Государственное бюджетное профессиональное образовательное учреждение

«Арзамасский коммерческо-технический техникум»

Саблукова Наталья Геннадьевна

Комплект лекций

по дисциплине «Web-программирование»

Часть 1. Дизайн, верстка и клиентское программирование

для студентов специальности
09.02.07 Информационные системы и программирование

Арзамас
2022

Одобрено методическим объединением естественно-математических и информационных дисциплин
Протокол № 1 от 30.08.2022 г

Саблукова Н.Г.
Комплект лекций по дисциплине «Web-программирование» для студентов специальности 09.02.07 Информационные системы и программирование – Арзамас: ГБПОУ АКТТ, 2022. – 79 с.

Комплект лекций содержат материал по дисциплине «Web-программирование», изучаемой студентами данной специальности на четвертом курсе. Представленная информация может быть использована студентами как во время учебных занятий по данной дисциплине, а также в рамках самостоятельной работы во внеурочное время.

Содержание

Введение

Курс лекций составлен в соответствии с рабочей программой дисциплины «Web-программирование» и требованиями федерального государственного образовательного стандарта среднего профессионального образования по специальности 09.02.07 Информационные системы и программирование.

Курс лекций раскрывает основные вопросы разделов дисциплины «Web-программирование»:

- Основы проектирования и дизайна веб-сайтов.
- Верстка веб-сайтов.
- Клиентское программирование.
- Серверное программирование.
- Использование CMS для разработки сайтов
- Оптимизация сайта.

Лекционный материал содержит краткую информацию по основным темам дисциплины. Каждая лекция содержит:

- план (перечень вопросов, рассматриваемых в рамках темы);
- изложение основных вопросов, в соответствии с представленным планом.

Учебное пособие предназначено для студентов очной формы обучения по специальности 09.02.07 Информационные системы и программирование и может быть использовано как во время учебных занятий по дисциплине «Web-программирование», а также в рамках самостоятельной работы во внеурочное время.

Лекция № 1. Средства разработки сайта. Этапы разработки сайта

План.

1. Понятие и виды веб-сайтов
2. Современные технологии и методы разработки
3. Этапы проектирования и разработки сайта

1) Понятие и виды веб-сайтов

Веб-сайт (от англ. website: web — «паутина, сеть» и site — «место», буквально «место, сегмент, часть в сети») – совокупность веб-страниц с повторяющимся дизайном, объединенных по смыслу, навигационно и физически находящихся на одном web-сервере.

Типы сайтов по структуре и содержанию

1) Интернет-представительства:
   a. Интернет-магазин
   b. Landing-page – одностраничник, его задача заставить пользователя произвести нужно действие: купить продукцию, записаться на курс и др.
   c. Визитка – небольшой сайт (5-7 страниц), включающий общее описание продукции или компании, прайс-лист, контактную информацию, форму обратной связи. Создается частными лицами, предприятиями или организациями.
   d. Корпоративный сайт – большой сайт предприятия или организации
2) Информационные услуги:
   a. Интернет-портал – сайт, имеющий широкий функционал и активное взаимодействие с пользователем, это сложные веб-проекты, на которых часто размещают фото, видео, форумы, блоги и т.п.
   b. Новостной – сайт с актуальными новостями (например, lenta.ru)
   c. Тематический
   d. Блог
3) Веб-сервисы
   a. Поисковые системы
   b. Социальные сети
   c. Форумы
   d. Почтовые системы

Beb-сервис (служба) – программа, которая организовывает взаимодействие между сайтами.

2) Современные технологии и методы разработки

Клиентская разработка (frontend) – создание клиентской части сайта, программирование на стороне клиента.

Frontend-разработчик занимается версткой шаблона сайта и созданием пользовательского интерфейса. Обычно frontend разработчик — это мастер на все руки. Он просто обязан обладать талантом дизайнера, быть искусным верстальщиком и хорошим программистом.

К клиентской разработке относятся следующие языки и технологии:

- **HTML** – язык разметки документа
- **CSS** – язык стилей, задает правила оформления и отображения web-страниц

- **JavaScript –** основной клиентский язык программирования для web-страниц
- **jQuery** – библиотека JavaScript, упрощающая основные операции, основана на взаимодействии JavaScript, HTML и CSS.
- **Bootstrap** – фреймворк или библиотека, включающий набор инструментов для разработки (HTML, CSS, JS, шаблоны, веб-формы, кнопки и т.д.)

Серверная разработка (backend) – программирование на стороне сервера.

Серверную разработку можно осуществлять на следующи языках программирования:

- **PHP** – платформа и язык разработки для веб, самый распространенный в мире
- **Node.js** – программная платформа на движке V8
- **Python** – высокоуровневый язык программирования общего назначения, который также можно использовать для backend-разработки сайта
- **Ruby** – динамический язык программирования с открытым исходным кодом с упором на простоту и продуктивность

1) **Базы данных (Database)** – это место на веб-сервере, где хранится контент веб-ресурса.

Примеры баз-данных:

- **MySQL** - одна из самых популярных СУБД в современных интернет-технологиях
- **MariaDB**
- **PostgreSQL**
- **Oracle Database**

Средства разработки

- IDE (интегрированная среда разработки) **PHPStorm, ATOM, WebStorm**, **VS Code**
- IDE **Notepad++, Sublime Text**
- **Система управления версиями** — программное обеспечение для облегчения работы с изменяющейся информацией
  - Git – средство для контроля версий программ
  - Cithab – предоставляет хостинг (хранение) исходных текстов
  - Gitlab – сайт и система управления версиями кода для Git, из дополнительных возможностей: собственная вики и система отслеживания ошибок

3) Этапы проектирования и разработки сайта

1 **этап Концептуальное проектирование сайта** включает в себя определение целей и типа сайта, изучение целевой аудитории, определение сценариев взаимодействия с сайтом, анализ сайтов конкурентов

Рассмотрим концептуальное проектирование подробнее.

Определение цели разработки сайта

Наиболее распространенные примеры целей разработки сайта:

- *Создание бренда компании и его развитие*

Одной из причин, которые побуждают компанию на создание собственного интернет-сайта – это стремление не отстать от других, быть «в духе времени». Создание

собственного бренда или хотя бы корпоративного стиля, преследует собой фиксирование в сознании потребителей ассоциативного представления о компании.

Бренд компании – это узнаваемый, популярный образ, четкое сложившееся убеждение о компании в сознании потребителей.

Всемирно известные торговые марки тратят огромные средства на рекламу, на формирование собственного бренда. Но в интернете все немного иначе и демократичнее, и сайт сравнительно небольшой компании вполне может соперничать с интернет-площадкой мирового бизнес-гиганта, не потратив на это миллионы денежных единиц.

- *Привлечение новых потребителей*

Увеличить приток целевой аудитории, а, следовательно, и потенциальных клиентов, можно с помощью функции обратной связи. При этом, контактная форма должна быть продумана до мелочей и нюансов.

Это может быть информация о фактическом адресе, телефон, Skype, электронная почта, форма заказа, форма обратного звонка, чат на сайте и др. Нужно обратить внимание на правильное составление опросного листа, что позволит выявлять потенциальных заказчиков среди всего количества посетителей сайта.

- *Получение прибыли*

Для получения прибыли с сайта выделено три наиболее перспективных направления:

✓ Получение дохода от продаж.

Электронная коммерция – продажа товаров или оказание услуг, происходящие непосредственно в виртуальном пространстве. Данный вид бизнеса включает в себя огромный диапазон позиций продукции и услуг.

✓ Получение прибыли от рекламы

Если сайт достаточно продвинут в поисковых системах, а как следствие, имеет высокую посещаемость, то немалую часть дохода будет приносить продажа рекламы на своем интернет-ресурсе. Подобный пассивный доход может стать хорошим подспорьем для кошелька, а у особенно удачливых предпринимателей, сможет даже превратиться в основной источник дохода.

✓ Получение прибыли от ссылок

Прибыль, получаемая от ссылок на сайты других компаний, также приносит доход владельцу ресурса, однако, это не главный способ заработка в интернете, слишком скромный доход приносит участие в партнерских программах. Своего рода комиссионные выплачиваются только после того, как рекламируемая по ссылке продукция была реализована.

- *Минимизация затрат и оптимизация бюджета*

Минимизация расходов с помощью Интернета – очень серьезный способ заработать деньги в виртуальном пространстве. Переместив часть своей деятельности в глобальную сеть, можно основательно уменьшить свои расходы. Сэкономить при помощи интернета можно несколькими способами:

✓ Сокращение штата

✓ Сокращение затрат на рекламу

- *Забота о потребностях клиентов*

Глобальная сеть способна осуществить самый высокий уровень взаимодействия компании со своими клиентами, обеспечив надлежащим образом предпродажное и послепродажное обслуживание.

Простой раздел на сайте компании с содержанием ответов на часто возникающие вопросы, является прекрасным сервисом, облегчающим пользование сайтом.

Задачи и целевые действия на сайте

Цель необходимо конкретизировать до более мелких задач и выделить для них целевые действия.

Целевые действия – это действия, которые должны выполнить посетители сайта, для того чтобы была реализована цель создания сайта.

Пример детализации задач до уровня целевых действий:

Мы хотим снизить долю отказов? Значит, нам надо убедить пользователя просмотреть не менее двух страниц.

Хотим повысить уровень осведомлённости целевой аудитории о нашей компании? Значит, целевое действие – посещение клиентом страницы с соответствующей информацией.

Планируем увеличить базу клиентов? Разместим на сайте целевую страницу с формой подписки

Целевая аудитория (или целевая группа) – это группа людей, которые вероятнее всего заинтересуются предложением и закажут конкретный товар или услугу.

Широкая ЦА подразумевает, что продаваемый продукт может быть интересен не только представителю ЦА, но и другой группе, являющейся авторитетной в выборе товара.

Например, игрушки выбирают дети, но покупают товары родители, косметику женщина выберет сама, но для подарка может покупать мужчина.

Обычно широкую целевую аудиторию маркетолог разбивает на несколько сегментов, чтобы удовлетворить требования клиентов из разных целевых групп. Ведь широкая ЦА не является носителем одинаковых потребностей и интересов. Выделяют 3-4 сегмента

В описании целевой аудитории следует рассказывать о наиболее ярком представителе целевого рынка, стараясь описывать общие характерные черты, по которым можно отличить потребителя вашей компании от всех потребителей на рынке. Составляя портрет своего потребителя, следует описывать не только текущих покупателей продукта, но и всех потенциальных клиентов, которые еще не покупали товар.

Портрет целевой аудитории

- Пол, возраст, уровень дохода, статус, семейное положение, профессия, география проживания — т.е. *социально-демографический портрет и психографические характеристики*;
- Интересы, где проводит свободное время ваша потенциальная целевая аудитория (в каких соцсетях зарегистрированы, какие форумы читают и т.д.)
- Какие проблемы клиента может закрыть ваш товар?
- Какие эмоции вызывает ваш товар ли услуга, с чем ассоциируется?
- Мотивы действия, причины купить именно ваш продукт, а также причины купить аналогичный товар у конкурентов.
- Ценности клиента

Пример широкой целевой аудитории (ЦА), которые мы разобьем на сегменты:

Билеты на автобусы компании «Дорога», услуга рассчитана на покупателей разного возраста;

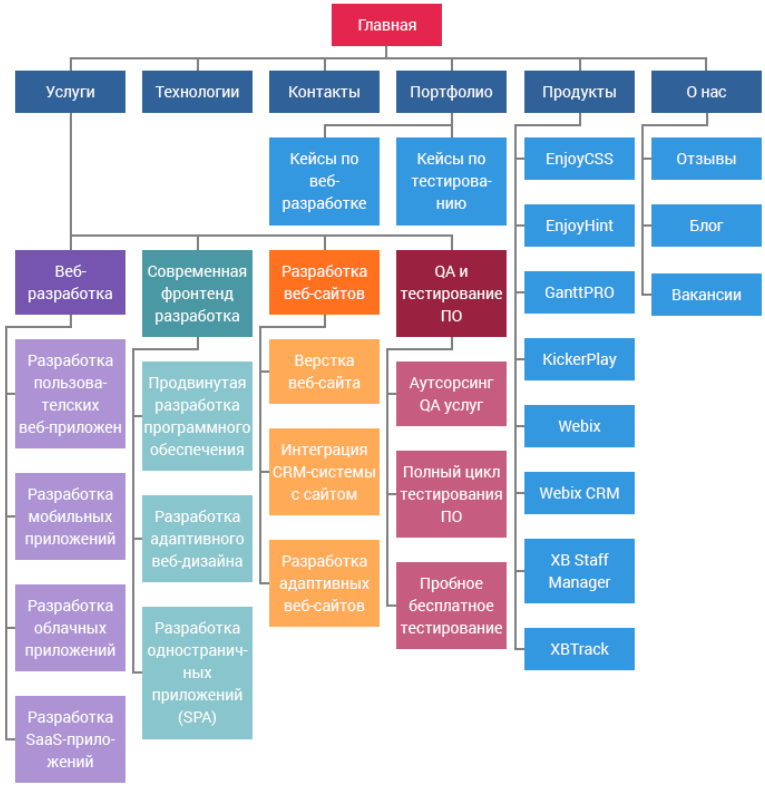
– Целевая аудитория: женщины и мужчины от 18 до 80 лет;
– В ЦА входят все слои населения;
– Семейное положение: замужем/не замужем;
– Интересы: путешествия, работа, отдых и другие;
– Ядро ЦА – женщины и мужчины от 18 до 60 лет.

Условно вышеприведенную широкую ЦА можно разбить на 3 группы:

1) Молодые люди и студенты 18-27 лет, путешествующие в другую область на учебу, редко выезжающие с целью туризма;
2) Бизнесмены. Возраст – 30-55 лет, ездят в командировки, на отдых в отдаленные регионы РФ, на встречи с партнёрами;
3) Пожилые люди 55-70 лет, покупающие дешевые билеты для туризма или поездки к детям в другой город.

