояний от 1 до 25 дюймов в сантиметры в формате:
   1" = 2.54 см
   2" = 5.08 см
   ...
2. Написать скриптовую программу, запрашивающую у пользователя целое число a (a<12) и выводящую произведение всех чисел в интервале от a до 12.
3. Написать скриптовую программу, запрашивающую у пользователя число n и выводящую на экран последовательность натуральных чисел от 1 до некоторого числа. Квадраты чисел не должны превышать число n.
4. Написать скриптовую программу, выводящую на экран последовательность натуральных чисел, сгенерированных случайным образом в интервале от 1 до 10, а также сумму чисел данной последовательности. Последовательность заканчивается, если выпадает число 4 или 7.

Вариант 2.

1. Написать скриптовую программу, выводящую на экран таблицу переводов расстояний от 1 до 22 футов в метры в формате:
   1` = 0.3048 м
   2` = 0.6096 м
   …
2. Написать скриптовую программу, запрашивающую у пользователя натуральное число b (10<b<20) и выводящую среднее арифметическое всех чисел в интервале от 1 до b.
3. Написать скриптовую программу, выводящую на экран последовательность чисел от 1 до некоторого числа. Кубы чисел должны содержать не более двух знаков (не превышают 99).
4. Написать скриптовую программу, выводящую на экран последовательность натуральных чисел, сгенерированных случайным образом в интервале от 5 до 25, а также произведение чисел данной последовательности. Последовательность заканчивается, если выпадает число 12 или число, кратное 5.

Вариант 3.

1. Написать скриптовую программу, выводящую на экран таблицу переводов ве-

сов от 1 до 25 фунтов в килограммы в формате:

1 lbs = 0.4535 кг

2 lbs = 0.907 кг

…

2. Написать скриптовую программу, запрашивающую у пользователя два натуральных числа `a` и `b` и выводящую сумму квадратов чисел в интервале от `a` до `b`, а также количество этих чисел.

3. Написать скриптовую программу, запрашивающую у пользователя число `n` и выводящую на экран последовательность натуральных чисел от 1 до некоторого числа. Для каждого выводимого числа значение выражения «удвоенное число - 7» не должны превышать число `n`.

4. Написать скриптовую программу, выводящую на экран последовательность натуральных чисел, сгенерированных случайным образом в интервале от -10 до 10, а также сумму чисел данной последовательности. Последовательность заканчивается, если выпадает число 0 или сумма становится больше 14.

Практическая работа 9. Массивы.

Вариант 1.

1. Написать скриптовую программу, выводящую количество положительных элементов массива.
2. Написать скриптовую программу, задающую пустой массив. Вписать в массив 10 элементов, значение которых рассчитываются по формуле $2i-3$, где i – индекс элемента. Вывести данный массив на экран.
3. Написать скриптовую программу, заполняющую массив случайными числами в диапазоне от 1 до 5. Умножить все элементы массива на последний элемент. Вывести первоначальный и измененный массивы на экран.
4. Написать скриптовую программу, выводящую сумму элементов массива, индексы которых четны.

Вариант 2.

1. Написать скриптовую программу, выводящую количество отрицательных элементов массива.
2. Написать скриптовую программу, задающую пустой массив. Вписать в массив 9 элементов, значение которых рассчитываются по формуле i^2-2i , где i – индекс элемента. Вывести данный массив на экран.
3. Написать скриптовую программу, заполняющую массив случайными числами в диапазоне от 3 до 8. Вычесть из всех элементов массива первый элемент. Вывести первоначальный и измененный массивы на экран.
4. Написать скриптовую программу, выводящую произведение элементов массива, индексы которых кратны 3.

Вариант 3.

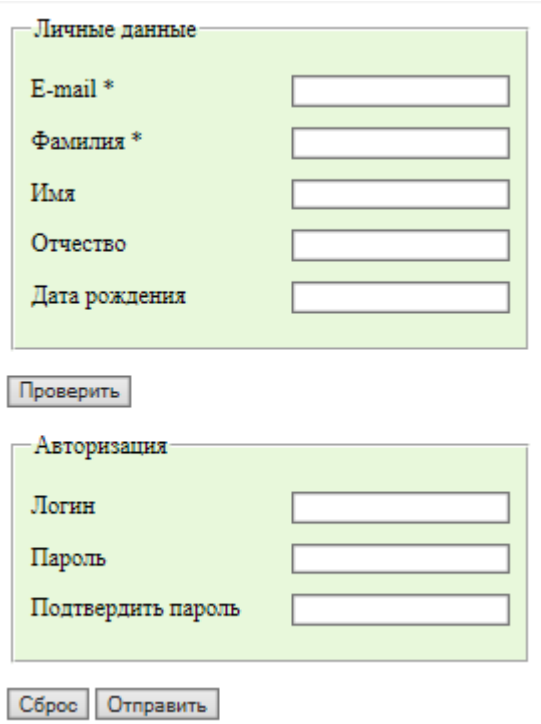
1. Написать скриптовую программу, выводящую количество четных элементов массива.
2. Написать скриптовую программу задающую пустой массив. Вписать в массив 12 элементов, значение которых рассчитываются по формуле $17-4i$, где i – индекс элемента. Вывести данный массив на экран.
3. Написать скриптовую программу, заполняющую массив случайными числами в диапазоне от 2 до 10. Прибавить ко всем элементам массива третий элемент. Вывести первоначальный и измененный массивы на экран.
4. Написать скриптовую программу, выводящую разность элементов массива, индексы которых кратны 4.

Практическая работа 10. Формы. Кнопки и текстовые поля

Создайте HTML-документ, содержащий регистрационную форму. Форма делится на две части: «Личные данные» и «Авторизация». После первой части располагается кнопка «Проверить», при нажатии на которую происходит проверка заполнения обязательных полей, помеченных звездочкой. В случае, если какое-то из обязательных полей не заполнено, должно появляться всплывающее окно с соответствующим напоминанием.

Во второй части формы в полях для ввода пароля и подтверждения пароля не должны быть показаны явно, а должны заменяться специальными символами (звездочками, точками…). Кнопка «Сброс» должна очищать форму, а кнопка «Отправить» должна отправлять форму на адрес вашей электронной почты.

Образец:



Практическая работа 11. Формы

Создайте интерактивный тест в виде HTML-документа. Тест начинается кнопкой «Начать», которая очищает форму, а также переменные, используемые для ее обработки.

Первый вопрос теста на установление последовательности тегов, которая осуществляется с помощью раскрывающихся списков. Второй и третий вопросы подразумевают множественный выбор нескольких вариантов ответов, здесь используются флажки. Четвертый вопрос подразумевает один вариант ответа, здесь использованы радиокнопки.

После каждого вопроса есть кнопка проверить, при нажатии на которую осуществляется проверка вариантов ответа и выводится в соседнее текстовое поле надпись «Верно» или «Неверно». Данные текстовые поля защищены от записи, информация в них выводится только программно.

Тест заканчивается кнопкой «Конечный результат». При нажатии на эту кнопку появляется модальное окно с надписью «Отличный результат» в том случае, если на все четыре вопроса пользователь ответил верно; «Хороший результат» в том случае, ели пользователь ответил верно на три вопроса; «Удовлетворительный результат» в том случае, ели пользователь ответил верно на два вопроса; «Плохой результат» в остальных случаях.

Образец: