**Требования по дисциплине «web-технологии». Весна 2022**

**Начисление баллов по дисциплине «web-технологии».**

8 лр \* 10 баллов = 80 баллов

**Правила начисления баллов за каждую работу:**

3 балла – оценка за качество работы (3 – выполнение всех пунктов задания и добавление своего материала, 2 – выполнение всех пунктов по заданию, 1 – отсутствие части заданий, нарушение условий варианта);

3 балла – оценка за качество ответа;

3 балла – оценка за качество отчета (наличие всех пунктов, оформление, наличие пояснений).

1 балл – общее впечатление.

Основная оценка – 8 баллов

**Условие получения «автомата» на экзамене:**

1..Выполнение всех лабораторных работ до экзамена

2.. наличие баллов: больше 70 – оценка 4, больше 60 – оценка 3.

**Урок 1. Введение в Web-технологии**

**Основные понятия и пояснения**

Одной из самых распространенных служб Интернета (наряду с электронной почтой) является World Wide Web (WWW, Web) — самая мощная информационная система, связывающая между собой миллионы документов (Web-страниц), размещенных на Web-серверах по всему миру. В расширении этой системы могут принимать участие практически все пользователи, имеющие доступ в Интернет.

***Web-страница*** — это документ (гипертекстовый документ), размеченный с помощью специальных элементов (тегов) языка HTML. Основное отличие этих гипертекстовых документов от хорошо известных нам текстовых состоит в том, что они могут включать ссылки на другие аналогичные документы.

Любой пользователь получает доступ к WWW-системе (к любой Web-странице) с помощью ***Web-браузера (программы-клиента)***. Web-браузер осуществляет http-запрос к серверу (о протоколе HTTP, определяющем правила взаимодействия клиента и сервера при передаче гипертекстовых документов, можно достаточно полно узнать, например, по адресу [*http://www.codenet.ru/webmast/http/*](http://www.codenet.ru/webmast/http/) ), отображает Web-страницу на экране компьютера пользователя. Современные популярные Web-браузеры ***Microsoft Internet Explorer*** (версии 4 — 6), ***Netscape Navigator*** (версии 4,6 и 7), ***Opera*** (версия 6) являются многофункциональными программами и поддерживают не только доступ к WWW, но и протоколы электронной почты, FTP, News и т. д.

***Web-сервер*** (более точно — http-сервер, реализующий поддержку протокола HTTP) — специальная программа, установленная на общедоступном компьютере. Web-сервер обеспечивает хранение Web-страниц в виде каталога файлов, структурированных определенным образом, и одновременную обработку запросов от нескольких Web-браузеров с целью поиска и/или формирования Web-страниц с последующей их отправкой клиентам.

Современные популярные Web-серверы:

***Apache*** — устанавливается в основном на ОС Unix, интегрируется с базами данных MySQL, используется большинством Web-провайдеров. Загрузить Apache можно с сайта http://www.apache.org/ совершенно бесплатно.

***Microsoft Internet Information Server (IIS)*** — устанавливается на ОС Windows NT/2000, интегрируется с базами данных Microsoft SQL Server, используется чаще в качестве отдельно стоящего Web-сервера, находящегося непосредственно в организации.

Для задания местоположения **Web-страниц** в World Wide Web используется **URL** (Uniform Resource Locator). Каждая страница имеет свой уникальный адрес.

Примеры URL:

1. [*http://www.microinform.ru/*](http://www.microinform.ru/) - http-сервер должен найти и отправить клиенту (браузеру) стартовую страницу сайта компании "Микроинформ";
2. [*http://www.microinform.ru/webschool/webschool.htm*](http://www.microinform.ru/webschool/webschool.htm) - http-сервер должен найти и отправить клиенту (браузеру) файл webschool.htm, который находится в каталоге webschool.

http://www.osp.ru/data/639/515/1234/084_0.gif

URL включает:

* Метод доступа к ресурсу, т. е. протокол доступа (http-доступ к Web-странице в сети с использованием соответствующего протокола. Это наиболее часто используемый метод доступа к какому-либо HTML-документу в сети.
* Cетевой адрес ресурса (имя хост-машины).
* Полный путь к файлу на сервере.

**Доменное имя сервера (Domain Name)** — уникальный идентификатор, который назначается определенному IP-адресу. Доменное имя дает возможность обращаться к компьютеру по имени типа *www.company.com* вместо его числового эквивалента.

**IP-адрес** представляет собой последовательность из четырех чисел, разделенных точками. Каждый компьютер, подключенный к Интернету (его еще называют хост), имеет уникальный IP-адрес, однозначно определяющий этот компьютер в Сети. Именно по IP-адресу происходит поиск и взаимодействие устройств.

Запоминать такие имена довольно сложно, поэтому и была разработана доменная система имен (Domain Name System, DNS), которая позволяет поставить в соответствие IP-адресу символьное имя.

Например, IP-адрес компьютера, на котором находится Web-сервер «Микроинформ»: 194.87.166.10. Этому IP-адресу соответствует доменное имя *www.microinform.ru*.

Для размещения Web-сайта в Интернете необходимо наличие доменного имени. Для этого его надо зарегистрировать (например, на сайте [*http://www.nic.ru/*](http://www.nic.ru/)).

**Web-сайт** — это набор Web-страниц, относящихся к одной фирме или одному пользователю. Есть понятие автора Web-сайта — это Web-мастер. За содержимое Web-сайта (контент) отвечает Web-редактор. В случае небольшого (до 10 страниц), а зачастую и среднего Web-сайта (до 100 страниц) Web-мастер и Web-редактор — это, как правило, одно лицо.

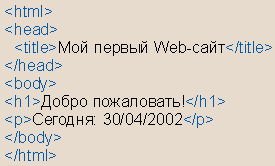
Web-сайт устанавливается на отдельно стоящий Web-сервер или на Web-сервер провайдера. На сервере может размещаться несколько сайтов. Установку, настройку и поддержку Web-сервера, включая обновление информации (замену или добавление файлов), осуществляет Web-администратор, которому контент Web-сайтов по большому счету безразличен.

**О Web-технологии клиент—сервер**

Теперь поговорим о технологии взаимодействия браузера и сервера (клиент-серверная технология).

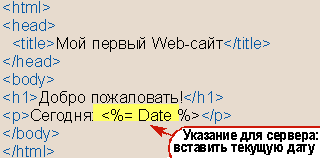
Здесь следует остановиться на понятии **статичной и динамичной Web-страницы**.

**Статичная Web-страница** состоит из «чистых» html-тегов (рис. 1).



|  |
| --- |
| http://www.osp.ru/data/643/515/1234/084_1.jpg |
| **Рис.1. Процесс взаимодействия клиент—сервер в случае статичной страницы** |

**Динамичная Web-страница** включает в себя «активные» коды (PHP, ASP, SSI...), которые при запросе страницы отрабатываются сервером, как бы формирующим страницу «на лету» перед отправкой клиенту. В примере, приведенном на рис. 2, динамичная Web-страница включает ASP-код формирования текущей даты (выделен желтым), который при запросах, естественно, будет давать каждый день другой результат.



|  |
| --- |
| http://www.osp.ru/data/647/515/1234/084_2.jpg |
| **Рис. 2. Процесс взаимодействия клиент—сервер в случае динамичной страницы** |

Обратите внимание, что в обоих случаях пользователь видит на экране один и тот же результат. Разница лишь в том, что в случае статичной страницы Web-мастеру придется редактировать ее каждый день (вписывать текущую дату), а в случае динамичной страницы при запросе ее клиентом эту функцию будет выполнять сервер.

**Основные элементы Web-страницы и файловая организация Web-сайта**

Для того чтобы получить представление об основных элементах Web-страницы и файловой организации Web-сайта на диске Web-сервера, рассмотрим простейший пример (рис. 3).

|  |
| --- |
| http://www.osp.ru/data/645/515/1234/084_3.jpg |
| **Рис. 3. Web-сайт, состоящий из трех связанных Web-страниц** |

Все эти страницы имеют единую структуру и единое оформление. Расположение элементов на экране: заголовок страницы (в данном случае рисунок); элементы навигации (меню) для перехода со страницы на страницу (в нашем случае каждый пункт меню является также отдельным рисунком); фотографии; текстовый блок.

|  |
| --- |
| http://www.osp.ru/data/619/515/1234/084_4.gif |
| **Рис. 4. Файловая структура сайта «Я и мой кот»** |

Каждая страница представляется отдельным файлом с расширением .html. Первая (стартовая) страница сайта получает, как правило, имя: index.html (для Web-сервера Apache); default.htm (для Microsoft IIS). Остальные файлы именуются произвольным образом, но рекомендуется делать это в соответствии с назначением документа или его содержимым, например aboutme. html и mycat.html. Обратите внимание, что именование папок и файлов допускается только в основной английской кодировке (в отличие от именования в Windows). Каждый рисунок представляется файлом с расширением .jpg или .gif.

Рекомендуемая для нашего случая файловая структура сайта представлена на рис. 4.

Здесь в корне каталога всегда хранится имя стартовой страницы. Мы сюда же поместили имена файлов и двух других страниц, что допустимо для небольшого сайта (до 10 страниц). Затем формируем папку для хранения рисунков (Pictures), а в ней — папку Navigation для хранения рисунков с элементами навигации, папку Photo для фотографий и папку Titles для рисунков с заголовками Web-страниц.

**Программы, необходимые для создания Web-сайта**

Выбор программ разработки не зависит от того, на какой Web-сервер (Apache или IIS) вы планируете устанавливать свой сайт.

1. Ключевая задача — выбор html-редактора. Они бывают двух видов.

**Кодовые редакторы.** Наиболее популярный и чаще других используемый профессионалами — редактор ***Home Site***. В данном редакторе Web-мастер контролирует создание и редактирование каждой строки html-кода.

**Визуальные редакторы.** Наиболее яркий представитель — Microsoft Front Page (работа с ним напоминает работу в Word) . С его помощью можно быстро и без знания языка HTML слепить свой первый сайт. Основные недостатки: порождает «массивные» Web-страницы, а также проблемы с браузерами, например с Netscape.

Начинающим Web-мастерам можно еще рекомендовать html-редактор ***Macromedia Dreamweaver***, занимающий промежуточную позицию между кодовыми и визуальными редакторами. К тому же по нему достаточно много литературы на русском языке.

2. При выборе графического редактора альтернатив практически нет — это **Adobe Photoshop**.

Вот, собственно, пока и достаточно. На следующем уроке мы займемся созданием рабочего места Web-мастера.

**Полезный совет 1**

Как пользователь Всемирной паутины вы можете предпочитать любой браузер. Но как создателю сайтов, пусть даже небольших, вам придется установить на свой компьютер все три (IE, Netscape и Opera) и тестировать в них созданные Web-странички. Ведь ваша задача сделать так, чтобы сайт выглядел практически одинаково при просмотре с помощью любого из этих браузеров. И все из-за так называемой войны браузеров Netscape Navigator и Microsoft Internet Explorer.

**Полезный совет 2**

Что касается сервера Apache, то есть версии и под Windows (Apache for Win32). Для знакомства и отладки на свой рабочий компьютер вполне можно установить именно ее. На странице [*http://httpd.apache.org/dist/httpd/binaries/win32/*](http://httpd.apache.org/dist/httpd/binaries/win32/) всегда можно найти последнюю версию Apache для платформы Windows.

Еще один полезный ресурс: [*http://apache.lexa.ru/*](http://apache.lexa.ru/) — Russian Apache. Это сайт поддержки русского Apache — модифицированной версии оригинального Web-сервера Apache, позволяющей корректно работать с русскими кодировками (Win1251, KOI8 ...).

**Несколько cоветов об URL и доменных именах**

* Отнеситесь серьезно к выбору доменного имени. По возможности выбирайте такое, которое легко запомнить, произнести и набрать.
* Как пользователи могут попасть к вам на сайт? Например, зная название фирмы, попытаются угадать доменное имя сайта. Поэтому закрепите за собой доменные имена, совпадающие с названием вашей компании. Этих имен может быть несколько. Не скупитесь. Регистрация в зоне .ru стоит порядка 25 долл., а поддержка - 20 долл. в год.
* Чем короче URL, тем лучше, так как часто приходится набирать их вручную.
* **При организации структуры сайта очень неплохо иметь такие URL, которые можно править, "откусывая" последнюю часть так, чтобы попадать на более высокие уровни иерархии сайта.**

**Вёрстка веб-страниц**

**Вёрстка веб-страниц** — создание структуры html-кода, размещающего элементы веб-страницы (изображения, текст и т. д.) в окне браузера, согласно разработанному макету, таким образом, чтобы элементы дизайна выглядели аналогично макету.

Вёрстка веб-страниц отличается от [полиграфической](https://ru.wikipedia.org/wiki/Вёрстка) тем, что необходимо учитывать разницу отображения элементов в различных [браузерах](https://ru.wikipedia.org/wiki/Браузер) и разницу в размерах рабочего пространства устройств.

Процесс сложен и имеет творческую основу, ни один из способов не является каноничным и принятым как основа. Все подходы к вёрстке имеют как преимущества, так и недостатки.

Хотя работа верстальщика скрыта от глаз, именно она обеспечивает бесперебойность при работе на различных устройствах, а также скорость загрузки каждой страницы сайта.

**История развития верстки веб-страниц**

[Тим Бернерс-Ли](https://ru.wikipedia.org/wiki/Тим_Бернерс-Ли), работая в [Европейском институте физики частиц](https://ru.wikipedia.org/wiki/CERN) придумал язык, который концентрировался не на средствах оформления, а на логике разметки текста. Некоторое время страницы оформлялись в так называемом «академическом дизайне», например, [lib.ru](https://ru.wikipedia.org/wiki/Lib.ru)[[3]](https://ru.wikipedia.org/wiki/Вёрстка_веб-страниц" \l "cite_note-samlayout-3).

С появлением браузера Netscape ситуация изменилась. Разработчики браузера внедряли нестандартные для html теги, призванные улучшить внешний вид документа. Браузер был настолько распространённым, что нестандартные теги не вызывали беспокойств: нет смысла беспокоиться о том, что другие браузеры не поддерживают эти теги, когда только 10 % посетителей пользуется другими браузерами. Сайты перестали разрабатываться в «академическом» стиле. Над дизайном стали задумываться, а значит, потребовалась более сложная верстка.

С середины 1990-х MS Internet Explorer захватывал всё большую часть потребителей. Подходы разработчиков браузеров различались, отличался и набор поддерживаемых технологий. Всё это лишь добавляло неудобств для разработчиков сайтов — увеличивая время разработки сайта.

Во время борьбы за рынок на смену устаревающим и плохо поддерживающим стандарты Netscape 4 и Internet Explorer 4 (IE 4) пришли пятые версии браузеров[. IE 5 должна была не просто исправить ошибки предшественника, а улучшить поддержку стандартов CSS. На этот шаг Майкрософт была обречена хотя бы потому, что вошла в группу](https://ru.wikipedia.org/wiki/Вёрстка_веб-страниц" \l "cite_note-rezhimyBrauzerov-4) [W3C](https://ru.wikipedia.org/wiki/W3C)[[4]](https://ru.wikipedia.org/wiki/Вёрстка_веб-страниц" \l "cite_note-rezhimyBrauzerov-4). В процессе разработки разница в отображении сайтов стала настолько значительной (а часть сайтов вовсе не работали), что разработчикам пришлось создать специальный «режим совместимости». В таком режиме браузер работал «по старинке». Для того, чтобы использовать режим поддержки стандартов необходимо было в html-код добавить элемент <!DOCTYPE>. Идея с режимом совместимости позднее перешла в остальные браузеры (например, Mozilla, Safari, Opera и т. д.). С развитием технологий разновидностей doctype оказалось слишком много, только к пятой версии html W3C предложили использовать короткую запись унифицировав стандарт.

Через некоторое время работать блоками стало слишком неудобно, так как поддержки CSS у браузеров фактически не было, приходилось вставлять стили прямо в блоки, что усложняло разработку. В это время для разработки стали применяться таблицы с невидимой границей, где многие стили не требовались — они были по умолчанию.

В 1998 году появляется спецификация Cascading Style Sheets Level 2 (CSS2), а потом и CSS2.1, что позволило выносить код оформления блоков в отдельные файлы и использовать на всех страницах сайта и даже [кэшировать](https://ru.wikipedia.org/wiki/Кэш) этот файл, а значит ускорять загрузку страниц. Развитие этой технологии и поддержка со стороны браузеров привела к использованию блочной вёрстки, вместо таблиц.

**Этапы вёрстки**

**Почему прямоугольники?**

Прямоугольник получил исключительное значение благодаря способности делиться на любое число прямоугольников любых пропорций.

Как правило верстальщик получает от дизайнера утверждённый дизайн-макет страницы. Далее, верстальщик анализирует полученный макет. Разбивает его на горизонтальные линии (полосы) — «этажи». Далее, каждый «этаж» анализируется отдельно и разбивается на прямоугольные блоки — колонки.

Далее происходит рекурсивный процесс вёрстки этих отдельных строк, а в них столбцов.

После вёрстки страница проверяется на [кросс-платформенность](https://ru.wikipedia.org/wiki/Кросс-платформенность). В общем случае можно отвечать на следующие вопросы:

* Одинаково ли отображается страница в браузерах и на разных операционных системах?
* Происходит ли критичное смещение блоков, если изменять размер шрифта в настройках браузера в меньшую и большую сторону?
* Происходит ли критичный сдвиг блоков, если отключить показ изображений в браузере?
* Оказывает ли существенное влияние на целостность страницы разрешение монитора?

Критические исправления вносятся в документ и проверка повторяется с самого начала.

**Верстка создаётся для шаблонов**

Для сайта, как правило, не используются сверстанные документы. Документ после верстки разрезается на повторяющиеся кусочки и используется по всему сайту. Например, верхняя полоса с логотипом и нижняя с важной информацией. Такие кусочки верстки называются шаблонами (от [англ.](https://ru.wikipedia.org/wiki/Английский_язык) template).

**Ограничения**

Одним из важных ограничений являются шрифты. Гарнитурные наборы у всех операционных систем отличаются. Выбор гарнитуры не ограничен ничем, но не найдя указанного набора браузер использует настройки по умолчанию.

**Многообразие форматов**

Распространение веба тотально. На момент 2010 года веб-ресурс может быть открыт на любом мониторе от телевизора с предустановленным браузером до мобильного телефона. Возможности по отображению цветов на всех устройствах также различаются. Однако, основным вопросом в верстке является — отображение при различной ширине рабочего пространства

Верстка отличается различными подходами к этому вопросу:

* Фиксированная ширина — сайт отображается с одинаковой шириной вне зависимости от ширины окна браузера. Соответственно, на широкоформатных браузерах такие сайты имеют большие поля по краям, а на мелких экранах смартфонов показывают горизонтальную прокрутку.
* «Резиновая» ширина — полосы сайта растягиваются на весь экран, занимая место пропорционально настройкам верстальщика.

Исторически сложилось, что сайт — вертикальный формат, который занимает не более чем ширину экрана, чтобы избежать горизонтальной прокрутки. Все возможности по управлению положения сайта (клавиатурные комбинации и полоса прокрутки и колесо мыши) настроены непосредственно на вертикальное передвижение.

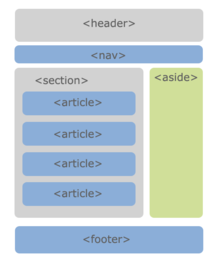
**Модульность и виды подходов к вёрстке**

|  |  |  |
| --- | --- | --- |
| **Типовые веб-макеты** | | |
|  |  |  |
| Одноколоночный | Двухколоночный | Трёхколоночный |

Ни один из способов не является каноничным и принятым как основной. Все подходы к верстке имеют как преимущества, так и недостатки. Верстальщик, как специалист, сам выбирает какие инструменты использовать, основываясь на взвешивании достоинств и недостатков для конкретного проекта.

Модульная сетка структурно разбивает материал на отдельные блоки, для упрощения добавления новых элементов на страницу. Некоторые верстальщики могут уйти от правил верстки, ради определённого проекта, но необходимость простого добавления вновь появляющихся элементов на странице обострит необходимость возврата к этим правилам.

Модульная сетка — набор направляющих, которые выравниваются относительно друг друга и делят собой макет на квадраты. Позволяют, таким образом, независимо от остальной части сайта изменять единицу информации.

[](https://commons.wikimedia.org/wiki/File:Html-5.png?uselang=ru)

[Семантическая вёрстка](https://ru.wikipedia.org/wiki/Семантическая_вёрстка) страницы

Основными инструментами для вёрстки являются [фреймы](https://ru.wikipedia.org/wiki/Фрейм_(HTML)), таблицы и div. В стандарте [HTML5](https://ru.wikipedia.org/wiki/HTML5) фреймы больше не поддерживаются, а для работы стали доступны [«семантические» блоки](https://ru.wikipedia.org/wiki/Семантическая_вёрстка): header, main, nav, section, article, aside, footer. Однако блоки не дают определённых преимуществ, а являются лишь «[синтаксическим сахаром](https://ru.wikipedia.org/wiki/Синтаксический_сахар)».

**Табличный подход**

Таблицы являются удобным инструментом для выравнивания блоков. В html таблица делится на строки, которые в свою очередь делятся на ячейки, таким образом можно создавать неограниченное количество строк с неограниченным числом ячеек в каждой. Ячейки таблицы имеют особые настройки выравнивания, которые не встречаются более в других элементах. Это значит, что некоторые элементы дизайна крайне сложно сверстать без использования таблиц.

В деловом мире таблицы используется для сравнения данных, а также для наглядности.

Недостатком табличного подхода является отрисовка (от [англ.](https://ru.wikipedia.org/wiki/Английский_язык) rendering) таблицы. До тех пор, пока не будет загружен закрывающий тег таблицы она и всё её содержимое не будет показано. Что является критичным при обрыве связи и медленном соединении.

На момент 2011 года табличная верстка считается устаревшей.

**Вёрстка слоями**

Слой (от имени [тега](https://ru.wikipedia.org/wiki/Тег_(языки_разметки)) <layer>) — разработка компании Netscape, использованная в браузере [Netscape Navigator](https://ru.wikipedia.org/wiki/Netscape_Navigator). Тег позволял показывать или скрывать его содержимое, устанавливать положение относительно окна браузера, накладывать слои друг на друга и включать содержимое блока из файла. В спецификацию HTML не был включён.

**Блочная вёрстка**

Блочная вёрстка — верстка с помощью [тега](https://ru.wikipedia.org/wiki/Тег_(языки_разметки)) <div> и описывающего их множество [таблицы стилей](https://ru.wikipedia.org/wiki/CSS) (CSS). Тег появился вследствие сознательного отказа консорциума «CSS Positioning (CSS-P)» от тега <layer>, чтобы создать аналогичный тег, но с поддержкой CSS в духе остальных тегов. С помощью блочной верстки реализуют концепцию [семантической вёрстки](https://ru.wikipedia.org/wiki/Семантическая_вёрстка).

**Вёрстка фреймами**

Вёрстка фреймами — верстка с помощью одноимённого тега <frame>, который имеет иной способ позиционирования, а внутри себя включает отдельные html-страницы.

Фреймы просты в применении, поскольку необходимо заботиться только об отображении и перерисовке части окна браузера — одного фрейма, — остальные части находятся в другом фрейме и не перерисовываются без взаимодействия с ними. Также возможно открытие ссылки в другом фрейме, что удобно при использовании главного меню в отдельном фрейме. Страницы поделённые на фреймы имеют меньше кода, вследствие отсутствия повторяемых частей, которые не перезагружаются. Они размещаются точно в нужном месте окна браузера.

Фреймы плохо индексируются поисковыми системами, поскольку на страницах с содержимым нет ссылок на другие страницы сайта и наоборот, в навигационной странице нет никакого содержимого. Переход из поисковой страницы происходит на одну страницу, без загрузки других фреймов — навигации и логотипа. Внутренние страницы нельзя добавить в закладки, потому что браузер не показывает изменения в адресной строке, отображая всегда только адрес сайта. Совместимость между браузерами противоречива. Одни и те же параметры интерпретируются браузерами всегда по-своему.

**Разметка страницы**

По принципам использования средств разметки [HTML](https://ru.wikipedia.org/wiki/HTML) различают логическую разметку и презентационную (физическую).

Например, *курсивный текст* можно получить как с помощью тега <i>, так и с помощью тега <em>.

В первом случае курсив задаётся явным образом, а во втором на текст производится логическое ударение, которое обычно отображается курсивом.

Иными словами, при первом подходе ориентируются на внешний вид, а во втором — на логическое предназначение.

Преимуществом второго подхода является независимость вёрстки от используемого типа устройств и дизайна веб-страниц.

Если придерживаться логической разметки, то можно использовать один и тот же вариант вёрстки для [экрана](https://ru.wikipedia.org/wiki/Монитор_(устройство)), [печати](https://ru.wikipedia.org/wiki/Принтер) и [мобильных устройств](https://ru.wikipedia.org/wiki/Мобильный_компьютер), регулируя внешний вид с помощью отдельных файлов стилей.

**Табличная вёрстка**

**Преимущества и недостатки**

[Браузеры](https://ru.wikipedia.org/wiki/Браузер) преднамеренно расценивают таблицу как один объект, из-за чего содержимое таблицы не отображается до тех пор, пока оно целиком не будет загружено на локальный компьютер.

При использовании таблицы в качестве каркаса для размещения элементов веб-страницы, её исходное преимущество обращается в недостаток, поскольку приводит к задержке вывода содержимого. Следует учитывать также и растущий объём веб-страниц при активном использовании таблиц, особенно в случае их вложенности друг в друга. Всё это приводит к тому, что табличная верстка вызывает ненужные задержки вывода информации в браузере.

Проблема возникает при множественной вложенности таблиц, что характерно для достижения определённых эффектов на веб-странице. Рост количества таблиц повышает шанс возникновения ошибок при вёрстке, увеличивает размер документов и снижает скорость загрузки файлов. Применение визуальных редакторов, вроде [Adobe Dreamweaver](https://ru.wikipedia.org/wiki/Adobe_Dreamweaver) или [Microsoft FrontPage](https://ru.wikipedia.org/wiki/Microsoft_FrontPage), для создания и правки документов облегчает работу с таблицами, но из-за обилия их параметров и в этом случае разработчики не застрахованы от появления ошибок и лишней работы, связанной с индивидуальным редактированием каждой таблицы.

Кроме того, использование таблиц для целей оформления не соответствует концепции семантичной вёрстки, подразумевающей использование элементов (тегов) в соответствии с их смыслом, семантическим значением. Элемент <table> и сопутствующие (<tr>,<th>,<td> и др.) предназначены для разметки табличной информации (то есть такой, в которой имеется смысловая связь между элементами, принадлежащими одному столбцу или одной строке). W3C, как разработчик языка HTML, призывает использовать HTML для *логической разметки* информации, а оформление (в том числе позиционирование различных блоков) описывать отдельно от (X)HTML-разметки (с помощью CSS, например).

**Распорки**

При использовании таблиц широко известным приёмом стало применение распорок — прозрачных изображений высотой в один пиксель. Само изображение не отображается на веб-странице, но его можно масштабировать в любых пределах. Получается невидимая палочка определённой ширины или высоты, которая не дает ячейкам таблицы сблизиться меньше, чем на заданное расстояние. Распорки особенно были актуальны для браузера [Netscape](https://ru.wikipedia.org/wiki/Netscape_Navigator), который не показывал фон ячейки, если в ней ничего не размещалось. Чтобы избавиться от указанной особенности, и помещали в ячейку маленький прозрачный рисунок в формате [GIF](https://ru.wikipedia.org/wiki/GIF).

Подобные техники не только усложняют разработку универсальных веб-страниц, но и ведут к снижению скорости загрузки документа. Браузеру в этом случае приходится загружать элементы, которые не видны пользователю и, по сути, ему не нужны, но они входят в общий трафик сайта.

**Блочная вёрстка**

[Слои](https://ru.wikipedia.org/wiki/Слой) представляют собой структурные элементы, которые можно размещать на веб-странице путём наложения их друг на друга с точностью до [пикселя](https://ru.wikipedia.org/wiki/Пиксель). В [HTML 4](https://ru.wikipedia.org/wiki/HTML_4) и [XHTML](https://ru.wikipedia.org/wiki/XHTML) слой — это элемент веб-страницы, созданный с помощью тега <div>, к которому применяется стилевое оформление.

При этом придерживаются следующих принципов:

* Разделение содержимого и оформления;
* Активное применение тега <div>;
* Таблицы применяются только для представления табличных данных.

Слои можно размещать в окне браузера с точностью до пикселя. Положение слоя задаётся двумя координатами относительно любого угла окна браузера, родительского элемента или документа.

[Скрипты](https://ru.wikipedia.org/wiki/Скрипт) позволяют изменять параметры слоя динамически. Это даёт возможность создавать на странице разные эффекты, такие как выпадающие меню, игры, разворачивающиеся [баннеры](https://ru.wikipedia.org/wiki/Баннер), плавающие окна и прочее.

Свойства слоя удобно задавать и настраивать через стили. Возможности [CSS](https://ru.wikipedia.org/wiki/CSS) расширяют спектр оформительских изысков. Использование стилевых таблиц позволяет несложными методами получить компактный и эффективный код.

Более современные версии [браузеров](https://ru.wikipedia.org/wiki/Браузер) стали строже придерживаться стандартов и содержать средства по работе со слоями.

**Преимущества и недостатки**

|  |  |
| --- | --- |
|  |  |

Слой можно перемещать, прятать и показывать без перезагрузки всей страницы. С помощью всего нескольких инструкций можно создавать разные эффекты, вроде выпадающих меню, всплывающих подсказок, движущихся элементов и другое. Добавление подобных трюков хотя и увеличивает объём кода, но не требует повторной загрузки и обновления документа и происходит без лишних задержек со стороны браузера. Кроме того, выразительность и привлекательность сайта во многом повышается благодаря использованию подобных приёмов со слоями.

Слои можно накладывать друг на друга, что упрощает размещение элементов на веб-странице и предоставляет больше возможностей при вёрстке.

Слои по сравнению с таблицами отображаются быстрее. Более высокая скорость достигается за счёт компактного кода и того, что отображение содержимого слоя происходит по мере его загрузки. Правда это может привести к «скачкам» элементов страницы по мере их подгрузки.

Не следует считать, что использование слоёв это панацея от всех бед. К сожалению, стандарты работы со слоями ещё не до конца устоялись и браузеры по-разному реализуют определённые возможности. Из-за этого основная сложность вёрстки слоями — создать универсальный код, который бы одинаково и без ошибок работал в разных браузерах («[кросс-браузерность](https://ru.wikipedia.org/wiki/Кросс-браузерность)») и при разных [разрешениях экрана](https://ru.wikipedia.org/wiki/Разрешение_(компьютерная_графика)). Приходится вникать в тонкости поведения браузеров при использовании различных элементов стилей.

**Верстальщик**

Вёрсткой веб-страниц занимаются [верстальщики](https://ru.wikipedia.org/wiki/Верстальщик). В общем случае в задачу верстальщика входят:

* создание кода веб-страницы с помощью соответствующего [языка разметки](https://ru.wikipedia.org/wiki/Язык_разметки). Таковыми могут быть, например, [HTML](https://ru.wikipedia.org/wiki/HTML), [XHTML](https://ru.wikipedia.org/wiki/XHTML), [XML](https://ru.wikipedia.org/wiki/XML).
* [оформление](https://ru.wikipedia.org/wiki/Оформление_книги) ранее созданного кода страницы с помощью встроенных средств языка разметки, либо же с помощью каскадных таблиц стилей [CSS](https://ru.wikipedia.org/wiki/CSS)

**Инструменты верстальщика**

Верстальщик использует следующее ПО:

* [текстовый редактор](https://ru.wikipedia.org/wiki/Текстовый_редактор) или [редактор HTML](https://ru.wikipedia.org/wiki/Редактор_HTML) для написания и редактирования кода
* [графическая программа](https://ru.wikipedia.org/wiki/Графические_программы) для так называемой «нарезки» графического макета, полученного верстальщиком от [веб-дизайнера](https://ru.wikipedia.org/wiki/Веб-дизайнер)

А также иногда прибегает к помощи:

* [WYSIWYG](https://ru.wikipedia.org/wiki/WYSIWYG) редакторов, в которых пользователь располагает все элементы, которые должны были быть получены с помощью HTML, используя [графический интерфейс пользователя](https://ru.wikipedia.org/wiki/Графический_интерфейс_пользователя). После чего программа преобразует визуальное представление в HTML код. В данном случае автору не обязательно обладать исчерпывающим знанием HTML.
* программы автоматической верстки сайтов, которые извлекают слои с изображениями и текстом из дизайн-макета, созданного в графическом редакторе [Adobe Photoshop](https://ru.wikipedia.org/wiki/Adobe_Photoshop) или ему подобном и формируют из этих слоёв HTML код. Таким образом создается базовый каркас веб страницы, готовый к последующей доработке.

Использование [WYSIWYG](https://ru.wikipedia.org/wiki/WYSIWYG) редакторов и программ автоматической верстки часто осуждается из-за низкого качества получаемого кода. Тем не менее, зачастую для пользователей их использование удобнее ручного кодирования, а также не требуют глубокого знания HTML, и поэтому широко используются.

**Валидность HTML-верстки**

[Валидность](https://ru.wikipedia.org/wiki/Валидность) HTML-верстки — это её соответствие стандартам организации The World Wide Web Consortium ([W3C](https://ru.wikipedia.org/wiki/Консорциум_Всемирной_паутины)). Отсутствие ошибок в вёрстке документа — один из основных показателей качества вёрстки. Автоматическая проверка вёрстки на ошибки может быть проведена как с помощью [онлайн сервиса W3C](http://validator.w3.org/), так и различными программами «валидаторами». Разные версии спецификации HTML предполагают различный синтаксис, поэтому тест верстки на валидность должен обязательно учитывать её [Document Type](https://ru.wikipedia.org/wiki/Элементы_HTML).

**Кроссбраузерность**

[Кроссбраузерность](https://ru.wikipedia.org/wiki/Кроссбраузерность) сайта — близкое к исходному дизайнерскому и функциональному виду отображение параметров страниц при использовании разных браузеров и их различных версий и модификаций.

Веб-разработчики принимают всевозможные усилия по унификации гипертекстовой разметки, направленные на разработку единого стандарта отображения в [браузерах](https://ru.wikipedia.org/wiki/Браузер), но реализация этого — сложный технологический процесс из-за ряда конфронтаций среди разработчиков.

Web-программирование

Лекция 2. Язык HTML

**План**

Определение

Теги и атрибуты

Структура документа

Теги раздела head

Таблица

Список

Верстка

Html4 и html5

**Определение**

HTML – HyperText Markup Language – Гипертекстовый язык разметки

Используется для разметки веб-страниц при помощи тегов. Теги задают структуру страницы (описывают абзац, список, таблицу, блок и т.д.). Внутри тега (между открывающей и закрывающей частями) находится контент (текст).

Оформление страницы сейчас выполняется с помощью правил css.

**Теги и атрибуты**

Тег – стандартное название элемента страницы. Заключено в угловые скобки.

<имя тега> контент </ имя тега>

Теги обычно состоят из двух частей - открывающей и закрывающей. Например, у тега <div>Hello world!</div>

<div> - открывающая часть

Hello world! – содержимое тега

</div> - закрывающая часть

Есть и одинарные теги:

<img> – вывод изображения <img src=”my.jpg”>

<br> - перевод строки

Атрибут – свойство тега. Записывается в открывающей части. Форма записи: имя\_атрибута = «значение».

<img src=”my.jpg” > - src – атрибут, содержащий имя графического файла.

У каждого тега есть свой фиксированный набор атрибутов. Атрибуты применяются по необходимости. Записываются в произвольном порядке.

**Структура документа**

Любая веб-страница всегда имеет две секции:

<head> – содержит служебную информацию (название страницы, стили, сценарии)

<body> – содержит непосредственно разметку веб-страницы

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>Моя первая веб-страница</title>

</head>

<body> <h1>Заголовок страницы</h1> <p>Основной текст.</p> </body></html>

**Основные теги**

<a> – вывод гиперссылки <a href=”http://google.ru”>Открыть поисковик Google</a>

<p> – вывод параграфа текста <p>Hello world!</p>

<h1> –вывод заголовка первого уровня <h1>Мой заголовок</h1>

<button> – вывод кнопки <button>Нажми меня!</button>

**Теги раздела head**

Раздел head содержит служебную информацию: сведения для поисковых систем и браузера, ссылки на подключаемые файлы, таблицы стилей и сценарии.

Теги: meta, link, style, script

<head>

<title>Название вкладки со страницей</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<link rel="stylesheet" type="text/css" href="style.css">

<style>

body {

background: #FFF;

color: #000;

font-family: Arial, sans-serif;

font-size: 14px;

}

</style>

<script type= "text/javascript">

prompt ('Это диалоговое окно prompt.', '') </script>

</head>

**Таблица**

Основные теги таблицы: <table> <th> <tr> <td>

Таблица формируется как набор строк (<tr>) из ячеек (<th> или <td>).

Для расширения ячейки таблицы на несколько строк и/или столбцов применяются атрибуты colspan и rowspan.

<table border =2 > <caption> Объединение строк и столбцов</caption>

<tr> <th>столбец 1</th> <th>столбец 2</th> <th>столбец 3</th></tr>

<tr> <td> ячейка 1 </td> <td> ячейка 2 </td><td> ячейка 3</td></tr>

<tr>

<td colspan = 2 rowspan = 2> ячейка 4 </td>

<td> ячейка 6</td></tr>

<tr> <td> ячейка 9</td></tr>

</table>

**Список**

<ol> + <li> нумерованный список

<ol> <li> декабрь </li>

<li> январь </li>

<li> февраль </li>

</ol>

<ul> + <li> список с маркером

<ul> <li> декабрь </li>

<li> январь </li>

<li> февраль </li>

</ul>

<dl> + <dt> + <dd> список определений

<dl> <dt>пункт 1 </dt>

<dd> первое определение пункта 1</dd>

<dd> второе определение пункта 1</dd>

<dt>пункт 2 </dt>

<dd> первое определение пункта 2</dd>

<dt>пункт 3 </dt>

<dd> первое определение пункта 3</dd>

</dl>

<p>Пример двухуровневого списка с комбинацией нумерации и маркеров</p>

<ol> <li> зима

<ul> <li> декабрь </li>

<li> январь </li>

<li> февраль </li>

</ul> </li>

<li> весна </li>

<li> лето </li> </ol>

***Листинг HTML документа***

<!DOCTYPE HTML>

<html> <head>

<meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />

<title>Основы HTML</title></head>

<body>

<h1>Название сайта </h1>

<a href = "ff.htm">Блок навигации </a>

<p>Пример двухуровневого списка с комбинацией нумерации и маркеров</p>

<ol> <li> зима

<ul> <li> декабрь </li>

<li> январь </li>

<li> февраль </li>

</ul> </li>

<li> весна </li>

<li> лето </li> </ol>

<table border =2 >

<caption>Таблица без объединенных ячеек</caption>

<tr> <th>столбец 1</th> <th>столбец 2</th> <th>столбец 3</th></tr>

<tr> <td> ячейка 1 </td> <td> ячейка 2 </td><td> ячейка 3</td></tr>

<tr> <td> ячейка 4 </td> <td> ячейка 5 </td><td> ячейка 6</td></tr>

<tr> <td> ячейка 7 </td> <td> ячейка 8 </td><td> ячейка 9</td></tr>

</table> <br>

<table border =2 >

<caption> Объединение строк</caption>

<tr> <th>столбец 1</th> <th>столбец 2</th> <th>столбец 3</th></tr>

<tr> <td> ячейка 1 </td> <td> ячейка 2 </td><td> ячейка 3</td></tr>

<tr> <td> ячейка 4 </td> <td> ячейка 5 </td><td rowspan = 2> ячейка 6</td></tr>

<tr> <td> ячейка 7 </td> <td> ячейка 8 </td></tr>

</table><br>

<table border =2 >

<caption> Объединение столбцов</caption>

<tr> <th>столбец 1</th> <th>столбец 2</th> <th>столбец 3</th></tr>

<tr> <td> ячейка 1 </td> <td> ячейка 2 </td><td> ячейка 3</td></tr>

<tr> <td colspan = 2> ячейка 4 </td> <td> ячейка 6</td></tr>

<tr> <td> ячейка 7 </td> <td> ячейка 8 </td><td> ячейка 9</td></tr>

</table><br>

<table border =2 > <caption> Объединение строк и столбцов</caption>

<tr> <th>столбец 1</th> <th>столбец 2</th> <th>столбец 3</th></tr>

<tr> <td> ячейка 1 </td> <td> ячейка 2 </td><td> ячейка 3</td></tr>

<tr> <td colspan = 2 rowspan = 2> ячейка 4 </td> <td> ячейка 6</td></tr>

<tr> <td> ячейка 9</td></tr>

</table> </body> </html>

**Web-программирование**

**Лекция DOM, javascript**

План

браузерные войны

доступ к элементу на странице html

знакомство с js

Лабораторная работа 3

**DOM** (от [англ.](https://ru.wikipedia.org/wiki/Английский_язык) *Document Object Model* — «объектная модель документа») — это не зависящий от платформы и языка [программный интерфейс](https://ru.wikipedia.org/wiki/Интерфейс_программирования_приложений), позволяющий [программам](https://ru.wikipedia.org/wiki/Компьютерная_программа) и [скриптам](https://ru.wikipedia.org/wiki/Скрипт) получить доступ к содержимому [HTML](https://ru.wikipedia.org/wiki/HTML)-, [XHTML](https://ru.wikipedia.org/wiki/XHTML)- и [XML](https://ru.wikipedia.org/wiki/XML)-документов, а также изменять содержимое, структуру и оформление таких документов.

Модель DOM не накладывает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями «родительский-дочерний».

Изначально различные [браузеры](https://ru.wikipedia.org/wiki/Браузер) имели собственные модели документов (DOM), несовместимые с остальными. Для обеспечения взаимной и обратной совместимости специалисты международного консорциума [W3C](https://ru.wikipedia.org/wiki/W3C) классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название «W3C DOM».

Ещё один интерфейс прикладного программирования, называемый «[JDOM](https://ru.wikipedia.org/wiki/JDOM)», обеспечивает более высокий, чем «W3C DOM», уровень для работы с XML-документами на [Java](https://ru.wikipedia.org/wiki/Java).

**История**

История DOM переплетается с историей «[браузерных войн](https://ru.wikipedia.org/wiki/Война_браузеров)» в конце 1990-х годов между [Netscape Navigator](https://ru.wikipedia.org/wiki/Netscape_Navigator) и [Microsoft Internet Explorer](https://ru.wikipedia.org/wiki/Microsoft_Internet_Explorer) (и между первыми скриптовыми языками [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) и [JScript](https://ru.wikipedia.org/wiki/JScript)) за то, чтобы широко использоваться в механизме [вёрстки веб-страниц](https://ru.wikipedia.org/wiki/Вёрстка_веб-страниц).

**Традиционный DOM**

Язык программирования «[JavaScript](https://ru.wikipedia.org/wiki/JavaScript)» был выпущен фирмой «[Netscape Communications](https://ru.wikipedia.org/wiki/Netscape_Communications)» в [1995](https://ru.wikipedia.org/wiki/1995_год) году в рамках [веб](https://ru.wikipedia.org/wiki/Всемирная_паутина)-[браузера](https://ru.wikipedia.org/wiki/Браузер) «[Netscape Navigator](https://ru.wikipedia.org/wiki/Netscape_Navigator) 2.0». Конкурент фирмы «Netscape» — фирма «[Microsoft](https://ru.wikipedia.org/wiki/Microsoft)» выпустила позже в том же году веб-браузер «[Internet Explorer](https://ru.wikipedia.org/wiki/Internet_Explorer) 3.0» с [портом](https://ru.wikipedia.org/wiki/Портирование_программного_обеспечения) JavaScript, названным «[JScript](https://ru.wikipedia.org/wiki/JScript)». JavaScript и JScript позволяют разработчикам создавать [веб-страницы](https://ru.wikipedia.org/wiki/Веб-страница), которые были бы интерактивны со стороны [клиента](https://ru.wikipedia.org/wiki/Клиент_(информатика)). Ограниченные возможности обнаружения создаваемых пользователем [событий](https://ru.wikipedia.org/wiki/Событийно-ориентированное_программирование) и изменения документа [HTML](https://ru.wikipedia.org/wiki/HTML) в первом поколении этих языков в итоге стали известны как «DOM уровня 0» или «традиционный DOM». Для DOM уровня 0 не было разработано никакого независимого стандарта, однако он был частично описан в спецификации [HTML4](https://ru.wikipedia.org/w/index.php?title=HTML4&action=edit&redlink=1).

Традиционный DOM был ограничен в типах элементов, к которым можно получить доступ. К таким элементам как форма ([form](https://ru.wikipedia.org/wiki/Элементы_HTML" \l ".D0.A4.D0.BE.D1.80.D0.BC.D1.8B)), ссылка ([link](https://ru.wikipedia.org/wiki/Гипертекст)) и изображение ([image](https://ru.wikipedia.org/wiki/Элементы_HTML" \l ".D0.98.D0.B7.D0.BE.D0.B1.D1.80.D0.B0.D0.B6.D0.B5.D0.BD.D0.B8.D1.8F)) можно было обращаться с помощью иерархических имён, которые начинались с корня объекта документа. Иерархическое имя могло использовать либо имя, либо последовательный индекс общего элемента. Например, элемент «form input» может быть доступен как «document.formName.inputName» или как «document.forms[0].elements[0]».

Традиционный DOM давал возможность подтверждения формы с [клиентской](https://ru.wikipedia.org/wiki/Клиент_(информатика)) стороны и популярный эффект «трансформации объекта».

**Промежуточный DOM**

В 1997 году — фирмы «Netscape» и «Microsoft» выпустили веб-браузеры, соответственно, «Netscape Navigator» и «Internet Explorer» версий 4.0, добавив поддержку «[Dynamic HTML](https://ru.wikipedia.org/wiki/Dynamic_HTML)» (DHTML), предоставляющего возможность изменения функциональности [HTML](https://ru.wikipedia.org/wiki/HTML)-документа при его загрузке. DHTML требовал расширений для элементарного объекта «document», имевшегося в традиционной реализации DOM. Поскольку «[JScript](https://ru.wikipedia.org/wiki/JScript)» был основан на «[JavaScript](https://ru.wikipedia.org/wiki/JavaScript)» — традиционные реализации DOM были в значительной степени совместимы, однако расширения DOM для DHTML были разработаны параллельно каждым из создателей браузера и остались несовместимыми. Эти версии DOM стали известны как «промежуточный DOM».

Промежуточный DOM давал возможность манипулировать свойствами [каскадных таблиц стилей](https://ru.wikipedia.org/wiki/Cascading_Style_Sheets) (англ. «CSS»), влияющими на отображение документа. Он также обеспечивал доступ к новому свойству под названием «слои» через свойства «document.layers» (в «[Netscape Navigator](https://ru.wikipedia.org/wiki/Netscape_Navigator)») и «document.all» (в «[Internet Explorer](https://ru.wikipedia.org/wiki/Internet_Explorer)»). Из-за исходной несовместимости в промежуточном DOM — разработка [веб-страниц](https://ru.wikipedia.org/wiki/Веб-страница) требовала специальной обработки для каждого случая.

Более поздние версии «[Netscape Navigator](https://ru.wikipedia.org/wiki/Netscape_Navigator)» отказались от поддержки промежуточного DOM. «[Internet Explorer](https://ru.wikipedia.org/wiki/Internet_Explorer)» продолжает поддержку своего промежуточного DOM для [обратной совместимости](https://ru.wikipedia.org/wiki/Обратная_совместимость).

**Стандартизация**

Организация «[World Wide Web Consortium](https://ru.wikipedia.org/wiki/World_Wide_Web_Consortium)» (W3C), основанная в [1994](https://ru.wikipedia.org/wiki/1994_год) году для поддерживания и развития открытых стандартов «[World Wide Web](https://ru.wikipedia.org/wiki/World_Wide_Web)», заставила «[Netscape Communications](https://ru.wikipedia.org/wiki/Netscape_Communications)», «[Microsoft](https://ru.wikipedia.org/wiki/Microsoft)», «[Apple](https://ru.wikipedia.org/wiki/Apple)» и другие компании разработать стандарт для [браузерных](https://ru.wikipedia.org/wiki/Браузер) [скриптовых языков](https://ru.wikipedia.org/wiki/Сценарный_язык) под названием «[ECMAScript](https://ru.wikipedia.org/wiki/ECMAScript)». Первая версия стандарта была опубликована в [1997](https://ru.wikipedia.org/wiki/1997_год) году. Последующие выпуски «[JavaScript](https://ru.wikipedia.org/wiki/JavaScript)» и «[JScript](https://ru.wikipedia.org/wiki/JScript)» стали реализовывать стандарт «[ECMAScript](https://ru.wikipedia.org/wiki/ECMAScript)» для лучшей [межбраузерной](https://ru.wikipedia.org/wiki/Браузер) [совместимости](https://ru.wikipedia.org/wiki/Совместимость_(информатика)).

После выхода «[ECMAScript](https://ru.wikipedia.org/wiki/ECMAScript)» — «[W3C](https://ru.wikipedia.org/wiki/W3C)» начала работу над стандартизацией DOM. Изначальный стандарт DOM, также известный как «DOM уровня 1», был рекомендован «[W3C](https://ru.wikipedia.org/wiki/W3C)» в конце [1998](https://ru.wikipedia.org/wiki/1998_год) года. Примерно в это же время вышел «[Internet Explorer 5.0](https://ru.wikipedia.org/wiki/Internet_Explorer_5)» с ограниченной поддержкой DOM уровня 1. DOM уровня 1 обеспечил полную объектную модель для всего [HTML](https://ru.wikipedia.org/wiki/HTML)- или [XML](https://ru.wikipedia.org/wiki/XML)-документа, включая способ изменения любой части документа. Неадаптированные браузеры (например: [«Internet Explorer 4.x»](https://ru.wikipedia.org/wiki/Internet_Explorer_4), «[Netscape](https://ru.wikipedia.org/wiki/Netscape_Navigator)» 4.x) широко использовались вплоть до [2000](https://ru.wikipedia.org/wiki/2000_год) года.

DOM уровня 2 был опубликован в конце [2000](https://ru.wikipedia.org/wiki/2000_год) года. Он ввёл функцию «getElementById», а также модель событий и поддержку «XML namespace» и «[CSS](https://ru.wikipedia.org/wiki/CSS)». DOM уровня 3, опубликованная в апреле [2004](https://ru.wikipedia.org/wiki/2004_год) года, — добавила поддержку «[XPath](https://ru.wikipedia.org/wiki/XPath)» и обработку [событий](https://ru.wikipedia.org/wiki/Событийно-ориентированное_программирование) [клавиатуры](https://ru.wikipedia.org/wiki/Клавиатура), а также [интерфейс](https://ru.wikipedia.org/wiki/Интерфейс) для [сериализации](https://ru.wikipedia.org/wiki/Сериализация) документа как «[XML](https://ru.wikipedia.org/wiki/XML)».

В [2005](https://ru.wikipedia.org/wiki/2005_год) году — большая часть «W3C DOM» поддерживалась основными [браузерами](https://ru.wikipedia.org/wiki/Браузер), удовлетворяющими «[ECMAScript](https://ru.wikipedia.org/wiki/ECMAScript)», в том числе «[Internet Explorer 6.x](https://ru.wikipedia.org/wiki/Internet_Explorer_6)» ([2001](https://ru.wikipedia.org/wiki/2001_год) год), «[Opera](https://ru.wikipedia.org/wiki/Opera)», «[Safari](https://ru.wikipedia.org/wiki/Safari)» и браузеры, основанные на [веб-движке](https://ru.wikipedia.org/wiki/Браузерный_движок) «[Gecko](https://ru.wikipedia.org/wiki/Gecko)» (в том числе «[Mozilla](https://ru.wikipedia.org/wiki/Mozilla)», «[Firefox](https://ru.wikipedia.org/wiki/Firefox)», «[SeaMonkey](https://ru.wikipedia.org/wiki/SeaMonkey)» и «[Camino](https://ru.wikipedia.org/wiki/Camino)»).

**Реализация DOM в веб-браузерах**

Учитывая существование различных реализаций DOM в [веб](https://ru.wikipedia.org/wiki/Всемирная_паутина)-[браузерах](https://ru.wikipedia.org/wiki/Браузер) — среди [программистов](https://ru.wikipedia.org/wiki/Программист) распространена привычка сперва проверять работоспособность тех или иных возможностей DOM для каждого конкретного [браузера](https://ru.wikipedia.org/wiki/Браузер) и только потом использовать их. Код ниже иллюстрирует способ проверки на поддержку стандартов «W3C DOM» — перед тем, как запустить [код](https://ru.wikipedia.org/wiki/Программный_код), зависящий от результата этой проверки.

if (document.getElementById && document.getElementsByTagName) {

// если методы getElementById и getElementsByTagName

// доступны, то можно относительно точно предположить поддержку W3CDOM.

obj = document.getElementById("navigation")

// далее идёт другой код с использованием возможностей W3CDOM.

// ….

}

Ещё один фрагмент [кода](https://ru.wikipedia.org/wiki/Программный_код), написанного на языке «[JavaScript](https://ru.wikipedia.org/wiki/JavaScript)» и позволяющего проверить заявленную поддержку различных расширений DOM в конкретном [веб](https://ru.wikipedia.org/wiki/Всемирная_паутина)-[браузере](https://ru.wikipedia.org/wiki/Браузер).

function domImplementationTest(){

var featureArray = ['HTML', 'XML', 'Core', 'Views',

'StyleSheets', 'CSS', 'CSS2', 'Events',

'UIEvents', 'MouseEvents', 'HTMLEvents',

'MutationEvents', 'Range', 'Traversal'];

var versionArray = ['1.0', '2.0', '3.0'];

var i;

var j;

if(document.implementation && document.implementation.hasFeature){

for(i=0; i < featureArray.length; i++){

for(j=0; j < versionArray.length; j++){

document.write(

'Поддержка расширения '+ featureArray[i] + ' версии ' + versionArray[j] + ': ' +

(document.implementation.hasFeature(featureArray[i], versionArray[j]) ?

'<font style="color:green">true</font>': '<font style="color:red">false</font>') + '<br/>'

);

}

document.write('<br/>');

}

}

}

Так как DOM поддерживает навигацию в любой области (например, родитель и предыдущий брат) и учитывает произвольные изменения — реализация должна по крайней мере буферизировать документ, который (или некоторая разработанная форма которого) читался до этого времени.

**Структура документа и навигация**

**DOM** (Document Object Model) - объектная модель, используемая для XML/HTML-документов. DOM представляет документ в виде дерева, элементами которого являются:

* теги, 2) текстовые фрагменты, 3) атрибуты тегов.

С помощью DOM элементы страницы можно динамически изменить.

**Нормализованный DOM** не содержит «пустые» текстовые элементы, состоящие только из пробелов.

|  |  |
| --- | --- |
| 1 | <html> |

|  |  |
| --- | --- |
| 2 | <head> |

|  |  |
| --- | --- |
| 3 | <title>Заголовок</title> |

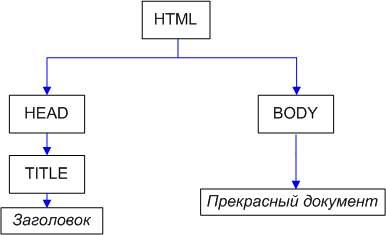
|  |  |
| --- | --- |
| 4 | </head> |

|  |  |
| --- | --- |
| 5 | <body> |

|  |  |
| --- | --- |
| 6 | Прекрасный документ |

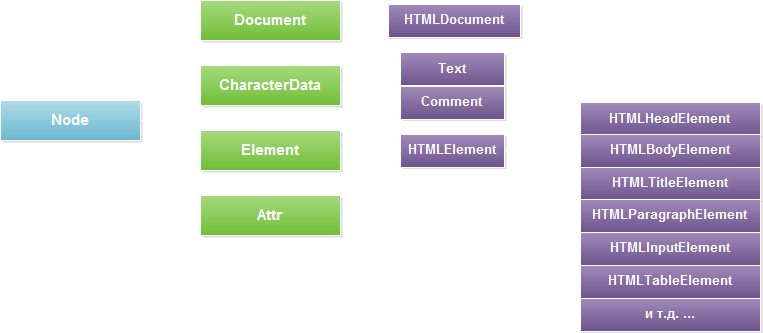
|  |  |
| --- | --- |
| 7 | </body> |

|  |  |
| --- | --- |
| 8 | </html> |



Объект document содержит в себе все дерево элементов страницы.

Каждый элемент дерева является объектом Node и содержит набор свойств и методов.



Иерархия классов для Node

**Свойства класса Node**

**childNodes** - массив с дочерними объектами (NodeList)

**firstChild, lastChild** - отдельно имеются ссылки на первого и последнего потомка.

null, если данный узел не имеет дочерних узлов.

**Илюстрация отношений между элементами страницы**

|  |  |
| --- | --- |
| 01 | <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> |

|  |  |
| --- | --- |
| 02 | <html> |

|  |  |
| --- | --- |
| 03 | <head><title>...</title></head> |

|  |  |
| --- | --- |
| 04 | <body> |

|  |  |
| --- | --- |
| 05 | <div id="dataKeeper">Циклы</div> |

|  |  |
| --- | --- |
| 06 | <ul> |

|  |  |
| --- | --- |
| 07 | <li style="background-color:red">Цикл for</li> |

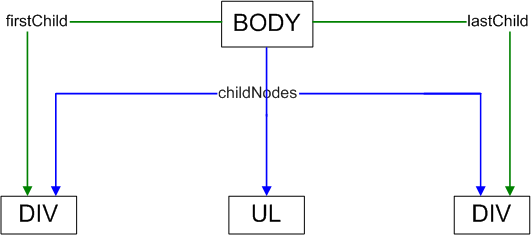
|  |  |
| --- | --- |
| 08 | <li class="info">Цикл while</li> |

|  |  |
| --- | --- |
| 09 | </ul> |

|  |  |
| --- | --- |
| 10 | <div id="footer">урок javascript</div> |

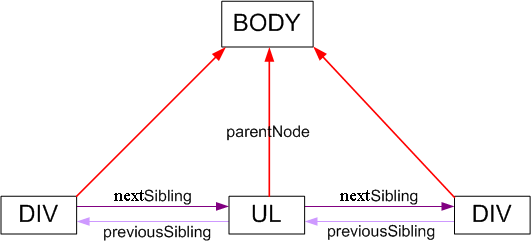
|  |  |
| --- | --- |
| 11 | </body> |

|  |  |
| --- | --- |
| 12 | </html> |



**parentNode -** родительский узел данного узла или null для узлов, не имеющих родителя, таких как Document.

**nextSibling, previousSibling** - следующий и предыдущий братские узлы. Братскими называются два узла, имеющие одного и того же родителя. Порядок их следования соответствует порядку следования в документе. Эти свойства связывают узлы в двусвязный список.



**nodeType**

Тип данного узла. Узлы типа Document имеют значение 9 в этом свойстве. Узлы типа Element - значение 1. Текстовые узлы типа Text - значение 3. Узлы типа Comments - значение 8 и узлы типа DocumentFragment - значение 11.

**nodeValue**

Текстовое содержимое узлов Text и Comment.

**nodeName**

Имя тега элемента Element, в котором все символы преобразованы в верхний регистр.

С помощью этих свойств класса Node можно сослаться на второй дочерний узел первого дочернего узла объекта Document, как показано ниже:

document.childNodes[0].childNodes[1] ==

document.firstChild.firstChild.nextSibling

Допустим, что рассматриваемый документ имеет следующий вид:

<html><head><title>Test</title></head><body>Hello World!</body></html>

Тогда вторым дочерним узлом первого дочернего узла будет элемент <body>. В свойстве nodeType он содержит значение 1 и в свойстве nodeName - значение «BODY».

Однако, обратите внимание, что этот прикладной интерфейс чрезвычайно чувствителен к изменениям в тексте документа. Например, если в этот документ добавить единственный перевод строки между тегами <html> и <head>, этот символ перевода строки станет первым дочерним узлом (текстовым узлом Text) первого дочернего узла, а вторым дочерним узлом станет элемент <head>, а не <body>.

**Свойства объектов Element. Работа с тегами и игнорирование текста**

Когда основной интерес представляют сами элементы документа, а не текст в них (и пробельные символы между ними), гораздо удобнее интерпретировать документ как дерево объектов Element, игнорируя узлы Text и Comment.

Перечисленные ниже свойства возвращают только вершины типа Element.

**children** - возвращает список дочерних вершин NodeList.

Обратите внимание, что узлы Text и Comment не имеют дочерних узлов. Это означает, что описанное выше свойство Node.parentNode никогда не возвращает узлы типа Text или Comment. Значением свойства parentNode любого объекта Element всегда будет другой объект Element или корень дерева - объект Document или DocumentFragment.

**firstElementChild, lastElementChild** – первый и последний потомок.

**nextElementSibling, previousElementSibling** - соседние элементы в списке потомков.

**childElementCount** - количество дочерних элементов. Возвращает то же значение, что и свойство children.length.

Эти свойства доступа к дочерним и братским элементам стандартизованы и реализованы во всех текущих браузерах, кроме IE.

Структура данных – конструкция, в которой хранится информация. В питоне: список, строка, словарь и кортеж

В лабораторной используем составные структуры данных:

список, элементами которого являются списки [[], [], [], …]

список, элементами которого являются словари [{}, {}, {}, …]

словарь, элементами которого являются словари {{}, {}, {}, …}

**список из списков**: список, элементами которого являются списки

[[], [], [], …]

**Формирование списка из простых данных**

f\_list = [] создание списка

f\_list.append (10) метод списка append добавляет новый элемент в конец списка

f\_list.append (20); f\_list.append (30); f\_list.append (40);

получим: f\_list: [10,20,30,40] список из целых чисел

**Формирование списка из списков**

f2\_list =[]

f2\_list.append ([1,2,3]); f2\_list.append ([10,20,30]); f2\_list.append ([100]);

получим: f2\_list: [ [1,2,3], [10,20,30], [100] ] список из списков

как обращаться к элементам списка из списков?

f2\_list[0] : это первый элемент списка f2\_list [1,2,3]

f2\_list[0][0] : это первый элемент первого элемента списка f2\_list 1

обращение к элементу в списке из списков выполняется через 2 индекса.

**Список из словарей**: список, элементами которого являются словари

f3\_list =[]

f3\_list.append ({1:”весна”}); f3\_list.append ({2:”лето”})

f3\_list.append ({3:”осень”})

получим: f3\_list: [ {1:”весна”}, {2:”лето”}, {3:”осень”} ]

обращение к элементу: имясписка [индекс по списку][ключ словаря]

f3\_list [0][2]

**Словарь из словарей**: словарь, элементами которого являются словари

Dic1 = {} создать пустой словарь

Dic1 [1] = ”весна”

Dic1 [2] = ”summer”

Имя\_словаря [ключ] = значение

получим: Dic1: {1:”весна”, 2:”summer”}

**словарь из словарей**

smalldic = {1: ‘март’, 2: ‘апрель’}

Dic1[3] = smalldic

получим: Dic1: {1:”весна”, 2:”summer”, 3: {1: ‘март’, 2: ‘апрель’} }

dic2 ={}

dic2[‘весна’] = smalldic

получим: dic2: {‘весна’: {1: ‘март’, 2: ‘апрель’} }

ключ‘весна’, элемент {1: ‘март’, 2: ‘апрель’}

dic2[‘лето’] = {1:’июнь’, 2:’июль’}

получим: dic2: {‘весна’: {1: ‘март’, 2: ‘апрель’}, ‘лето’: {1:’июнь’, 2:’июль’} }

обращение к элементу: имясловаря [ключ словаря]

dic2[‘лето’] : получим {1:’июнь’, 2:’июль’}

имясловаря [ключ внешнего словаря] [ключ внутреннего словаря]

dic2[‘лето’][2] : получим ’июль’

Методические указания для выполнения

лабораторной работы № 1 по курсу «Обработка текстовых массивов»

весенний семестр

**«Использование в программе структур данных разного типа»**

**Цель работы**: получение опыта разработки программы на языке питон с использованием составных структур данных.

**Задание**

Напишите программу, которая выполняет следующие действия:

* Вводит данные с клавиатуры и сохраняет их в структуре данных по варианту
* Выводит на экран все введенные значения в виде таблицы с заголовком
* Выполняет поиск данных по условию.

Исходный текст программы следует адаптировать под свой вариант, поменяв содержание полей таблицы и соответствующий код программы. Задание на поиск студент формирует самостоятельно. Во всех примерах используется таблица «Студент» с полями: Фамилия, Имя, Список оценок.

**Таблица 1. Варианты заданий для программы**

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Структура данных** | **Название таблицы** |
| 1 | Список из словарей | Поезд |
| 2 | Словарь из словарей | Языковой словарь |
| 3 | Список из списков | Писатель |
| 4 | Список из словарей | Город |
| 5 | Словарь из словарей | Море |
| 6 | Список из списков | Языковая группа |
| 7 | Список из словарей | Языковой словарь |
| 8 | Словарь из словарей | Писатель |
| 9 | Список из списков | Город |
| 10 | Список из словарей | Море |
| 11 | Словарь из словарей | Языковая группа |
| 12 | Список из списков | Поезд |
| 13 | Список из словарей | Здание |
| 14 | Словарь из словарей | Здание |
| 15 | Список из списков | Здание |

**Поля данных в таблицах**

«Поезд»: Название, Пункт отправления, Список остановок

«Языковой словарь»: Слово, Перевод, Список синонимов

«Писатель»: Фамилия, Имя, Список сочинений

«Город»: Название, численность жителей, Список пригородов

«Море»: Название, площадь, Впадающие реки

«Языковая группа»: Название, число человек, говорящих на языках этой группы, Язык

«Здание»: Адрес, Год постройки, список фамилий жильцов.

**Содержание отчета**

* Титульный лист
* Вариант задания и словесная формулировка условия поиска
* Перечисление всех переменных программы с указанием типа данных и назначения
* Текст программы
* Копии экрана с примерами работы

**Примеры программ**

**Программа № 1. Структура данных: словарь из словарей**

A = {}

ans = int (input ('сколько данных вводить?'))

for x in range (ans):

name = input ('Имя: ')

fam = input('Фамилия: ')

x =[] # список с оценками

for i in range (3):

print ('Оценка за курс % d' % i, end =' ')

a = int (input ())

x.append(a)

stud = {} # словарь

stud ['name'] =name

stud ['ball']= x

A[fam]=stud

#вывод таблицы на экран

print ('Фамилия Имя Баллы')

for el in A.keys() :

print ('%-10s %-10s'% (el, A[el]['name']), A[el]['ball'])

#поиск по условию

#сколько студентов с пятерками

count =0

for el in A.keys():

if 5 in A[el]['ball']:

count +=1

print ('Всего студентов с пятерками = ', count)

Часть 1. Ввод данных. Заполнение словаря

A = {}

ans = int (input ('сколько данных вводить?'))

for x in range (ans):

name = input ('Имя: ')

fam = input('Фамилия: ')

x =[] # список с оценками

for i in range (3):

print ('Оценка за курс % d' % i, end =' ')

a = int (input ())

x.append(a)

stud = {} # словарь

stud ['name'] =name

stud ['ball']= x

A[fam]=stud

A: {‘Бу’: {‘name’: ‘Хаосян’, ‘ball’ : [5, 5, 5] }, ‘Дэн’: {‘name’: ‘Шиянь’, ‘ball’ : [5, 5, 5] }}

В словаре А 2 элемента с ключами ‘Бу’ и ‘Дэн’.

Обращение к данным:

А[‘Бу’] получу: {‘name’: ‘Хаосян’, ‘ball’ : [5, 5, 5] }, строка таблицы

А[‘Бу’][‘name’] получу: ‘Хаосян’ , элемент строки

#вывод таблицы на экран

print ('Фамилия Имя Баллы')

for el in A.keys() : цикл по всему словарю

print ('%-10s %-10s'% (el, A[el]['name']), A[el]['ball'])

A.keys() - .keys()- метод словаря, дает список всех ключей словаря,

for el in A.keys() : el – это ключ словаря, фамилия

el – ключ = фамилия, A[el]['name'] - имя , A[el]['ball'] – список с оценками

#поиск по условию

#сколько студентов с пятерками

count =0

for el in A.keys():

if 5 in A[el]['ball']:

count +=1

print ('Всего студентов с пятерками = ', count)

**Программа № 2. Структура данных: список из словарей**

Отличия по сравнению с вариантом «список из списков»:

* При вводе данных формируется словарь (stud)
* Вместо индексов используются ключи

Список из списков: А: [ [“Хаосян”, “Бу”, [5,5,5] ], [“Инь”, “Ци” , [5, 5, 5]] ]

Список из словарей: А: [ {“family”:“Хаосян”, “name”:“Бу”, “ball”:[5,5,5] }, {“family”:“Инь”, “name”:“Ци” , “ball”: [5, 5, 5]} ]

A = []

ans = int (input ('сколько данных вводить?'))

for x in range (ans):

name = input ('Имя: ')

fam = input('Фамилия: ')

x =[] # список с оценками

for i in range (3):

print ('Оценка за курс % d' % i, end =' ')

a = int (input ())

x.append(a)

stud = {} # словарь

stud ['family']=fam

stud ['name'] =name

stud ['ball']= x

A.append(stud)

#вывод таблицы на экран

print ('Фамилия Имя Баллы')

for el in A :

print ('%-10s %-10s'% (el['family'], el ['name']), el['ball'])

#поиск по условию

#сколько студентов с пятерками

count =0

for el in A:

if 5 in el['ball']:

count +=1

print ('Всего студентов с пятерками = ', count)

**Программа № 3. Структура данных: список из списков**

**Части программы**

* Ввод данных и формирование списка А
* Вывод на экран
* Задание на поиск

Структура данных программы:

Основной список А, каждый элемент списка А является списком с 3 элементами: имя, Фамилия и список с оценками

А: [ [“Хаосян”, “Бу”, [5,5,5] ], [“Инь”, “Ци” , [5, 5, 5]], [] ]

A = []

ans = int (input ('сколько данных вводить?'))

ans - переменная, которая хранит ответ пользователя. Ответ – это целое число

input ('сколько данных вводить?') input - функция для ввода данных с клавиатуры. Фраза 'сколько данных вводить? Будет выведена на экран

int () функция преобразует значение в скобках в целое число

for x in range (**ans**):

цикл ввода данных (записей). Количество записей задано переменной ans. конец цикла - A.append(stud)

name = input ('Имя: ')

fam = input('Фамилия: ')

переменные name, fam - содержат имя и фамилию

x2 =[] # список с оценками

for i in range (3):

print ('Оценка за курс % d' % i, end =' ')

a = int (input ())

x2.append (a)

stud = [] # список с одной записью

формируем одну строку таблицы

stud .append(fam)

stud .append(name)

stud .append( x2)

A.append(stud)

Добавляем строку в таблицу

**Вывод на экран данных в таблице**

print ('Фамилия Имя Баллы')

печать заголовка

for el in A :

цикл по списку А, el – переменная, которая перебирает все элементы списка А

print ('%-10s %-10s'% (el[0], el [1]), el[2])

вывод на экран с форматированием

print ( '%-10s %-10s'% (el[0], el [1]), el[2])

print ('%фамилия %имя '% (el[0], el [1]), список оценок без форматирования)

print ('строка для вывода %-10s' % ( переменные, которые надо вывести) )

А: [ [“Хаосян”, “Бу”, [5,5,5] ], [“Инь”, “Ци” , [5, 5, 5]], [] ]

el [“Хаосян”, “Бу”, [5,5,5] ]

el [“Инь”, “Ци” , [5, 5, 5]]

**поиск по условию**

задание: подсчитать количество студентов с пятерками

count - переменная для подсчета количества студентов с пятерками

count =0

for el in A:

if 5 in el[2]: если среди оценок есть 5, то увеличить счетчик

count +=1

print ('Всего студентов с пятерками = ', count)

A = []

ans = int (input ('сколько данных вводить?'))

for x in range (ans):

name = input ('Имя: ')

fam = input('Фамилия: ')

x =[] # список с оценками

for i in range (3):

print ('Оценка за курс % d' % i, end =' ')

a = int (input ())

x.append (a)

stud = [] # список

stud .append(fam)

stud .append(name)

stud .append( x)

A.append(stud)

#вывод таблицы на экран

print ('Фамилия Имя Баллы')

for el in A :

print ('%-10s %-10s'% (el[0], el [1]), el[2])

#поиск по условию

#сколько студентов с пятерками

count =0

for el in A:

if 5 in el[2]:

count +=1

print ('Всего студентов с пятерками = ', count)

пример выполнения работы

вариант 15, список из списков,

«Здание»: Адрес, Год постройки, список фамилий жильцов.

В примере тема «Студент», поля: Имя, Фамилия, список оценок

поля: Имя ( строка), Фамилия (строка), список оценок (список целых чисел)

Адрес (строка), Год постройки (целое число), список фамилий жильцов (список строк).

A = []

ans = int (input ('сколько данных вводить?'))

for x in range (ans):

name = input ('Имя: ')

fam = input('Фамилия: ')

Переделки: input возвращает строку, а нужно целое число

yearbild = int(input('Год постройки: '))

x =[] # список с оценками

for i in range (3):

print ('Оценка за курс % d' % i, end =' ')

a = int (input ())

Переделки: a = input ()

x.append (a)

stud = [] # список

stud .append(fam)

stud .append(name)

stud .append( x)

A.append(stud)

#вывод таблицы на экран

print ('Фамилия Имя Баллы')

for el in A :

print ('%-10s %-10s'% (el[0], el [1]), el[2])

#поиск по условию

#сколько студентов с пятерками

count =0

for el in A:

if 5 in el[2]:

count +=1

print ('Всего студентов с пятерками = ', count)

«Поезд»: Название, Пункт отправления, Список остановок

Таблица «поезд»

|  |  |  |
| --- | --- | --- |
| Название | Пункт отправления | Список остановок |
| Красная стрела | Санкт-Петербург | Бологое, Акуловка, Тверь, Москва |
| Северный | Мурманск | Архангельск, Петрозаводск, Санкт-Петербург |
|  |  |  |

«Языковой словарь»: Слово, Перевод, Список синонимов

|  |  |  |
| --- | --- | --- |
| Слово | Перевод | Список синонимов |
| pen | карандаш | Ручка, письменный прибор, палочка |
|  |  |  |

Распределение вариантов для выполнения лабораторной работы № 1 по курсу «Обработка текстовых массивов». Группа **6933к**

|  |  |
| --- | --- |
| Вариант из таблицы 1 | ФИО |
| 1 | Бычкова |
| 2 | Величко |
| 3 | Кабанова |
| 4 | Киселева |
| 5 | Лошак |
| 6 | Милованова |
| 7 | Песля |
| 8 | Чалганская |
| 9 | Шестаков |
| 10 | Шульман |
| 11 | Эгембердиева |
|  |  |
|  |  |

## https://professorweb.ru/my/javascript/js\_theory/level2/2\_3.php

## Выбор элементов документа

Работа большинства клиентских программ на языке JavaScript так или иначе связана с манипулированием элементами документа. В ходе выполнения эти программы могут использовать глобальную переменную document, ссылающуюся на объект Document. Однако, чтобы выполнить какие-либо манипуляции с элементами документа, программа должна каким-то образом получить, или выбрать, объекты Element, ссылающиеся на эти элементы документа. Модель DOM определяет несколько способов выборки элементов.

Для выбора элемента используют методы объекта document.

Выбрать элемент или элементы документа можно:

* по значению атрибута id; **getElementById()**
* по значению атрибута name; **getElementsByName()**
* по имени тега;  **getElementsByTagName()**
* по имени класса или классов CSS; **getElementsByClassName()**

**Выбор элементов по значению атрибута id**

Все HTML-элементы имеют атрибуты id. Значение этого атрибута должно быть уникальным в пределах документа - никакие два элемента в одном и том же документе не должны иметь одинаковые значения атрибута id. Выбрать элемент по уникальному значению атрибута id можно с помощью метода **getElementById()** объекта Document:

var section1 = document.getElementById("section1");

Это самый простой и самый распространенный способ выборки элементов. Если сценарию необходимо иметь возможность манипулировать каким-то определенным множеством элементов документа, присвойте значения атрибутам id этих элементов и используйте возможность их поиска по этим значениям.

В версиях Internet Explorer ниже IE8 метод getElementById() выполняет поиск значений атрибутов id без учета регистра символов и, кроме того, возвращает элементы, в которых будет найдено совпадение со значением атрибута name.

**Выбор элементов по значению атрибута name**

HTML-атрибут name первоначально предназначался для присваивания имен элементам форм, и значение этого атрибута использовалось, когда выполнялась отправка данных формы на сервер. Подобно атрибуту id, атрибут name присваивает имя элементу. Однако, в отличие от id, значение атрибута name не обязано быть уникальным: одно и то же имя могут иметь сразу несколько элементов, что вполне обычно при использовании в формах радиокнопок и флажков. Кроме того, в отличие от id, атрибут name допускается указывать лишь в некоторых HTML-элементах, включая формы, элементы форм и элементы <iframe> и <img>.

Выбрать HTML-элементы, опираясь на значения их атрибутов name, можно с помощью метода **getElementsByName()** объекта Document:

var radiobuttons = document.getElementsByName("favorite\_color");

Метод getElementsByName() определяется не классом Document, а классом HTMLDocument, поэтому он доступен только в HTML-документах и не доступен в XML-документах. Он возвращает объект **NodeList**, который ведет себя, как доступный только для чтения массив объектов Element.

В IE метод getElementsByName() возвращает также элементы, значения атрибутов id которых совпадает с указанным значением. Чтобы обеспечить совместимость с разными версиями браузеров, необходимо внимательно подходить к выбору значений атрибутов и не использовать одни и те же строки в качестве значений атрибутов name и id.

**Выбор элементов по типу**

Метод **getElementsByTagName()** объекта Document позволяет выбрать все HTML или XML-элементы указанного типа (или по имени тега). Например, получить подобный массиву объект, доступный только для чтения, содержащий объекты Element всех элементов <span> в документе, можно следующим образом:

var spans = document.getElementsByTagName("span");

Подобно методу getElementsByName(), getElementsByTagName() возвращает объект NodeList. Элементы документа включаются в массив NodeList в том же порядке, в каком они следуют в документе, т.е. первый элемент <p> в документе можно выбрать так:

var firstParagraph = document.getElementsByTagName("p")[0];

Имена HTML-тегов не чувствительны к регистру символов, и когда getElementsByTagName() применяется к HTML-документу, он выполняет сравнение с именем тега без учета регистра символов. Переменная spans, созданная выше, например, будет включать также все элементы <span>, которые записаны как <SPAN>.

Можно получить NodeList, содержащий все элементы документа, если передать методу getElementsByTagName() шаблонный символ «\*».

Кроме того, классом Element также определяет метод getElementsByTagName(). Он действует точно так же, как и версия метода в классе Document, но выбирает только элементы, являющиеся потомками для элемента, относительно которого вызывается метод. То есть отыскать все элементы <span> внутри первого элемента <p> можно следующим образом:

var firstParagraph = document.getElementsByTagName("p")[0];

var firstParagraphSpans = firstParagraph.getElementsByTagName("span");

По историческим причинам класс HTMLDocument определяет специальные свойства для доступа к узлам определенных типов. Свойства *images*, *forms* и *links*, например, ссылаются на объекты, которые ведут себя как массивы, доступные только для чтения, содержащие элементы <img>, <form> и <a> (но только те теги <a>, которые имеют атрибут href). Эти свойства ссылаются на объекты HTMLCollection, которые во многом похожи на объекты NodeList, но дополнительно могут индексироваться значениями атрибутов id и name.

Объект HTMLDocument также определяет свойства-синонимы **embeds** и **plugins**, являющиеся коллекциями HTMLCollection элементов <embed>. Свойство **anchors** является нестандартным, но с его помощью можно получить доступ к элементам <a>, имеющим атрибут name, но не имеющим атрибут href. Свойство **scripts** определено стандартом HTML5 и является коллекцией HTMLCollection элементов <script>.

Кроме того, объект HTMLDocument определяет два свойства, каждое из которых ссылается не на коллекцию, а на единственный элемент. Свойство **document.body** представляет элемент <body> HTML-документа, а свойство **document.head** - элемент <head>. Эти свойства всегда определены в документе: даже если в исходном документе отсутствуют элементы <head> и <body>, браузер создаст их неявно. Свойство **documentElement** объекта Document ссылается на корневой элемент документа. В HTML-документах он всегда представляет элемент <html>.

**Выбор элементов по классу CSS**

Значением HTML-атрибута class является список из нуля или более идентификаторов, разделенных пробелами. Он дает возможность определять множества связанных элементов документа: любые элементы, имеющие в атрибуте class один и тот же идентификатор, являются частью одного множества. Слово class зарезервировано в языке JavaScript, поэтому для хранения значения HTML-атрибута class в клиентском JavaScript используется свойство className.

Обычно атрибут class используется вместе с каскадными таблицами стилей CSS, с целью применить общий стиль отображения ко всем членам множества. Однако кроме этого, стандарт HTML5 определяет метод **getElementsByClassName()**, позволяющий выбирать множества элементов документа на основе идентификаторов в их атрибутах class.

Подобно методу getElementsByTagName(), метод getElementsByClassName() может вызываться и для HTML-документов, и для HTML-элементов, и возвращает «живой» объект NodeList, содержащий все потомки документа или элемента, соответствующие критерию поиска.