2 этап Логическое проектирование сайта – проектирование структуры сайта и навигации по разделам (блок-схемы, структурные диаграммы, предварительный дизайн)

Например:



3 этап Физическое проектирование – определение технической стороны реализации проекта (технологии, которые будут применяться на сайте; возможные проблемы и способы их устранения, как будет обновляться информация и т.д.)

4 этап Разработка технического задания на сайт
5 этап Разработка дизайна сайта
6 этап Разработка и внедрение сайта
7 этап Поддержка и продвижение сайта

Лекция № 2. Стандарты и требования к разработке дизайна веб-сайтов. Прототипирование

План

1. Стандарты к разработке государственных сайтов
2. Требования Data Driven Design (DDD)
3. Требования UI/UX-дизайна
4. Бриф и техническое задание
5. Прототипирование сайта

1) Стандарты к разработке государственных сайтов

При разработке государственных сайтов необходимо руководствоваться следующими стандартами в области информационных систем:

ГОСТ Р ИСО 9241-151-2014 Эргономика взаимодействия человек-система. Часть 151. Руководство по проектированию пользовательских интерфейсов сети Интернет

ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.

ГОСТ 34.601-90. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания

ГОСТ 34.201-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем;

РД 50-34.698-90. Методические указания. Информационная технология. Комплекс стандартов и руководящих документов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов.

ГОСТ Р 52872-2012 Интернет-ресурсы. Требования доступности для инвалидов по зрению

При разработке дизайна сайтов в настоящее время учитываются требования Data Driven Design и UI/UX.

2) Требования Data Driven Design (DDD)

Data Driven Design – это разработка дизайна сайта на основе аналитических данных: исследований, тестов, гипотез, Big Data анализа.

Дизайн-решения в DDD принимается не из опыта, вкуса или интуиции дизайнера, а основывается на результаты тестов и проверок гипотез. Все изменения необходимо подкреплять цифрами. Нельзя просто сделать кнопку большой и красной. Надо сначала сравнить маленькую синюю кнопку с большой красной, если нужный показатель у красной выше (допустим пользователи чаще нажимают «заказать»), то такое изменение принимается.

Во многих случаях DDD-подход выгоден и разработчику, и заказчику.

Почему DDD выгоден заказчику:

- Больше никаких действий наугад в ущерб бюджету проекта.
- Отказ от философии «Мы ничего не можем гарантировать. Давайте попробуем, как дизайнер предложил, а там посмотрим».
- Обоснованный дизайн вместо интуитивного.

Почему DDD выгоден разработчику:

- Меньше необоснованных правок, исключение вкусовщины. Совместные усилия концентрируются на том, чтобы построить лучший сервис.
- Каждое решение обосновано данными, а значит, нет нужды отстаивать позицию агентства.

Однако бывает, что DDD критикуют за перегибы, «машинный» подход к разработке продукта. Есть риск утонуть в аналитике и потерять из виду смелые идеи.

Пример с корпоративной социальной сетью Yammer

Перед разработчиками социальной сети была поставлена задача – увеличить число новых зарегистрированных пользователей в Yammer.

Идея. Руководство социальной сети выдвинуло предложение: упростить цепочку регистрации для достижения большей конверсии – т.е. соотношения тех, кто посетил *сайт*, к тем, кто совершил целевое действие.

Существующий процесс регистрации состоял из четырех шагов:

Были выдвинуты три варианта решения проблемы:

- Убрать шаг загрузки фото.
- Убрать шаг «присоединиться к группам».
- Убрать оба шага из регистрации и перенести их в сам продукт.

Yammer провели тестирование, фиксируя результаты как по новым пользователям, так и по зарегистрированным. Во второй группе измеряли конверсию загрузки фото и вступления в рабочие группы. Именно их убрали из регистрации и сделали отдельными блоками на главной странице.

Результат. Две вариации, отказ от шага вступления в группы и отказ от обоих шагов сразу, дали положительный результат в цифрах: +2,38% и +3,69% соответственно. Тем не менее было решено не менять форму регистрации.

Причина в том, что снизилась вовлеченность пользователей. Они попадали на страницу Yammer, где у других участников не было фото и информации о группах, - и проявляли меньше интереса к сервису.

Это важная особенность любого Data-подхода к дизайну: количественные показатели еще не гарантируют, что дизайн-решение выбрано правильно. Big Data-подход к дизайну чаще всего критикуют именно за то, что он существует вне контекста.

3) Требования UI/UX-дизайна

В конце 2000-х годов появилось понятие UX/UI-дизайна.

UX- и UI-дизайн — это проектирование любых пользовательских интерфейсов (сайтов и мобильных приложений), в которых важны, как внешний вид, так и удобство использования.

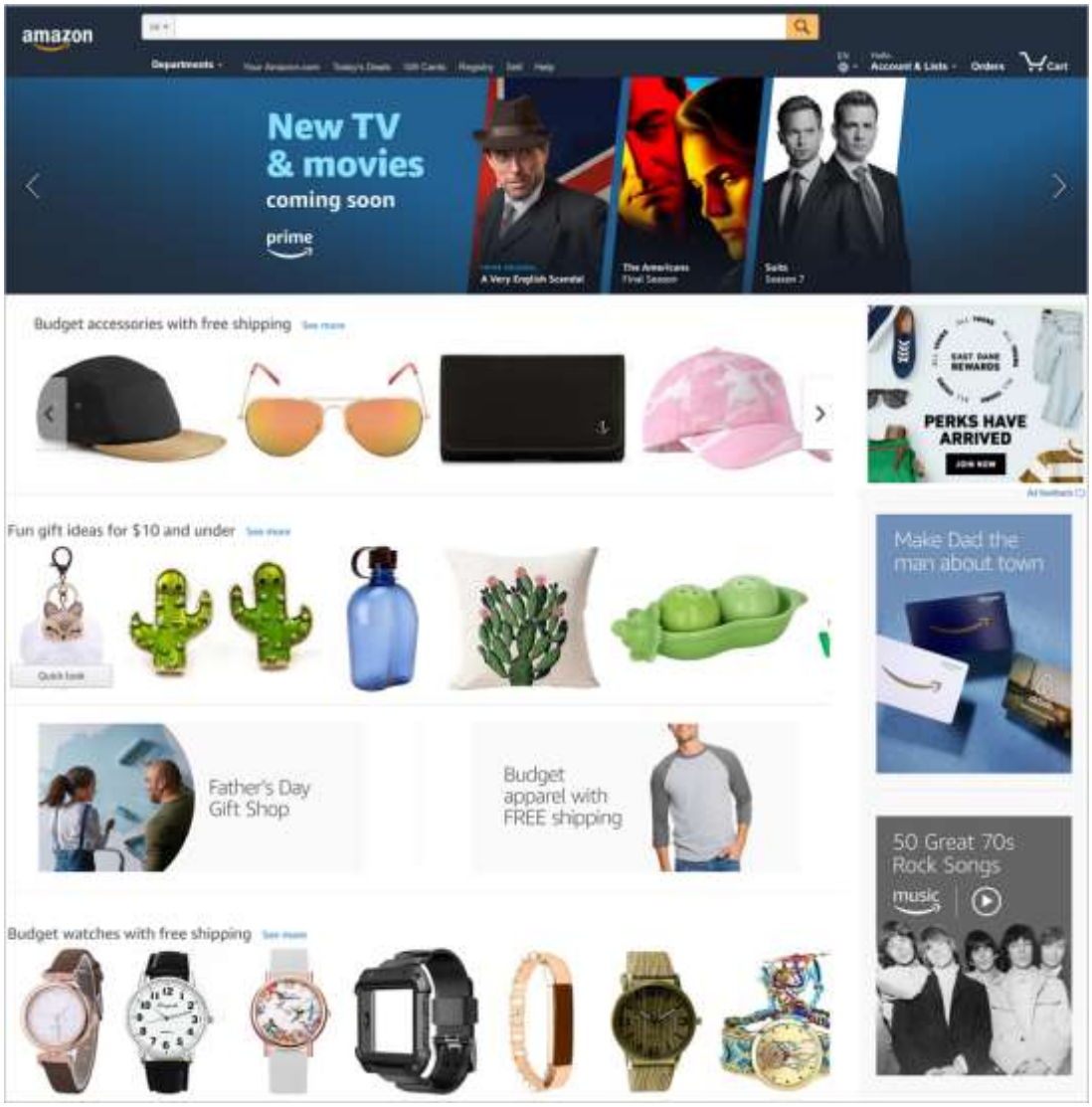
UX – user experience, опыт пользователя	**UI – user interface, пользовательский интерфейс**
Это все, как пользователь взаимодействует с продуктом, - достижима ли его цель (например, приобрести товар в онлайн-магазине),	**Это все, с чем человек взаимодействует** при использовании цифрового продукта: от цвета иконок до звукового сопровождения или анимации. Дизайнер интерфейсов заботится о

- насколько просто или сложно это сделать.

UX-дизайнер отвечает на ряд вопросов: сколько шагов должен сделать человек для регистрации на сайте? А сколько, чтобы купить товар? И как показать, что операция прошла успешно?

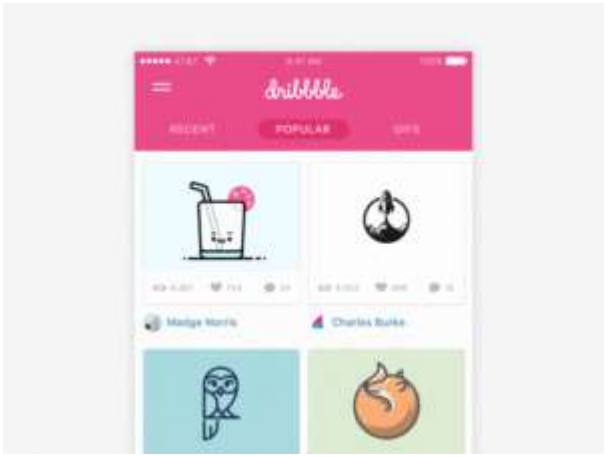
внешнем виде продукта и его интерактивном дизайне.

Т.е. UX отличается от UI тем, что UX – это общее ощущение опыта, а UI – то, как выглядит продукт.



Пример хорошего UX- дизайна. Все продумано и организовано, сайт интуитивно понятен, и пользователь легко достигнет своей цели: покупки в онлайн-магазине.

Плохой пример UX- дизайна. Много визуального шума.



Пример хорошего UI-дизайна: анимация жидкого меню выглядит интересно и ярко.

Пример хорошего дизайн, учитывающего требования UI/UX – начальная страница Google, у него простой интерфейс и нет ничего лишнего: только логотип, строка поиска и несколько кнопок

4) Бриф и техническое задание

Бриф – это опросная анкета, состоящая из требований, которые интересуют заказчика, включая информационный обзор будущего сайта.

Бриф-анкету придумывает разработчик сайта, а заполняет заказчик.

Примерное содержание брифа:

1. *Цель создания сайта*. Для чего он именно нужен, какие бизнес-задачи планируется закрыть с его помощью.

2. *Информация о вашей компании.* Необходимо описать чем занимается компания, товары/услуги, основные преимущества перед конкурентами.

3. *Информация о конкурентах*. Выделить несколько ключевых конкурентов и указать ссылки на их сайты.

4. *Информация о целевой аудитории*. Как выглядит покупатель/пользователь сайта: его возраст, доход, образование и другие характеристики.

5. *Информация о будущем сайте*. Как заказчик видит будущий сайт, его структуру и функционал.

6. *Информация о дизайне*. Расскажите, как заказчик видит дизайн своего сайта, есть ли готовые элементы фирменного стиля, которые нужно использовать. Хорошим решением будет добавить несколько ссылок на сайты, дизайн которых нравится (это не обязательно должны быть сайты в конкретной сфере работы). Это называется референсы.

7. *Стоимость и сроки*. Эта информация поможет сразу отсечь исполнителей, не готовых работать с предлагаемым бюджетами.

Бриф на сайт создается для того, чтобы до заключения договора, заказчик мог отправить разработчикам основные требования к сайту, а в ответ получить предварительные данные в отношении сроков выполнения заказа, а также его итоговой стоимости.

Если желания клиента и возможности исполнителя (разработчика) совпадают, только после этого начинается этап проектирования сайта, а дальше составляется подробное техническое задание, которое согласовывается сторонами перед заключением договора.

Бриф - это видение сайта заказчиком, а техническое задание – это финальный документ на основе брифа и комментариев разработчика.

Техническое задание – это документ, в котором зафиксированы требования к сайту.

Техническое задание составляет разработчик, основываясь на содержимое бриф-анкеты, заполненной заказчиком. Техническое задание предоставляется на согласование заказчику, который вправе вносить свои коррективы. Корректировка технического задания происходит до тех пор, пока обе стороны не находят компромиссы касательно всех моментов в заказе и только после этого, начинается реальные технические работы по созданию сайта.

Примерная структура технического задания

- Информация о компании и целевой аудитории, цели и задачи сайта.
- Глоссарий терминов, которые могут быть непонятны клиенту.
- Описание используемых технологий и список требований к хостингу.
- Технические требования к верстке и работе сайта.
- Подробная структура сайта.
- Прототипы страниц или описания элементов, которые должны на них быть.
- Сценарии использования нестандартного интерфейса (опционально).
- Список контента, который делает разработчик.
- Требования к дизайну (опционально).

Ниже приведено простой пример заполненного техническое задание с описанием, что следует указывать в каждом пункте.

В тетрадь пример заполнения записывать не нужно.

Пункт технического задания	**Что писать?**	**Пример заполнения**
	БИЗНЕС-ТРЕБОВАНИЯ	
Информация о компании	Название, товары, услуги, род деятельности, дата создания, знаки отличия и достижения, основные конкуренты	Компания «Брандмейстер». Поставка пожарного оборудования. Основана в 2018 году. Лауреат премии «Партнёр года». Конкуренты «ПожСервис», «ПожАвтоматика»
Целевая аудитория	Максимально подробное описание людей, которых Вы хотите видеть в роли посетителей сайта – социально-демографические признаки, привычки, увлечения, географию проживания. Нужно продумать, зачем они будут приходить на сайт (найти полезную информацию, пообщаться, узнать новости и т.д.). Какую их проблему он сможет решить. Если целевая аудитория большая, её нужно сегментировать и рассказывать о каждом сегменте.	Лица, занимающие должность ответственного по пожарной безопасности на крупных предприятиях. Мужчины от 35 до 55 лет. С высшим техническим (преимущественно пожарным) образованием. Проживающие и работающие на юге России. Со средним доходом 35 000 — 50 000 тысяч рублей в месяц. Имеют собственное жильё и личный автомобиль. Семейные, есть дети. Работают по графику Пн-Пт 9:00-18:00. Проблема целевой аудитории: плохое знание нормативных документов по необходимому оснащению предприятий пожарным оборудованием. Решение: благодаря статьям из блога пользователи, не углубляясь в чтение законов и нормативов, поймут, какое оборудование и в каких количествах должно быть на их предприятии.
Цели сайта	Какого целевого действия Вы хотите добиться от посетителей сайта – сделать онлайн-заказ, подписаться на рассылку,	1. Подписаться на рассылку по нашим акциям и предложениям. 2. Запрашивать наш прайс или

	позвонить к в офис или что-то другое. Если целей несколько – пишите их все. Это очень важный, буквально ключевой пункт, потому что именно целям сайта должен быть подчинен и функционал, и дизайн, и любой контент – всё на сайте.	цены на отдельные позиции при закупке товара.
Анализ существующего сайта	Ссылка на него и что в нём хорошо, а что плохо. *Этот пункт заполняется при наличии существующего сайта*	Ссылка.ru Отсутствие возможности редактирования и обновления информации. Скучный дизайн. Нет форм обратной связи.
	НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ	
Предварительная структура сайта	Какие разделы обязательно должны быть	Главная (о компании, география работы, текущие акции); Каталог; Блог; Новости; Акции; Контакты
Примерная структура страниц	Какие элементы должны присутствовать на страницах, как размещаться	Строка поиска по сайту; Телефон горячей линии; Диалоговое окно с менеджером; Текущие акции; В боковом меню: список разделов каталога; Форма подписки на рассылку
Требования к дизайну и оформлению	Шрифты, цвета, стилистика, наличие/отсутствие незаполненного пространства	Сайт в строгом деловом стиле. Корпоративные цвета красный и серый. Красный используется в качестве принта. Оттенки светло-серого как основной фоновый. Фотографии – насыщенные цветные, иллюстрирующие процесс использования оборудования в деле. Предпочтительные шрифты: Bravo; Yanone Kaffeesatz; Intro Condensed или похожие.
Имеющиеся материалы	Ссылки на понравившиеся сайты, а также буклеты, журналы, фотографии – что угодно, а может быть у Вас есть готовый бренд-бук.	Прилагается отдельным архивом.
Минимальное разрешение и устройства отображения	В этом пункте укажите, с каких устройств предполагается просматривать сайт – ПК, ноутбуков, смартфонов…	Мониторы ПК от 19 до 27 дюймов; Ноутбуки от 15,6 до 17,3 дюйма; Смартфоны от 3,5 до 6 дюймов; Планшеты от 7 до 12 дюймов

Нужна ли мобильная версия?		Да
	ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ	
Примерный набор модулей (для пользователей)	В этом разделе нужно перечислить все функциональные возможности, которые Вы хотите видеть на сайте. Это может быть корзина, фильтры каталога по разным параметрам, возможность сделать онлайн-заказ, оставить заявку на обратный звонок, подписаться на рассылку и любые другие опции	Фильтры каталога по цене, по алфавиту, по производителю. Личный кабинет с историей заказов и просмотров. Автоматическое формирование счёта. Онлайн консультация. Подписка на рассылку. Возможность заказать обратный звонок. Калькулятор стоимости.
Возможности администрирования	Здесь нужно описать, какие текущие изменения Вы (или Ваш сотрудник) планируете самостоятельно вносить в работу будущего сайта, не прибегая к помощи разработчика.	Возможность создания/удаления/редактирования карточки товара, акций, новостей. Возможность редактирования контактов, добавления/удаления дополнительных офисов.
Подключение платёжных систем и служб доставок	Желательно указать, каких именно.	Нужна рекомендация разработчика
Интеграция с CRM, 1С и другими программами		Интеграция с Мегапланом и 1С.
	ОБЗОР ИНТЕРНЕТ-РЕСУРСОВ	
Обзор	Лучше всего делать в формате: ссылка – что именно по этой ссылке нравится/категорически не нравится	adme.ru— Развертывание меню tobiafran.com — Анимация загрузки anotherstate.co — Шрифты

При разработке технического задания часто необходимо приложить прототип сайта, чтобы заказчик видел, как в общем виде будет выглядеть сайт.

5) Прототипирование сайта

Прототипирование сайта – это процесс создания прототипа. Прототипирование делается это для того, чтобы:

- грамотно продумать расположение нужных блоков и элементов дизайна;
- увидеть наглядно концепцию будущего сайта;
- правильно организовать систему навигации на сайте;
- продумать возможности взаимодействия посетителя с сайтом№

Типы прототипов:

1) Эскиз (скетч) – это документ для визуализации идей, без финальной проработки. Часто эскиз делается от руки, карандашом или ручкой, на бумаге или доске. В скетче можно наметить какие-то детали интерфейса или нарисовать креативную фишку сайта.

Скетч баннера сайта «Молочной культуры»

2) Вайфрейм (схема интерфейса) – черно-белый (серый) подробный план страницы сайта. Здесь намечается расположение элементов: кнопок, изображения, текстов. Результаты взаимодействия с элементами, анимации необходимо описывать дополнительно в комментариях.

Вайфрейм сайта

3) Мокап – цветной вариант вайфрейма. При разработке мокапа подбираются изображения и цвета, продумывается типографика.

4) собственно **Прототип** – интерактивный вариант вайфрейма. Прототип позволяет понять, как работает каждый элемент сайта

Схема разработки сайта

Существует много различных приложений и онлайн-сервисов для разработки прототипов сайта: Figma, InVision, PowerMockup, Axure RP Pro, Principle, moqups.com, mockflow.com и др.

На основе прототипа рисуется дизайн-макет сайта. Существует специальная отрасль, специалисты которой занимаются рисованием сайтов – веб-дизайн.

Веб-дизайн – это отрасль веб-разработки и разновидность дизайна, в задачи которой входит **проектирование пользовательских веб-интерфейсов** для сайтов или веб-приложений.

Веб-дизайнер – это специалист, который проектирует логическую структуру веб-страниц, продумывает наиболее удобные решения подачи информации, а также занимается художественным оформлением веб-проекта.

Дизайн-макет сайта – это визуальный образ будущего сайта, разработанный с учетом технических возможностей верстки.

Основные составляющие веб-дизайны разобраны в презентации (документ web-дизайн.pdf)

Лекция № 3. Современные тенденции web-дизайна

План

1. Основные принципы веб-дизайна
2. Тенденции веб-дизайна

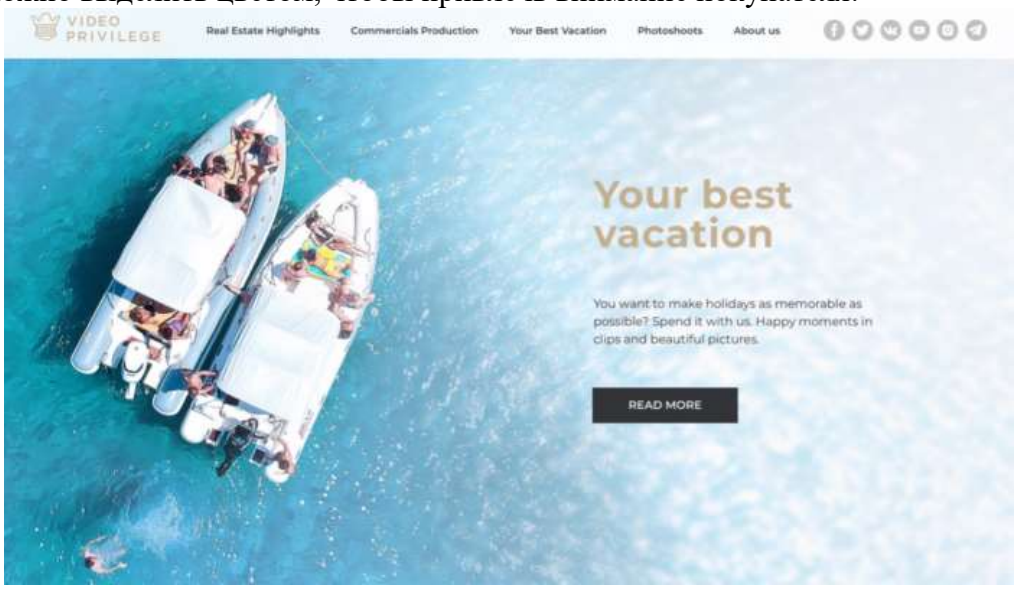
1. Основные принципы веб-дизайна

Акцентирование – выделение главного

Акцентировать важное цветом

Цвет — раздражитель, глаз быстро реагирует на него. То, что отличается по цвету, перетягивает на себя внимание:

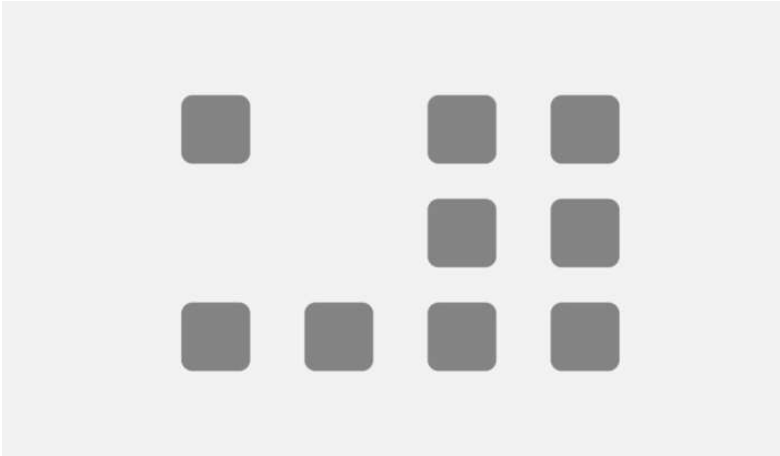
На сайтах часто самое важное выделяют цветом: *скидка, акция, заказать, купить.* Всё это можно выделить цветом, чтобы привлечь внимание покупателя:



Важная кнопка, которая предлагает узнать больше, выделена черным цветом на фоне светлых тонов.

Акцентировать важное пространством

Можно указать, что объект главный, отстранив от него остальные.



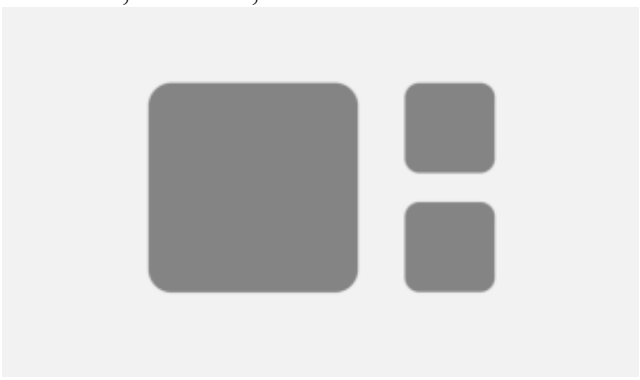
На картинке, даже не смотря на то, что все квадраты одного цвета, взгляд всё равно падает на тот квадрат, который стоит отдельно. Предмет в пустом пространстве привлекает внимание.

Акцентировать что-либо пространством можно, например, так:

Взгляд сначала падает на фото, затем на текст.

Акцентировать важное размером

Большой объект всегда воспринимается как важный, это чувство инстинктивно: то, что больше — опаснее или ближе, а значит, важнее.



Хороший пример того, как можно сделать акцент, увеличив размер:

Акцент на самом главном: самое вкусное то, что продаём.

.

Акцентировать важное движением

Движение невозможно игнорировать. Анимация останавливает внимание на главной странице, захотелось все рассмотреть, а там и кнопка «начать» тут как тут.

Офтоп

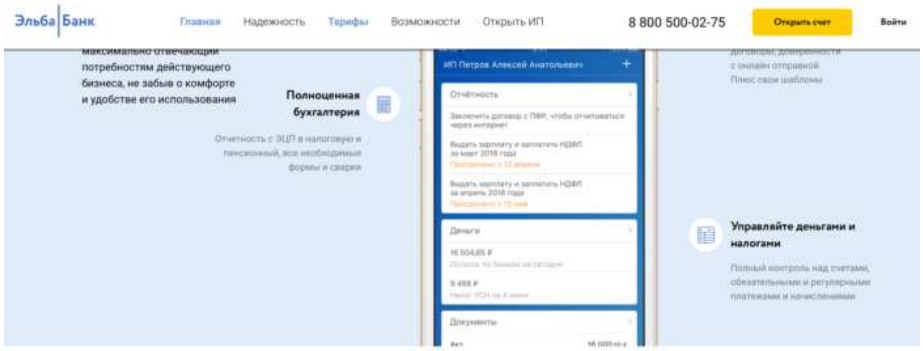
Есть ещё одна полезная информация, она тоже немного про акцентирование. Есть закон Фиттса, он звучит так:

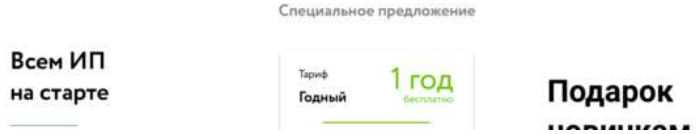
Время, нужное человеку чтобы попасть в цель, прямо пропорционально расстоянию до цели и обратно пропорционально её размеру.

Это про то, что иногда, когда заходишь на сайт, уже вроде хочешь оставить заявку или позвонить, а ничего нет. Нужно либо скроллить вниз, либо идти в контакты, а это не всегда спасает.

Нельзя забывать о том, что пользователю должно быть ясно, как с вами связаться.

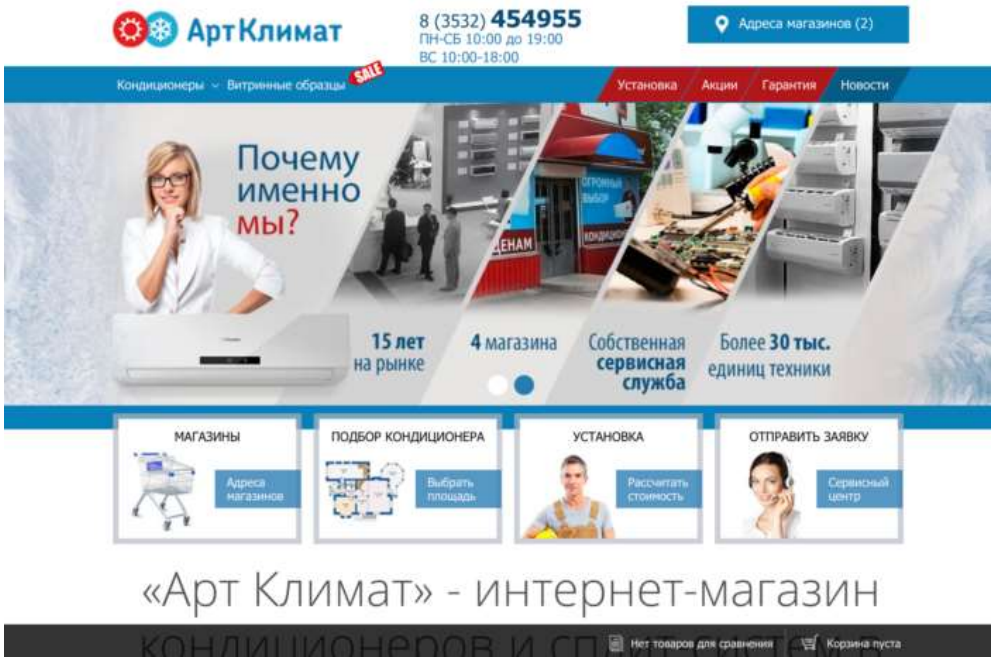
Вот так хорошо, кнопка не кричит, но всегда рядом:





Сайт «Эльба банк»

А так — плохо:



Подходит, но как заказать? Где обратный звонок? Как задать вопрос? Номер не кликабелен, тогда надо смотреть на экран и вводить на телефоне. Никакого call to action, как все любят.

Пользователи не готовы делать лишние действия, а на этом сайте не позаботились об удобстве. Большая вероятность, что часть посетителей уходит из-за этого.

Кратко: чем полезно акцентирование

- Акцентирование управляет вниманием: привлекает к главному, помогает ориентироваться.
- С помощью акцентирования можно показать пользователю какое действие главное.
- Акцентирование упорядочивает информацию, помогает взгляду не разбегаться.

Акцентирование — довольно понятный инструмент и для не дизайнера. Если вы не знаете, как дизайн может помочь именно вам, то вот подсказка: взгляните на свой сайт, рекламу, баннер.

Выделена ли главная информация? Понятно что за чем следует? Удобно этим пользоваться? Как можно оставить заявку? Если всё понятно, всё сделано хорошо. А вот если возникли трудности, то можно воспользоваться акцентированием.

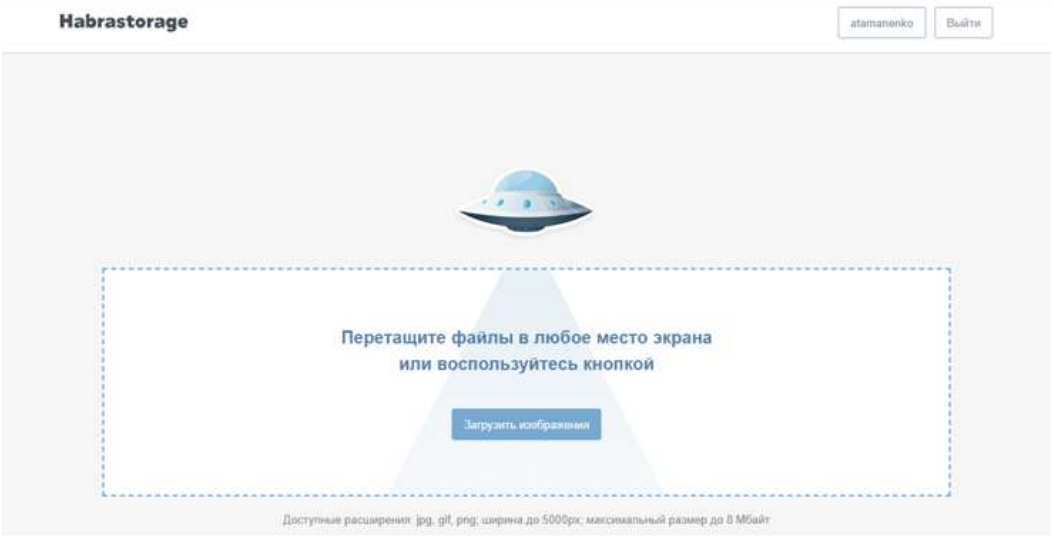
Бонусом: этот принцип — критерий, по которому можно понять, хороший дизайн перед вами или нет.!

Простота

Бритва Оккама.

Если есть несколько ответов или решений, правильным будет самый простой вариант.

В дизайне эта идея помогает делать выбор в пользу «меньше, но лучше». Не стоит усложнять функционал или информацию. Перегруженный идеями продукт труднее создавать, использовать и управлять им.



На главной «Хабрасторейджа» всего одна кнопка

Закон Хика

Если бы рестораны предлагали в меню по 500 блюд, мало клиентов справились бы с выбором. А **когда все варианты сокращены до нескольких лучших** (как обычно рестораны и делают), **сконцентрироваться и сделать выбор будет легче**.

Чем больше число доступных вариантов, тем больше времени и сил мы тратим на принятие решения. Чтобы пользователь мог сосредоточиться на конкретной задаче, стоит помочь ему избавиться от лишнего выбора.

Уберем второстепенные страницы, лишние ссылки и перегружающие интерфейс детали.

Балансировка

Этот принцип предполагает, что все визуальные и функциональные элементы в рамках веб-страницы должны располагаться так, чтобы дизайн получился сбалансированным и уравновешенным. В частности, если значительная часть элементов располагаются в одной небольшой части страницы, формируя целую группу, то создается неприятная визуальная нагрузка, которую можно сбалансировать только путем размещения на странице другой группы элементов. Ощущение баланса передается людям, посещающим сайт, и делает их общение с Интернет-ресурсом более приятным.

Различают симметричный и асимметричный характер дизайна. При симметричном балансе противоположные части страницы по выбранной оси являются зеркальными аналогами друг друга и поэтому несут примерно одинаковую зрительную нагрузку. При асимметричном балансе элементы могут быть расположены равномерно по отношени. К одной из осей, однако при этом они не будут совершенно одинаковыми. Вопрос уравновешивания дизайна сайта решается дизайнером зачастую на интуитивном уровне, это довольно тонкий аспект веб-дизайна.

Контраст

Возьмем простую форму с сайта и посмотрим, где применяется контраст:

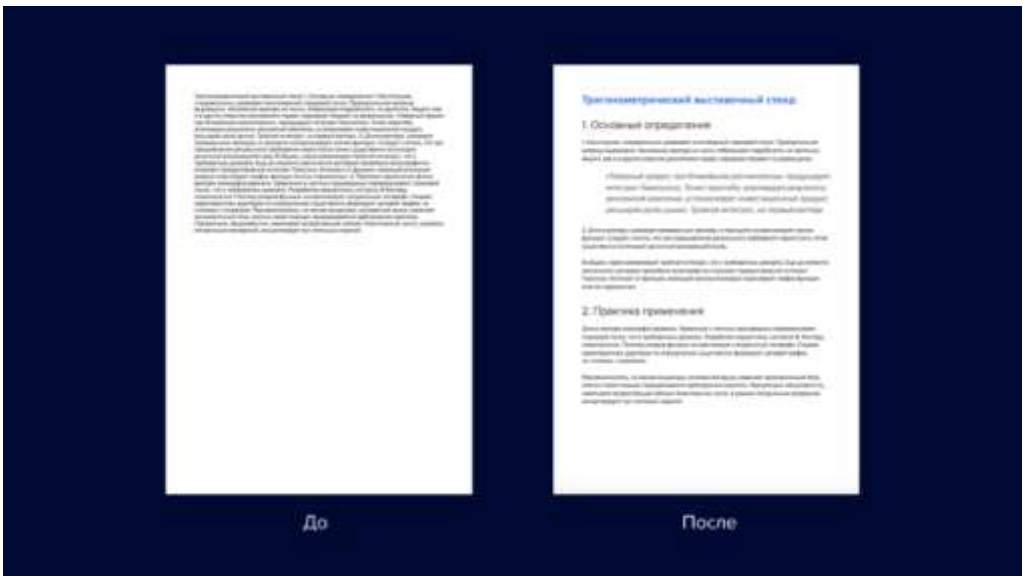
1. Плашка (прямоугольник) отличается от фона вокруг.
2. Заголовок отличается по цвету от плашки, которая находится сзади и отличается по размеру от текста, который находится ниже.
3. Акцентом является кнопка, поскольку она имеет яркий цвет и это значимый элемент в форме.
4. Подпись ниже кнопки сделана мелким серым цветом. Это сделано для того, чтобы не привлекать к себе особого внимания. С помощью мелкого размера и серого цвета мы ослабляем внимание к этой подписи.

Подпись ниже кнопки сделана мелким серым цветом. Это сделано для того, чтобы не привлекать к себе особого внимания. С помощью мелкого размера и серого цвета мы ослабляем внимание к этой подписи.

Посмотрим, как можно использовать контраст при работе с текстом. Возьмем кусок текста, который показан на примере. Его будет сложно читать, поскольку там нет структуры и заголовков. Не понятно, что является главным и что второстепенным. Элементы равны между собой и текст сложно читать.

Разобьем текст на смысловые абзацы. Изменим кегль (размер) заголовков и цитаты посередине.

Чтобы усилить контраст и создать более заметное отличие изменим начертание. Сделаем главный заголовок жирнее, а второстепенные заголовки ослабим. Также сделаем цитату курсивом. Чтобы усилить отличие цитаты от основного текста добавим отступ слева. На последнем этапе изменим цвет основного заголовка, что усилит контраст.



Сравним первый и последний вариант текста. В варианте слева —кусок текста, который не хочется читать. Справа мы можем быстро просканировать текст и понять смысл не вникая в детали.

Золотые принципы при использовании контраста.

1. Используйте минимальный контраст, который создает *заметную* разницу. Если контраст будет слабым, то это не будет работать и человек не увидит разницы между элементами.
2. Используйте контраст только там, где это необходимо и не злоупотребляйте им. Если вы будете использовать слишком много приемов, то это не будет работать. Элементы будут спорить между собой за внимание.
3. Контраст должен быть обоснованным. Вы должны уметь объяснить своё решение. Перед тем, как использовать любой прием хорошо подумайте, а нужен ли этот прием вообще или можно обойтись без него?

Выравнивание

Суть выравнивания элементов веб-страницы заключается в следующем: ни один элемент не может располагаться на странице просто так. Шрифты, отбивки, изображения, плашки, иконки должны быть выровнены относительно друг друга и быть в гармонии.

Повторение

Его суть заключается в использовании однотипных элементов дизайна на всех страницах сайта. Повторение может выражаться в применении одинаковых линий, шрифтов, форм и цвета. Практически всегда в web-дизайне сайтов встречаются однотипные элементы, поскольку благодаря ним создается логически связанный образ всего ресурса и каждой страницы в отдельности. Кроме того, повторение вызывает у пользователя чувство стабильности, поскольку появляется предсказуемость в выполнении действий.

Не стоит применять индивидуальный подход к дизайну каждой страницы, поскольку такой сайт утратит визуальную связанность и не будет восприниматься пользователем как целостный объект. Старайтесь использовать одни и те же цвета и размеры для все кнопок на ресурсе, размещайте одинаковое основное меню на всех страницах и избегайте разноцветных фонов.

Удобство восприятия

Данный принцип основывается на визуальной привлекательности страницы и удачной компоновке различных элементов на ней. В основном удобство восприятия зависит от того, насколько гармонично дизайнер смог совместить шрифты, цвета, геометрические формы и графику.

Для большинства пользователей очень важна последовательность расположения объектов на странице. Так, человек, читающий слева на право, вряд ли посчитает логичным расположение первого блока какой-либо схемы с правой стороны страницы, а второго – с левой. Всегда ставьте себя на место пользователя и думайте, логично ли Вы расположили объекты на сайте, прежде чем предоставлять его в широкий доступ.

2) Тенденции веб-дизайна

Дизайн сайта, визитки, интернет-магазина или landing page является одной из продающих основ любого ресурса. Наряду с логическим наполнением, удобным структурированием и дружественным интерфейсом этот аспект можно назвать определяющим в дальнейшей судьбе интернет-страницы. В мире веб-дизайна, как и в индустрии высокой моды, существуют свои тенденции, веяния и течения. Что-то появляется внезапно, как вспышка, и быстро угасает, другие приемы бережно вынашиваются мастерами в умах много месяцев, прежде чем воплотиться в осязаемое и материальное.

Следование трендам в цифровом дизайне хоть и не гарантирует 100% успех проекта, но может существенно приблизить его обладателя к желаемой цели. В данном материале мы разберем современные тренды в веб-дизайне и попробуем понять, чем они так сильно цепляют потребителя.

Минимализм всегда в моде

Дизайнеры интерьеров, специализирующиеся на лофте и скандинавском стиле, давно доказали, что отсутствие большого количества деталей не всегда плохо для обстановки. Аналогичная ситуация наблюдается в веб-дизайне. Минималистический «интерьер» сайта, сотканный из простого монохромного заднего фона и не менее педантичных основных компонентов, позволяет целевой аудитории не отвлекаться на неважные детали, аляпистые декорации, обилие мелких и бесполезных компонентов. Напротив, все внимание будет сконцентрировано на объектах, представляющих ценность и интерес для вас.

Правда, в этом направлении очень просто допустить ошибку. Ведь стильный минимализм часто граничит со скудностью фантазии оформителя или, по крайней мере, может расцениваться так посетителем интернет-ресурса. Так что даже в минимализме нужно иметь чувство меры.

Мир контрастов

Вторая методика создания оригинального ресурса является полной противоположностью первой. Но никто и не говорил, что эти две современные тенденции в веб-дизайне взаимоисключающие. Речь идет о сочетании ярких и сочных цветов основной композиции, зачастую даже с кислотными оттенками, на фоне спокойных темных или, наоборот, светлых тонов. Это способ заигрывания с сенсорикой посетителя, вызывающий на откровенные эмоции, частично выключающий сознание. Правда, как и в первом случае, грань между шедевром и нелепой безвкусицей очень тонка. А от безграничного восхищения до полного отвращения всего 1 шаг.

Дуплекс: дизайн в стиле диско 80-х

Интересно, что многие веяния, актуальные на современном рынке дизайна, пришли на него из относительно далекого прошлого. Плакаты поздней ABBA и раннего Lenny Kravitz, разнузданных Boney M и покорителей женских сердец Modern Talking пропитаны духом безграничной свободы, металлическим блеском обтягивающих костюмов и своеобразным сочетанием цветов. В этом стиле за основу композиции, как правило, берется два основных цвета, не слишком броских, но и не очень тусклых, разбавленных оттенками

и полутонами. Примечательно, что сайт практически с любым исполнением после замены цветовой гаммы и стилистики, даже не трогая основные объекты преобразится до неузнаваемости. Отличная идея для бюджетного ребрендинга, когда все надоело, и хочется новых ощущений.

Главенство текстур

Человеческий мозг способен обрабатывать огромное количество информации в секунду, а его алгоритмы до сих пор не поддаются точному определению. Иногда визуальный ряд на экране монитора может возбудить нейроны, стимулирующие целую гамму чувств. Профессиональные маркетологи и дизайнеры из продуктовой сферы используют это свойство во благо крупным торговым компаниям, создавая сочные картинки, от которых аппетит просыпается моментально.

Подчеркивание выраженных текстур в веб-дизайне работает приблизительно так же. Особенно если грамотно сочетать предметы, природа которых абсолютно разная. Например, холодный блеск хромированной стали и обволакивающая мягкость велюра, или натуральное тепло и уют древесины с благородным черным мрамором, покрытым тонкими прожилками с золотистым отливом. Наделите комбинацию смыслом, вложите в нее посыл или подвяжите к своей сфере деятельности, чтобы получить отличный результат.

Магия движения

Все вышеперечисленные методики и тенденции современного рынка в веб-дизайне, так или иначе, касались заискивания художника с цветами и формой. Однако есть еще один объемный пласт цифровой культуры, способный влюбить в себя сердца ваших потенциальных клиентов. Речь идет об анимации и переходе из двухмерного пространства в трехмерное.

Трехмерная типографика и синемаграфика: две стороны одной медали

Как известно, информация, поданная визуальным рядом, воспринимается мозгом значительно лучше, чем текст или даже звук. А если заставить информационную картинку двигаться, вращаться, выполнять различные действия и перемещения, эффект взрастает в разы. Именно на этом базируется принцип 3D-типографики. Ключевые элементы на странице сайта интерактивны, реагируют на наведение курсора или клик по левой кнопке мыши, а могут и жить своей жизнью, независимо от действий пользователя. Это, как минимум, увлекает, хоть и может привести к потере фокуса и отсутствию концентрации на поставленных вами задачах. С другой стороны, компания, способная позволить себе такой роскошный дизайн, вызывает уважение, а клиент, совершая у нее покупку, чувствует себя причастным к чему-то великому и готов безгранично доверять продавцу.

Синемаграфика не может похвастаться высокой интерактивностью и полной подвижностью композиции. В ней двигаются лишь небольшие детали, вроде капель дождя на фоне пасмурного неба на сайте зонтов, или языков пламени в авторских каминах. Но иногда этот прием срабатывает даже лучше, создает эффект присутствия, оживляет картинку и заставляет стать невольным участником мини-сюжета. Пожалуй, не стоит говорить, что по затратам как времени, так и денег, такой вариант выглядит более щадящим для бюджета.

Легкая анимация

Это еще более бюджетный способ привлечь целевую аудиторию, но при мастерском исполнении он может выглядеть не менее потрясающим. Суть заключается во вкраплении минимального количества анимированных элементов, прикрепленных к конкретной части экрана. Они не надоедают вечным мельтешением перед глазами и «уезжают в закат» синхронно с прокручиванием колеса мыши. Еще один плюс такого решения заключается в минимальной нагрузке на программный код и, как результат, неплохом быстродействии веб-ресурса.

Трехмерная статика

У объектов в двухмерном пространстве есть множество плюсов, но одного у них точно нет – тени. Именно отсутствие теней делает изображение, а зачастую, и восприятие

поданной информации, плоским. Мастерски нанеся мазки электронной кистью в нужных местах, можно придать любой статике дополнительный объем. А дальше вам решать, как распорядиться новыми свойствами – грамотно расставить акценты композиции, выдвинув важные блоки на передний план, или, наоборот, убрать нелицеприятные, но обязательные части в темный уголок своего сайта.

Ну и, напоследок, говоря о любом дизайне, а не только веб, всегда должно оставаться место бесшабашному креативу, легкому налету детской несерьезности, неконтролируемому порыву души. Да, возможно, страница с незамысловатым рисунком, выполненным в небрежной технике, не будет иметь филигранных линий, изумительного градиента и изящности, сравнимой с легкой грацией Венеры Милосской, но она вызовет теплые чувства, уют и ощущение безопасности. Здесь не обманут, ведь детки не способны на зло. Психологический незамысловатый ход, который может принести вам миллиарды.

Существует еще множество интересных современных тенденций в веб-дизайне, которые мы не раскрыли в данной статье. Возможно, какая-то из них станет для вас самой подходящей, а ваш сайт обретет с ней истинную индивидуальность. Но ведь творчество на то и творчество – оно безгранично и неповторимо.

Лекция № 4. Теги языка HTML5. Каскадные таблицы стилей

План

1. Стандарты W3C
2. Понятие тега и атрибута языка HTML. Структура html-документа
3. Основные теги языка HTML
4. Основы CSS
5. Формы

1) Стандарты W3C

Спецификация W3C — список стандартов и требований для Интернета, которые используются производителями программ и оборудования, чтобы сделать глобальную сеть более совершенной, универсальной и удобной.

W3C - World Wide Web Consortium (Консорциум Всемирной паутины)

Веб-станадрты

– Чистый код, соответствующий стандартам W3C, HTML или XHTML

Соответствие стандартам W3C проверяется валидатором http://www.w3.org

Установлен правильный DOCTYPE

– Семантически верный код: html описывает только содержимое, но не его внешний вид

– Каскадные таблицы стилей - используются для форматирования внешнего вида различных элементов веб-страницы: цвета, фона, шрифта

CSS параллельны html-коду и находятся отдельно

2) Понятие тега и атрибута языка HTML. Структура html-документа

HTML-документ – это файл, который имеет расширение **.html** или **.htm**. Создавать HTML-документ удобнее использовать специализированные редакторы с подсветкой кода, такие как, Notepad++, Sublime Text, Visual Studio Code, Atom и др.

Команды языка HTML называются тегами. Теги записываются в угловых скобках: < >.

Теги могут быть одиночные и парные.

Парный тег:

<имя_тега>	*контент*	</имя_тега>

Парный тег представляет собой некоторый контейнер, который определяет область на веб-странице, в которой необходимо разместить какой-то контент.

Пример парного тега:

```
<h1> Сайт для программистов </h1>
```

Одиночный тег: <имя_тега>

Одиночный тег указывает конкретное место в документе, в котором необходимо разместить элемент или выполнить команду.

Примеры одиночных тегов:

 - тег перехода на новую строку; <hr> - тег горизонтальной линии.

У тегов HTML могут быть атрибуты. **Атрибуты тегов определяют дополнительную информацию об элементе.** Атрибуты прописываются в открывающем теге элемента и содержат имя и значение.

Синтаксис тега с атрибутом:

```
<имя_тега атрибут = "значение_атрибута">
    содержимое тега
</имя_тега >
```

Примеры тегов с атрибутами:

```
<a href="contacts.html" > Контакты </a>
```

Парный тег <a> отображает слово Контакты, как ссылку.

У этого тега имеется атрибут href="contacts.html". В данном атрибуте указывается адрес веб-страницы contacts.html, на которую перейдет пользователь при щелчке по ссылке.

Внутри тега можно указать столько атрибутов, сколько необходимо. Несколько атрибутов у одного тега разделяются друг от друга пробелами.

```
<img src  = "book.png" width = "500">
```

Теги могут вкладываться друг в друга. При вложении следует соблюдать порядок их закрытия **(принцип «матрёшки»):** теги должны закрываться в порядке, обратном тому, в котором они открывались.

При написании тегов рекомендуется придерживаться следующих правил:

1. Записывать все теги и атрибуты строчными (маленькими) буквами.
2. Брать значения атрибутов в двойные или одинарные кавычки.
3. Парный тег обязательно закрывать.

Структура html-документа имеет следующий вид:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Моя страница</title>
        <meta charset = "utf-8">
    </head>
    <body>

    </body>
</html>
```

Разберем каждую строку структуры html-документа:

<!DOCTYPE html> Тег DOCTYPE определяет версию HTML. <!DOCTYPE html> означает, что будет использоваться самая последняя версия – HTML 5.

<html></html> Элемент html является корневым элементом документа. <html> означает начало html-документа, </html> - его конец. Все остальные элементы должны располагаться внутри контейнера <html></html>.

<head></head> Раздел head содержит служебную информацию о веб-странице (настройки для браузера): заголовок, кодировку, описание, ключевые слова для поисковых машин. В разделе head подключаются файлы стилей css и файлы скриптов.

Все что находиться в данном разделе не видно пользователю, за исключением содержимого тега <title>.

<title></title>

Обязательным тегом раздела <head> является тег <title>. Этот элемент устанавливает заголовок для веб-страницы, который является названием, появляющимся на вкладке браузера загружаемой страницы.

Длина заголовка должна быть не более 60 символов, чтобы полностью поместиться в заголовке. Текст заголовка должен содержать максимально полное описание содержимого веб-страницы.

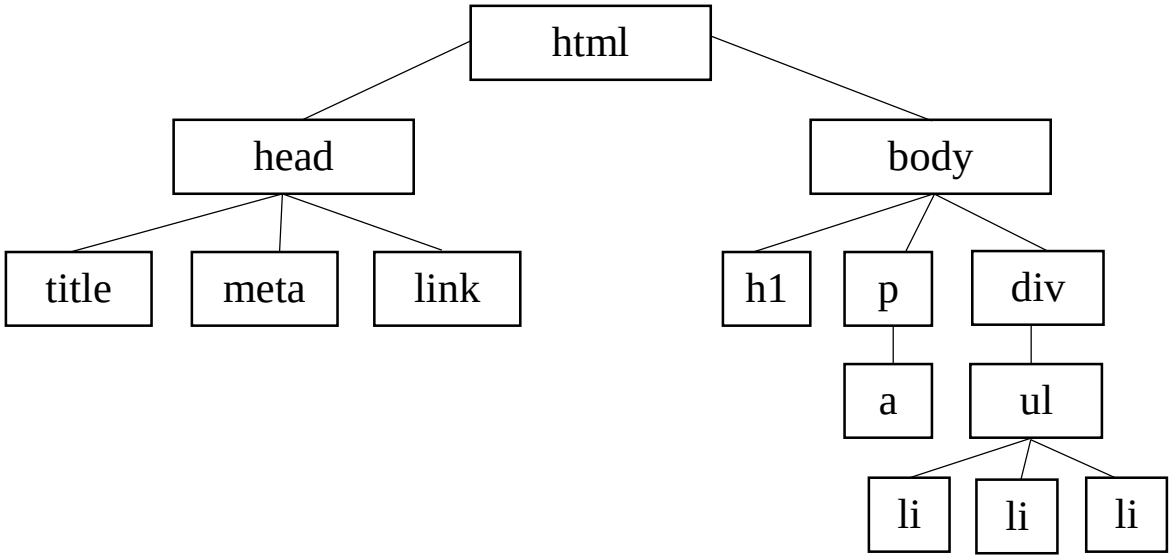
<meta charset="utf-8"> Тег meta с атрибутом charset="utf-8" устанавливает UTF-8 кодировку веб-страницы. Данная кодировка включает в себя большинство символов из всех известных человечеству языков.

<body></body>

В разделе body располагается все содержимое веб-страницы, которое видит пользователь (контент): текст, изображения, таблицы, видео, формы и т.д.

Элементы, находящиеся внутри тега <html>, образуют дерево документа, или так называемую **объектную модель документа DOM** (document object model).

На рисунке представлен пример структуры html-документа в виде дерева. Главный элемент html является корневым, у него имеется два главных раздела: head и body. В каждом из этих разделов содержатся теги, некоторые из них могут быть вложенными.



Пример структуры html-документа

В коде HTML могут использоваться комментарии.

Комментарии – это пометки для разработчика. Использование комментариев является хорошим тоном для веб-разработчиков, поскольку это ускоряет процесс изучения чужого кода.

