Оглавление

[**1. Обзор современного стандарта HTML5. Теги и атрибуты. Отличия языков гипертекстовой разметки HTML и XHTML. Устаревшие теги и атрибуты. Поддержка тегов и атрибутов браузерами. Локализация. Метатеги.** 1](#_Toc125067677)

[**2. Структурирование HTML текста. Абзацы, заголовки, списки. Гиперссылки и якоря.** 2](#_Toc125067678)

[**3. Теги для оформления таблиц и блоков. Отступы. Границы. Табличная и блочная верстка.** 3](#_Toc125067679)

[**4. Графика и мультимедиа. HTML теги вставки изображений. Фон и оформление страницы.** 5](#_Toc125067680)

[**5. Использование векторной графики SVG в HTML 5.** 7](#_Toc125067681)

[**6. HTML формы. Поля ввода. Кнопки, селекторы и другие интерактивные элементы.** 8](#_Toc125067682)

[**7. Способы стилизации элементов страницы. Обзор стандарта CSS3. Наследование классов и атрибутов. Приоритет стилей. Комбинаторы и селекторы. Псевдоэлементы и псевдоклассы. Стилизация для различных устройств.** 10](#_Toc125067683)

[**8. Стилизация CSS. Параметры шрифта и фона. Блоки. Параметры абзацев, списков и отображения. Курсоры.** 12](#_Toc125067684)

[**9. Основы дизайна страниц. Стилизация CSS. Свойства фона. Вертикальное и горизонтальное меню. Стилизация блочных и табличных элементов. Отступы. Границы. Выделения.** 14](#_Toc125067685)

[**10. Стилизация CSS. Виды позиционирования и выравнивания элементов. Видимость элементов. Слои. Прозрачность. Текстовые и графические эффекты. Анимация.** 17](#_Toc125067686)

[**11. Язык сценариев JavaScript. Синтаксис языка. Переменные. Операторы. Операции. Функции. Массивы. Ссылки. Объекты.** 18](#_Toc125067687)

[**12. Подключение скриптов на странице. Обработка событий. Область видимости переменных. Замыкания. Ввод и вывод данных. Отладка программ.** 22](#_Toc125067688)

[**13. Объектная модель браузера** 22](#_Toc125067689)

[**14. Доступ сценариев к элементам формы. Проверки формы javascript** 24](#_Toc125067690)

[**15. Стандарты представления данных XTM. Формат обмена JSON.** 26](#_Toc125067691)

[**16. Использование функций DOM (я не знаю про что тут). Генерация элементов страницы JS** 28](#_Toc125067692)

[**17. Отправка данных на сервер** 29](#_Toc125067693)

[**22. PHP классы и объекты** 31](#_Toc125067694)

[**25. Трехуровневая архитектура приложений** 33](#_Toc125067695)

[**26. Основы информационной безопасности. Защита Web – приложений от несанкционированного доступа и SQL инъекций. Средства проверки на безопасность. Использование плейсхолдеров и драйвера PDO.** 35](#_Toc125067696)

# **1. Обзор современного стандарта HTML5. Теги и атрибуты. Отличия языков гипертекстовой разметки HTML и XHTML. Устаревшие теги и атрибуты. Поддержка тегов и атрибутов браузерами. Локализация. Метатеги.**

HTML5 — это не продолжатель языка разметки гипертекста, а новая открытая платформа, предназначенная для создания веб-приложений использующих аудио, видео, графику, анимацию и многое другое.

Тег – набор ключевых слов, заключенных в скобки <>

одиночный тег:

<br> - здесь будет перевод строки

контейнерный тег:

<p>загадки разума</p>

Атрибуты тега – именованный массив параметров и их значений

<p class="pod">Александр Великий</p>

Вложенные теги:

правильно:

<p><strong>Что в имени тебе моем?</strong></p>

Несколько атрибутов:

<meta http-equiv="content-type" content="text/html">

Атрибут без значения:

<td nowrap>Пиррова победа</td>

Универсальные атрибуты:

id — уникальный идентификатор элемента;

class — имя класса стилевой таблицы;

src — источник встраиваемого содержимого;

href — ссылка на связанный ресурс.

 Различия: **HTML** построен на основе SGML, а **XHTML** — на основе XML. **XHTML** является более строгим по сравнению с **HTML** и не позволяет отступать от правил написания кода.

Согласно синтаксису XHTML:

• все элементы должны быть закрыты. Теги, которые не имеют закрывающего тега (например, *<br>*), должны завершаться символом */* (слеш): *<br />*;

• недопускаются атрибуты без содержания. Логические атрибуты всегда

должны иметь значения: *< td nowrap="nowrap">*;

• имена тегов и атрибутов должны быть записаны только строчными буквами;

• XHTML гораздо строже относится к ошибкам в коде, например, символы < и & везде, включая URL адреса, должны замещаться специальными сущностями &lt; и &amp; соответственно. По рекомендации W3C. браузеры, встретив ошибку в XHTML, должны сообщить о ней и не обрабатывать документ. В случае с HTML браузеры пытаются обработать страницу;

• кодировкой по умолчанию является UTF-8.

Инфраструктура локализации HTML считывает локализованные строки из файлов *локализации*. Файл локализации — это набор значений, привязанных к ключам и последовательно собранных в текстовый файл. Файл локализации иногда называют *пакетом*.

В версии HTML 4.0, опубликованной в 1999 г. произошло обновление стандарта. Многие элементы были отмечены как устаревшие и нерекомендованные, например, тег *font*, используемый для изменения свойств шрифта, был помечен как устаревший (вместо него рекомендовано использовать таблицы стилей CSS).

Устаревшие теги – это теги, которые не воспринимаются браузером.

Примеры:

<bgsound /> вместо него следует использовать тег audio

<dir> вместо него следует использовать тег ul

<frame />, <frameset>, <noframes> вместо них следует использовать тег iframe

Примеры устаревших атрибутов: *align (тип выравнивания),* clear (Устанавливает, с какой стороны элемента запрещено его обтекание другимиэлементами.)

**<meta>** определяет **метатеги**, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Разрешается использовать более чем один метатег, все они размещаются в контейнере **<head>**. Как правило, атрибуты любого метатега сводятся к парам «имя=значение», которые определяются ключевыми словами content, name или http-equiv.

# **2. Структурирование HTML текста. Абзацы, заголовки, списки. Гиперссылки и якоря.**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Заголовок страницы</title>

</head>

<body>

... содержание страницы ...

</body>

</html>

Согласно спецификации HTML 5 первым тегом документа является метатег *<!DOCTYPE html>*, после которого размещается сам HTML документ. Данный метатег представляет собой команду для корректного отображения документа в браузере. Тег *<html>* является главным корневым тегом веб-страницы, он содержит два контейнерных тега *<head>* и *<body>*, задающих заголовок страницы и ее содержание. Заголовок *<head>* содержит служебную часть, которая содержит различные теги, определяющие тип и формат содержимого для браузера (метатеги), а также ссылки на различное подгружаемое содержимое: каскадные стилевые таблицы CSS, скрипты со сценарием поведения Javascript и др. В данном примере указано минимальное содержимое веб-страницы в HTML 5: тег *<meta>* с помощью атрибута *charset* задает кодировку документа; тег *<title>* задает заголовок окна. Тело документа *<body>* представляет собой непосредственно выводимое содержи)мое на экран браузера.

**Абзацы**

Основной структурной единицей текста является абзац, выделяемый в контейнерный тег *<p>*.

*Простой абзац:*

*<p>абзац<p>*

HTML предлагает шесть **заголовков** разного уровня, которые показывают относительную важность секции, расположенной после заголовка. Так, элемент *<h1>* представляет собой наиболее важный заголовок первого уровня, а *<h6>* служит для обозначения наименее важного заголовка шестого уровня.

Элементы *<h1>*,...,*<h6>* относятся к контейнерным элементам, они всегда начинаются с новой строки, как и элементы, следующие за ними. До и после заголовка добавляется абзацный интервал.

Стандарт HTML позволяет представлять два вида **списков**: нумерованных и маркированных.

Нумерованный список задается контейнерным тегом *<ol>*. Списки состоят из произвольного количества вложенных элементов *<li>*, также представляющих собой контейнеры.

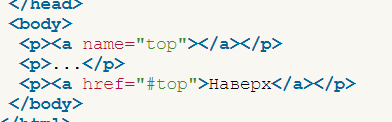
Маркированный список задается контейнерным тегом *<ul>*, содержащим элементы списка *<li>*.

**Гиперссылки** в HTML представляются строчным контейнерным тегом *<a>*. По умолчанию, гиперссылки отображаются в браузере подчеркнутым текстом синего цвета. Посещенные гиперссылки отображаются сиреневым цветом.

В качестве гиперссылок могут использоваться как абсолютные, так и относительные URL адреса. При использовании абсолютной адресации в URL рекомендуется указывать протокол.

**Якорем** называется закладка с уникальным именем на определенном месте веб-страницы, предназначенная для создания перехода к ней по ссылке. Якоря удобно применять в документах большого объема, чтобы можно было быстро переходить к нужному разделу.

Для создания якоря следует вначале сделать закладку в соответствующем месте и дать ей имя при помощи атрибута name тега **<a>**. В качестве значения href для перехода к этому якорю используется имя закладки с символом решетки (#) впереди.



# **3. Теги для оформления таблиц и блоков. Отступы. Границы. Табличная и блочная верстка.**

Таблица в HTML представляет собой отдельный блочный элемент *<table>*. В составе таблицы можно выделить отдельные структурные элементы: *<caption>* - заголовок; *<thead>* - начальный колонтитул; *<tbody>* - содержание; *<tfoot>* -конечный колонтитул. Все эти элементы являются опциональными и, заисключением тега *<caption>*, могут включать одну или несколько строктаблицы *<tr>*. Если в таблице содержание и колонтитулы не указаны, то строки *<tr>* могут включаться непосредственно в содержание тега *<table>*.

Каждая строка таблицы *<tr>* может включать одну или несколько ячеек. Ячейкимогут быть представлены двумя тегами: *<td>* - ячейка; *<th>* - заголовок столбца. По умолчанию текст заголовка столбца отображается в браузереполужирным ширфтом, выровненным по центру; текст простой ячейки) —выровненный по левому краю; текст заголовка — выровненным по центру.

Размер таблицы по умолчанию определяется ее содержимым, границы таблицыне отображаются, по вертикали содержимое всех ячеек выравнивается поцентру.

Таблица и ячейки внутри неё по-умолчанию отображаются в браузере без видимых границ.

**Границы таблицы** задаются свойством border. Для задания рамки вокруг блочного элемента используется атрибут *border*:

*border: < толщина> < стиль> < цвет>,* например:

*td { border: 1px solid black }*

Толщина рамки *< толщина>* задается атрибутом *border-width*, имеющим вид:

*border-width: <толщина1> [<толщина2>] [<толщина3>] [<толщина4>]*

В стандарте CSS используются два вида отступов:

• внутренний отступ, между содержимым и внешней границей элемента;

• внешний отступ, между элементом и соседними блочными элементами.

Величина внутренних отступов определяется свойством *padding*:

*padding: < отступ1> [<отступ2>] [<отступ3>] [<отступ4>]*

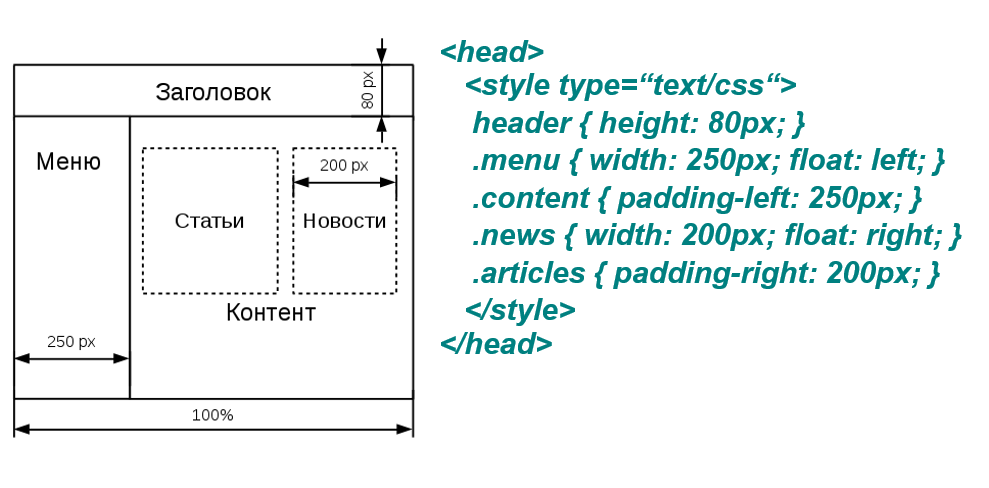
Если задан один параметр *<отступ1>*, то он применяется ко всем сторонам элемента; если) два параметра, то первый указывает верх-низ, второй — правую-левую стороны; если три — первый указывает верх, второй — правую-левую стороны, третий - низ; если четыре параметра — первый указывает верх, второй — правую сторону, третий — низ, четвертый — левую сторону. Пример:

*.indented { padding: 0cm 2cm 2cm 2cm }*

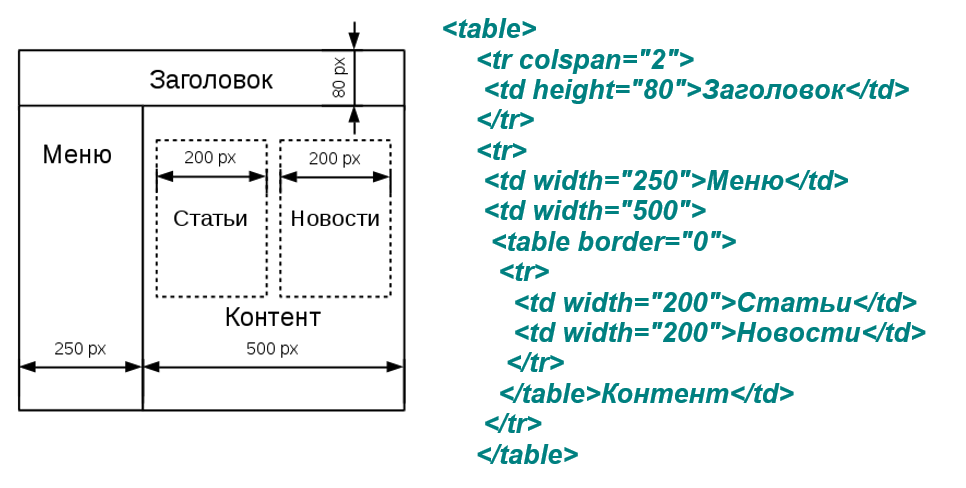
Величина внешних отступов определяется свойством margin: *margin: <отступ1>* *[<отступ2>] [<отступ3>]* *[<отступ4>]*

Параметры внешних отступов атрибута *margin* аналогичны параметрам внутренних отступов *padding*.

**Блочная верстка**

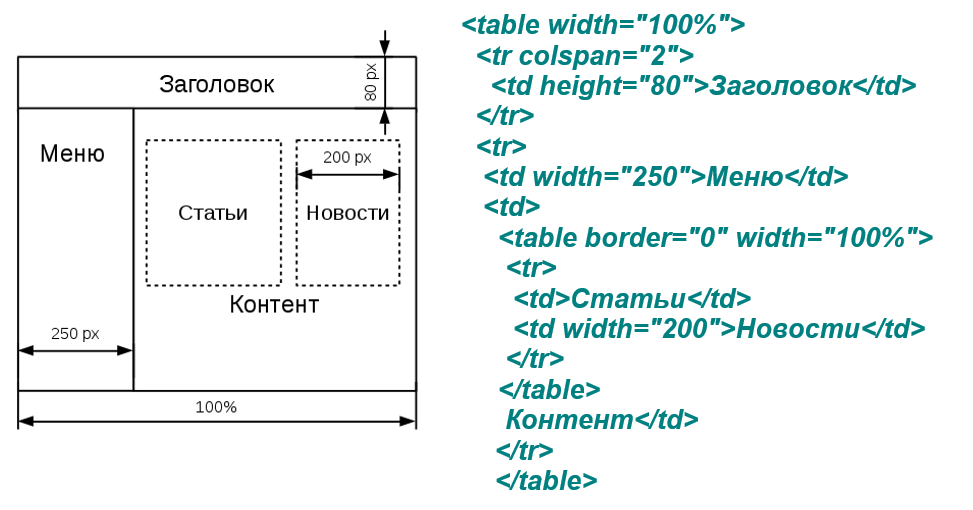


**Табличная верстка фиксированная**



|  |  |
| --- | --- |
| Достоинство фиксированного дизайна:  Везде выглядит одинаково  Недостатки:  Появление прокрутки на малом экране  Появление пустых мест на большом экране | Достоинство плавающего дизайна:  Подстраивается под размер экрана  Недостатки:  Выглядит по разному на разных устройствах  Содержимое плавающей колонки расползается |

**Табличная верстка фиксированная**



# **4. Графика и мультимедиа. HTML теги вставки изображений. Фон и оформление страницы.**

Для вставки изображения на страницу используется одиночный тег

*<img src="URL\_изображения">*

Некоторые используемые атрибуты:

*alt* — обозначение альтернативного текста, выводимого на месте изображения в браузере в случае отключения загрузки изображений.

*title* — текст подсказки, всплывающей при наведении указателя на изображение.

*<img src="images/flowers.jpg" title="цветы" alt="цветы">*

*Для определения холста изображения используется контейнерный тег <figure>*

Тег <figcaption> задает заголовок холста

<figure>  
 <img src="images/flowers.png" alt="">

<figcaption>Цветы</figcaption>

</figure>

Форматы графических данных:

GIF (Graphics Interchange Format) – 256 цветов, сжатие LZW, индексирование палитры, сохранение растра (устарел);

PNG (Portable Network Graphics) – 16M цветов, сжатие Deflate, индексирование палитры, сохранение растра;

JPEG (Joint Photographic Experts Group) – 16М цветов, сжатие JFIF, без палитры, оптимизация растра.

Форматы графических данных:

**SVG** (Scalable Vector Graphics) – векторная графика в формате XML;

Формат тега:

*<svg>... команды ...</svg>*

Атрибуты:   
width – ширина холста;

height – высота холста.

Для вставки изображения на страницу используется одиночный тег *img>*.

Обязательный атрибут *src* указывает URL изображения. Изображение будет

вставлено на место размещения тега в тексте HTML страницы. Тег *<img>*

является встроенным строчным элементом.

Некоторые используемые атрибуты:

*alt* — обозначение альтернативного текста, выводимого на месте изображения в

браузере в случае отключения загрузки изображений. Рекомендуемый атрибут.

*title* — текст подсказки, всплывающей при наведении указателя на изображение.

Пример:

*<img src="images/flowers.jpg" title="цветы" alt="цветы">*

В данном примере изображение содержится в файле *flowers.jpg* в каталоге

images на веб-сервере. Изображение загружается и выводится на экран браузера при отображении страницы.

Изображение также может быть содержимым гиперссылки. В этом случае активация перехода происходит при щелчке мыши на изображении:

*<a href="description.html"><img sr>c="tour.png" alt=""></a>*

Для определения холста изображения используется контейнерный тег семантической разметки *<figure>*, в который могут включаться теги изображения *<img>*, подписи *<figcaption>* и другие теги. Тег *<figcaption>* содержит текст подписи и также является контейнерным. Пример:

*<figure>*

*<img src="images/flowers.png" alt="">*

*< figcaption>Цветы</figcaption>*

*</ figure>*

Дополнительное оформление фона и стиля страницы с помощью изображений, а также выравнивание, обтекание текстом, изменение размера изображений рекомендуется осуществлять с помощью таблиц CSS.

# **5. Использование векторной графики SVG в HTML 5.**

Формат SVG (Scalable Vector Graphics ) - разработан в 2001 году для представления векторной 2D графики) в Интернете. Данные об объектах представляются в формате XML, версия SVG 1.1 поддерживается большинством современных браузеров. Отрисовка графических объектов производится средствами браузера на основе описанных в файле команд. Имеется поддержка анимации, векторных примитивов, градиентной заливки, стилизация средствами CSS, объектами SVG можно управлять с помощью Javascript. Достоинством является относительно малый размер файлов и наличие открытого интерфейса для взаимодействия объектов с пользователями.

**Достоинства SVG**

* Графика в формате SVG создается с использованием математических формул, которые при изменении размера изображения можно скорректировать. Таким образом векторные изображения масштабируются лучше, чем растровые.
* Размер векторной картинки обычно меньше, чем у сравнимых по качеству изображений в форматах JPEG, GIF или PNG.
* SVG графика имеет текстовый формат, который можно и править в блокноте, и рисовать в графических векторных редакторах Adobe Illustrator, CorelDRAW.
* Cкрипты и анимация в SVG позволяют создавать динамичную и интерактивную графику.
* Текст в графике SVG является текстом, а не изображением, поэтому он индексируется поисковыми системами.
* В SVG изображение можно добавить несколько ссылок.
* К SVG формату можно подключать внешние таблицы стилей CSS, глобальные стили внутри контейнера <style>...</style> или добавлять внутренние стили с помощью атрибута style в тегах фигур и путей.

**Векторная графика.**

Для вставки векторной графики в формате SVG на HTML страницу используется контейнерный тег *<svg>*, содержащий команды отрисовки объектов. На экране браузера тег SVG отображается в виде блочного объекта - холста. Ширина и высота холста может быть задана дополнительными атрибутами *width* и *height.*

Для задания ширины и цвета линий при отрисовке векторных объектов в командах используется атрибуты *stroke-width* — ширина линии в пикселах и *stroke* — цвет линии (в формате браузера), по умолчанию, черный. Атрибут *fill* задает цвет заполненной области (в формате браузера) или значение *none*, если заполнение не предусмотрено.

Список некоторых *SVG* команд и их атрибутов:

*<line>* - отображение прямой линии.Основные атрибуты:

*x1* и *y1* - координаты начальной точки);

*x2* и *y2* - координаты конечной точки;

*<polyline>* - отображение ломаной линии. Основные атрибуты:

*points* — пары координат опорных точек, разделенных запятыми и пробелами.

Например: *points="0,50 50,0 100,50".*

*<rect>* - отображение прямоугольника. Основные атрибуты:

*x, y* - координаты левого верхнего угла, *width* — ширина; *height* — высота.

*<circle>* - отображение окружности. Основные атрибуты:

*cx* и *cy* — координаты центра; *r* — радиус.

*<ellipse>* - отображение эллипса. Основные атрибуты:

*cx* и *cy* — координаты центра; *rx* — радиус по горизонтали; *ry* – радиус по вертикали.

Пример линии, красного треугольника и синей окружности:

*<svg width="120" height="150">*

*<line x1="0" y1="30" x2="100" y2="30" stroke-width="1" stroke="#000"/>*

*<polyline points="0,50 50,0 100,50" fill="rgb(120,0,0)"/>*

*<circle r>="50" cx="50" cy="100" fill="blue"/>*

*</svg>*

К объектам SVG также может быть применена трансформация. Для этого используются атрибуты

*transform-origin* и *transform* вида: *transform-origin="x y*" transform="функция()".

Вид и параметры функции аналогичны параметрам CSS трансформации.

# **6. HTML формы. Поля ввода. Кнопки, селекторы и другие интерактивные элементы.**

Для взаимодействия клиента с веб-сервером в HTML используются клиентские формы. Форма, имеющая поля ввода и другие элементы управления, заполняется клиентом с помощью браузера, проверяется обработчиком Javascript на стороне клиента и затем отправляется на веб-сервер, где управление передается отдельному серверному приложению, написанному на одном из серверных скриптовых языков, например, PHP. Для создания формы используется контейнерный тег *<form>*. Атрибут *action* указывает URL адрес серверного приложения.

Стандартные атрибуты формы:

*name* — имя формы. Используется для идентификации формы в Javascript.

*method* — метод HTTP отправки формы на сервер. Атрибут может при)ни)мать

значения *GET* или *POST*.

**Текстовое поле ввода**

Предназначение: ввод символов с клавиатуры. Пример:

*< input name="family" type="text">*

Некоторые используемые атрибуты:

*value* — значение поля по умолчанию.

*size* — ширина поля в моноширных символах.

*maxlength* — максимальное количество вводимых символов.

**Поле ввода пароля**

Предназначение: маскировка вводимых символов браузером. Пример:

*<input name="passw" type="password">*

Обратите внимание, что несмотря на маскировку символов браузером, при отправке данных на сервер значения полей ввода паролей передаются по протоколу HTTP в незашифрованном виде. Для передачи конфиденциальной информации используйте шифрование трафика SSL (*Secure Socket Layer*) и протокол HTTPS.

**Скрытое поле ввода**

Предназначение: передача не отображаемых браузером данных на сервер.

Может включать атрибут *value*, содержащий предустановленное значение.

Пример:

*< input name="userid" type="hidden" value="135">*

**Текстовая кнопка**

Предназначение: связывает событие *onclick* элемента с обработчиком Javascript.

Используется для предваритель ной обработки данных формы перед отправкой их на сервер. Пример:

*<input name="btn" type="button" value="Ввод">*

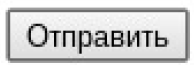
Некоторые используемые атрибуты:

*value* – текст кнопки, отображаемый браузером;

*onclick* – функция обработчика события Javascript.

**Кнопка отправки данных**

Предназначение: отправка данных формы на сервер. Пример:

*<input name="btn" type="submit">*

Атрибуты те же, что и у обычной кнопки.

**Кнопка сброса**

Предназначение: возвращение данных формы в первоначальное состояние.

Пример:

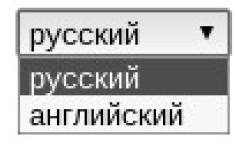
*<input name="btn" type="reset">*

Атрибуты те же, что и у обычной кнопки.

**Выпадающий список**

Предназначение: выбор одного или нескольких элементов из заданного списка. Список задается контейнерным тегом *<select>*, включающим контейнерные элементы *<option>* с текстовым содержимым. При выборе элемента из списка,

его значение value устанавливается для контейнера *<select>* и передается на

сервер. Пример использования:

*< select name="lang">*

*< option value="RU">русский</option>*

*< option value="EN">английский</ option>*

*</ select>*

**Комбинированный список**

Предназначение: выбор при наборе в текстовом поле. Список задается контейнерным тегом *<datalist>*, включающим контейнерные элементы *<option>* с текстовым содержимым. С помощью атрибута *id* тега *<datalist>* список связывается с текстовым полем *<input>*, имеющим соответствующий атрибут *list*. Список становится доступным при получении полем фокуса или при наборе текста.

**Поле выбора чисел из диапазона**

Предназначение: выбор целых чисел из диапазона. Отображает ползунок, изменяющий значение на один шаг.

**Поле ввода чисел**

Предназначение: ввод целых чисел. Содержит кнопки-стрелки вверх-вниз, изменяющие введенное значение на один шаг. Пример:

*<input name="num" type="number">*

Дополнительные атрибуты: *value* – значение по умолчанию; *min* – минимальное значение; *max* – максимальное значение; *step* – шаг приращения (по умолчанию, 1).

**Флажок выбора опций (***checkbox)*

Предназначение: установка и снятие флажка опции. Используется при выборе

нескольких элементов из предложенных. Если при отправке формы флажок не

установлен, данные элемента не передаются.

**Переключатель (***radio)*

Предназначение: выбор одного из нескольких предложенных элементов из группы. Все элементы переключателя должны иметь одинаковое имя name.

Если при отправке формы переключатель не установлен, его данные не передаются.

**Поле для отправки файлов (***attach)*

Предназначение: прикрепление содержания файла к отправляемой форме. Форма должна отправляться на сервер с помощью метода *POST*, способ кодрования данных *enctype* должен быть установлен как *multipart/form-data*. При нажатии на кнопку *"Обзор"* в браузере открывается диалог выбора файла. После выбора файла его имя появляется в поле ввода, а содержи мое прикрепляется к форме и подготавливается для отправки на сервер.

# **7. Способы стилизации элементов страницы. Обзор стандарта CSS3. Наследование классов и атрибутов. Приоритет стилей. Комбинаторы и селекторы. Псевдоэлементы и псевдоклассы. Стилизация для различных устройств.**

**Каскадной стилевой таблицей** CSS (Cascading Style Sheets) называется набор правил - параметров форматирования, которые применяются к элементам документа, чтобы изменить их внешний вид.

***<head>***

***<style type=“text/css“>***

***header { height: 80px; } // высота 80 px***

***menu { width: 250px; } // ширина 250 px***

***</style>***

***</head>***

**Внешняя таблица стилей:**

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

***</head>***

**Локальная таблица стилей:**

***<div style="text-align: center">Текст по центру</div>***

**Порядок применения стилей:**

* ***Внешние стили.*** Стиль определяется во внешнем CSS файле.
* ***Страничные стили.*** Стиль определяется внутри контейнерного тега вида: *<style type="text/css">*, размещаемого в заголовке HTML документа. Переопределяет внешние стили. Правило распространяется на весь HTML документ.
* ***Локальные стили.*** Стиль определяется в атрибуте *style* тега.

В рассматриваемой в настоящее время спецификации CSS 3 добавлено управление отображением векторной графики SVG, трансформацией и анимацией без использования языка Javascript.

**Наследованием** называется перенос форматирования для элементов, находящихся внутри других

*table { color: red; }* - цвет для всех ячеек таблицы

*body { font-size: 12pt; }* - шрифт для всего текста

Наследование применяется не для всех тегов

Приоритеты наследования:

1. Стиль браузера (по умолчанию)

2. Стиль автора (CSS оформление)

3. Стиль пользователя (из настроек в браузере)

4. Стиль автора с добавлением !important

5. Стиль пользователя с добавлением !important

**Селекторы**

Описание стилей:

*селектор1, селекторN {*

*свойство1: значение1;*

*свойство2: значение2;*

*}*

Селектор применяется к одному или нескольким тегам страницы:

*div { text-align: center; }* // для всех тегов div

**Комбинаторы** — разновидность специальных селекторов, привязывающая стиль к элементу веб-страницы на основании его местоположения относительно других элементов.

**Комбинаторы**, как и следует из названия, предназначены для сочетания (комбинирования) различных селекторов

В CSS есть четыре разных типа комбинатора:

* селектор потомок (пробел)
* селектор дочерний (>)
* селектор соседний родственный (+)
* селектор общий (~)

Псевдоклассы – специальный вид классов

*селектор:псевдокласс { свойство: значение; }*

* определяют динамическое состояние элемента
* относятся к дереву элементов
* задают язык текста

Псевдоклассы, определяющие динамическое состояние элементов

* *a:active { color: #f00; }* - активная ссылка
* *a:link { color: #f00; }* - непосещенная ссылка
* *a:visited { color: #f00; }* - посещенная ссылка

Псевдоклассы относящиеся к дереву элементов

* *b:first-child { color: #f00; }* - первый элемент из списка
* *<p><b>Дихотомический метод</b> заключается в разбиении <b>выделенной</b> области решений пополам</p>*

Псевдоэлементы определяют новое содержимое

* *p::after { content: “новое содержимое”; }* - вставка после
* *p::before { content: “\20aa”; }* - вставка до абзаца

# **8. Стилизация CSS. Параметры шрифта и фона. Блоки. Параметры абзацев, списков и отображения. Курсоры.**

*Параметры шрифта*

Для стилизации текста могут использоваться как универсальные, так и специальные свойства, задающие характеристики шрифта и текста. По умолчанию все стили шрифта принимают значение inherit (наследуемое). Стиль шрифта может быть задан универсальным составным свойством font. Упрощенный вид свойства:

font: [<наклон>] [<строч>] [<толщина>] [<размер> [/h\_стр]] <имя>

Обязательно указывается имя шрифта. например: p { font: bold italic 14pt sans-serif; }

Для абзаца значение bold устанавливает жирное начертание, italic — курсив, размер шрифта 14pt и семейство шрифтов sans-serif. Каждое из этих свойств может быть задано отдельной строкой, например:

p

{ font-style: italic;

font-weight: bold;

font-size: 12pt;

font-family: sans-serif; }

Если имя шрифта содержит пробелы, то оно заключается в кавычки. Пример:

h1 { font-family: "Times New Roman", serif }

Параметр <наклон> устанавливается атрибутом font-style. Возможные значения: normal (по умолчанию, italic курсив и oblique (небольшой наклон).

Параметр <строч> устанавливается атрибутом font-variant. Возможные значения: normal (по умолчанию), small-caps - малые строчные буквы.

Толщина шрифта <толщина> устанавливается атрибутом font-weight.

Размер шрифта <размер> устанавливается атрибутом font-size. Размеры шрифтов, а также других элементов CSS, можно задать в абсолютных и относительных единицах: px — пикселах, pt — типографских пунктах, cm — сантиметрах, mm — милиметрах.

Высота строки текста <h\_стр> устанавливается с помощью атрибута line- height. При этом можно задать как абсолютное, так относительное значение. Пример двойной высоты строки: p { line-height: 2 }

Атрибут стиля color задает цвет текста. Цвет в CSS может задаваться в виде имени, полных #HHHHHH или сокращенных #HHH шестнадцатеричных чисел, а также в виде функций rgb(r, g, b)

h1 { color: #f00 } /\* Красный цвет. То же: red, #ff0000, rgb(255,0,0) \*/

***Параметры абзацев***

Для блочных элементов можно задать параметры горизонтального и вертикального выравнивания. Для горизонтального выравнивания текста используется атрибут text-align. Возможные значения: left — по левому краю (по умолчанию), right — по правому краю, center — по центру, justify — по ширине. Кроме этого, можно задать отступ для красной строки абзаца с помощью атрибута text-ident, например:

p { text-ident: 1cm }

Для вертикального выравнивания элементов используется атрибут vertical- align. Он определяет выравнивание текущего фрагмента текста относительно родительского элемента. Возможные значения: baseline — по базовой линии текста (по умолчанию); top — по верхней границе; middle — по центру; bottom — по нижней границе; text-top и text-bottom — по верхней или нижней границе текста; sub и super> — по базовой линии нижнего или верхнего индекса.

***Параметры фона***

Фон можно указывать для фрагмента текста, блочного элемента, таблицы, строки или ячейки таблицы или для всей страницы. Для задания фона элемента обычно указывается цвет или фоновое изображение. Атрибут background-color устанавливает цвет фона. Значение атрибута transparent устанавливает прозрачный фон. По умолчанию цвет страницы задается браузером. Пример задания желтых букв на синем фоне:

body { color: yellow; background-color: blue }

Атрибут background-image устанавливает фоновое изображение. Возможные значения атрибута: none (по умолчанию) или) url(URL\_адрес\_изображения). Форматы GIF и PNG обеспечивают поддержку прозрачности.

Пример: body { background-color: yellow;

background-image: url("images/fon.png") }

***Параметры списков***

Для задания вида маркеров или нумерации пунктов списка используется атрибут list-style-type. *Возможные значения для маркированных списков:* disk — маркер в виде символа круга (по умолчанию), circle — маркер в виде символа окружности, square — маркер в виде квадрата. *Возможные значения для нумерованных списков:* decimal — десятичная нумерация (по умолчанию), decimal-leading-zero — десятичная с начальным нулем, lower-alpha и lower- latin — строчные латинские буквы, upper-alpha и) upper-latin — заглавные латинские буквы, lower-roman и upper-roman — строчные и заглавные римские цифры. Для отмены маркировки и нумерации списка используется значение none.

***Параметры курсора***

Форма и вид курсора, отображаемого при наведении на определенный элемент страницы, устанавливается с помощью атрибута cursor. Наиболее часто используемые значения: auto — устанавливаемое автоматически браузером по умолчанию, default — указатель стрелка, none — курсор не отображается, help — стрелка с вопросительным знаком, pointer— указующий перст (используется при наведении на гиперссылку), progress — стрелка с песочными часами (используется при ожидании), wait — песочные часы (используется при ожидании), text — текстовый курсор (используется при вводе текста).

***Параметры и вид отображения***

Вид отображения элемента устанавливаются с помощью атрибута display. Это многоцелевое свойство, которое определяет, как элемент должен быть показан в документе. Оно устанавливает, будет ли элемент блочным, строчным, строчно- блочным и т.д. Возможные значения: inline — строчный элемент (по умолчанию для строчных элементов); block — блочный элемент (по умолчанию для блочных элементов); none — не отображается, место под него не резервируется; inline-block — строчно-блочный элемент, комбинирует внутреннее поведение блочного и внешнее поведение строчного элемента; list- item — элемент списка; table — таблица; inline-table — таблица с внешним поведением строчного элемента; table-caption — заголовок таблицы; table- column — столбец таблицы; table-row — строка таблицы; table-cell — ячейка таблицы; table-column-group — группа столбцов таблицы.

Наиболее часто используется свойство none, позволяющее делать элемент невидимым: .hidden { display: none }

***Блоки***

Позиция блочного элемента на странице обеспечивается его поведением, заданным с помощью атрибута position. Возможные значения: static — блок не позиционируемый (по умолчанию); absolute — абсолютно позиционируемый, место на странице не выделяется. Если родительский элемент позиционирован, то текущий элемент позиционируется от него, иначе — от края экрана; fixed — подобный абсолютному позиционированию, но не меняет позицию элемента при прокрутке содержимого родительского элемента; relative — относительно позиционируемый, положение элемента устанавливается относительно его исходного места, место под элемент выделяется. Для установки позиции элемента используются атрибуты top, left, right и bottom, устанавливающие расстояние от верхней, левой, правой или нижней

границы текущего элемента до соответствующей границы родительского элемента или экрана. Значение auto вышеуказанных атрибутов не изменяет положение текущего элемента. Пример — левый верхний угол элемента с

id="search" на экране имеет координаты (200, 100):

#sear>ch { position: absolute; left: 200px; top: 100px }

# **9. Основы дизайна страниц. Стилизация CSS. Свойства фона. Вертикальное и горизонтальное меню. Стилизация блочных и табличных элементов. Отступы. Границы. Выделения.**

*Стилизация CSS*

· Внешние стили. Стиль определяется во внешнем CSS файле.

· Страничные стили. Стиль определяется внутри контейнерного тега вида: <style type="text/css">, размещаемого в заголовке HTML документа. Переопределяет внешние стили. Правило распространяется на весь HTML документ.

· Локальные стили. Стиль определяется в атрибуте style тега.

*Параметры фона*

Фон можно указывать для фрагмента текста, блочного элемента, таблицы, строки или ячейки таблицы или для всей страницы. Для задания фона элемента обычно указывается цвет или фоновое изображение. Атрибут background-color устанавливает цвет фона. Значение атрибута transparent устанавливает прозрачный фон. По умолчанию цвет страницы задается браузером. Пример задания желтых букв на синем фоне:

body { color: yellow; background-color: blue }

Атрибут background-image устанавливает фоновое изображение. Возможные значения атрибута: none (по умолчанию) или) url(URL\_адрес\_изображения). Форматы GIF и PNG обеспечивают поддержку прозрачности.

Пример: body { background-color: yellow;

background-image: url("images/fon.png") }

*Параметры отступов*

В стандарте CSS используются два вида отступов:

• внутренний отступ, между содержимым и внешней границей элемента;

• внешний отступ, между элементом и соседними блочными элементами

Величина внутренних отступов определяется свойством padding:

padding: <отступ1> [<отступ2>] [<отступ3>] [<отступ4>]

Если задан один параметр <отступ1>, то он применяется ко всем сторонам элемента; если два параметра, то первый указывает верх-низ, второй — правую левую стороны; если три — первый указывает верх, второй — правую-левую стороны, третий - низ; если четыре параметра — первый указывает верх, второй — правую сторону, третий — низ, четвертый — левую сторону.

Пример: .indented { padding: 0cm 2cm 2cm 2cm }

Атрибуты внутренних отступов могут быть указаны отдельно: padding-left, padding-top, padding-right, padding-bottom.

Пример: .indented { padding-left: 2cm }

Величина внешних отступов определяется свойством margin: margin: <отступ1> [<отступ2>] [<отступ3>] [<отступ4>]

Параметры внешних отступов атрибута margin аналогичны параметрам внутренних отступов padding.

Пример задания отступов сверху и снизу: h1 { margin: 5mm 0mm }

Атрибуты внешних отступов могут быть указаны отдельно:

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

Например: h1 { margin-top: 5mm }

Необходимо отметить, что для таблицы внешние отступы ячеек задаются отдельным атрибутом border-spacing, вида: border-spacing: <отступ1> [<отступ2>]

Если задан один параметр <отступ1>, то он применяется ко всем сторонам ячейки таблицы;

если два параметра, то первый указывает отступ слева-справа, второй — верх-низ.

*Параметры рамок*

Для задания рамки вокруг блочного элемента используется атрибут border:

border: <толщина> <стиль> <цвет>, например:

td { border>: 1px solid black }

Толщина рамки <толщина> задается атрибутом border>-width, имеющим вид:

border>-width: <толщина1> [<толщина2>] [<толщина3>] [<толщина4>]

Параметры толщины атрибута border>-width применяются к границам рамки в последовательности, аналогичной параметрам отступов. Значения толщины могут задаваться одной из констант: thin — тонкая, medium — средняя, thick — толстая или одним из размерных значений CSS.

Пример задания тонкой границы рамки сверху и снизу: h1 { border>-width: thin 0 } Атрибуты толщины границы также могут быть указаны отдельно: border>-leftwidth — для левой границы, border-top-width — для верхней границы, border - right-width — для правой границы, border-bottom-width — для нижней границы.

Например, для нижней границы рамки: h1 { border>-bottom-width: 2px } /\* толщина 2px \*/

Стиль рамки <стиль> задается атрибутом border-style, имеющим вид: border-style: <стиль1> [<стиль2>] [<стиль3>] [<стиль4>]

Параметры стиля атрибута border-style применяются к границам рамки в последовательности, аналогичной параметрам отступов. Значения стиля могут задаваться одной из констант: none или hidden — рамка отсутствует (по умолчанию), solid — сплошная линия, dotted — пунктирная линия, dashed — штриховая линия, double — двойная линия, groove и ridge— вдавленная или выпуклая 3D линия, inset и outset — вдавленная или выпуклая 3D плоскость.

Пример задания выпуклой 3D границы рамки вокруг всего элемента: h1 { border>-style: ridge } Атрибуты стиля границы также могут быть указаны отдельно: border-left-style — для левой границы, border-top-style — для верхней границы, border-right-style — для правой границы, border>-bottom-style — для нижней границы.

Например, для нижней границы рамки): h1 { border>-bottom-style: solid } /\* сплошная линия \*/ Цвет рамки <цвет> задается атрибутом border-color, имеющим вид: border-color>: <цвет1> [<цвет2>] [<цвет3>] [<цвет4>] Параметры цвета атрибута border-color применяются к границам рамки в последовательности, аналогичной параметрам отступов.

Пример: h1 { border-color: lightgreen } /\* светло-зеленый цвет \*/

Атрибуты цвета границы также могут быть указаны отдельно: border-left-color> — для левой границы, border-top-color> — для верхней границы, border-rightcolor — для правой границы, border-bottom-color— для нижней границы.

Например, для нижней границы рамки: h1 { border>-bottom-color>: #0F0 } /\* светло-зеленый цвет \*/ Радиус закругления углов рамки указывается с помощью атрибута border-radius, имеющим вид: border-radius: <r1> [<r2>] [<r3>] [<r4>] Параметры радиусов атрибута border>-radius применяются к границам рамки в последовательности аналогичной параметрам отступов, начиная с верхнего левого угла. Каждый параметр может быть как числом, указывающим радиус круга, так и числовой дробью, указывающей размеры полуосей эллипса. Размеры могут быть заданы как в размерных единицах CSS, так и)в процентах от ширины блока.

Пример: h1 { border-radius: 2px/3px } /\* радиус углов рамки 2 на 3 пиксела \*/ *Параметры выделения*

Для задания выделения блочного элемента используется атрибут outline. Выделение представляет собой границу, окружающую данный элемент. В отличие от рамки, выделение не увеличивает размер элемента. Параметры выделения такие же, как и параметры рамок. outline: <толщина> <стиль> <цвет>, например: p { outline: thin dotted black } Для задания параметров выделения также можно использовать отдельные свойства: outline-width, outline-color>, outline-style. Параметры свойств выделения аналогичны параметрам свойств рамок.

# **10. Стилизация CSS. Виды позиционирования и выравнивания элементов. Видимость элементов. Слои. Прозрачность. Текстовые и графические эффекты. Анимация.**

Позиция блочного элемента на странице обеспечивается его поведением, заданным с помощью атрибута position. Возможные значения: static — блок не позиционируемый (по умолчанию); absolute — абсолютно позиционируемый, место на странице не выделяется. Если родительский элемент позиционирован, то текущий элемент позиционируется от него, иначе — от края экрана; fixed — подобный абсолютному позиционированию, но не меняет позицию элемента при прокрутке содержимого родительского элемента; relative — относительно позиционируемый, положение элемента устанавливается относительно его исходного места, место под элемент выделяется. Для установки позиции элемента используются атрибуты top, left, right и bottom, устанавливающие расстояние от верхней, левой, правой или нижней

границы текущего элемента до соответствующей границы родительского элемента или экрана. Значение auto вышеуказанных атрибутов не изменяет положение текущего элемента. Пример — левый верхний угол элемента с

id="search" на экране имеет координаты (200, 100):

#sear>ch { position: absolute; left: 200px; top: 100px }

*Видимость элементов*

Область видимости позиционированного элемента определяется атрибутом clip, задающим прямоугольник маски и имеющим вид: clip: rect(<верх>,<право>,<низ>,<лево>)

Параметры <верх>,<право>,<низ> и <лево> определяют расстояние от верхней, правой, нижней и левой границы текущего элемента до области маски. Прямоугольник маски задает видимую область элемента. Значение auto атрибута clip убирает маску и делает видимым весь элемент. Пример:

#search { position: absolute; left: 200px; top: 100px

clip: rect(50px, 20px, 0px, 50px) }

*Текстовые и графические эффекты*

**Градиенты**

Функция linear-gradient() добавляет линейный градиент к фону элемента. Она выступает значением свойства background-image или background. Вид:

background-image: linear-gradient( <направл>, <цветн>, <цветк>);

Параметры <цветн> и <цветк> задают начальный и конечный цвет градиента с необязательной позицией цвета относительно оси градиента в процентах. Параметр <направл> задает направление градиента. Направление может задаваться с помощью угла наклона deg, отсчитываемого от вертикали по часовой стрелке, напри)мер 45deg

**Трансформация**

Трансформации позволяют выполнять над элементом различные геометрические преобразовани)я: масштабирование, переносы, повороты и др. Для этого используется атрибут transform

Вид атрибута transform: transform: <функция>

Параметр <функция> определяет функцию трансформации. Для отмены трансформации используется значение none. Виды функций трансформации: rotate(<угол>) — поворот элемента на заданный угол по часовой стрелке, относительно точки привязки, заданной дополнительным атрибутом transform- origin. Обычно значения атрибута определяются координатами, задаваемыми в виде двух чисел или констант: left, center, right – по горизонтали и top, center, bottom – по вертикали

*Анимация*

Анимация по кадрам К элементам страниц можно применить вид анимации по ключевым кадрам. Для этого используется атрибут animation.

Общий вид атрибута animation: animation: <переменная> [<время>] [<функция>] [<задержка>] [<итер>] [<направление>] [<режим>] [<состояние>]

Кроме параметра <переменная>, каждый из этих параметров может быть в любой комбинации установлен с помощью отдельных атрибутов: animation-duration, animation-timing-function, animation-delay, animation-iteration- count, animation-direction, animation-fill-mode и animation-play-state.

Параметр <время> задается атрибутом animation-duration и определяет время действия цикла анимации в секундах s или миллисекундах ms, например animation-duration: 3s — три секунды.

Параметр <функция> задается атрибутом animation-timing-function и определяет временную функцию анимирования.

Параметр <задержка> задается атрибутом animation-delay и определяет время ожидания перед выполнением анимации (по умолчанию 0).

Параметр <итер> задается атрибутом animation-iterаtion-count и определяет число итераций (1 по умолчанию). Бесконечное число итераций задается константой infinite.

Параметр <направление> задается атрибутом animation-direction и определяет направление движения анимации.

Параметр <режим> задается атрибутом animation-fill-mode и определяет состояние элемента после окончания анимации. Возможные значения: none — стили не применяются, по умолчанию элемент будет находится в исходном состоянии; forwards — к элементу применяется стиль последнего ключевого кадра; backwards — к элементу применяется стиль первого ключевого кадра.

Параметр <состояние> задается атрибутом animation-play-state и определяет состояние анимации. Возможные значения: running — проигрывать анимацию (по умолчанию); paused — приостановить анимацию.

# **11. Язык сценариев JavaScript. Синтаксис языка. Переменные. Операторы. Операции. Функции. Массивы. Ссылки. Объекты.**

Язык Javascript является объектно-ориентированным интерпретатором, встроенным в браузер, и служит для обработки программных сценариев страницы.

В основе синтаксиса языка Javascript лежит интерпретируемый код, заключенный в функции — обработчики. Код скриптов обычно загружается вместе с загрузкой страницы и размещается в специальном контейнерном теге:

<script>

alert('Hello, world!');

</script>

Контейнер <script> чаще всего размещается в заголовке страницы <head> и сразу же интерпретируется.

Часто скрипты размещают в отдельных текстовых файлах с расширением .js которые подключают к контейнеру <script> с помощью атрибута src: <script src="js/form.js"></scr>ipt>

Синтаксис сценариев на языке Javascript включает набор команд, разделенных символом ; (точка с запятой): alert('Hello'); alert('world!!!'); // два текстовых сообщения

*Переменные*

Для объявления переменных в Javascript используется ключевое слово var:

var <идентификатор> [=<значение>];

Идентификатор представляет собой имя переменной, состоящее из латинских

букв, цифр, символов: $ (денежный знак) и \_ (подчеркивание), начинающееся не с цифры.

В качестве идентификаторов нельзя использовать зарезервированные команды языка. При объявлении нескольких переменных, идентификаторы разделяются запятым). Пример:

Var author = 'Ivan', age = 25, message\_12 = 'Hello!';

Традиционно в Javascript заглавными буквами объявляются константы: var GREEN = '#00FF00';

*Операторы*

Язык Javascript имеет С-подобные операторы и команды.

**Оператор присвоения** служит для присвоения значений переменным: c = a + b; // присвоение суммы a и b

c = d = x; // сначала d = x, потом c = d

**Арифметические операторы**: + (сложение), - (вычитание), \* (умножение), / (деление), %HH (остаток от деления) подобные используемым в других языках.

**Операторы сравнения**: < (меньше), > (больше), == (равно), != (не равно), <>= (меньше или равно), >= (больше или равно), === (строго равно), !== (строго не равно). Последние два оператора отличаются от обычного равенства и неравенства тем, что при их выполнении над аргументами не производится преобразование типов.

Пример: c = (2 == '2'); // результат c = true

c = (2 === '2'); // результат c = false

**Логические операторы**: && (логическое И, конъюнкция), || (логическое ИЛИ, дизъюнкция), ! (логическое НЕ, отрицание).

**Унарные операторы** представляют собой операторы с одним аргументом: + (унарный плюс), - (унарный минус), ++ (инкремент), -- (декремент), ! (логическое отрицание), ~ (побитовое НЕ).

**Простой условный оператор** (если-то) имеет вид: if (<выражение>) <действие>; Если) выражение в круглых скобках истинно, то выполняется действие. Пример: if (a < 0) alert("Число а - отрицательное!");

if (<выражение>) <действие1> else <действие2>;

**Оператор параметрического цикла** имеет вид: for ([<начало>]; [<условие>]; [<шаг>]) <тело цикла>;

**Оператор цикла с предусловием** имеет вид: while (<условие>) <тело цикла>;

*Операции*

Ввод — вывод значений

1. Функция alert выводит информационное сообщение. Формат: alert(<сообщение>); Пример: alert("Сообщение"); // вывод информационного сообщения
2. Функция prompt выводит окно с сообщени)ем и полем ввода. Формат: <переменная> = prompt(<сообщение>, <значение\_по\_умолч>); Пример: var year> = prompt("Сколько вам лет?", 100);
3. Функция confirm выводит запрос подтверждения с двумя кнопками. Формат: <переменная> = confirm(<запрос>); Пример:

Var admin = confirm("Вы - администратор?");

*Массивы*

Простой массив в языке Javascript — это переменная, содержащая однотипные элементы.

Нумерация элементов массива начинается с 0.

Пример:

var empty = [ ]; // пустой массив

var odd = [0, 2, 4, 6, 8]; // массив из 5 чисел

var list = ["Иван", "Андрей", "Наталья"]; // массив из 3 слов

var c = odd[2]; // значение с = 4;

alert(list[1]); // вывод "Андрей"

var a = new Array(2, 3, 4); // массив из 3 чисел

К элементам массива также можно обращаться с помощью именованных ключей. Такой массив называется ассоциативным.

var earth = { name: "Земля", period: 36>5.25, radius: 6378};

может быть объявлен с помощью конструктора: var earth = new Array();

earth.name = "Земля";

earth.radius = 6378;

alert(earth.name); // вывод Земля

alert(earth["radius"]); // вывод 6378

Для перебора всех ключей ассоциативного массива используется цикл for и команда in: for (key in earth)

*Функции*

Функции являются основными строительными блоками программы.

Пользовательские функции требуют обязательного объявления:

function <имя\_функции> ([<аргументы>,...]) { <тело функции> }

Функция вызывается по ее имени, например:

// объявление функции

function sum(a, b) { // сумма двух чисел

retur>n (a+b); // возвращаемое значение }

var c = sum(2,3); // вызов функции

alert("результат:" + c); // результат: 5

Если аргументы не заданы, то они принимают значения undefined.

Переменные, объявленные вне функции, являются глобальными.

Функциональное выражение, которое не присваивается переменной называется анонимной функцией. Анонимная функция вызывается в том месте, где она описана. Пример:

function ask(question, yes, no) { // описание функции

if (confirm(question)) yes();

else no(); }

ask("Вы согласны?", // вызов функции

function() { alert("Да"); }, // анонимная функция

function() { alert("Нет"); } // анонимная функция );

*Объекты*

Для описания пользовательского объекта используются фигурные скобки { }, в которых перечисляются свойства, подобно ассоциативному массиву:

person = { name: "Иван", family: "Иванов", age: 20 };

Также для описания объекта можно использовать встроенный класс Object:

person = new Object(); person.name = "Иван";

person.family = "Иванов";

person.age = 20;

После описания класса экземпляры объектов можно задавать с помощью команды new:

var alex = new Person("Алексей", "Сидоров", 22);

Для добавления нового свойства заданного объекта достаточно указать его в программе: alex.group = 144; // появилось новое свойство group

Для удаления свойств заданного объекта можно использовать команду delete:

delete alex.group; // удалено свойство group

Для удаления объекта целиком используется указатель на пустую строку null:

alex = null; // удаление объекта alex

**Объект Number.**

Представляет собой обычное число. Основные методы:

Number(<параметр>) — представляет собой конструктор числа. В качестве параметра может передаваться любое значение. Пример: var a = new Number("12.24"); // a = 12.24

**Объект String.** Представляет собой текстовую строку, заключенную в кавычки или апострофы. При необходимости, строка может содержать управляющие последовательности: \n — перевод строки), \DDD — десятичный код символа \ xHH — шестнадцатиричный код символа.

Основные методы и свойства: String(<параметр>) - представляет собой конструктор строки. В качестве параметра инициализации может передаваться любое значение.

Пример: var str = new String(12.45);

alert(str.length); // вывод 5

Свойство length определяет длину строки в символах:

"Привет!".length; // 7

Обращение по индексу строки) [i] возвращает символ на позиции i. Нумерация позиций начинается с 0: "Привет!"[3]; // символ "в"

**Объект Array.**

Представляет собой массив. Основные свойства и методы: Array(<n>) - конструктор массива из n элементов. Если параметр n является числом, то резервируется память для элементов массива. Все элементы принимают значения undefined.

**Объект Date.** Представляет собой дату. Основные свойства и методы: Date(<дата>) - конструктор даты.

# **12. Подключение скриптов на странице. Обработка событий. Область видимости переменных. Замыкания. Ввод и вывод данных. Отладка программ.**

**. Ввод и вывод данных**

1. Функция alert выводит информационное сообщение. Формат: alert(<сообщение>); Пример: alert("Сообщение"); // вывод информационного сообщения
2. Функция prompt выводит окно с сообщени)ем и полем ввода. Формат: <переменная> = prompt(<сообщение>, <значение\_по\_умолч>); Пример: var year> = prompt("Сколько вам лет?", 100);
3. Функция confirm выводит запрос подтверждения с двумя кнопками. Формат: <переменная> = confirm(<запрос>); Пример:

Var admin = confirm("Вы - администратор?");

**Отладка программы**

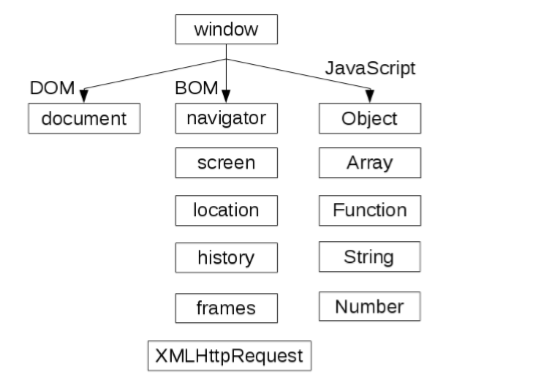
Для отладки Javascript программы и вывода значений переменных обычно используется функция alert, выводящее текстовое сообщение. Однако в некоторых случаях вывод окна влияет на поведение сайта и такой способ является нежелательным. Гораздо более удобным и гибким инструментом является вывод с помощью встроенного метода console.log(<сообщение>). С помощью этого метода можно выводить сообщения в отладочную панель браузера (для этого она должна быть предварительно включена). Пример: console.log("Тестовое сообщение");

Для вывода значений объектов и массивов программы также можно использовать метод console.dir(<bпеременная>).