Метод getElementsByClassName() принимает единственный строковый аргумент, но в самой строке может быть указано несколько идентификаторов, разделенных пробелами. Соответствующими будут считаться все элементы, атрибуты class которых содержат все указанные идентификаторы. Порядок следования идентификаторов не имеет значения. Обратите внимание, что и в атрибуте class, и в аргументе метода getElementsByClassName() идентификаторы классов разделяются пробелами, а не запятыми.

Ниже приводится несколько примеров использования метода getElementsByClassName():

// Отыскать все элементы с классом "warning"

var warnings = document.getElementsByClassName("warning");

// Отыскать всех потомков элемента с идентификаторам "log"

// с классами "error" и "fatal"

var log = document.getElementById("log");

var fatal = log.getElementsByClassName("fatal error");

**Лекция 3**

**CSS – Cascading Stylesheets – каскадные таблицы стилей**

План

**Формат правила**

**Способы включения в html: атрибут style, тег style , через link подключение отдельного файла**

**Селекторы: тег, класс, идентификатор, составной**

**Задание размера**

**Задание цвета**

**Структура элемента (поля и отступы)**

**Формат правила**

CSS используются для оформления веб-страниц при помощи правил CSS.

Правило CSS состоит из:

- левой части – селектора (от слова select – англ. «выбирать»)

- правой части – свойств оформления

Список\_селекторов {параметр : значение; параметр : значение}

Пример правила CSS:

div {color: red}

Здесь div – селектор, color: red – свойство оформления, задающее цвет шрифта красным.

При помощи селекторов указывается, к каким тегам применять определенные свойства оформления. В примере выше, красный цвет шрифта будет применяться только к тегам div.

Некоторые свойства оформления:

color – цвет шрифта

font-weight – выделение шрифта (нормальный, жирный)

font-size – размер шрифта

background-color – цвет фона

background-image – путь к фоновому изображению

text-align – расположение содержимого по горизонтали

**Способы включения в html: атрибут style, тег style , через link подключение отдельного файла**

1.. атрибут style внутри тега. Высший приоритет.

<h1 style = “color : navy”> Заголовок</h1>

2.. тег style. Второй приоритет

рекомендуется располагать внутри раздела head.

<style type = “text/css”>

body {

background: #FFF;

color: #000;

font-family: Arial, sans-serif;

font-size: 14px;

}

</style>

3.. через link подключение отдельного файла с правилами, расширение .css. Третий приоритет.

Внутри head

<link rel="stylesheet" type="text/css" href="style.css">

**Селекторы: тег, класс, идентификатор, составной**

- \* Все элементы (тег)

\* {background-color: blue;}

- тега

Задает оформление для всех элементов с указанным тегом

div {color: red; text-align: right;}

p {background-color: blue;}

- класса

Задает оформление для определенного класса тегов, перечисленных в левой части правила. При этом перед названием класса обязательно ставится символ точки. Несколько элементов могут иметь одинаковый класс.

.myclass {font-size: 14px; font-weight: bold; background-image:url(‘my.jpg’)}

В HTML-коде класс указываетcя у тега через атрибут class:

<div class=”myclass”>Hello world!!!</div>

<img class=”myclass” src=”my.jpg”>

- идентификатора

Идентификатор задается в теге через атрибут id. Идентификатор должен быть уникальным. Также применяется в скриптах для указания элемента. Перед именем в правиле ставится знак #.

#myid {font-size: 14px;}

<div id = “myid” >

- составной селектор

селектор.класс div.small

селектор#идентификатор div#myid

- контекстный селектор

Учитывает расположение элементов в тексте html относительно друг друга.

Потомок – элемент, который вложен в другой элемент. Предок - элемент в котором есть вложения других элементов. Родитель – находится на один уровень выше и связан с данным элементом. Дочерний элемент – находится на один уровень ниже и связан с данным элементом.

Селектор1 Селектор2 селекторы подряд через пробел.

Селектор1 – предок для Селектора2.

li a {color : red} красные ссылки внутри списка

Селектор1 > Селектор2 Селектор1 – родитель. Селектор 2 – дочерний элемент.

Селектор1 + Селектор2 селекторы элементов-соседей (теги идут подряд). Селектор2 идет сразу после Селектор1.

Селектор [атрибут] все теги, где есть такой атрибут

**Группировка селекторов**

1.. один стиль на несколько тегов:

Вместо:

div {color: red}

p { color: red}

пишем:

div, p {color: red}

2.. Группировка атрибутов по типам:

Вместо:

p {font-size: 12pt; font-weight: bold; font-family: courier}

пишем:

p {font: 12pt bold courier}

Псевдоселекторы



Пример. Задание разного цвета для ссылок с разным статусом.

a:link {color : red}

a:visited {color : blue}

a:hover {color : black}

**Задание размера**

1.. Абсолютные – не зависят от устройства отображения: cm - сантиметры, mm - миллиметры, in – дюймы, 1in = 2.54см; pt – пункты 1pt=1/72 in, pc – пики 1 pc = 12 pt

2.. Относительные – размер определяется относительно другого элемента.

em – размер родительского шрифта, ex – высота буквы x родительского шрифта,

%

3..Пикселы – размер зависит от устройства, но не зависит от размера других элементов, px

**Задание цвета**

1.. Английское название ()

rgb = red + green + blue (красный+зеленый+синий)

Самый темный цвет - 0 или 00, самый яркий - 255 или ff

Черный #000000, белый #ffffff

Оттенки серого: одинаковые числа для каждого цвета: #343434

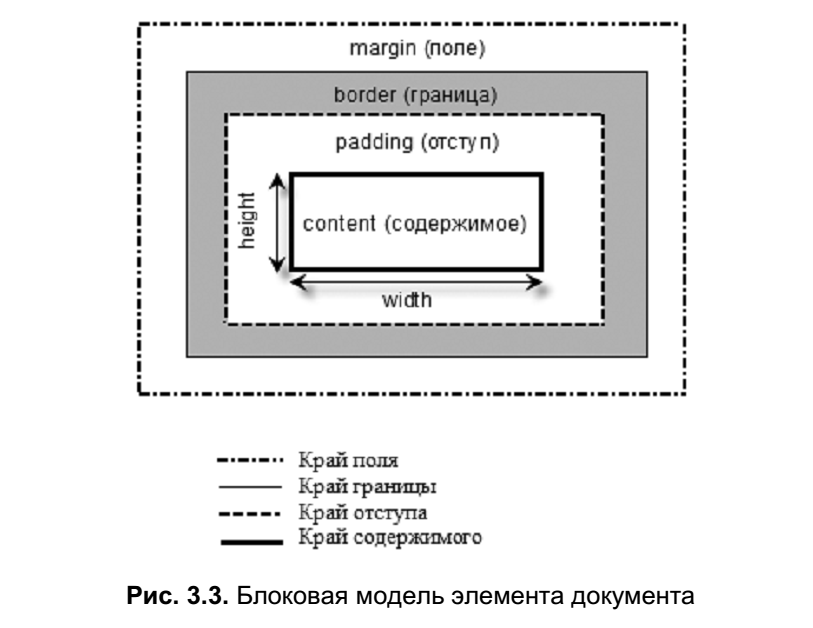
2.. 16-ричный формат rgb. Цвет кодируется 3 16-ричными числами. На каждое число 2 символа. В начале ставится знак #. Примеры: #75C3DD. Сокращенная запись, если пара символов одинакова: #77ccdd = #7cd

3.. 10-тичный формат rgb (125, 7, 7). В скобках три десятичных числа. Значения каждого от 0 до 255 (один байт)

4.. rgb в процентах rgb (10%, 70%, 20%)



**Структура элемента (поля и отступы)**



Каждое поле имеет 4 стороны, свойства для которых можно задавать отдельно: top - верх, bottom - низ, left - лево, right – право

margin-top, margin-right, margin-bottom, margin-left

margin: 10px 20px 10px 20px

**Примеры программ на javascript**

**Пример 1. Простая работа с массивом**

//это комментарий

//Задача: найти максимум

b = new Array (3, 5, 12, 3); //создаем новый массив и инициализируем его

max = b[0]; // создаем переменную max и присваиваем ей первый элемент массива

for (i=1; i< b.length; i++) //цикл по массиву начиная со второго элемента

if (b[i] > max)

max = b[i];

alert(max) // это функция выводит окно со значением переменной max

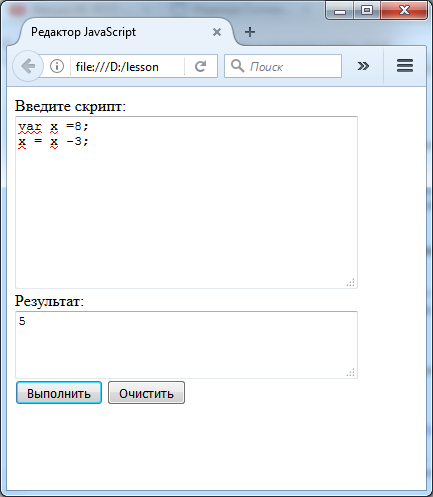
max // чтобы результат отобразился в поле myrezult можно просто написать имя //переменной

**Пример 2. Редактор javascript**

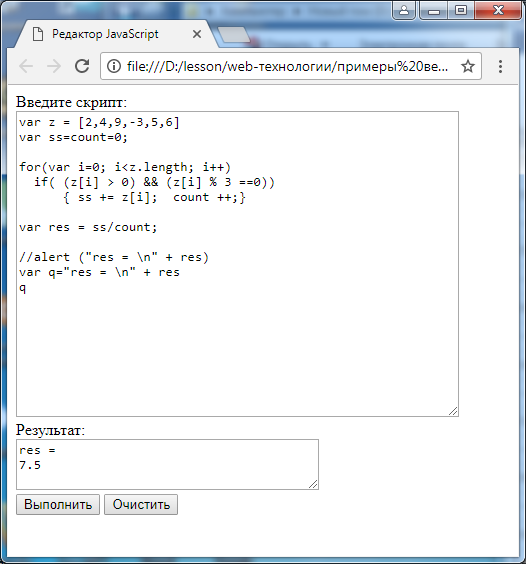
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html>  
<head><title>Редактор JavaScript</title></head>  
<body> Введите скрипт:<br/>  
<textarea id="mycode" rows=10 cols=40></textarea>  
<br/>Результат:<br/>  
<textarea id="myrezult" rows=3 cols=40></textarea> <br/>

<button onclick= "document.getElementById('myrezult').value =  
eval(document.getElementById('mycode').value)">  
Выполнить  
</button>

<button onclick=  
"document.getElementById('mycode').value='';  
document.getElementById('myrezult').value=''"> Очистить  
</ button ></body></html>



**Пример 3. Вычислить среднее арифметическое положительных элементов, кратных трем**

var z = [2,4,6,-3,5,6]

var ss=count=0;

for(var i=0; i<z.length; i++)

if( (z[i] > 0) && (z[i] % 3 ==0))

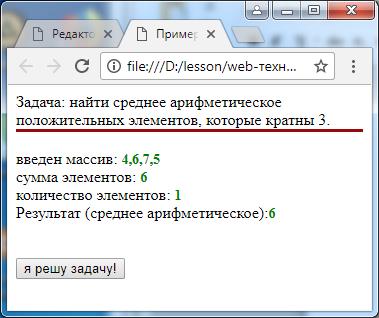
{ ss += z[i]; count ++;}

var res = ss/count;

//alert ("res = \n" + res)

var q="res = \n" + res

q

 **Пример 4. Ввод и вычисления на массиве**

<html> <head> <title>Пример работы с массивом</title>

<script>

function avg() {

strmm = prompt("Введите элементы массива через пробел.");

mm = strmm.split (" ")

el= document.getElementsByTagName("span")

el[0].innerHTML= "<b style='font-size:14;color:green;'>"+mm +"</b>"

var ss=count=0;

for(var i=0; i<mm.length; i++)

{ if( (mm[i] > 0) && (mm[i] % 3 ==0))

{ ss += Number(mm[i]); count ++;}}

var res = ss/count;

el[1].innerHTML= "<b style='font-size:14;color:green;'>"+ss +"</b>"

el[2].innerHTML= "<b style='font-size:14;color:green;'>"+count+"</b>"

el[3].innerHTML= "<b style='font-size:14;color:green;'>"+res +"</b>"

//alert ("ss= "+ss+" count = "+ count+" res =" + res)

} </script>

</head>

<body>

Задача: найти среднее арифметическое положительных элементов, которые кратны 3.<br>

<div style="width: 100%; height: 3px; background-color: #97070B;"></div><br>

введен массив: <span id="input"></span><br> сумма элементов: <span id="input"></span><br>

количество элементов: <span id="input"></span><br>

Результат (среднее арифметическое):<span id="input"></span><br>

<br><br>

<button onclick = "avg()"> я решу задачу!</button>

</body></html>

**Пример 5. Часы**

<html>

<head>

<script type= "text/javascript">

var tmrid;

function bigben (nameel)

{

var datecur, hour, minute;

var time ="";

datecur = new Date();

hour = datecur.getHours();

minute= datecur.getMinutes();

sec = datecur.getSeconds();

time = ((hour <10) ? "0" : "" ) + hour+"\_";

time += ((minute <10) ? "0" : ":\_" ) + minute;

time += ((sec <10) ? "0" : ":\_" ) + sec;

tmr = document.getElementById (nameel);

tmr.innerText = time;

//tmrid = setTimeout ("bigben()", 100);

window.onload=time;

}

bigben ();

</script>

</head>

<body>

<h1> Time now : span:

<span id = "tmr1"> </span><br>It is div:

<div id = "tmr2"> </div>

</h1>

<button onclick = "bigben('tmr1')"> я покажу время span</button>

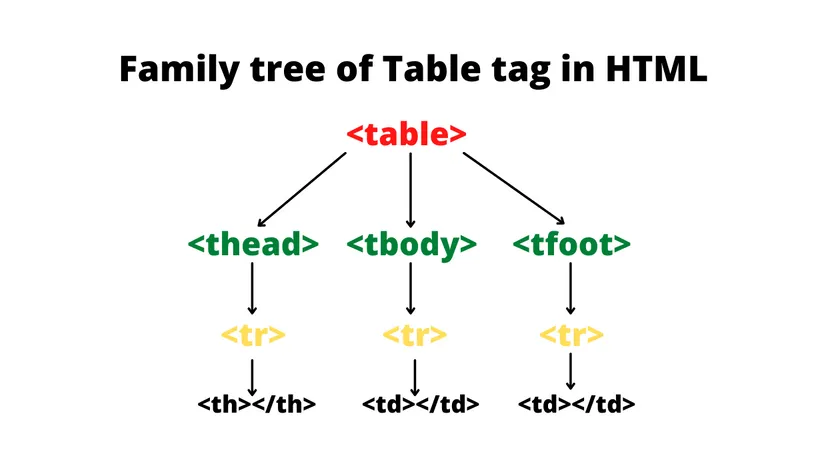
<button onclick = "bigben('tmr2')"> я покажу время div</button>

</body>

</html>

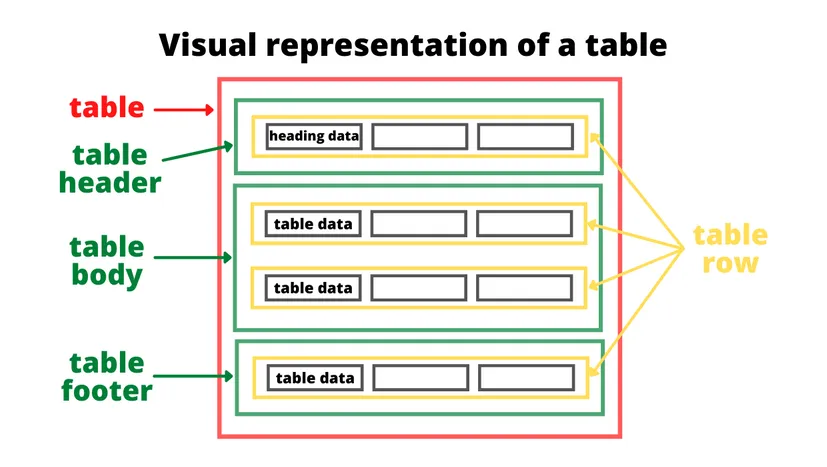
**Работа с таблицами**

По стандарту DOM у таблицы должен быть тег <tbody>, но в HTML можно написать без него. В этом случае браузер добавляет <tbody> в DOM самостоятельно.



Теги для таблицы

* table: HTML-тег table представляет всю таблицу в HTML. Это начальный тег таблицы, внутри которого будут использоваться все остальные теги, относящиеся к таблице (например, thead, tr, td и т. Д.), В зависимости от того, как вы хотите структурировать таблицу.
* thead: thead означает заголовок таблицы. Этот тег используется для добавления заголовка к таблицам. Обычно это первая строка таблицы. Чтобы добавить данные внутрь тега thead, мы используем тег th. Если вы не хотите добавлять заголовок к своим таблицам, вы можете пропустить тег thead.
* tbody: tbody означает тело таблицы. Этот тег представляет все тело или содержимое таблицы.
* tfoot: tfoot обозначает нижний колонтитул таблицы. Обычно это последняя строка таблицы. Этот тег не является обязательным. Мы не так часто используем этот тег.
* tr: tr обозначает строку таблицы. Это представляет собой всю строку таблицы. Чтобы вставить данные в таблицу внутри заголовка, тела или нижнего колонтитула таблицы, мы сначала должны создать строку, а затем внутри этой строки мы можем вставить данные с помощью тега td.
* th: th используется только внутри тега thead. Он представляет собой единственную ячейку строки заголовка. Заголовки таблицы выделены жирным шрифтом.
* td: td обозначает данные таблицы. Он представляет собой единственную ячейку таблицы. Он похож на th, но с той лишь разницей, что th используется только внутри тега thead, а td используется где-либо еще. Тег td можно использовать с тегами tbody и tfoot.

Таблица со всеми возможными элементами

**Пример**

<!DOCTYPE html>

<html >

<head>

<meta charset="UTF-8">

<title>Document</title>

<style>

table{

border-collapse: collapse;

border-spacing: 0;

}

th, td{

padding: 10px 20px;

border: 1px solid #00f;

}

</style>

<script >

let count =0;

function createtable(n,m)

{ let n2 = 3;

let table = document.createElement('table');

let thead = document.createElement('thead');

let tbody = document.createElement('tbody');

table.appendChild(thead);

table.appendChild(tbody);

document.getElementById('body').appendChild(table);

let row\_1 = document.createElement('tr');

for (let j=0; j< m; j++)

{ let heading\_1 = document.createElement('th');

row\_1.appendChild(heading\_1);

}

row\_1.children[0].innerHTML = "Header 1";

row\_1.children[1].innerHTML = "Header 2";

row\_1.children[2].innerHTML = "Header 3";

thead.appendChild(row\_1);

for (let i=0; i< n; i++)

{

row\_1 = document.createElement('tr');

for (let j=0; j< m; j++)

{ let a = document.createElement('td');

a.innerHTML = "Nomer " +i +"\_" + j;

row\_1.appendChild(a);

}

thead.appendChild(row\_1);

}

}

function insertetable(m)

{ let secondtable = document.getElementById("table2");

let row\_1 = document.createElement('tr');

for (let j=0; j< m; j++)

{ let a = document.createElement('td');

a.innerHTML = "Nomer " +count +"\_" + j;

row\_1.appendChild(a);

}

secondtable.appendChild(row\_1);

count ++;

}

</script>

</head>

<body >

<div> Создание таблицы</div><br>

<div id = "body"></div> <br>

<button onclick = "createtable(5,3)">Создать таблицу</button>

<br><br>

<div >

<table id = "table2">

<tr><td></td><td></td><td></td><td></td><td></td></tr>

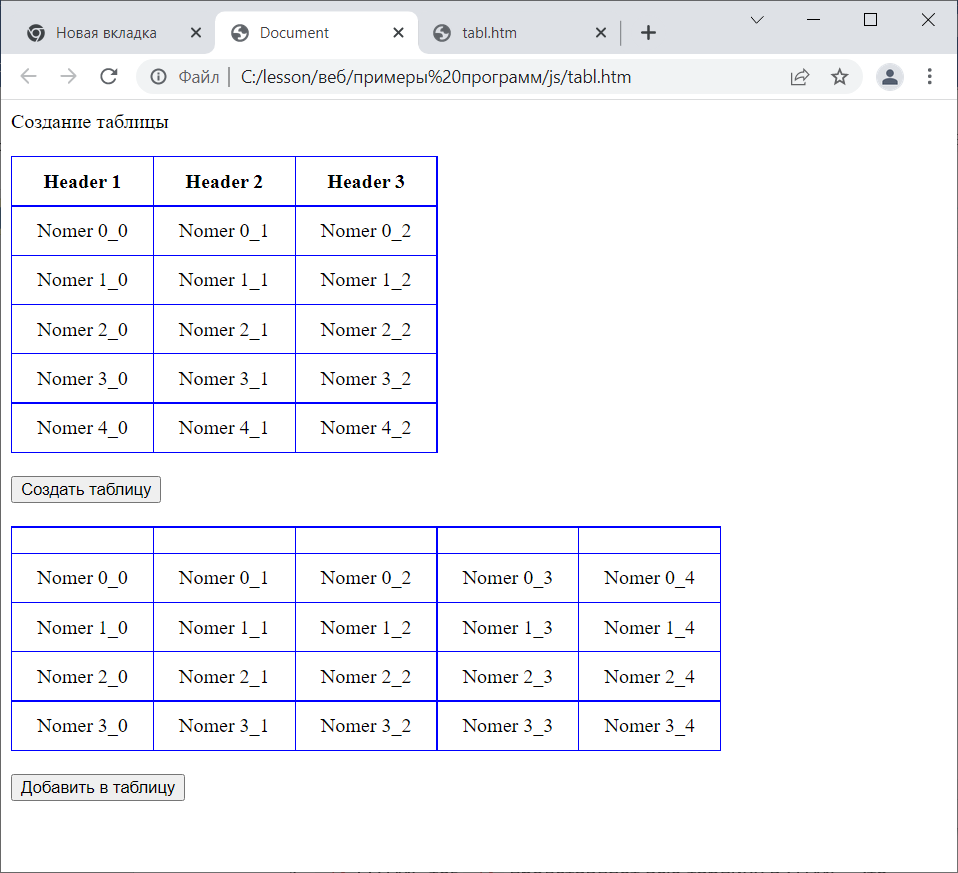
</table>

</div> <br>

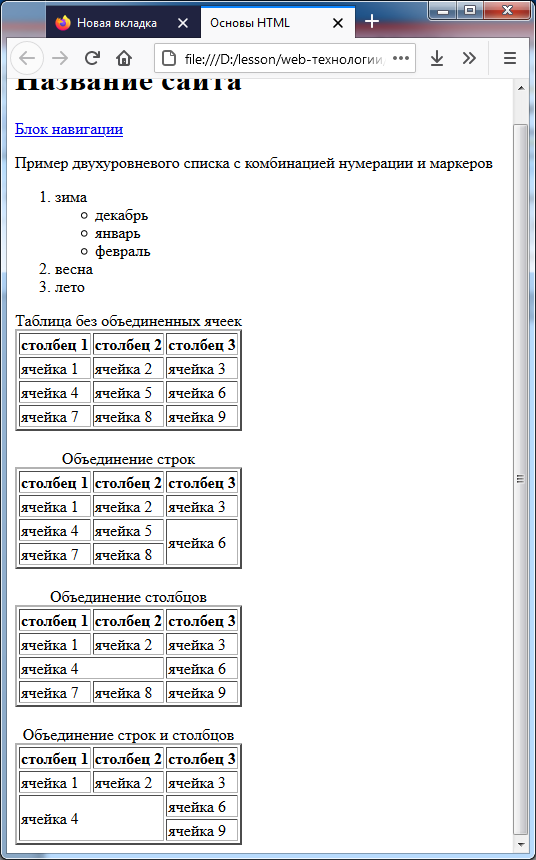
<button onclick = "insertetable(5)">Добавить в таблицу</button>

</body>

</html>



***Отображение страницы в браузере***



**Верстка**

**Табличная верстка (невидимая table)**

Элементы сайта помещаются в ячейки невидимой таблицы. Часто внутри ячейки располагается еще одна таблица, для более точного позиционирования элемента. Сейчас табличная верстка не используется, потому что страница будет отображаться только после полной загрузки таблицы, что замедляет работу.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">

<title> Табличная верстка </title>

</head>

<body>

<table width="1000" border="0" align="center" height="700">

<tr> <td valign="top" height="121" style="border:#6699FF solid 1px">

Шапка сайта </td> </tr>

<tr> <td>

<table width="100%" height="100%" style="border:#6699FF solid 1px">

<tr>

<td valign="top" width="250" style="border:#0000FF solid 1px">

<p class="font\_blue\_18"> Меню сайта</p>

<div class="left\_menu\_my">

<ul>

<li><a href="[../../html\_step.php](view-source:http://www.webpupil.ru/html_step.php)"> HTML по шагам</a></li>

<li><a href="[../../html\_tag.php](view-source:http://www.webpupil.ru/html_tag.php)"> Теги HTML</a></li>

<li><a href="[../../css\_step.php](view-source:http://www.webpupil.ru/css_step.php)"> CSS по шагам</a></li>

<li><a href="[../../css\_property.php](view-source:http://www.webpupil.ru/css_property.php)"> Свойства CSS</a></li>

<li><a href="[../../view\_articles.php?id=1](view-source:http://www.webpupil.ru/view_articles.php?id=1)"> Путь к файлу</a></li>

<li><a href="[../../view\_articles.php?id=2](view-source:http://www.webpupil.ru/view_articles.php?id=2)">Блочные элементы</a></li>

</ul>

</div>

</td>

<td valign="top" style="border:#0000FF solid 1px">

<p class="font\_blue\_18" align="center"> Главная страница сайта</p>

<p>&nbsp;</p>

<p> Контент страницы </p>

<p>&nbsp;</p></td>

<td width="150" style="border:#0000FF solid 1px"><p>Здесь Вы можете разместить рекламу...</p></td>

</tr>

</table>

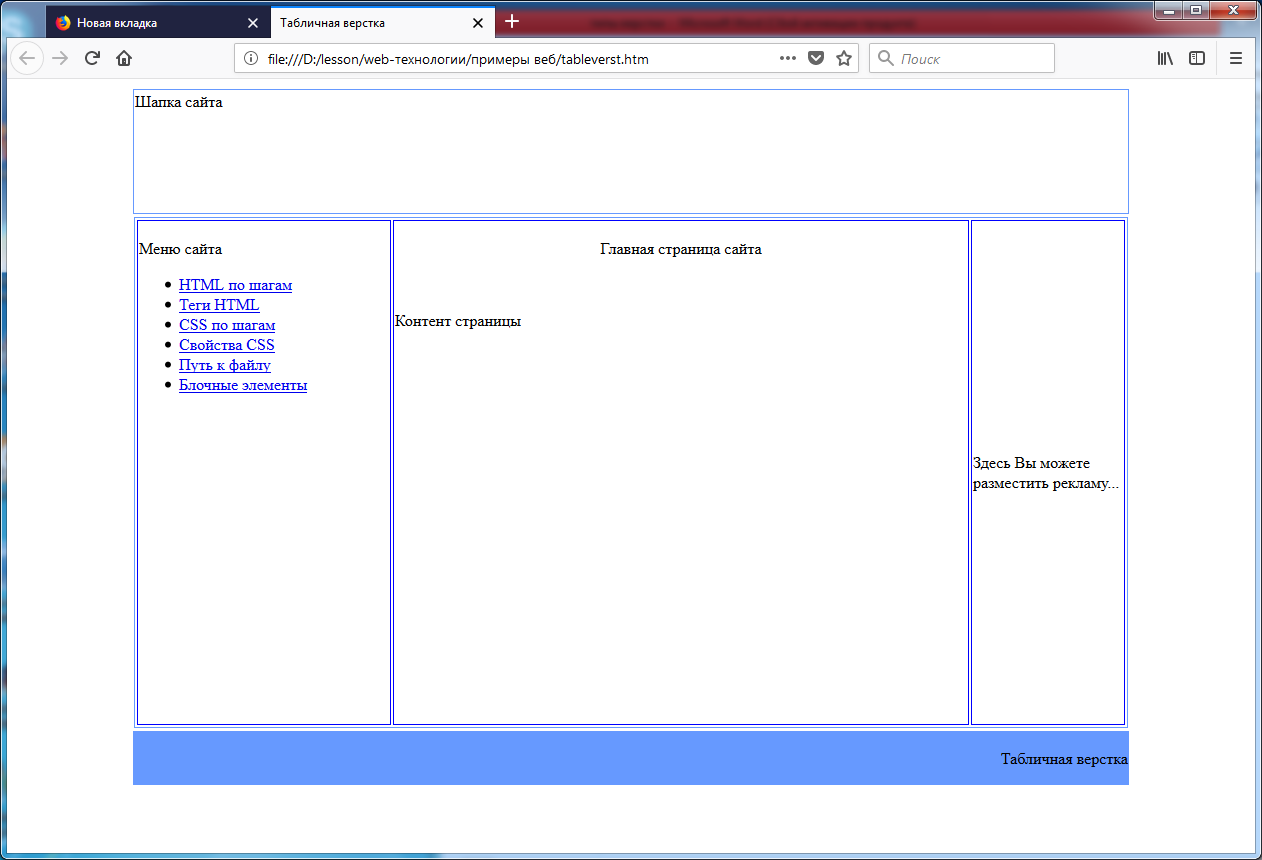
</td> </tr>

<tr> <td bgcolor="#6699FF" height="35px"> <p align="right" class="font\_small\_lsilver">Табличная верстка </p> </td> </tr>

</table>

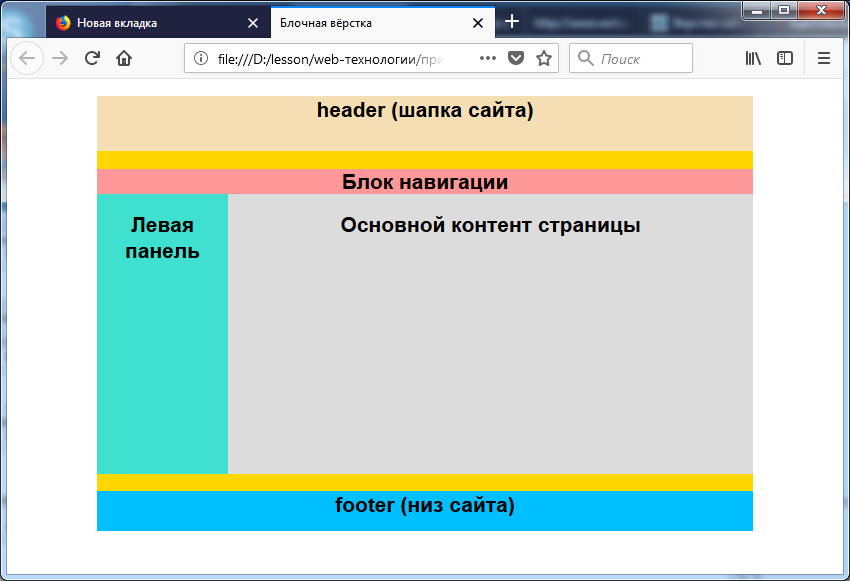
</body>

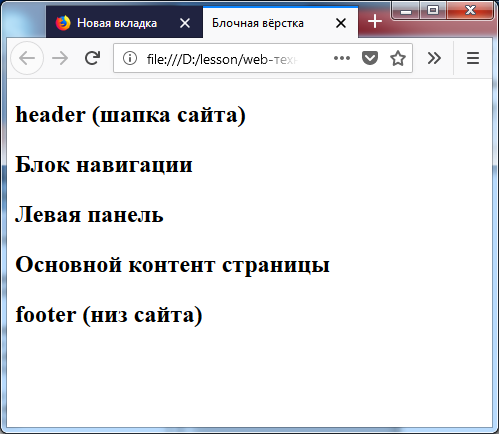
</html>



**Блочная верстка (div + id + css)**

Блочная верстка строится на основе тега div, которому с помощью правил css задают нужное положение на странице.

Вид страницы без применения css



<!DOCTYPE html>

<html>

<head> <title>Блочная вёрстка</title>

<link rel="stylesheet" type="text/css" href="style.css"> </head>

<body>

<div id="container">

<div id="header"> <h2>header (шапка сайта)</h2> </div>

<div id="navigation"> <h2>Блок навигации</h2> </div>

<div id="sidebar"> <h2>Левая панель</h2> </div>

<div id="content"> <h2>Основной контент страницы</h2> </div>

<div id="clear"> </div>

<div id="footer"> <h2>footer (низ сайта)</h2> </div>

</div>

</body> </html>

body {

background: #FFF;

color: #000;

font-family: Arial, sans-serif;

font-size: 14px;

}

#header {

background: #F5DEB3;

width: 100%;

height: 55px;

}

#container {

background: #FFD700;

margin: auto auto;

text-align: center;

width: 80%;

height: 400px;

}

#navigation {

background: #FE9798;

width: 100%;

height: 25px;

}

#sidebar {

background: #40E0D0;

float: left;

width: 20%;

height: 280px;

}

#content {

background: #DCDCDC;

float: right;

width: 80%;

height: 280px;

}

#clear {

clear: both;

}

#footer {

background: #00BFFF;

width: 100%;

height: 40px;

}

**#clear { clear:both; }** запрещает обтекание элемента слева и справа. Если убрать это правило, вёрстка «поедет» и низ сайта перестанет корректно отображаться.

**Семантическая верстка (html5 + css)**

Версия языка html5 содержит специальные теги, структурирующие конструктивные элементы страницы (шапка, панель навигации, боковая панель, раздел, подвал). Теги html5 для верстки: <header>, <nav>, <aside>, <section>, <footer>. Расположение блоков задается через css.

**Файл html5blockverst.htm**

<!DOCTYPE html>

<html>

<head>

<title>Семантическая вёрстка</title>

<link rel="stylesheet" type="text/css" href="style5.css">

</head>

<body>

<div id="container" >

<header> <h2>header (шапка сайта)</h2> </header>

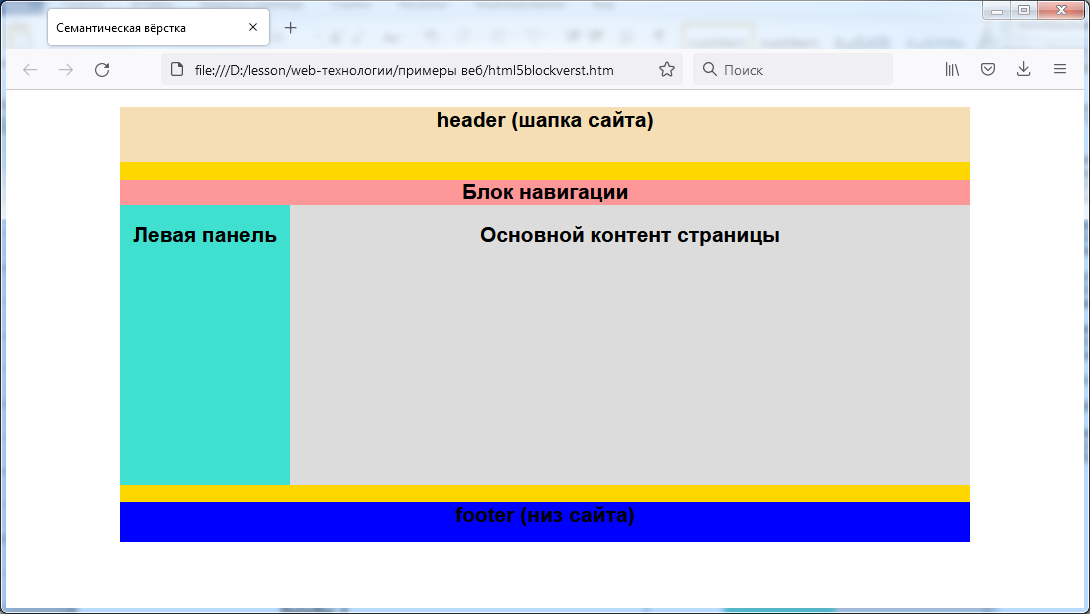
<nav> <h2>Блок навигации</h2> </nav>

<aside> <h2>Левая панель</h2> </aside>

<section> <h2>Основной контент страницы</h2> </section>

<div id="clear"> </div>

<footer> <h2>footer (подвал)</h2> </footer>

</div> </body> </html>

**Файл style5.css**

body {

background: #FFF;

color: #000;

font-family: Arial, sans-serif;

font-size: 14px;

}

header {

background: #F5DEB3;

width: 100%;

height: 55px;

}

#container {

background: #FFD700;

margin: auto auto;

text-align: center;

width: 80%;

height: 400px;

}

nav {

background: #FE9798;

width: 100%;

height: 25px;

}

aside {

background: #40E0D0;

float: left;

width: 20%;

height: 280px;

}

section {

background: #DCDCDC;

float: right;

width: 80%;

height: 280px;

}

#clear {

clear: both;

}

footer {

background: #0000FF;

width: 100%;

height: 40px;

}

**Сравнение html4 и html5**

Самое значимое различие между старыми версиями HTML и HTML5 заключается в интеграции видео и аудио в спецификации языка. Кроме того, в HTML5 вошли следующие обновления:

были удалены устаревшие элементы, такие как center, font и strike;

улучшение правил парсинга сделало его более гибким и совместимым;

появились новые элементы video, time, nav, section, progress, meter, aside и canvas;

новые атрибуты для тега input, в том числе email, URL, dates и times ;

новые атрибуты, в том числе charset, async и ping;

новые API с офлайн кэшированием и поддержкой drag-and-drop и т.д;

поддержка векторной графики без сторонних программ типа Silverlight или Flash;

поддержка MathML улучшила отображение математических обозначений;

благодаря JS Web worker API, JS теперь может работать в фоновом режиме;

глобальные атрибуты типа tabindex, repeat и id теперь можно применять ко всем элементам.

**Лекция 6. Формы и элементы управления**

**План**

1.. Форма

2.. get, post

3.. элементы управления через тег input

4.. элементы управления через специальные теги

5.. примеры форм

**Форма**

Формы – это элемент веб-страницы. Она служат для оправки данных на веб-сервер. Форма заключается в парный тег <form>. Внутри этого тега помещаются элементы интерфейса пользователя – кнопки, поля ввода и т.п. Элементы интерфейса пользователя могут находиться и вне формы – в этом случае для того, чтобы интерфейс был полезен, к элементам прикрепляется сценарий на языке javascript.

Пример первого тега формы:

<FORM method="POST" action="/cgi-bin/tutor5.py">

Основные атрибуты формы: action, name, method.

action – содержит адрес (url) файла, который получает данные из формы. В примере это файл на языке питон tutor5.py, расположенный в каталоге cgi-bin.

name – имя формы.

method – содержит название метода (get или post) передачи данных на веб-сервер. Данные передаются в виде пары:

имя\_элемента\_name = значение\_элемента\_value через разделитель &.

В описании элемента «переключатель» :

<input type=radio name=shoesize value=маленький>

имя элемента задано атрибутом name, а значение атрибутом value.

Метод get передает данные из формы внутри url, этот вариант означает, что передаваемые данные открыты и доступны всем, поскольку отображаются в адресной строке браузера. Пример передачи данных из формы при выполнении запроса get:

<http://localhost/cgi-bin/horiztable.py?name=Аделина&shoesize=маленький&job=студент&language=Python&comment=люблю+программировать>

Чтобы получить читаемый текст при использовании кириллицы, надо зайти на один из сайтов, которые выполняют кодировку/расшифровку url с кириллицей (например, <https://www.design-sites.ru/utility/url-encoding.php>) и преобразовать текст.

Метод post передает данные в виде отдельного файла. Рекомендуется использовать именно его.

Форма будет передавать данные после нажатия кнопки типа submit. Атрибут value здесь содержит надпись на кнопке. Пример кнопки:

<input type=submit value="ОТПРАВИТЬ">

Создать **элемент интерфейса** можно разными способами: с помощью непарного тега **<input>,** в котором через атрибут type задается тип элемента, и с помощью **специальных тегов**. Все элементы интерфейса имеют атрибуты name и value.

Атрибут name – имя элемента интерфейса, необходимо для обращения к элементу при отправке данных на веб-сервер. Атрибут value – значение элемента. Для элемента text – значение по умолчанию, для кнопки – надпись на ней.

**Создание интерфейса через тег <input>**

Основные атрибуты: type, src.

Атрибут **type** – задает вид элемента интерфейса.

Варианты значений:

button - кнопка,

checkbox - флажок,

file – текстовое поле для ввода имени файла, есть кнопка для открытия диалогового окна выбора файла.

hidden – невидимый элемент, но его данные можно передать на сервер,

image – графическая кнопка типа «submit»; атрибут src содержит имя графического файла,

password – текстовое поле для ввода пароля, аналогично элементу text; все символы при печати скрыты.

radio - переключатель,

reset – кнопка для установки значений по умолчанию,

submit – кнопка, которая отправляет данные формы на сервер,

text – текстовое поле ввода.