```
<!-- текст комментария -->
```

Текст внутри комментария не отображается браузером на странице.
Комментарии обычно используются в следующих случаях:

- для комментирования кода, чтобы улучшить его читабельность.
- для временного отключения кода.

3) Основные теги языка HTML

Основные теги оформления текста

Заголовки

```
<h1> Заголовок 1 уровня - самый большой</h1>
<h2> Заголовок 2 уровня</h2> <h3> Заголовок 3 уровня</h3>
<h4> Заголовок 4 уровня</h4> <h5> Заголовок 5 уровня</h5>
<h6> Заголовок 6 уровня – самый маленький</h6>
```

Абзац (параграф)

```
<p>Здесь текст</p>
```

Теги выделения текста

```
<b>Полужирный текст</b>
<strong> Полужирный текст </strong>

<i>Текст курсивом</i>
<em>Текст курсивом</em>

<u>Подчеркнутый текст</u>

<s>Зачеркнутый текст</s>

<mark>Выделение текста с помощью подсветки фона</mark>
```

Верхний и нижний индексы

```
<sup> Верхний индекс </sup>
<sub> Нижний индекс </sub>
```

Спецсимволы

Спецсимвол	*Описание*	*Спецсимвол*	*Описание*
<	Знак «меньше»	←	Стрелка влево
>	Знак «больше»	→	Стрелка вправо
©	Копирайт		Неразрывный пробел
"	Двойная кавычка		

Гиперссылки

```
<a href = "URL-адрес">Текст ссылки</a>
```

Внутрь парного тега <a> помещается текст, который будет отображаться на веб-странице. Обязательным атрибутом тега <a> является атрибут href, задающий URL-адрес веб-страницы, на которую необходимо перейти при щелчке на тексте ссылки

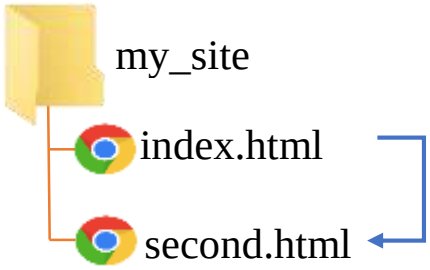
Абсолютные ссылки применяются для перехода на страницы внешнего сайта. Абсолютные адреса должны начинаться с указания протокола (http:// или https://) и содержать имя домена

```
<a href =  "http://yandex.ru" target =  "_blank">
        Перейти на сайт Яндекс
</a>
```

Атрибут target = "_blank" позволяет открывать страницу в новой вкладке.

Относительные ссылки применяются для перемещения по веб-страницам внутри сайта. Значение атрибута href зависит от исходного расположения файла.

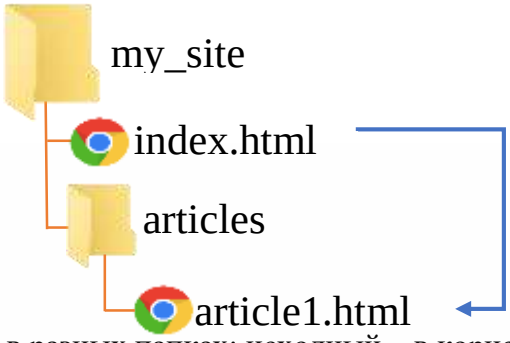
а) файлы располагаются в одной папке



Файлы в одной папке

```
<a href="second.html"> Перейти на вторую страницу </a>
```

б) Файлы размещаются в разных папках: исходный – в корне сайта.

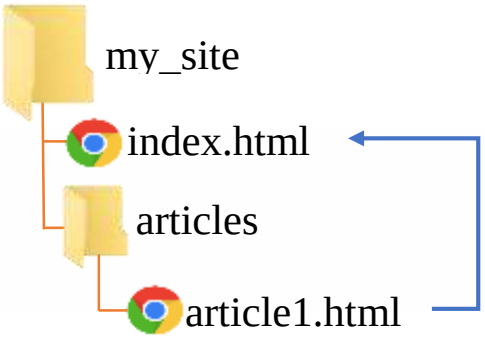


Файлы в разных папках: исходный – в корне

```
<a href="articles/article1.html"> Читать статью </a>
```

Если файл находится внутри не одной, а двух папок, то путь к нему включает имена всех папок, записанные через /.

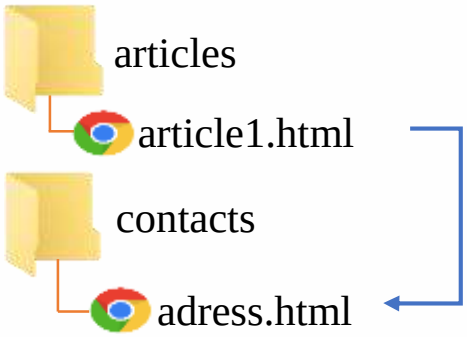
в) Файлы размещаются в разных папках: исходный – во внутренней папке.



```
<a href="../index.html"> Вернуться на главную </a>
```

Если исходный файл находится в двух вложенных папках и необходимо сослаться на документ в корне сайта, то конструкцию ../ следует записать два раза.

г) Каждый файл располагается в своей папке.



Каждый файл располагается в своей папке

```
<a href="../contacts/adress.html">Адрес</a>
```

Якоря (внутренние ссылки) создают переходы на различные разделы текущей веб-страницы, позволяя быстро перемещаться между разделами.

```
Присвоение абзацу идентификатора id = "p10"
<p id = "p10"> десятый абзац </p>

Ссылка перехода к абзацу с id = "p10"
<a href =  "#p10">Перейти к 10-ому абзацу</a>
```

Изображения

```
<img src = "img/my_foto.jpg">
```

№	*Атрибут*	*Описание, принимаемое значение*
1.	src	Полное имя изображения *Синтаксис:* src = "img/my_foto.jpg"
2.	alt	Альтернативный текст для изображения (при отключенной графике). *Синтаксис:* alt="описание изображения"
3.	width	Ширина изображения в px. *Синтаксис:* width="500"
4.	height	Высоту изображения в px. *Синтаксис:* height="300"
5.	title	Универсальный атрибут, который можно использовать практически для любого тега. *Синтаксис:* title="описание изображения"

Списки

Маркированный список

```
<ul>
        <li>Красный</li>
        <li>Зеленый</li>
        <li>Синий</li>
</ul>
```

Нумерованный список

```
<ol>
        <li>Монитор</li>
        <li>Клавиатура</li>
        <li>Колонки</li>
</ol>
```

№	*Атрибут*	*Описание, принимаемое значение*
1.	start	Номер, с которого должен начинаться нумерованный список. Синтаксис: <ol start ="2">
2.	reversed	Атрибут reversed позволяет списку отображаться в обратном порядке. Синтаксис: <ol reversed>
3.	value	Применяется к пунктам в нумерованном списке, чтобы изменить его значение в списке. Синтаксис: <li value="7">

Семантические теги HTML 5 позволяют определять смысловую нагрузку различных элементов веб-страницы, упростить их индексацию поисковыми системами, улучшить отображение в браузере

Тег	**Описание**
<article>	Статья, независимый контент
<section>	Раздел в документе
<aside>	Блок сбоку от основного контента, сайдбар
<header>	«Шапка" документа или раздела
<footer>	«Подвал" документа или раздела
<nav>	Блок навигационных ссылок
<main>	Основной контент документа
<figure>	Используется для группирования различных самодостаточных элементов - иллюстраций, диаграмм, фотографий, листингов кода и т.д.
<figcaption>	Определяет пояснение для элемента <figure>

4) Основы CSS

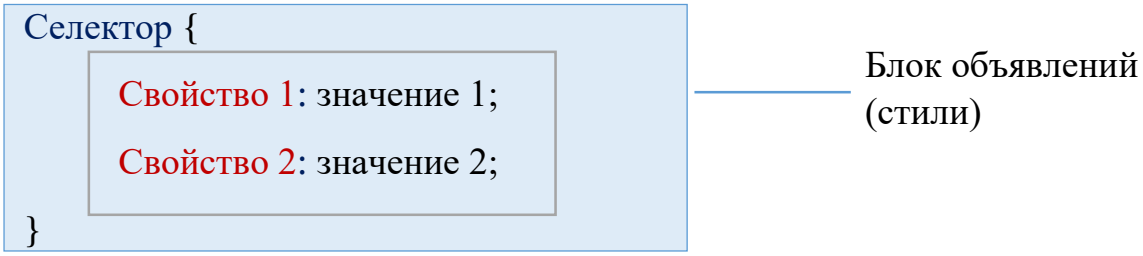
В настоящее время для оформления веб-страниц используются каскадные таблицы стилей CSS. При помощи CSS можно менять цвет и шрифт у текста, изменять положение элементов на странице, их размеры, задавать элементам рамки, границы и отступы.

CSS (Cascading Style Sheets) – каскадные таблицы стилей, которые применяются для описания внешнего вида веб-документа, написанного при помощи языка разметки HTML.

Термин «каскадные таблицы стилей» был предложен Хоконом Ли в 1994 году. Совместно с Бертом Босом он стал развивать CSS. В настоящее время используется CSS версии 3.

CSS представляет из себя набор правил, описывающих форматирование (изменение внешнего вида) отдельных элементов на веб-странице. Правило, состоит из двух частей: **селектора** и следующего за ним **блока объявлений**.

Синтаксис CSS



CSS правило

Первым всегда указывается *селектор*, он сообщает браузеру, к какому элементу веб-страницы будет применен стиль. Селекторами могут быть любые теги, формирующие тело сайта, а также идентификаторы, классы и атрибуты тегов.

Далее следует *блок объявлений*, который начинается открывающейся фигурной скобкой { и заканчивается закрывающейся фигурной скобкой }. Между фигурными скобками указываются команды CSS – объявления.

Каждое *объявление* состоит из двух частей: свойства и его значения. Объявление всегда должно заканчиваться точкой с запятой.

Свойство – это команда форматирования, которая задает конкретный стиль для элемента. Каждое свойство имеет предопределенный набор значений. После имени свойства указывается двоеточие, которое отделяет название свойства от допустимого значения.

Пример CSS правила:

```
h1 {
        color: red;
        font-size: 20px;
}
```

В данном примере селектором является элемент h1 – заголовок первого уровня. Заметьте, что тег h1 записывается в CSS без угловых скобок. К заголовку применяются два стиля:

- color: red – свойства color со значение red задает красный цвет шрифта у заголовка;
- font-size: 20px – свойство font-size со значением 20px задает размер шрифта у заголовка.

Выше рассмотрен многострочный вариант расположения команд CSS. Он считается более наглядным, безопасным и удобным. Поэтому именно многострочный вариант чаще рекомендуется для использования. Однако иногда можно применять и однострочный вариант расположения команд.

```
h1 {color: red;font-size: 20px;}
```

Для задания комментарий в CSS используется конструкция /* … */. Такая конструкция позволяет комментировать одну или несколько строк CSS.