# **13. Объектная модель браузера**

**Объектная модель браузера.**

Javascript использует стандартную объектную модель, вершиной которой является объект window. Он также является главным объектом всей иерархии Javascript.



Объектная модель браузера (Browser Object Model, BOM) состоит и)з следующих основных объектов:

1. window - окно браузера.
2. navigator - объект свойств браузера и операционной системы
3. screen - объект экрана
4. location - URL адрес загруженной HTML страницы
5. history - объект списка посещенных страниц браузером

**Таймер**

Для управления временем в Javascript имеется несколько полезных глобальных методов, которые с успехом применяются при анимации объектов.

<переменная> = setTimeout(<функция>, <задержка>- устанавливает пользовательскую функцию, которая будет выполнена с заданной задержкой (в миллисекундах).

Пример:

var func = function() { alert("1 секунда"); };

setTimeout(func, 1000); // вывод сообщения через 1 секунду

clearTimeout(<переменная>) - отменяет выполнение пользовательской функции, установленной методом setTimeout. Переменная представляет собой идентификатор таймера setTimeout.

Пример: var timer = setTimeout(function() { aler>t("1 секунда") }, 1000);

clearTimeout(timer);

В данном случае сообщение не будет выведено.

<переменная> = setInterval(<функция>, <задержка>) - устанавливает пользовательскую функцию, которая будет периодически выполняться с заданной задержкой (в миллисекундах).

Пример:

Var time = 0; var func = function() { time++; aler>t(time+" секунда"); };

setInterval(func, 1000); // вывод сообщения каждую секунду

В отличие от метода setTimeout, пользовательская функция в данном примере будет выполняться больше одного раза, увеличивая каждый раз счетчик секунд на 1.

clearInterval(переменная>) - отменяет выполнение пользовательской функции, установленной методом setInterval. Переменная представляет собой идентификатор таймера setInterval.

Пример: var time = 0; var func = function() { time++; alert(time+" секунда");

if (time >= 5)

clearinterval(timer); // отмена выполнения };

var timer = setInterval(func, 1000); // вывод 5 сообщений через секунду

**Объектная модель документа DOM**

Объект document хранит объектную структуру HTML документа (Document Object Model, DOM). С помощью него можно получить доступ к отдельным HTML тегам, добавлять и изменять их содержи)мое, а также определять поведение этих элементов.

**Доступ к элементам страницы.**

document.getElementById(<Id>) - возвращает HTML элемент по его идентификатору (атрибуту id); document.getElementsByName(<имя>) - возвращает массив HTML элементов по имени) (атрибуту name).

document.getElementsByTagName(<тег>) - возвращает массив HTML элементов по имени тега. document.getElementsByClassName(<класс>) - возвращает массив HTML элементов по имени класса (атрибуту class).

document.querySelector>(<селектор>) - возвращает первый подходящий HTML элемент по его CSS селектору. Параметр <селектор> описывается так же, как в стилевой таблице

**Динамическое изменение атрибутов**

Пример по изменению атрибутов тэга

<img src="picture1.png" alt="изображение" onclick="change()"> <script> function change() { // функция изменения изображения

document.images[0].src = "picture2.png";

} </script>

# **14. Доступ сценариев к элементам формы. Проверки формы javascript**

Существенным моментов программирования форм является идентификация их элементов. Рассмотрите еще раз иерархическую схему объектов раздела «[Объектная модель документа DOM0](http://vvz.nw.ru/Lessons/JavaScript/DOM0.htm)». Массив форм forms[] вложен в объект document; элементы форм являются вложенными объектами самой формы; элементы списка являются вложенными объектами объекта select.

Вспомним о том, что объекты JavaScript по сути представляют собой ассоциативные массивы со строковыми ключами – именами свойств.

Пример обращения к элементам формы из сценария. По типу идентификации элементы форм можно разделить на три группы:

* радиокнопки;
* элементы списка;
* прочие элементы форм.

Нетрудно догадаться о причинах именно такого разделения. В отличие от прочих элементов форм, группы радиокнопок и элементы списка представляют собой массивы. А массив элементов списка к тому же является вложенным по отношению к объекту select.

Пусть в документе HTML задана следующая форма (это мог бы быть фрагмент online-анкеты). Из группы «прочих элементов форм» в ней представлено однострочное текстовое поле ввода. Указаны минимально необходимые атрибуты.

<form name="f">

  Ваше имя:<br>

  <input type="text" name="yourName">

  Ваш пол:<br>

  <input type="radio" name="sex" value="male">мужской<br>

  <input type="radio" name="sex" value="female">женский<p>

  Город:<br>

  <select name="town">

    <option value="msk">Москва</option>

    <option value="spb">Санкт-Петербург</option>

    <option value="other">другой</option>

  </select>

 </form>

Во всех случаях при идентификации элементов форм используются имена (значения атрибута name)  заданных элементов HTML. По имени идентифицируется и сама форма; для нашей формы это обращение

document.f

Идентификация большинства элементов формы. Группу «прочих элементов форм» в нашей форме представляет текстовое поле yourName.  Варианты идентификации:

* document.f.yourName
* document.f['yourName']

**Идентификация групп радиокнопок.** В нашей форме есть группа их двух радиокнопок с именем sex. Группа радиокнопок представляет собой массив, в котором к отдельной кнопке с индексом i (индексация начинается с нуля) можно обращаться следующим образом:

* document.f.sex[i]
* document.f['sex'][i]

**Идентификация элементов списка**. В нашей форме представлен список с именем town. Его элементы представляет собой массив options[]. К элементу этого массива с индексом i (индексация начинается с нуля) можно обращаться следующим образом:

* document.f.options[i]
* document.f['options'][i]

**Универсальная идентификация**. Есть и альтернативная возможность идентификации. Все элементы формы представлены массивом elements[], в котором содержатся в порядке их объявления в документе HTML.

Проверка форм

Существует два разных типа проверки формы, с которыми вы столкнетесь в Интернете:

* **Проверка на стороне клиента -**это проверка, которая происходит в браузере, прежде чем данные будут отправлены на сервер. Это удобнее, чем проверка на стороне сервера, так как дает мгновенный ответ. Ее можно далее подразделить на:
  + **JavaScript** проверка выполняется с использованием JavaScript. Полностью настраиваемая.
  + **Встроенная проверка формы,**используя функции проверки формы HTML5. Для этого обычно не требуется JavaScript. Встроенная проверка формы имеет лучшую производительность, но она не такая настраиваемая, как с использованием JavaScript.
* **Проверка на стороне сервера** - это проверка, которая возникает на сервере после отправки данных. Серверный код используется для проверки данных перед их сохранением в базе данных. Если данные не проходят проверку валидности, ответ отправляется обратно клиенту, чтобы сообщить пользователю, какие исправления должны быть сделаны. Проверка на стороне сервера не такая удобная, как проверка на стороне клиента, поскольку она не выдает ошибок до тех пор, пока не будет отправлена вся форма. Тем не менее, проверка на стороне сервера - это последняя линия защиты вашего приложения от неправильных или даже вредоносных данных. Все популярные [серверные фреймворки](https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks) имеют функции для проверки и очистки данных (что делает их безопасными).

При помощи JavaScript можно проводить валидацию (проверку данных) форм HTML.

Например, в следующем примере если поле формы (fname) пусто, то функция выводит сообщение с предупреждением и возвращает значение false, чтобы предотвратить отправку данных формы:

JavaScript

function validateForm() {

var x = document.forms["myForm"]["fname"].value;

if (x == "") {

alert("Необходимо ввести имя");

return false;

}

}

# **15. Стандарты представления данных XTM. Формат обмена JSON.**

Документ XML представляет собой дерево, состоящее из узлов - элементов. Для доступа к XML узлам и0спользуются функции DOM, применяемые ранее к тегам HTML страниц. Одним из важных достоинств представления данных в виде XML является его универсальность.

Недостаток использования формата XML заключается в некоторой избыточности передаваемых данных (каждый XML узел представляет собой пару текстовых тегов), а также трудоемкость и ресурсоемкость разбора данных XML формата с помощью функций DOM или SAX (Simple API XML). Пример использования XML данных cодержание файла books.xml на сервере:

<?xml ver>sion="1.0"?>

<books>

<book id="1020"> <

<title>Изучаем Perl</title>

<author>>Рэндал Шварц</author>

</book>

<book id="1021">

<title>HTML, скрипты и стили</title>

<author>>Дунаев В.</author>

<book> <

</books>

Обработка подгружаемого списка из XML файла:

<ol id="books" onclick="select()">Список книг</ol>

<script> function select() {

Var xhr = new XMLHttpRequest();

xhr.open('GET', 'http://domain.r>u/books.xml', false);

xhr.send(); // выполнение запроса

if (xhr.status != "200") { // обработка ошибки alert("Ошибка сервера"); return false; }

var xml = xhr.responseXML;

var ol = document.getElementById("books");

var book = Array();

for(var i=0; i < xml.childNodes.length; i++) {

var title = xml.getElementsByTagName("title")[i].innerHTML;

var author> = xml.getElementsByTagName("author>")[i].innerHTML; book[i] = document.createElement("li");

book[i].innerHTML = author.italics() + title;

ol.appendChild(book[i]); } }

</script>

В результате после нажатия на текстовый элемент мы получаем вывод списка книг из заданного XML файла.

**JSON**

Альтернативой использования стандарта XML является представление данных в формате JSON. Это удобный краткий текстовый формат описания Javascript объектов. Данные, передаваемые от серверной части веб-приложения, конвертируются в объектные переменные Javascript и могут непосредственно использоваться в программных обработчиках на стороне клиента. Данные в формате JSON представляют собой набор терм, содержащих:

• Javascript объекты { ... };

• масси)вы [ ... ];

• строки, заключенные в двойные кавычки;

• числа;

• логические значения true/false;

• пустое значение null.

В основе представления JSON данных являются Javascript объекты, содержащие свойства и их значения. Содержание файла books.json на сервере:

{ "book": [

{ "id": 1020, "title": "Изучаем Perl",

"author": "Рэндал Шварц"

},

{ "id": 1021,

"title": "HTML, скрипты и стили",

"author>": "Дунаев В."

}

] }

В отличие от объектов Javascript, все атрибуты JSON данных должны представлять собой текстовые строки, заключенные в кавычки. Апострофы не допускаются. Такое представление короче и проще, чем XML, и потому завоевало большую популярность у разработчиков. Для чтения и разбора данных JSON в языке Javascript используется глобальный объект JSON. Основные методы объекта JSON:

JSON.parse(<строка>[,<функция>(ключ,значение)]) - преобразование текстовой строки JSON в объект Javascript. Дополнительный параметр <функция>(ключ, значение) вызывает функцию для каждого свойства объекта, которая должна вернуть измененное значение. JSON.strngify(<объект>[,<массив>]) - преобразование (сериализация) объекта Javascript в текстовую строку формата JSON. Дополнительный параметр <массив> содержит массив заданных свойств для сериализации объекта.