Примеры элементов управления:

<input type=checkbox name=language value=Python>

<input type=radio name=shoesize value=маленький>

<input type=text name=name>

<input type=submit value="ОТПРАВИТЬ">

**Другие теги для элементов интерфейса**

В парном теге кнопки **<button>** могут размещаться другие теги (текст, изображение, таблица). Для указания типа кнопки используется атрибут type со значениями: button – обычная кнопка, reset – отмена всех изменений в элементах формы, submit – оправка данных формы, внутри которой находится кнопка, на веб-сервер.

Раскрывающийся список создается через парный тег **<select>**, внутри которого размещаются теги <option>, содержащие пункты списка.

<select name=job>

<option>студент <option>преподаватель <option>библиотекарь

<option>другое </select>

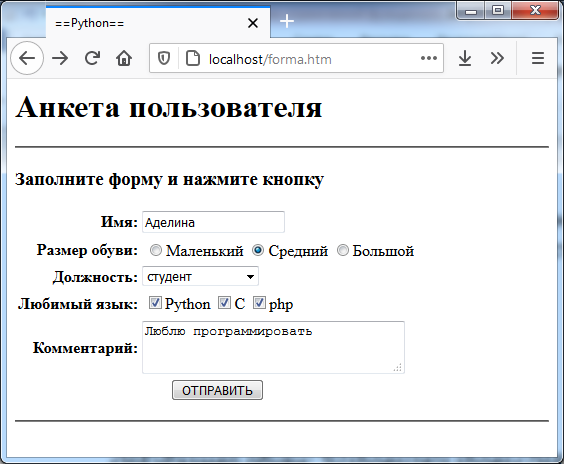
Область ввода текста **<textarea>** - поле для ввода многострочного текста, может содержать полосы прокрутки. Атрибуты cols и rows определяют размер поля в количестве символов по горизонтали и вертикали.

<textarea name=comment cols=30 rows=2>

Для аккуратного размещения элементов интерфейса используется невидимая таблица.

**Листинг 1. Файл forma.htm с формой.**

Запуск файла: <http://localhost/forma.htm> при работающем веб-сервере

****

<HTML><TITLE>==Python==</TITLE>

<BODY>

<H1>Анкета пользователя</H1>

<HR>

<FORM method="POST" action="/cgi-bin/tutor5.py">

<H3>Заполните форму и нажмите кнопку</H3>

<P><TABLE>

<TR>

<TH align=right>Имя:

<TD> <input type=text name=name>

<TR>

<TH align=right>Размер обуви:

<TD><table>

<td><input type=radio name=shoesize value=маленький>Маленький

<td><input type=radio name=shoesize value=средний>Средний

<td><input type=radio name=shoesize value=большой>Большой

</table>

<TR>

<TH align=right>Должность:

<TD><select name=job>

<option>студент

<option>преподаватель

<option>библиотекарь

<option>другое

</select>

<TR>

<TH align=right>Любимый язык:

<TD><table>

<td><input type=checkbox name=language value=Python>Python

<td><input type=checkbox name=language value=C>C

<td><input type=checkbox name=language value=php>php

</table>

<TR>

<TH align=right>Комментарий:

<TD><textarea name=comment cols=30 rows=2>

введите текст</textarea>

<TR>

<TD colspan=2 align=center>

<input type=submit value="ОТПРАВИТЬ">

</TABLE>

</FORM>

<HR>

</BODY></HTML>

**Изменение свойств изображения с помощью элементов управления (Дунаев 18.6.3)**

Установить или изменить свойства изображения, такие как src, width, height и др., можно с помощью выражения:

document. getElementById("*id\_элемента\_img*"). *свойство*=*значение*;

При установке нового значения свойства sr (URL-адрес файла изображения) браузер посылает на сервер соответствующий запрос и пытается загрузить и отобразить графическое изображение без перезагрузки текущего документа. Иначе говоря, содержимое элементов <img> изменяется динамически.

В листинге 2 приведен пример документа с элементом <img> и формой, в которой можно выбрать графическое изображение и ввести значения его ширины и высоты. При щелчке на кнопке **Применить** вызывается функция changeimg(), которая устанавливает заданные в форме значения для соответствующих свойств изображения. Результат можно сразу же увидеть в окне браузера. Щелчок на кнопке **Отмена** возвращает содержимое формы в исходное состояние и отображает исходную картинку.

Листинг 2.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html>  
<head><title>Изменение свойств изображения</title>  
<script type="text/javascript">

function **changeimg()** { // изменение параметров изображения  
document.getElementById("myimg").src=document. myform. mysrc. value;  
document.getElementById("myimg").width=document.myform.mywidth.value;  
document.getElementById("myimg").height=document.myform.myheight.value;  
}  
</script> </head>  
<body>

<form id="myform" name="myform">  
<!-- Ввод значений свойств изображения -->  
Файл:  
<select id="mysrc" name="mysrc" >  
<option value="picture1.jpg"/>Картинка1  
<option value="picture2.jpg"/>Картинка2  
<option value="picture3.jpg"/>Картинка3  
</select>  
<br/>  
Ширина:  
<input id="mywidth" name="mywidth" type="text" value="200" size="4"/>  
Высота:  
<input id="myheight" name="myheight" type="text" value="100" size="4"/> <br/>

<input id="applay" name="applay" type="button" value="Применить" **onclick="changeimg()"**/>  
<input id="cancel" name="cancel" type="button" value="Отмена"  
**onclick="document.myform.reset(); changeimg()"**/> </form>  
<img id="myimg" name="myimg" src="picture1.jpg" alt="Картинка" width="200" height="100"/></body></html>



Некоторые свойства графического изображения (например, width и height) могут быть установлены не только с помощью атрибутов тега <img> или программным путем как свойства соответствующего объекта, но и посредством CSS. Параметры CSS имеют приоритет перед соответствующими параметрами, задаваемыми атрибутами и с помощью скриптов. Например, если в листинге 2 для <img> задать стилевой параметр width, то ширина всех картинок будет определяться именно его значением и сценарий в данном листинге не сможет ее изменить. Чтобы все же это сделать, необходимо в функции changeimg() заменить выражение document. getElementById("myimg"). width=document. myform. mywidth. value;  
на такое:  
document.getElementById("myimg"). **style**.width=  
document.myform.mywidth.value+**"px"**

**Проверка данных перед отправкой (Дунаев 18.4.1)**

Если перед отправкой данных формы требуется предварительно их проверить или еще что-нибудь сделать, то необходим сценарий. В примере из листинга 3 выводится форма и проверяется, имеется ли символ @ в поле ввода адреса электронной почты отправителя и не пусты ли другие поля ввода. Если символа @ в адресе нет или хотя бы одно поле пусто, то отправка не производится. Функция validator() обрабатывает событие onsubmit, возникающее при щелчке на кнопке типа **Отправить** (submit), и возвращает значение true, если все условия выполнены, и false — в противном случае. Чтобы возвращаемое значение могло разрешить или запретить действие обработчика onsubmit по умолчанию, в теге <form> указан атрибут onsubmit="return validator()".

Форма содержит еще одно невидимое поле <input name="browser" type="hidden" value="">, значение которого (название браузера пользователя) формируется программно. Это сделано лишь для демонстрации возможности передачи на сервер данных, неконтролируемых пользователем.

Листинг 3

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html>  
<head><title>Отправка данных</title>

<meta http-equiv="content-type"  
content="text/html; charset=windows-1251"/>  
</head>  
</body>  
<form id="myform" name="myform" method="post"  
action="http://www.anyserver.ru/program.php"  
onsubmit="return **validator()** ">

Ваше имя:  
<input name="username" id="username" type="text" value=""/> <br/><br/>  
Ваш e-mail:  
<input name="email" type="text" value=""/> <br/>  
Ваше сообщение:<br/>  
<textarea name="message" type="text" value=""> </textarea><br/>

<input name="browser" **type="hidden"** value=""/>

<br/>  
<input name="send" type="submit" value="Отправить"/>  
</form> </body>  
<script type="text/javascript">  
document.getElementById('username').focus();

/\* устанавливаем курсор ввода \*/

function **validator()** { // проверка данных формы  
var nousername=document. myform. username. value=="";  
**var noemail=(document.myform.email.value.indexOf('@')** == **- 1);**  
var nomsg=document.myform.message.value=="";

if (nousername || noemail || nomsg){  
errortext="Одно из полей не заполнено,";  
errortext+="\nили неверно указан адрес e-mail.";  
errortext+="\nСообщение не отправлено";  
alert(errortext);  
return false // отменить отправку  
} else { // название браузера в скрытом поле  
myform.browser.value=document.navigator.appName;  
return true // разрешить отправку  
} } </script> </html>



В данном примере правильность адреса электронной почты проверяется весьма примитивно. Для более тщательной проверки требуется либо более сложный сценарий, либо так называемые регулярные выражения. Язык регулярных выражений не прост, но его применение для поиска соответствия строк заданному шаблону, замены подстрок и т. п. очень эффективно. Более полную проверку правильности введенного пользователем адреса электронной почты можно выполнить с помощью следующих двух выражений:  
var re=new RegExp(/[0-9a-z\_]+@[0-9a-z\_^.]+.[a-z]{2,3}/i);  
var noemail=document.myform.email.value.search(re)==-1;  
Здесь сначала создается экземпляр объекта RegExp и устанавливается регулярное выражение (шаблон), в соответствии с которым введенная пользователем строка должна состоять из двух частей, разделенных символом @ (имя пользователя и имя домена). В имени пользователя могут быть латинские буквы в любом регистре, цифры, знаки подчеркивания, минуса и точки. Последнее доменное имя (ru, com и т. п.) должно состоять из двух или трех латинских букв. Выражение вида

*строка*. search(*шаблон*)  
определяет, соответствует ли *строка* указанному в методе search() шаблону; если нет, то метод возвращает –1.  
Приведенные два выражения можно подставить в листинг 3 вместо строки  
var noemail=(document.myform.email.value.indexOf('@') == - 1) ;

и получить более качественную проверку правильности введенного пользователем адреса электронной почты.

**Баннер как форма (Дунаев 18.4.2)**

Баннер — небольшое графическое изображение (часто анимированное), которое что-то рекламирует. Нередко щелчок на баннере просто загружает в браузер некоторую страницу. В рассматриваемом случае это просто графическая ссылка, создаваемая HTML-элементом <a href=…><img src=…/></a>. Поскольку функции баннера реализуются тривиально, дизайнеры сосредотачиваются на его внешнем виде.  
Но не в этом главная сущность данного элемента для программиста. Баннер — конструкция, содержащая что-то видимое и обеспечивающая некоторые действия. С точки зрения функциональности баннер должен обеспечить передачу каких-то данных на сервер, который как-то их обработает. Для создания баннера очень подходит HTML-элемент <form>, т. е. контейнер для элементов пользовательского интерфейса, таких как поля ввода данных, а также кнопки и различного рода переключатели. В баннер, как правило, ничего не нужно вводить, а потому он, как форма, должен содержать только невидимые (скрытые) элементы, получаемые из обычных с помощью атрибута type="hidden". Однако они могут содержать данные(значения атрибута value), например, URL-адрес страницы, сведения о браузере пользователя и др., необходимые серверному приложению для учета щелчков и, в частности, для начисления бонусов владельцу страницы с данным баннером.

Элементы формы могут содержать предустановленные данные (значения атрибутов value) или сформированные программно путем присвоения свойству value требуемого значения.  
Форма для баннера, кроме скрытых полей, должна содержать элемент  
<input type="image" src=*url-адрес\_изображения/*>.  
Это графическое изображение, которое в форме играет ту же роль, что и кнопка типа submit. Иначе говоря, щелчок на ней приводит к передаче данных, содержащихся во всех элементах формы, в том числе и скрытых. Разумеется, файл графического изображения, указанный в атрибуте src, должен иметь надлежащее содержание, соблазняя посетителя страницы щелкнуть на его картинке. Но сейчас не об этом. Пример кода баннера приведен в листинге 4. При щелчке на картинке будет вызвана функция send() , которая запишет в скрытое поле адрес страницы с данным баннером. Поскольку картинка играет роль кнопки типа submit, произойдет отправка на сервер скрытых данных.

Листинг 4

<! DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 transitional//EN"  
"http://www. w3. org/TR/xhtml1/DTD/xhtml1-transitional. dtd">  
<html>  
<head><title>Баннер</title></head>  
<body>

<form name="myform"  
action="http://www.anyserver.ru/anyprogram.php" method="post" onsubmit="**send()** ">

<input type="hidden" name="mydata"/>  
<input type="image" src="banner.jpg"/>  
</form> </body>

<script type="text/javascript">

function **send()**

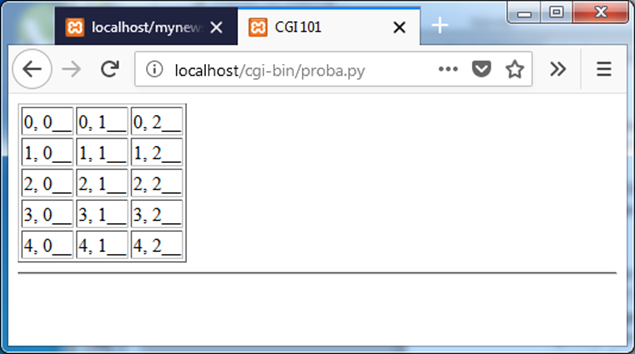
{ // запись адреса страницы в скрытое поле  
document.myform.mydata.value=window.location.href;}  
</script> </html>

**Таким образом, если форма содержит только скрытые элементы, то посетитель Web-страницы может и не догадаться, что, щелкнув на картинке или кнопке, он отправляет какие-то данные, сформированные программно.**

**Лекция 8. Примеры применения языка питон в веб-программировании**

**Задание 1. Формирование таблицы**

text = """ content­type: text/html

<TITLE>CGI 101</TITLE>

<table border =1>

"""

print (text)

for i in range (5):

print ('<tr>')

for j in range (3):

print ('<td> %d, %d \_\_</td> ' % (i,j))

print ('</tr>')

print("""

</table>

<HR>

""")

**Задание 2. Программно сформировать страницу html с двухуровневым списком с перечислением времен года и месяцев. Названия времен года и месяцев предварительно поместить в списки питона и заполнять пункты в циклах.**

**Задание 2. Вариант решения 1**

seasons = ["зима", "весна", "лето", "осень"]

months = ["декабрь", "январь", "февраль", "март", "апрель", "май", \

"июнь", "июль", "август", "сентябрь", "октябрь", "ноябрь"]

text = """ content­type: text/html

<TITLE>CGI 101</TITLE>

<ul>

"""

print (text)

j = 0 # счетчик месяцев

for i in range(len(seasons)):

    print ('<li>' + seasons[i] + '</li>')

    print ('<ul>')

    j = i \* 3 # 0     1     2     3       - номера времен года

              # 0 1 2 3 4 5 6 7 8 9 10 11 - номера месяцев

    for \_ in range(3):

        print ('<li>' + months[j] + '</li>')

        j += 1

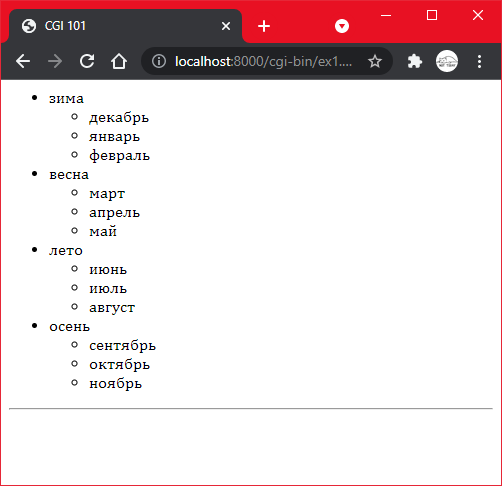
    print ('</ul>')

print("""

</ul>

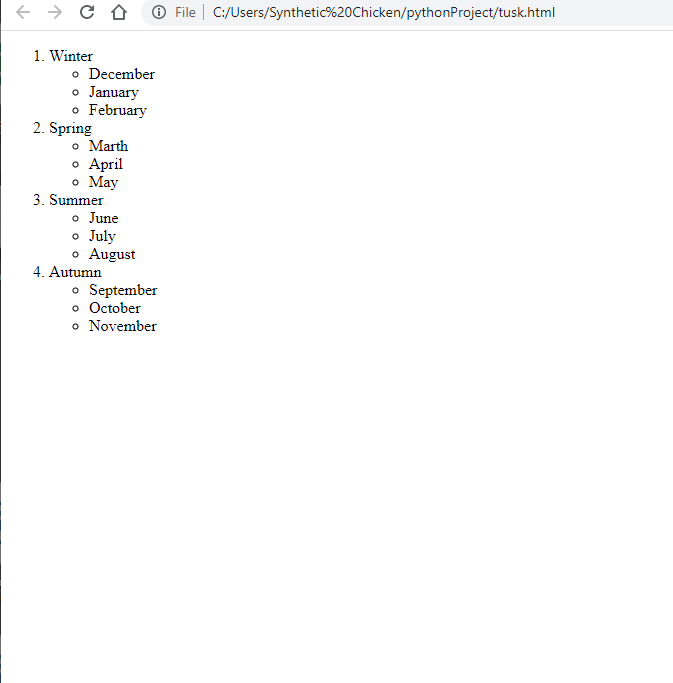
<HR>

""")



**Задание 2. Вариант решения** 2

seasons = ['Winter', 'Spring', 'Summer', 'Autumn']  
months = ['December', 'January', 'February', 'Marth', 'April', 'May', 'June', 'July', 'August', 'September', 'October',  
 'November']  
  
with open("tusk.html", "w") as file:  
 file.write(  
 "<!DOCTYPE html>" + "\n" + "<html>" + "\n" + "\t<head>" + "\n" + "\t" \* 2 + "<title>Seasons</title>" + "\n" + "\t" \* 2 + "<style></style>" + "\n" +  
 "\t</head>" + "\n" + "\t<body>" + "\n" + "\t" \* 2 + "<header>" + "\n" + "\t" \* 2 + "</header>" + "\n" + "\t" \* 2 + "<main>" + "\n")  
  
 file.write("\t" \* 3 + "<ol>" + "\n")  
  
 for s in seasons:  
 file.write("\t" \* 4 + f"<li>{s}" + "\n")  
 file.write("\t" \* 4 + "<ul>" + "\n")  
 for i\_m in range(3):  
 file.write("\t" \* 5 + f"<li>{months[i\_m]}</li>" + "\n")  
 del months[:3]  
 file.write("\t" \* 4 + "</ul>" + "\n")  
 file.write("\t" \* 4 + "</li>" + "\n")  
  
 file.write("\t" \* 3 + "</ol>" + "\n")  
  
 file.write(  
 "\t" \* 2 + "</main>" + "\n" + "\t" \* 2 + "<footer>" + "\n" + "\t" \* 2 + "</footer>" + "\n" + "\t</body>" + "\n" + "</html>")



**Задание 2. Вариант решения 3**

import cgi, sys

Style = """

li { text-align: left; font-family: "Century Gothic", serif; }

ol { font-family: "Century Gothic", serif; }

ol > li { font-weight: bold; font-family: "Century Gothic", serif; }

"""

print('<html>')

print('<head>')

print('<title>Seasons</title>')

print('<style>')

print(Style)

print('</style>')

print('</head>')

class MonthProvider:

def \_\_init\_\_(self, monthList, yearSeasons):

self.months = monthList

self.yearSeasons = yearSeasons

self.monthState = 0

self.seasonState = 0

def getNext(self):

if self.monthState == len(self.months):

return None, None

if 0 <= self.monthState < 2:

self.seasonState = 0

elif 3 <= self.monthState < 6:

self.seasonState = 1

elif 6 <= self.monthState < 9:

self.seasonState = 2

elif 9 <= self.monthState < 12:

self.seasonState = 3

month = self.months[self.monthState]

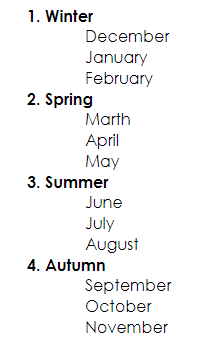
season = self.yearSeasons[self.seasonState]

self.monthState += 1

return month, season

Seasons = ['Winter', 'Spring', 'Summer', 'Autumn']

Months = ['December', 'January', 'February', 'Marth', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November']

Provider = MonthProvider(Months, Seasons)

print('<ol>')

CurrentSeason = 'None'

while True:

month, season = Provider.getNext()

if month == None or season == None:

break

if CurrentSeason != season:

print('\t<li>', season, '</li>')

CurrentSeason = season

print('\t<ul>', month, '</ul>')

print('</ol>'); print('</html>')

**Программы для отправки электронной почты**

**Некоторые сведения про строки в питоне**

Строка – это неизменяемый объект

>>> s = "hello"