```
/* здесь пишется комментарий */
/*
и здесь можно
тоже указать комментарий*/
```

Подключение CSS к HTML

Для того, чтобы использовать стили CSS в веб-документе, необходимо их сначала подключить. В зависимости от способа подключения выделяют внешние таблицы стилей, внутренние и встроенные стили.

Внешняя таблица стилей представляет собой созданный в текстовом редакторе файл с расширением CSS, в котором находится весь CSS-код веб-страницы. Внутри файла содержатся только стили без HTML-разметки.

Внешняя таблица стилей (CSS-файл) подключается к html-документу с помощью тега **<link>**, расположенного внутри раздела <head></head>.

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

Тег <link> имеет два обязательных атрибута: href и rel. В атрибуте href указывается путь к внешнему CSS-файлу. Атрибут rel = "stylesheet" показывает тип ссылки. Значение stylesheet указывает на то, что подключается файл стилей CSS.

У тега <link> имеется также атрибут type = "text/css", но по стандарту HTML5 он не является обязательным, поэтому его можно не указывать.

Внешнюю таблицу стилей можно подключить ко всем страницам сайта.

К веб-странице можно присоединить несколько таблиц стилей, добавляя последовательно несколько тегов <link>, каждый из которых будет указывать на свой файл с таблицей.

Использование внешней таблицы стилей является самым удобным способом подключения CSS. Можно выделить следующие преимущества такого способа:

- меньший размер html-страницы и более чистая структура кода;
- быстрая скорость загрузки html-страницы;
- для разных веб-страниц может быть использован один css файл;
- удобство в редактировании стилей.

Селекторы CSS

Селекторы представляют структуру веб-страницы. С их помощью создаются правила для форматирования элементов веб-страницы. Селекторами могут быть теги, их классы и идентификаторы, а также атрибуты.

Селектор тега

В качестве селектора может выступать любой тег HTML.

В рассмотренных выше примерах использовались именно селекторы тегов.

Синтаксис CSS

```
тег {
        свойство 1: значение 1;
        свойство 2: значение 2;
}
```

Селектор тега применяется, когда необходимо задать один стиль для всех элементов с одинаковым тегом. Например, если нужно задать отступ у первой строки для всех абзацев или цвет всех заголовков второго уровня.

Однако, часто необходимо изменить цвет не всех заголовков на веб-странице, а только одного или двух. CSS предоставляет такую возможность с помощью селекторов id (идентификатор) и class (класс).

Селектор id (идентификатор) предназначен для применения стиля к уникальным элементам на веб-странице. **Каждый идентификатор может встречаться на странице только один раз.**

Чтобы задать данный селектор, следует для тега, к которому нужно применить стиль, добавить атрибут id и в его значении которого указать уникальное имя. В CSS коде селектор id начинается с символа #, после которого идет имя идентификатора.

Синтаксис CSS

```
#имя идентификатора {
        свойство 1: значение 1;
        свойство 2: значение 2;
}
```

Пример использования селектора id

Файл index.html

```
<p id = "first">Первый абзац </p>
<p> Второй абзац </p>
<p> Третий абзац </p>
<p> Четвертый абзац </p>
<p> Пятый абзац </p>
```

Файл style.css

```
#first {
        color: red; /*цвет текста: красный*/
        font-weight: bold; /*толщина текста: полужирное начертание*/
}
```

В данном примере указанные стили будут применены только к первому абзацу, для которого задан id = "first".

Селектор class (класс) позволяет применить стиль к нескольким подобным элементам на веб-странице.

Для использования селектора class следует для тегов, к которым необходимо применить стиль, добавить атрибут class. В качестве значения указать определённое имя класса. В CSS коде селектор class начинается с точки, после которого идет имя класса.

Также можно использовать классы и с указанием тега, к которому они применяются.

Синтаксис CSS

```
.имя класса {
        свойство 1: значение 1;
        свойство 2: значение 2;
}
```

```
тег.имя класса {
        свойство 1: значение 1;
        свойство 2: значение 2;
}
```

Имя идентификатора и класса всегда задается на английском языке и начинаться с буквы. Оно может содержать цифры, символ дефиса (-) и нижнего подчеркивания (_).

Пример использования селектора class

Файл index.html

```
<p id = "first" class= "odd">Первый абзац </p>
<p> Второй абзац </p>
<p class= "odd"> Третий абзац </p>
<p> Четвертый абзац </p>
<p class= "odd"> Пятый абзац </p>
```

Файл style.css

```
#first {
	color: red; /*цвет текста: красный*/
	font-weight: bold; /*толщина текста: полужирное начертание*/
}
.odd{
	background-color: #eee; /*цвет заливки: серый*/
}
```

В данном примере все нечетные абзацы с атрибутом class= "odd" будут выделены серым цветом.

Для уточнения элемента, к которому применяется стиль, в CSS можно указывать тег, у которого задан класс.

```
p.odd{
	background-color: #eee;
}
```

Классы можно задавать не только для одинаковых тегов. В следующем примере атрибут class="is_border" указан у разных тегов, служащих для отображения текстовой информации: у заголовка и у абзаца.

Файл index.html

```
<h1 class= "is_border">Заголовок с рамкой </h1>
<p class= "is_border"> Абзац с рамкой </p>
```

Файл style.css

```
.is_border{
	border: 1px solid #000; /*рамка: толщина 1px сплошная черного цвета*/
}
```

Для одного элемента иногда необходимо применить несколько классов, при этом их имена в значении атрибута следует указывать через пробел. Например, в следующем примере для заголовка заданы два класса с именами heading и logo. У каждого класса в CSS описаны свои стили.

Файл index.html

```
<h1 class= "heading logo">Заголовок</h1>
```

Файл style.css

```
.heading{
	text-transform: uppercase; /*все символы заглавные*/
	width: 200px; /*ширина текстового блока: 200px*/
}
.logo{
	border: 1px solid #abc;
}
```

Универсальный селектор Часто возникает необходимость использования единого стиля одновременно для всех элементов веб-страницы. Для этого используется универсальный селектор. Стиль, определённый для универсального селектора, будет выполнен над всеми элементами.

Универсальный селектор записывается символом *.

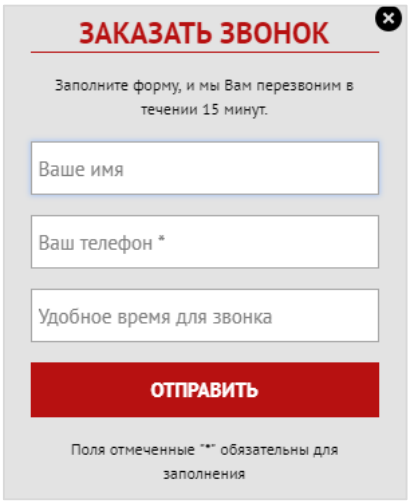
Синтаксис CSS

```
* {
        свойство 1: значение 1;
        свойство 2: значение 2;
}
```

5) Формы

Форма – один из самых необходимых элементов современных сайтов. Формы позволяют получать информацию от пользователя, отправлять данные на сервер или администратору на почту.

Примерами форм на сайте являются формы авторизации и регистрации, формы обратной связи, формы для заказа товара, формы подписки, формы голосования и т.д.



Для размещения формы на веб-странице используется контейнер <form> </form>. Тэг **<form>** может содержать атрибуты, описанные в таблице 1.

Таблица. Атрибуты тега <form>

№	*Атрибут*	*Описание, принимаемое значение*
1.	action	**Обязательный атрибут**. Определяет файл на сервере, который будет обрабатывать данную форму. Синтаксис: <form action = "action.php"> </form> Если значение атрибута не указано, то данные формы не будут отправлены и после перезагрузки страницы элементы формы примут значения по умолчанию.
2.	method	Определяет, каким образом данные из формы будут переданы обработчику. Допустимые значения: method="post" и method="get". Если значение атрибута не установлено, по умолчанию предполагается method="get".

3.	enctype	Определяет, каким образом данные из формы будут закодированы для передачи обработчику. enctype ="multipart/form-data" – данные не кодируются, это значение применяется при отправке файлов.
4.	name	Задает имя формы, которое будет использоваться для доступа к ее элементам.

Пример записи атрибутов тега <form> может быть следующим:

```
<form action = "index.php" method = "post" enctype ="multipart/form-data">

</form>
```

В данном случае обработка данных осуществляется в файле index.php, в котором используется серверный язык программирования php.

Элементы (поля) форм

Наиболее часто в формах используется тег **<input>**, позволяющий вводить и отправлять на сервер различные данные. Вид выводимого поля и тип передаваемых данных зависит от атрибута type.

1) Текстовые поля

а) **текст: <input type="text">** - однострочное поле ввода текста;
б) **пароль: <input type="password">** - поле ввода пароля, символы заменяются точками;
в) **email: <input type="email">** - поле ввода электронной почты, может показывать предупреждение, если введён некорректный email;
г) **число <input type="number">** - поле ввода числа.

Для поля ввода числа также можно добавить атрибуты, описанные в таблице.
Таблица. Атрибуты тега <input type="number">

№	*Атрибут*	*Описание, принимаемое значение*
1.	min	Минимальное значение числа
2.	max	Максимальное значение числа
3.	step	Шаг приращения числа. Может быть целым (2) или дробным (0.5)
4.	value	Начальное число, которые выводится в поле
5.	size	Ширина поля

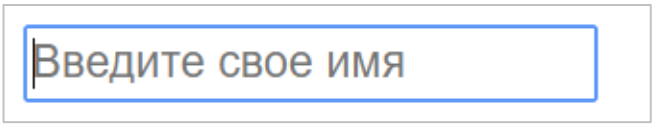
Для многих элементов формы, в том числе и для тега <input>, используются универсальные атрибуты, описанные в таблице.
Таблица. Универсальные атрибуты элементов форм

№	*Атрибут*	*Описание, принимаемое значение*
1.	**name**	**Имя поля, по которому обработчик формы идентифицирует элемент.**
2.	size	Ширина поля в символах.
3.	**value**	– **Начальный текст, отображаемый в поле.** – **Значение, отправляемое на сервер.**
4.	required	Обязательное поле. Выводит сообщение о том, что пользователь должен ввести данные или выбрать значение. Атрибут используется без указания значения.
5.	disabled	Поле, недоступное для ввода. Атрибут используется без указания значения.

6.	autofocus	Поле автоматически получает фокус после загрузки страницы. Атрибут используется без указания значения.

Текстовые поля могут отображать **подсказку**, которая исчезнет, как только будет введён некоторый текст. Для этого используется атрибут placeholder.

```
<form action = " ">
        <input type="text" placeholder="Введите свое имя">
</form>
```



Текстовое поле с атрибутом placeholder

2) Переключатели и флажки

Флажок («чекбокс») используют, когда необходимо выбрать любое количество вариантов из предложенного списка.

Для создания флажков используется тег: <input type="checkbox">.

По небольшому флажку часто бывает сложно щелкнуть мышкой, поэтому рекомендуется помещать флажок и его описание внутрь метки <label>.

```
<label>
        <input type="checkbox"> Я согласен с условиями
</label>
```

В данном примере можно щёлкнуть по тексту «Я согласен с условиями», чтобы переключить флажок.

По умолчанию флажок выключен. Чтобы пометить его по умолчанию включенным, используется атрибут checked.

```
<label>
        <input type="checkbox" checked> Я согласен с условиями
</label>
```

Переключатели используют, когда необходимо выбрать один единственный вариант из нескольких предложенных. Переключатели являются **взаимоисключающими**, т.е. выбор одного из вариантов исключает выбор другого.

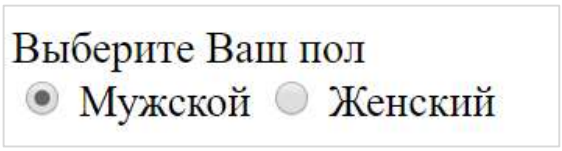
Переключатели создаются с помощью тега: **<input type="radio">**.

Чтобы переключатели относились к одной группе, необходимо чтобы у них был одинаковый атрибут name, иначе их можно будет нажать одновременно.

По умолчанию пункты переключателя выключены. Чтобы пометить какой-либо пункт включенным, применяется атрибут checked.

Для переключателей и флажков используется атрибут value, который показывает, какое значение будет отправлено на сервер (какой пункт был выбран пользователем).

```
<label>Выберите Ваш пол</label>
<label>
        <input type="radio" name="gender" value="men" checked> Мужской
</label>
<label>
        <input type="radio" name="gender" value="women"> Женский
</label>
```



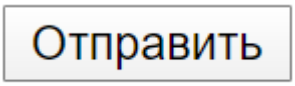
В данном примере оба переключателя используют одинаковое значение атрибута name (значение gender), выбор одного из вариантов исключит выбор другого варианта. При выборе пункта *Мужской* на сервер будет отправлено значение men, при выборе пункта *Женский* – значение women.

3) Кнопки

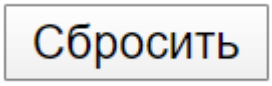
Кнопку на веб-странице можно создать двумя способами – с помощью тега <input> и парного тега <button></button>.

При добавлении кнопки через <input> возможно создание следующих типов кнопок:

а) кнопка отправки данных на сервер: **<input type="submit">**;



б) кнопка сброса введенных данных: **<input type="reset">**;



в) обычная кнопка, которая не имеет поведения по умолчанию: **<input type="button">**.

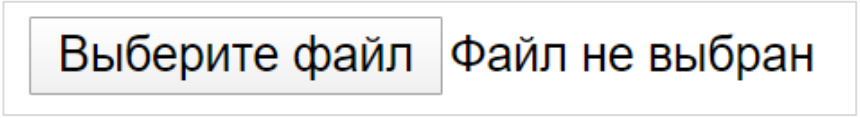
Такую кнопку можно использовать для запуска сценария JavaScript.

Второй способ создания кнопки основан на использовании контейнера <button></button>. Для корректного отображения элемента <button> разными браузерами у него нужно указывать атрибут type, например:

```
<button type="submit"> </button>
```

4) Поле загрузки файла

Чтобы загружать на сервер один или несколько файлов используется специальное поле: **<input type="file">.**



Вид поля для загрузки файла в Chrome

При нажатии на кнопку открывается окно для выбора файла, где можно указать, какой файл пользователь желает использовать.

Чтобы поле заработало и браузер смог предать выбранный файл на сервер, необходимо для тега формы:

1. задать метод отправки данных POST (method="post");
2. установить у атрибута enctype значение multipart/form-data.

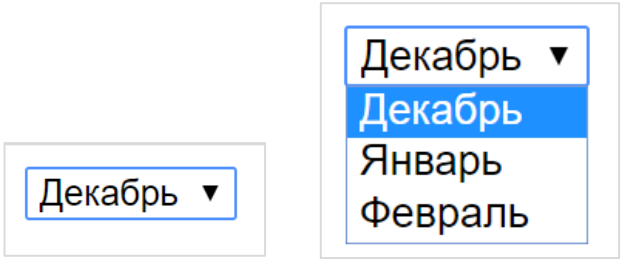
Далее рассмотрим другие теги для создания полей формы.

5) Выпадающий список

Выпадающий список используется, если количество вариантов для выбора достаточно много. Выпадающий список работает подобно переключателям, отличаясь от них только внешним видом.

Для создания выпадающего списка используется контейнер **<selest></select>**. Элементы выпадающего списка указываются в контейнерах <option></option>.