# **16. Использование функций DOM (я не знаю про что тут). Генерация элементов страницы JS**

DOM структура – глобальная структура. (document) - управляет dom структурой, дом структура - набор всех HTML тегов, которые есть внутри файла.

Основным инструментом работы и динамических изменений на странице является DOM (Document Object Model) - объектная модель, используемая для XML/HTML-документов.

Согласно DOM-модели, документ является иерархией.  
Каждый HTML-тег образует отдельный элемент-узел, каждый фрагмент текста - текстовый элемент, и т.п.

Проще говоря, DOM - это представление документа в виде дерева тегов. Это дерево образуется за счет вложенной структуры тегов плюс текстовые фрагменты страницы, каждый из которых образует отдельный узел.

Для создания элементов объект document имеет следующие методы:

* **createElement(elementName)**: создает элемент html, тег которого передается в качестве параметра. Возвращает созданный элемент
* **createTextNode(text)**: создает и возвращает текстовый узел. В качестве параметра передается текст узла.

|  |  |
| --- | --- |
| 1  2 | var elem = document.createElement("div");  var elemText = document.createTextNode("Привет мир"); |

Таким образом, переменная elem будет хранить ссылку на элемент div. Однако одного создания элементов недостаточно, их еще надо добавить на веб-страницу.

Для добавления элементов мы можем использовать один из методов объекта Node:

* **appendChild(newNode)**: добавляет новый узел newNode в конец коллекции дочерних узлов
* **insertBefore(newNode, referenceNode)**: добавляет новый узел newNode перед узлом referenceNode

Иногда элементы бывают довольно сложными по составу, и гораздо проще их скопировать, чем с помощью отдельных вызовов создавать из содержимое. Для копирования уже имеющихся узлов у объекта Node можно использовать метод **cloneNode()**:

|  |  |
| --- | --- |
| 1  2  3  4  5 | var articleDiv = document.querySelector("div.article");  // клонируем элемент articleDiv  var newArticleDiv = articleDiv.cloneNode(true);  // добавляем в конец элемента body  document.body.appendChild(newArticleDiv); |

* В метод cloneNode() в качестве параметра передается логическое значение: если передается true, то элемент будет копироваться со всеми дочерними узлами; если передается false - то копируется без дочерних узлов. То есть в данном случае мы копируем узел со всем его содержимым и потом добавляем в конец элемента body.

Для замены элемента применяется метод **replaceChild(newNode, oldNode)** объекта Node. Этот метод в качестве первого параметра принимает новый элемент, который заменяет старый элемент oldNode, передаваемый в качестве второго параметра.

# **17. Отправка данных на сервер**

Взаимодействие с сервером происходит с помощью Get и post запросов

GET запрос используется чтобы получить данные а POST чтобы отправить. (Напоминаю что технически они работают одинаково).

GET запрос пример:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8" />

</head>

<body>

<div id="output"></div>

<script>

// объект для отправки

var user = {

    name: "Tom",

    age: 23

};

var request = new XMLHttpRequest();

function reqReadyStateChange() {

    if (request.readyState == 4) {

        var status = request.status;

        if (status == 200) {

            document.getElementById("output").innerHTML=request.responseText;

        }

    }

}

// строка с параметрами для отправки

var body = "name=" + user.name + "&age="+user.age;

request.open("GET", "<http://localhost:8080/postdata.php?>"+body);

request.onreadystatechange = reqReadyStateChange;

request.send();

</script>

</body>

</html>

POST запрос пример:

var user = {

    name: "Tom",

    age: 23

};

var request = new XMLHttpRequest();

function reqReadyStateChange() {

    if (request.readyState == 4 && request.status == 200)

        document.getElementById("output").innerHTML=request.responseText;

}

var body = "name=" + user.name + "&age="+user.age;

request.open("POST", "<http://localhost:8080/postdata.php>");

request.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

request.onreadystatechange = reqReadyStateChange;

request.send(body);

Технология AJAX

AJAX (аббревиатура от «**A**synchronous **J**avascript **A**nd **X**ml») – технология обращения к серверу без перезагрузки страницы.

За счёт этого уменьшается время отклика и веб-приложение по интерактивности больше напоминает десктоп.

Несмотря на то, что в названии технологии присутствует буква X (от слова XML), использовать XML вовсе не обязательно. Под AJAX подразумевают любое общение с сервером без перезагрузки страницы, организованное при помощи JavaScript.

Вот код кнопки в примере выше:

|  |
| --- |
| <input value="Голосовать!" onclick="vote()" type="button" /> |

При нажатии она вызывает функцию vote, которая отправляет запрос на сервер, ждет ответа, а затем показывает сообщение об этом в div'е под кнопкой:

|  |
| --- |
| <div id="vote\_status">Здесь будет ответ сервера</div> |

jQuery.ajax

Это самый основной метод, а все последующие методы лишь обертки для метода jQuery.ajax. У данного метода лишь один входной параметр — объект включающий в себя все настройки (выделены параметры которые стоит запомнить):

\* async — асинхронность запроса, по умолчанию true  
\* cache — вкл/выкл кэширование данных браузером, по умолчанию true  
\* contentType — по умолчанию “application/x-www-form-urlencoded”  
\* data — передаваемые данные — строка иль объект  
\* dataFilter — фильтр для входных данных  
\* dataType — тип данных возвращаемых в callback функцию (xml, html, script, json, text, \_default)  
\* global — тригер — отвечает за использование глобальных AJAX Event’ов, по умолчанию true  
\* ifModified — тригер — проверяет были ли изменения в ответе сервера, дабы не слать еще запрос, по умолчанию false  
\* jsonp — переустановить имя callback функции для работы с JSONP (по умолчанию генерируется на лету)  
\* processData — по умолчанию отправляемые данный заворачиваются в объект, и отправляются как “application/x-www-form-urlencoded”, если надо иначе — отключаем  
\* scriptCharset — кодировочка — актуально для JSONP и подгрузки JavaScript’ов  
\* timeout — время таймаут в миллисекундах  
\* type — GET либо POST  
\* url — url запрашиваемой страницы

Локальные AJAX Event’ы:

\* beforeSend — срабатывает перед отправкой запроса  
\* error — если произошла ошибка  
\* success — если ошибок не возникло

\* complete — срабатывает по окончанию запроса

Для организации HTTP авторизации (О\_о):

\* username — логин

\* password — пароль

Пример javaScript’а:

$.ajax({

url: '/ajax/example.html', // указываем URL и

dataType : "json", // тип загружаемых данных

success: function (data, textStatus) { // вешаем свой обработчик на функцию success

$.each(data, function(i, val) { // обрабатываем полученные данные

/\* ... \*/

});

}

});

# **22. PHP классы и объекты**

class SimpleClass  
{  
    // объявление свойства  
    public $var = 'значение по умолчанию';  
  
    // объявление метода  
    public function displayVar() {  
        echo $this->var;  
    }  
}

Для создания экземпляра класса используется директива new. Новый объект всегда будет создан, за исключением случаев, когда он содержит [конструктор](https://www.php.net/manual/ru/language.oop5.decon.php), в котором определен вызов [исключения](https://www.php.net/manual/ru/language.exceptions.php) в случае ошибки. Рекомендуется определять классы до создания их экземпляров (в некоторых случаях это обязательно).

Если с директивой new используется строка ([string](https://www.php.net/manual/ru/language.types.string.php)), содержащая имя класса, то будет создан новый экземпляр этого класса. Если имя находится в пространстве имен, то оно должно быть задано полностью.

**Замечание**:

В случае отсутствия аргументов в конструктор класса, круглые скобки после названия класса можно опустить.

**Пример #3 Создание экземпляра класса**

<?php  
$instance = new SimpleClass();  
  
// Это же можно сделать с помощью переменной:  
$className = 'SimpleClass';  
$instance = new $className(); // new SimpleClass()  
?>

Конструктор[¶](https://www.php.net/manual/ru/language.oop5.decon.php#language.oop5.decon.constructor)

**\_\_construct** ([ [mixed](https://www.php.net/manual/ru/language.pseudo-types.php" \l "language.types.mixed) $args [, $... ]] ) : void

PHP 5 позволяет объявлять методы-конструкторы. Классы, в которых объявлен метод-конструктор, будут вызывать этот метод при каждом создании нового объекта, так что это может оказаться полезным, например, для инициализации какого-либо состояния объекта перед его использованием.

Наследование[¶](https://www.php.net/manual/ru/language.oop5.inheritance.php#language.oop5.inheritance)

Наследование — это хорошо зарекомендовавший себя принцип программирования, и PHP использует этот принцип в своей объектной модели. Этот принцип повлияет на то, как многие классы и объекты связаны друг с другом.

Например, при расширении класса дочерний класс наследует все общедоступные и защищенные методы из родительского класса. До тех пор пока эти методы не будут переопределены, они будут сохранять свою исходную функциональность.

Это полезно для определения и абстрагирования функциональности и позволяет реализовать дополнительную функциональность в похожих объектах без необходимости реализовывать всю общую функциональность.

**Замечание**:

Если не используется автозагрузка, классы должны быть объявлены до того, как они будут использоваться. Если класс расширяет другой, то родительский класс должен быть объявлен до наследующего класса. Это правило применяется к классам, которые наследуют другие классы или интерфейсы.

**Пример #1 Пример наследования**

<?php  
class Foo  
{  
    public function printItem($string)  
    {  
        echo 'Foo: ' . $string . PHP\_EOL;  
    }  
    public function printPHP()  
    {  
        echo 'PHP просто супер.' . PHP\_EOL;  
    }  
}  
class Bar extends Foo  
{  
    public function printItem($string)  
    {  
        echo 'Bar: ' . $string . PHP\_EOL;  
    }  
}  
$foo = new Foo();  
$bar = new Bar();  
$foo->printItem('baz'); // Выведет: 'Foo: baz'  
$foo->printPHP();       // Выведет: 'PHP просто супер'   
$bar->printItem('baz'); // Выведет: 'Bar: baz'  
$bar->printPHP();       // Выведет: 'PHP просто супер'  
  
?>

Свойства[¶](https://www.php.net/manual/ru/language.oop5.properties.php#language.oop5.properties)

Переменные, которые являются членами класса, называются "свойства". Также их называют, используя другие термины, такие как "атрибуты" или "поля", но в рамках этой документации, мы будем называть их свойствами. Они определяются с помощью ключевых слов public, protected или private, могут сопровождаться необязательным объявлением типа и следуют правилам правильного объявления переменных.

// правильное определение свойства с PHP 5.6.0:  
   public $var1 = 'hello ' . 'world';

# **18) Библиотека jQuery. Основные функции библиотеки jQuery. Обработка событий. Проверка валидности данных формы. Динамическое изменение CSS стилей.**

Одной из самых популярных JavaScript библиотек, использующихся в разработке веб-приложений, является библиотека **JQuery**. Если CSS отделяет визуализацию от структуры HTML, то JQuery отделяет поведение от структуры HTML документа. Эта библиотека позволяет значительно упростить и ускорить написание JavaScript кода, сделать его более прозрачным и понятным.

Для добавления библиотеки JQuery на HTML страницу веб- приложения используется вставка скрипта в заголовок страницы в виде отдельного файла, например (для предварительной загрузки и локального использования):

<script src="jquery-2.2.4.min.js"></script>

или (для использования ссылки Google):

<script src="https://ajax.googleapis.com/ajax/libs/jquery/ 2.2.4/jquery.min.js"> </script>

Вся работа с JQuery производится с помощью функции $(), что является синонимом JQuery.