>>> s[0]

'h'

Нельзя менять элементы строки

>>> s[0] = 'H'

Traceback (most recent call last):

File "<pyshell#15>", line 1, in <module> s[0] = 'H'

TypeError: 'str' object does not support item assignment

**Удаляет пропуски в начале и в конце строки**

>>> s1 =" spring time "

>>> s1.strip()

'spring time'

**Разбиение строки по разделителям**

>>> s = "a aa aaa b"

>>> li = s.split (' ') # по пробелам

>>> li

['a', 'aa', 'aaa', 'b']

>>> li2 = s.split () # пробел – это разделитель по умолчанию

>>> li2

['a', 'aa', 'aaa', 'b']

>>> li3 = s.split('a')

>>> li3

['', ' ', '', ' ', '', '', ' b']

**Объединение элементов из списка в строку**

>>> l

['h', 'e', 'l', 'l', 'o', ',', 'N', 'a', 'd']

>>> sl = ''.join(l)

>>> sl

'hello,Nad'

**Объединение элементов из списка в строку с применением разделителя**

>>> li = ["one", "two"]

>>> s2 = " \_ "

>>> s2.join (li)

'one \_ two'

**Комментарии к программам отправки почты**

‘adress.txt' файл со списком адресов

[eeeee@list.ru](mailto:eeeee@list.ru) Иван

[qqqq@mail.ru](mailto:qqqq@mail.ru) Софья

[rrrrr@gmail.com](mailto:rrrrr@gmail.com) Света

listadress = []

for line in open(‘adress.txt','r'):

line = line.strip()

listadress.append(line)

listadress = [] создание пустого списка

open(‘adress.txt','r') функция открытия файла в режиме чтения

for line in open(‘adress.txt','r'): цикл по всем строкам открытого файла

line - переменная, содержит строку из файла

parts = line.split(' ') разбиваем строку line по пробелам и записываем части строки в список parts

listadress.append(parts) добавляем новый элемент в список listadress

listadress - список из списков

for line in open(‘adress.txt','r'):

line = line.strip()

parts = line.split(' ')

listadress.append(parts)

**отправка почты:**

for x in listadress:

msg["To"] = x[0]

print ('adress = ', x[0])

server.sendmail(FROM, x[0], msg.as\_string())

server.quit()

for x in listadress: цикл по всем элементам списка listadress. X – элемент списка, который является списком

msg["To"] = x[0] берем первый элемент

**Вариант 1. Простое письмо**

import smtplib

from email.mime.text import MIMEText

from email.header import Header

HOST = "smtp.mail.ru" почтовый сервер для отправителя

SUBJECT = "Test email from Python"

TO = "mail1@list.ru"

FROM = "mail2@mail.ru"

text = "DAY OPEN"

password = "123456"

msg = MIMEText('body… русский язык', 'plain', 'utf-8')

msg['Subject'] = Header('тема…', 'utf-8')

msg['From'] = FROM

BODY = "\r\n".join((

"From: %s" % FROM,

"To: %s" % TO,

"Subject: %s" % SUBJECT ,

"",

text

))

server = smtplib.SMTP(HOST)

server.starttls()

server.login (FROM, password)

#server.sendmail(FROM, TO, BODY)

server.sendmail(FROM, TO, msg.as\_string())

server.quit()

**Вариант 2. Письмо с вложением файла**

import smtplib

from email.mime.text import MIMEText

from email import encoders

from email.header import Header

from email.mime.base import MIMEBase

from email.mime.multipart import MIMEMultipart

from email.utils import formatdate

HOST = "smtp.mail.ru"

SUBJECT = Header('день дверей', 'utf-8')

TO = "mail1@list.ru"

FROM = "mail2@mail.ru"

text = "Здравствуйте!" текст письма

password = "123456"

file\_to\_attach = 'note.pdf' файл для отправки

header = 'Content-Disposition', 'attachment; filename="%s" ' % file\_to\_attach

msg = MIMEMultipart()

msg["From"] = FROM

msg["Subject"] = SUBJECT

msg.attach( MIMEText(text, 'plain','utf-8') )

msg["To"] = TO

attachment = MIMEBase('application', "octet-stream")

try:

with open(file\_to\_attach, "rb") as fh:

data = fh.read()

attachment.set\_payload( data )

encoders.encode\_base64(attachment)

attachment.add\_header(\*header)

msg.attach(attachment)

except IOError:

msg = "Error opening attachment file %s" % file\_to\_attach

print(msg)

sys.exit(1)

server = smtplib.SMTP(HOST)

server.starttls()

server.login (FROM, password)

server.sendmail(FROM, TO, msg.as\_string())

server.quit()

**Вариант 3. Отправка группы писем с вложением файла.**

import smtplib

from email.mime.text import MIMEText

from email import encoders

from email.header import Header

from email.mime.base import MIMEBase

from email.mime.multipart import MIMEMultipart

from email.utils import formatdate

HOST = "smtp.mail.ru"

SUBJECT = Header('день открытых дверей институт 4 ГУАП', 'utf-8')

FROM = "abiturientguap4@mail.ru"

text1 = "Здравствуйте! Приглашаем Вас в ГУАП на день открытых дверей института вычислительных систем и сетей (№ 4) 29 ноября, пятница, начало в 16 часов."

text2 = ' <http://new.guap.ru/messages/596>'; text = text1+text2

password = "123456"

file\_to\_attach = 'note29.pdf'

header = 'Content-Disposition', 'attachment; filename="%s" ' % file\_to\_attach

msg = MIMEMultipart()

msg["From"] = FROM

msg["Subject"] = SUBJECT

msg.attach( MIMEText(text, 'plain','utf-8') )

attachment = MIMEBase('application', "octet-stream")

try:

with open(file\_to\_attach, "rb") as fh:

data = fh.read()

attachment.set\_payload( data )

encoders.encode\_base64(attachment)

attachment.add\_header(\*header)

msg.attach(attachment)

except IOError:

msg = "Error opening attachment file %s" % file\_to\_attach

print(msg)

sys.exit(1)

server = smtplib.SMTP(HOST)

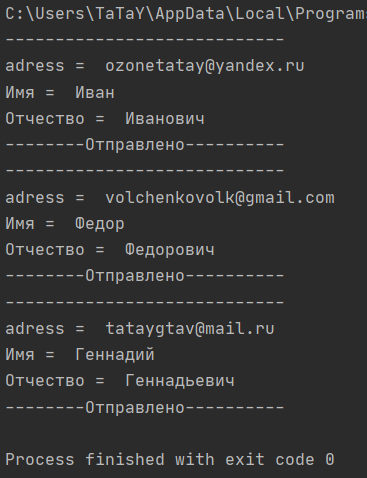
server.starttls()

server.login (FROM, password)

**Вариант 4.**

В тексте письма использовать обращение к адресату по имени и отчеству (после слова «Здравствуйте» следует разместить имя и отчество. Например, «Здравствуйте, Иван Эдуардович!»). Для этого в файле adress.txt кроме адреса почты нужно вписать И.О. получателя. Последующий текст письма (после приветствия) следует сохранить таким, как дано в примере.

**Вариант 4.1**

import smtplib  
from email.mime.text import MIMEText  
from email.header import Header  
from email.mime.base import MIMEBase  
from email.mime.multipart import MIMEMultipart  
  
HOST = "smtp.mail.ru"  
SUBJECT = Header('Вариант задания №4. Волченков Сергей', 'utf-8')  
FROM = "volk@mail.ru"  
text1 = "Здравствуйте, "  
text2 = 'Приглашаем Вас в ГУАП на день открытых дверей института вычислительных систем и сетей (№ 4) 29 ноября, ' \  
 'пятница, начало в 16 часов. http://new.guap.ru/messages/596 '  
password = "qUK36"  
  
attachment = MIMEBase('application', "octet-stream")  
  
server = smtplib.SMTP(HOST)  
server.starttls()  
server.login(FROM, password)  
  
listadress = []  
for line in open('address.txt', 'r', encoding='UTF-8'):  
 line = line.strip()  
 parts = line.split(' ')  
 listadress.append(parts)  
for x in listadress:  
 msg = MIMEMultipart()  
 msg["From"] = FROM  
 msg["To"] = x[0]  
 msg["Subject"] = SUBJECT  
 text = text1 + x[1] + ' ' + x[2] + '!\n' + text2  
 msg.attach(MIMEText(text, 'plain', 'utf-8'))  
 print('----------------------------')  
 print('adress = ', x[0])  
 print('Имя = ', x[1])  
 print('Отчество = ', x[2])  
 server.sendmail(FROM, x[0], msg.as\_string())  
 print('--------Отправлено----------')  
server.quit()

Вариант 4.2

import smtplib  
from email.mime.text import MIMEText  
from email.header import Header  
from email.mime.multipart import MIMEMultipart  
  
HOST = "smtp.mail.ru"  
SUBJECT = Header('Занятие по Web-технологиям 26.10', 'utf-8')  
FROM = "999@mail.ru"  
text1 = "Здравствуйте, "  
text2 = "!\n\nЗавтра 26.10 лабораторные работы по Web-технологиям будут проходить очно по расписанию.\n\nС уважением," \  
 "\nВиктор"  
with open('password.txt', 'r', encoding='utf-8') as file:  
 password = file.readline()  
  
server = smtplib.SMTP(HOST)  
server.starttls()  
server.login(FROM, password)  
  
file\_address = open('address.txt', 'r', encoding='utf-8')  
for line in file\_address:  
 line = line.strip()  
 parts = line.split(' ')  
 msg = MIMEMultipart()  
 msg["From"] = FROM  
 msg["Subject"] = SUBJECT  
 msg["To"] = parts[0]  
 msg.attach(MIMEText(text1 + parts[1] + ' ' + parts[2] + text2, 'plain', 'utf-8'))  
 server.sendmail(FROM, parts[0], msg.as\_string())  
file\_address.close()  
server.quit()

Вариант 4.3

import smtplib

from email.mime.text import MIMEText

from email.header    import Header

HOST = "smtp.gmail.com: 587"

SUBJECT = "Test email from Python"

FROM = "aleksandervoynov@gmail.com"

text = "DAY OPEN"

password  = "1234"

server = smtplib.SMTP(HOST)

server.starttls()

server.login (FROM, password)

file = open('adress.txt', 'r',encoding='utf-8')

lines = file.readlines()

for line in lines:

    length = len(line)

    temp=0

    TO=""

    name=""

    for i in range (length):

        if (line[i]=="\n"):

            break

        if (temp==0):

            TO+=line[i]

        if (temp==1):

            name+=line[i]

        if (line[i]==" "):

            temp=1

            continue

    BODY = "\r\n".join((

    "Здравствуйте, %s" % name +"!",

    "From: %s" % FROM,

    "To: %s" % TO,

    "Subject: %s" % SUBJECT ,

    "",

    text

    ))

    msg = MIMEText('%s' %BODY, 'plain', 'utf-8')

    msg['Subject'] = Header(SUBJECT, 'utf-8')

    msg['From'] = FROM

    server.sendmail(FROM, TO, msg.as\_string())

server.quit()

Вариант 4.4

from email.mime.text import MIMEText  
from email.header import Header  
  
HOST = "smtp.mail.ru"  
  
SUBJECT = "Test email from Python"  
list\_address = []  
with open('address.txt', encoding='utf-8') as fi:  
 for line in fi:  
 list\_address.append(line.strip().split(' '))  
FROM = "011@bk.ru"  
password = "463346"  
server = smtplib.SMTP(HOST)  
server.starttls()  
server.login(FROM, password)  
for s in list\_address:  
 msg = MIMEText('Здравствуйте, ' + s[1] + '! Приглашаем Вас в ГУАП на день открытых дверей института вычислительных систем и сетей (№ 4) 29 ноября, пятница, начало в 16 часов.', 'plain', 'utf-8')  
 msg['Subject'] = Header('Test email from Python', 'utf-8')  
 msg['From'] = FROM  
 server.sendmail(FROM, s[0], msg.as\_string())  
server.quit()

Вариант 4.5

import sys  
import re  
import smtplib  
from email.mime.text import MIMEText  
from email.header import Header  
  
  
def print\_help():  
 *"""  
 Вывод справки  
 """* print("Параметры:\n"  
 "[1] - путь к файлу со списком адресатов\n"  
 "[2] - путь к файлу с текстом письма\n"  
 "[3] - путь к файлу с логином, паролем и хостом почты адресанта\n")  
 print("Структура строки файла со спском адресатов:\n"  
 "<адрес> <Фамилия> <Пол (М или Ж)>\n")  
 print("Структура файла с параметрами адресанта:\n"  
 "<HOST> (например, smtp.yandex.ru)\n"  
 "<адрес>\n"  
 "<пароль>\n")  
 print("В файле с текстом письма первая строка указывает тему")  
  
  
def check\_mail(mail):  
 *"""  
 Проверка формата ввода почтового адреса* ***:param*** *mail: строка для проверки* ***:return****: результат проверки по регулярному выражению (True или False)  
 """* reg = re.compile(r"^[\w\.\-]+\@[\w]+\.[a-z]{2,3}$")  
 return reg.search(mail)  
  
  
def parse(file\_name):  
 *"""  
 Обработка списка адресатов* ***:param*** *file\_name: путь к файлу со списком адресатов* ***:return****: addresses - списко адресатов в формате [адрес, фамилия, пол (М или Ж)]  
 """* addresses = list()  
 gender = ['М', 'Ж']  
  
 try:  
 for line in open(file\_name, encoding = 'UTF-8'):  
 temp\_list = line.rstrip('\n').split()

# По умолчанию split разбивает строку по пробелам  
 temp\_list = [s for s in temp\_list if s]

# Убираем пустые строки, если они есть  
 if len(temp\_list) != 3 or not check\_mail(temp\_list[0]) or not (temp\_list[2] in gender):  
 continue  
 else:  
 addresses.append(temp\_list)  
 except Exception as e:  
 print("Error: ", str(e))  
 return addresses  
  
  
def get\_sender(sender\_file):  
 *"""  
 Обработка информации об отправителе* ***:param*** *sender\_file: путь к файлу с информацией отправителя (хост, логин, пароль)* ***:return****: sender\_data: список с информацией из файла  
 """* sender\_data = list()  
 for line in open(sender\_file, encoding = 'UTF-8'):  
 sender\_data.append(line.rstrip('\n'))  
 if len(sender\_data) < 3 or not check\_mail(sender\_data[1]):  
 raise Exception("Некорректные данные отправителя")  
 return sender\_data  
  
  
def send(sender\_data, addresses, text\_way):  
 *"""  
 Отправка письма* ***:param*** *sender\_data: список с информацией отправителя* ***:param*** *addresses: список адресатов* ***:param*** *text\_way: путь к файлу с текстом письма, который должен идти после приветствия  
 """* text = list()  
 for line in open(text\_way, encoding = 'UTF-8'):  
 text.append(line)  
 subject = Header(text[0].rstrip('\n'))  
 server = smtplib.SMTP(sender\_data[0])  
 server.starttls()  
 server.login(sender\_data[1], sender\_data[2])  
  
 for address in addresses:  
 if address[2] == "М":  
 text[0] = "Здравствуйте, господин " + address[1] + "!"  
 else:  
 text[0] = "Здравствуйте, госпожа " + address[1] + "!"  
  
 print("Отправка: ", address[0])  
  
 msg = MIMEText("\n".join(text), 'plain', 'utf-8')  
 msg["From"] = sender\_data[1]  
 msg["Subject"] = subject  
 server.sendmail(sender\_data[1], address[0], msg.as\_string())  
 server.quit()  
  
  
# Параметры командной строки:  
# [1] - путь к файлу со списком адресатов  
# [2] - путь к файлу с текстом письма  
# [3] - путь к файлу с логином, паролем и хостом почты адресанта  
def main():  
 if len(sys.argv) == 4:  
 try:  
 send(get\_sender(sys.argv[3]), parse(sys.argv[1]), sys.argv[2])  
 except Exception as e:  
 print("Error: ", str(e))  
 elif len(sys.argv) == 2 and sys.argv[1] == "-help":  
 print\_help()  
 else:  
 print("Пожалуйста, укажите параметры. Справка: '-help'")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вариант 4.6

# Установка параметров

FILE **=** "adress.txt" # Файл с адресатами

HOST **=** "smtp.yandex.ru" # Адрес сервера для отправки почты

FROM = "ansekunov@yandex.ru"

PASS = "password"

# Тема сообщения

SUBJECT = "День открытых дверей институт 4 ГУАП"

# Текст письма

TEXT = """

Приглашаем Вас в ГУАП на день открытых дверей института вычислительных систем и сетей (№ 4) 29 ноября, пятница, начало в 16 часов.

http://new.guap.ru/messages/596

"""

**import smtplib**

**from email.header import Header**

**from email.mime.text import MIMEText**

# Простая проверка названия почты

# Возвращает true - если всё верно

**def check\_email(email):**

is\_correct = False # Флаг корректности строки

l = len(email) - 1 # Индекс последнего символа

indices = [] # Для хранения индексов на @ и . после @

**for i, c in enumerate(email):**

# Проверка, что есть символ @ и он не вначале и не повторялся

**if c == '@':**

if i != 0 and len(indices) == 0:

indices.append(i)

else:

break

# Фиксирует наличие точки после @

elif c == '.' and len(indices) > 0:

indices.append(i)

# Если просмотрели всю строку и есть {@, .}

**elif i == l and len(indices) > 1:**

is\_correct = True

# Проверка, что символы @ и . не стоят рядом друг с другом

**for i in range(0, len(indices) - 1):**

if indices[i+1] - indices[i] < 2:

is\_correct = False

break

return is\_correct

# Обрабатывает i-строку файла line

# Возвращает кортеж (почта, сообщение)

**def parse\_data(i, line):**

email = None # Выходные параметры

message = None

line.strip() # Удаление пробелов с обоих концов

values = line.split() # Разбиение строк по пробелам

# Проверка количества параметров

if len(values) == 3:

# Проверка значения почты

if check\_email(values[0]):

email = values[0]

else:

print("Строка {0}. Почта {1} не соответствует формату."**.format(**i**,** values**[**0**]))**

**if values[1] == "M":**

message = "Здравствуйте, господин {0}!".format(values[2]) + TEXT

elif values[1] == "W":

message = "Здравствуйте, госпожа {0}!".format(values[2]) + TEXT

**else:**

**print(**"Строка {0}. Значение \"{1}\" должное быть M или W."**.format(**i**,** values**[**1**]))**

else:

print("Строка {0}. Несоответствие числа параметров.".format(i))

**return (email, message)**

# Отправка message на email

**def send\_email(email, message):**

# Формирование тела запроса на отправление почты

msg **=** MIMEText**(**message**,** 'plain'**,** 'utf-8'**)**

msg**[**'Subject'**] =** Header**(**SUBJECT**,** 'utf-8'**)**

msg['From'] = FROM

# Отправка запроса

server = smtplib.SMTP(HOST)

server**.**starttls**();** # Включение защищённого протокола передачи

server.login(FROM, PASS)

server.sendmail(FROM, email, msg.as\_string())

server.quit()

# Функция для отправки писем из файла

**def main():**

i = 1 # Для вывода номера строки с ошибкой

**try:**

**for** line **in open(**FILE**,** 'r'**,** encoding **=** 'UTF-8'**):**

(email, message) = parse\_data(i, line)

if email != None and message != None:

send\_email(email, message)

**print("Отправлено сообщение на почту {0}\n".format(email))**

i += 1

except Exception as e:

**print(e)**

# Выполнение скрипта

main()

Дополнительные задания:

1) отправить почту с разных почтовых серверов. Проанализировать результаты

2) отправить файлы разных типов и разных объемов. Проанализировать результаты

3) в тексте письма использовать обращение к адресату по имени (после слова «Здравствуйте» следует разместить имя. Например, «Здравствуйте, Антон!»). Для этого в файле adress.txt кроме адреса почты нужно вписать имя получателя. Последующий текст письма (после приветствия) следует сохранить таким, как дано в примере.

4) в тексте письма использовать обращение к адресату по имени и отчеству (после слова «Здравствуйте» следует разместить имя и отчество. Например, «Здравствуйте, Иван Эдуардович!»). Для этого в файле adress.txt кроме адреса почты нужно вписать И.О. получателя. Последующий текст письма (после приветствия) следует сохранить таким, как дано в примере.

5) в тексте письма использовать обращение к адресату в форме «Здравствуйте, господин Иванов!» или «Здравствуйте, госпожа Смирнова!». Для этого в файле adress.txt кроме адреса почты нужно вписать фамилию и пол получателя. Последующий текст письма (после приветствия) следует сохранить таким, как дано в примере.