Работу с jQuery можно разделить на 2 типа:

• Получение jQuery объекта с помощью функции $(). Например, передав в нее CSS-селектор, можно получить jQuery объект всех элементов HTML, попадающих под критерий, и далее работать с ними с помощью различных методов jQuery объекта. В случае если метод не должен возвращать какого-либо значения, он возвращает ссылку на jQuery объект, что позволяет вести цепочку вызовов методов.

• Вызов глобальных методов объекта $, например итераторов для массива.

Таким образом, стандартный синтаксис выполнения команд JQuery:

$(<селектор>).<метод>(<параметры>)

После выбора элементов можно задать обработчики событий и привязать их к данным элементам. Общий синтаксис обработчика:

$(<селектор>).<событие>(function() { ...код обработчика... } );

Например, для изменения текста раздела div по нажатию на кнопку button определим следующий обработчик:

$(document).ready(function() {

$(":button").click(function() {

$("div#par").html("новый текст"); });

});

Существует **два вида валидации**: на стороне клиента и на стороне сервера. Рекомендуется использовать оба типа проверки. Валидация на стороне клиента чаще всего реализуется с помощью JavaScript. Она предоставляет пользователям быстрый ответ об ошибках ввода, прежде чем отправить данные на сервер.

Один из подключаемых модулей jQuery, пользующихся наибольшей популярностью, называется **Validation**, и уже само это название говорит о том, для чего он предназначен, поскольку валидация данных — это процесс проверки введенных данных на предмет их соответствия определенным правилам, условиям или ограничениям проще говоря — проверки их корректности.

CSS определяет режим воспроизведения отдельных элементов в документе. Объектная модель манипулирования свойствами таблицы стилей основана на рекомендациях CSS. Когда атрибут или правило изменяются посредством сценария, то статическая таблица стилей и страница обновляются.

Данная модель динамических стилей отличается от модели JASS, поддерживаемой в Netscape Navigator 4.0. JASS является процедурной моделью определения таблицы стилей документа во время его анализа, а не моделью программирования для манипулирования стилем документа. Например, JASS может быть использована для написания кода условия, который применим к различным таблицам стилей в зависимости от размера экрана в ходе загрузки документа. JASS не может быть использован для изменения стиля элемента в ответ на событие без перезагрузки или запроса новой страницы с сервера.

**19) Элементы языка PHP. Переменные. Типы данных. Операции. Операторы и основные конструкции.**

**PHP** — C-подобный скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов.

В синтаксисе PHP **имя переменной** записывается латинскими символами, но первым символом всегда должен быть знак доллара $, а затем идёт имя. Не допускается начинать имя переменной с цифры, а также использовать любые значения, кроме букв алфавита и знака подчеркивания. Примеры допустимых имён переменных:

• $age;

• $favorite\_color;

• $name2.

Как правило, названия переменный начинаются с маленькой буквы или символа подчеркивания. Стоит учитывать, что PHP является регистрозависимым языком, а значит, переменные $num и $Num будут представлять две разные переменные.

Также при наименовании переменных нам надо учитывать следующие правила:

• Имена переменных должны начинаться с алфавитного символа или с подчеркивания

• Имена переменных могут содержать только символы: a–z, A–Z, 0–9, и знак подчеркивания

• Имена переменных не должны включать в себя пробелы

С помощью **операции** **присвоения** (=) переменной присваивается определенное значение: $num = 10;

Здесь определена переменная $num, которая хранит число 10.

После определения переменной и присвоения ей значения мы можем использовать ее в выражениях PHP. Например, вывести ее значение на веб-страницу:

<?php // определение переменной $num

$num = 10; // вывод значения переменной $num на веб-страницу

echo $num;

?>

В PHP есть **десять базовых типов** данных:

• bool (логический тип)

• int (целые числа)

• float (дробные числа)

• string (строки)

• array (массивы)

• object (объекты)

• callable (функции)

• mixed (любой тип)

• resource (ресурсы)

• null (отсутствие значения)

Из этих типов данных первые четыре являются скалярными: bool, int, float, string.

В PHP мы можем использовать **различные операторы**: арифметические, логические и т.д. Рассмотрим каждый тип операций.

**Арифметические операции**

• + (операция сложения) $a + 5

• - (операция вычитания) $a - 5

• \* (умножение) $a \* 5

• / (деление) $a / 5

• % (получение остатка от деления) $a = 12; echo $a % 5; // равно 2

• \*\* (возведение в степень) $a \*\* 2

**Операции сравнения**

Операции сравнения, как правило, применяются в условных конструкциях, когда надо сравнивать два значения, и в зависимости от результата сравнения выполнить некоторые действия. Имеются следующие операции сравнения.

• == Оператор равенства сравнивает два значения, и если они равны, возвращает true, иначе возвращает false: $a == 5

• === Оператор тождественности также сравнивает два значения, и если они равны, возвращает true, иначе возвращает false: $a === 5

• != Сравнивает два значения, и если они не равны, возвращает true, иначе возвращает false: $a != 5

• !== Сравнивает два значения, и если они не равны, возвращает true, иначе возвращает false: $a !== 5

• > Сравнивает два значения, и если первое больше второго, то возвращает true, иначе возвращает false: $a > 5

• < Сравнивает два значения, и если первое меньше второго, то возвращает true, иначе возвращает false: $a < 5

• >= Сравнивает два значения, и если первое больше или равно второму, то возвращает true, иначе возвращает false: $a >= 5

• <= Сравнивает два значения, и если первое меньше или равно второму, то возвращает true, иначе возвращает false: $a <= 5

**Логические операции**

Логические операции обычно применяются для объединения результатов двух операций сравнения. Например, нам надо выполнить определенное действие, если истинны будут несколько условий. Имеются следующие логические операции:

• && Возвращает true, если обе операции сравнения возвращают true, иначе возвращает false: $a && $b

• and Аналогично операции &&: $a and $b

• || Возвращает true, если хотя бы одна операция сравнения возвращают true, иначе возвращает false: $a || $b

• or Аналогично операции ||: $a or $b

• ! Возвращает true, если операция сравнения возвращает false: !$a

• xor Возвращает true, если только одно из значений равно true. Если оба равны true или ни одно из них не равно true, возвращает false: $a xor $b

**Приоритет операций**

Если одно выражение содержит несколько разных операций, то при выполнении выражения учитывается приоритет операций. Сначала выполняются операции с большим приоритетом и в конце с меньшим приоритетом.

К основным операторам языка PHP относятся операторы обработки данных, такие как математические, операторы присваивания, операторы сравнения, логические операторы. Помимо этого, в языке PHP используются управляющие конструкции языка - **условные операторы и операторы циклов**.

**Условные операторы** являются наиболее распространенными конструкциями во всех алгоритмических языках программирования.

1. **Конструкция if**

<?php

if (логическое выражение)

оператор;

?>

Конструкция if содержит логическое выражение. Если логическое выражение истинно (true), то оператор, следующий за конструкцией if будет исполнен, а если логическое выражение ложно (false), то следующий за if оператор исполнен не будет. Приведем пример:

2. **Конструкция else**

Часто возникает потребность исполнения операторов не только в теле конструкции if, но и в случае, если условие конструкции if не выполнено. В данной ситуации нельзя обойтись без конструкции else. Такая конструкция будет называться if-else.

**25. Трехуровневая архитектура приложений** - это модульная клиент-серверная архитектура, которая состоит из уровня представления, уровня приложения и уровня данных. Уровень данных обеспечивает хранение информации, уровень приложений обрабатывает логику, а уровень представления являет собой графический интерфейс пользователя (GUI), который взаимодействует с двумя другими уровнями. Эти три уровня являются логическими, а не физическими, и могут работать как на одном физическом сервере, так и на разных машинах.

**Уровень представления.** Этот уровень, созданный с использованием HTML5, JavaScript и каскадных таблиц стилей (CSS), развертывается на вычислительном устройстве через веб-браузер или веб-приложение. Уровень представления связывается с другими уровнями посредством вызовов интерфейса прикладных программ (API).

**Уровень приложения.** Уровень приложения, который также можно назвать логическим уровнем, написан на языке программирования, таком как Java, Python или Ruby, и содержит бизнес-логику, которая поддерживает основные функции приложения. Базовый уровень приложений может быть размещен на распределенных серверах в облаке или на выделенном внутреннем сервере, в зависимости от того, сколько вычислительной мощности требуется приложению.

**Уровень данных.** Уровень данных состоит из базы данных и программы для управления доступом для чтения и записи в базе данных. Этот уровень также может называться уровнем хранения и может быть размещен локально или в облаке. Популярные системы баз данных для управления доступом для чтения / записи включают MySQL, Oracle, PostgreSQL, Microsoft SQL Server и MongoDB.

MVC

Модель - Model

Представление - View

Контроллер - Controller



Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида).

К одной модели можно присоединить несколько видов, при этом не затрагивая реализацию модели.

Не затрагивая реализацию видов, можно изменить реакции на действия пользователя (использовать другой контроллер)

Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

**Модель**. Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

* Пассивная модель — модель не может воздействовать на представление или контроллер, и используется в качестве источника данных.
* Активная модель — модель оповещает представление о том, что в ней произошли изменения, а представления, которые заинтересованы в оповещении, подписываются на эти сообщения.

**Представление**, вид. Отвечает за отображение информации (визуализацию).

**Контроллер**. Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

# **26. Основы информационной безопасности. Защита Web – приложений от несанкционированного доступа и SQL инъекций. Средства проверки на безопасность. Использование плейсхолдеров и драйвера PDO.**

SQL-инъекция – это тип атаки, ориентированный на базы данных, при которой злоумышленник успешно формирует и вставляет вредоносный код в строки. В свою очередь данные строки транслируются на сервер системы управления базой данных (СУБД) для синтаксического анализа и выполнения. В случае успешного исполнения данной атаки злоумышленник может легко преодолеть систему безопасности приложения, получить доступ, копировать, удалять и изменять конфиденциальную информацию, которая содержится в базе данных (БД).

Виды SQL инъекций

* Команда UNION
* Вставка кавычек и комментариев
* Логические условия OR и AND
* Использование подзапросов
* Механизм слепого внедрения
* Использование временных задержек

Пример SQL – инъекции:

**<html>**

**<head>**

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

**</head>**

**<body>**

**<?php**

**$id = $\_GET['uid'];**

**$conn = mysql\_connect('localhost','root','root');**

**if (!$conn) {**

**echo "Ошибка сервера mysql <br>";**

**exit;**

**}**

**mysql\_select\_db('http\_auth');**

**$sql = mysql\_query("select \* from students where username = ".$id);**

**while ($row = mysql\_fetch\_assoc($sql) ) {**

**echo "ФИО студента: $row[family] $row[name] $row[sename] <br>";**

**}**

**?>**

**</body>**

**</html>**

PDO (PHP Data Objects) — расширение для PHP, предоставляющее простой интерфейс для доступа к различным базам данных. Если говорить очень просто и коротко, при помощи PDO в PHP подключаются к базам данных разных типов.

Пример использования placeholder:

**<html>**

**<head>**

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

**</head>**

**<body>**

**<?php**

**$id = $\_REQUEST['uid'];**

**try {**

**$db = new**

**PDO('mysql:host=localhost;dbname=http\_auth','Adm001','psA#$671bN');**

**$sql = $db->prepare("select \* from students where username = ?");**

**$sql->execute(array($id));**

**while ($row = $sql->fetch(PDO::FETCH\_ASSOC)) {**

**echo "ФИО студента: $row[family] $row[name] $row[sename] <br>";**

**}**

**$db = null;**

**} catch (PDOException $e) {**

**echo "Ошибка сервера mysql ". $e->getMessage() . "<br>";**

**}**

**?>**

**</body>**

**</html>**