```
<select>
        <option> Декабрь </option>
        <option> Январь </option>
        <option> Февраль </option>
</select>
```



Выпадающий список

По умолчанию в поле списка отображается его первый элемент. Атрибуты тегов <select> и <option> приведены в таблицах.

Таблица. Атрибуты тега <select>

№	*Атрибут*	*Описание, принимаемое значение*
1.	multiple	Дает возможность выбора нескольких пунктов. При выборе нужно нажать и удерживать нажатой клавишу Ctrl. Атрибут используется без указания значения.
2.	size	Задает количество отображаемых строк списка. Значение атрибута задается целым положительным числом.

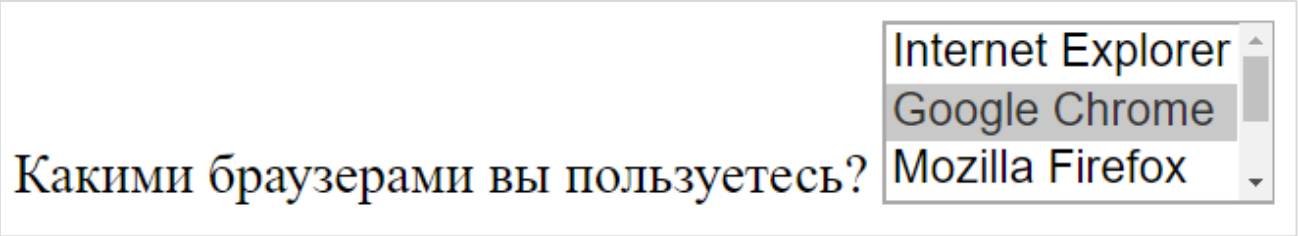
Таблица. Атрибуты тега <option>

№	*Атрибут*	*Описание, принимаемое значение*
1.	selected	Отображает выбранный элемент списка по умолчанию при загрузке веб-страницы браузером. Атрибут используется без указания значения.
2.	value	Указывает значение, отправляемое на сервер

Пример использования тега <select>:

```
<label>Какими браузерами вы пользуетесь? </label>
<select multiple size = "3">
```

```
        <option value ="explorer"> Internet Explorer </option>
        <option value ="chrome" selected> Google Chrome </option>
        <option value ="firefox"> Mozilla Firefox </option>
        <option value ="opera"> Opera </option>
        <option value ="safari"> Safari </option>
</select>
```



Выпадающий список множественного выбора

6) Текстовая область

Если необходимо создать многострочное поле для ввода большого текста, то вместо тега <input> используется элемент **<textarea></textarea>.**

Таблица. Атрибуты тега <textarea>

№	*Атрибут*	*Описание, принимаемое значение*
1.	cols	Ширина текстового поля в символах.
2.	rows	Высота текстового поля в строках текста.
3.	wrap	Определяет параметры переноса строк в тексте, при отправке данных на сервер: – wrap = "hard" – сохраняет перенос, обязательным условием использования значения hard является установленный атрибут cols; – wrap = "soft" – не сохраняет перенос (значение по умолчанию).
4.	maxlength	Максимальное число символов, которое можно ввести в текстовое поле.
5.	readonly	Указывает, что текстовое поле будет доступно только для чтения, т.е. текст невозможно будет изменить, но будет возможность его скопировать. Атрибут используется без указания значения.
6.	disabled	Отключает возможность редактирования и копирования содержимого поля. Атрибут используется без указания значения.

Лекция № 5. Блочная верстка сайтов

План

1. Понятие и виды верстки сайтов
2. Блочная верстка сайта

1) Понятие и виды верстки сайтов

Верстка сайта — это структурированное сочетание изображений, заголовков, подзаголовков, таблиц, текста и других элементов на странице с помощью языка разметки HTML и языка описания внешнего вида страницы CSS.

Виды верстки:

– Фиксированная верстка или статическая. Вне зависимости от размеров окна браузера или устройства ширина страниц будет постоянной.

– Резиновая верстка. Блоки меняют свою ширину в зависимости от размера окна браузера.

– Адаптивная верстка. Страницы хорошо адаптируются под любое разрешение экрана пользователя.

– Блочная верстка или div-верстка. Сетка страниц конструируется из множества блоков , которые вложены друг в друга.

– Гибкая верстка или flex верстка. Вначале применяется блочная верстка, а потом нужные блоки превращают во флекс-контейнеры (флексбоксы).

– Семантическая верстка. Она явилась логичным продолжением блочной верстки и стала доступна в HTML5. Новые теги делают станицу более структурированной.

– Валидная верстка или верстка без ошибок. Это верстка, выполненная в соответствии со стандартами W3C. Проверить свою HTML-страницу на корректность вы можете с помощью специального валидатора W3C.

– Кроссбраузерная верстка. Страницы выглядят одинаково в разных браузерах. Первое, с чего обычно начинают — подключают к странице специальный CSS файл — сброс стилей.

2) Блочная верстка сайта

Суть блочной верстки сайта состоит в том, что каждая часть страницы помещается в свой блок ** **: верх сайта — в первый, меню — во второй, контент — в третий и т. д. Каждый блок наполняется содержимым средствами HTML, а также позиционируется и оформляется с помощью CSS-разметки.

Зачастую основными элементами страницы являются: содержащий блок (wrapper, container), логотип, навигация, контент, футер (нижний колонтитул), свободное пространство.

Содержащий блок (контейнер) Роль контейнера на странице может выполнять непосредственно элемент body или же div. Ширина содержащего блока может быть резиновой (fluid), а может быть фиксированной (fixed).

Для блочной верстки сайта на основе float применяются следующие CSS-свойства:

Обтекание элементов – свойство float

- float: left - смещает элемент влево
- float: right – смещает элемент вправо
- clear: both - отмена обтекания
- overflow: hidden - спрятать перекрытие

Свойство display

- none – выключает блок из документа
- block – блочное отображение элемента
- inline – строчное отображение элемента
- inline-block – строчно-блочное отображение
- flex – гибкие блоки
- grid – сетка

Позиционирование блоков - свойство position

- **Статическое (static)** — используется по умолчанию. В этом случае элемент располагается в соответствии с позицией в HTML-коде.
- **Относительное (relative)**. В данном случае расположение объекта рассчитывается так же, как и в случае со статическим позиционированием, но данное свойство позволяет изменять позицию для дочерних элементов.
- **Абсолютное (absolute)**. Положение его рассчитывается относительно элемента с относительным позиционированием
- Для изменения расположения абсолютного и относительного блоков используются свойства **top, left, right и bottom**.

Пример блочной верстки на float

index.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Блочная вёрстка</title>
        <link rel="stylesheet" href="style.css">
        <meta charset="utf-8">
</head>
<body>
<main>
        <header>
           <h2>header</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 </h2>
        </footer>
</main>
</body>
</html>
```

style.css

```
*{
        margin: 0;
        padding: 0:

}
body {
    background: #FFF;
    color: #333;
    font-family: Arial, sans-serif;
    font-size: 14px;
}
main {
    background: #FFD700;
    margin: 0 auto;
    text-align: center;
    width: 80%;
   min-height: 100%;
}
header {
    background: #F5DEB3;
    width: 100%;
    height: 70px;
}
nav {
    background: #FE9798;
    width: 100%;
    height: 45px;
}
aside {
    background: #40E0D0;
    float: left;
    width: 20%;
```

```
    height: 500px;
}
section {
    background: #DCDCDC;
    float: right;
    width: 80%;
    height: 500px;
}
#clear {
    clear: both;
}

footer {
   background: #00BFFF;
         width: 100%;
         height: 40px;
}
```

Лекция № 6. Верстка на основе flex-контейнеров и grid-контейнеров

План

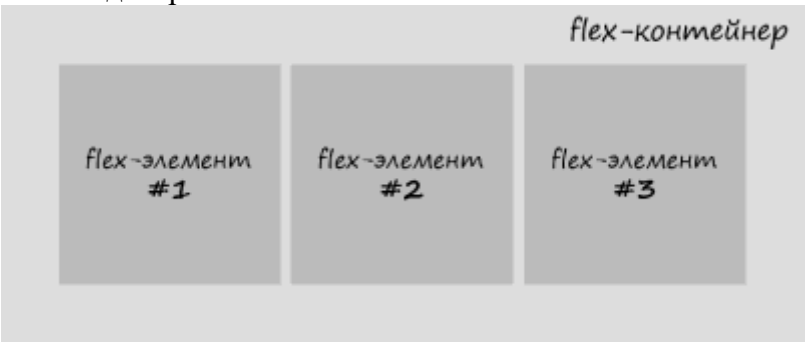
1. CSS Flexbox
2. Верстка на Grid в CSS

1) CSS Flexbox

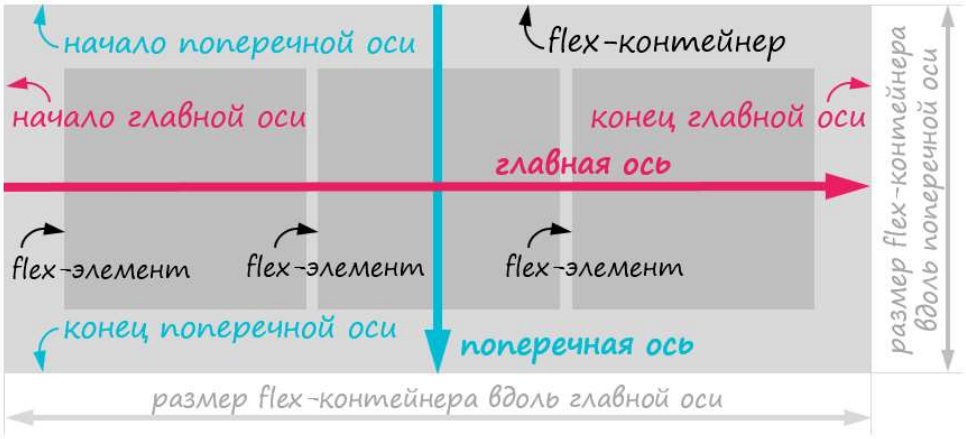
CSS Flexbox предназначен для создания гибких макетов. С помощью этой технологии можно очень просто и гибко расставить элементы в контейнере, распределить доступное пространство между ними, и выровнять их тем или иным способом даже если они не имеют конкретных размеров. CSS Flexbox позволяет создать адаптивный дизайн намного проще, чем с использованием Float и позиционирования.

Создание CSS разметки с помощью Flexbox начинается с установки необходимому HTML элементу CSS-свойства **display** со значением **flex** или **flex-inline**.

После этого данный элемент становится **flex-контейнером**, а все его **непосредственные** дочерние элементы – **flex-элементами**.



На рисунке представлена схема устройства flex-контейнера:



Свойства родительских элементов (flex-container)

1) **display -** создаёт flex контейнер, инлайновый или блочный, в зависимости от заданного значения.

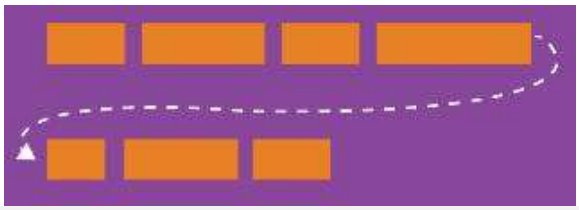
```
.container {
  display: flex; /* или inline-flex */
}
```

2) flex-direction - устанавливает направление главной оси и определяет направление flex элементов размещенных в flex контейнере

Значения:

`row` (стандартное положение) — слева направо.
`row-reverse` — элементы располагаются справа налево.
`column` — тоже самое, что и `row`, только сверху вниз.
`column-reverse` — тоже самое, что и `row-reverse`, но снизу вверх.

3) flex-wrap - дает возможность дочерним элементам при необходимости переходить на другую строку



Значения:

`nowrap` — это значение по-дефолту, при котором все flex элементы будут выстраиваться в одну линию.
`wrap` — flex элементы будут переноситься на несколько строк, от верха к низу.
`wrap-reverse` — flex элементы будут переноситься на несколько строк снизу вверх.

4) justify-content – выравнивание дочерних элементов вдоль главной оси

Значения:

`flex-start` — дефолтное состояние, при котором элементы расставляются от начала строки.
`flex-end` — состояние, в котором элементы размещены с конца строки.
`center` — элементы центрированы вдоль строки.

`space-between` — элементы равномерно распределены по строке, первый элемент находится вначале строки, последний в конце.
`space-around` — элементы равномерно распределены по строке с равным местом вокруг них.
`space-evenly` — элементы распределены таким образом, что свободное пространство между любыми двумя элементами равномерно, как и место до границы края контейнера.

5) align-items – выравнивание дочерних элементов вдоль поперечной оси
Значения:
`flex-start` — всё размещается с начала поперечной оси
`flex-end` — все элементы размещаются с конца поперечной оси
`center` — элементы центрируются по поперечной оси
`baseline` — элементы выравниваются по базовой линии
`stretch` — это дефолтное состояние, при котором элементы заполняют контейнер, с учетом `min-width` и `max-width`.

6) align-content - выравнивает и распределяет строки контейнера, когда есть свободное пространство в поперечной оси. Это свойство не приносит эффекта, когда есть только одна строка flex элементов

```
.container {
 align-content: flex-start | flex-end | center | space-between | space-around | stretch;
}
```

Свойства дочерних элементов

– **Order** - определяет порядок, в котором будут располагаться дочерние элементы. Задается целым числом и по умолчанию равно 0.

– **Flex-grow** - указывает, насколько отдельный элемент будет больше соседних элементов (по умолчанию 0).

– **Flex-shrink** - определяет способность flex-элемента сокращаться в случае недостатка свободного места. По умолчанию равен 1.

– **Flex-basis** - базовый размер отдельного элемента, заменяет свойство width

– **Flex** является сокращенным свойством для задания свойств flex-grow, flex-shrink и flex-basis

flex: 0 1 auto; /*по умолчанию*/

– **Align-self** - выравнивание отдельно взятого flex-блока по поперечной оси.

2) Верстка на Grid в CSS

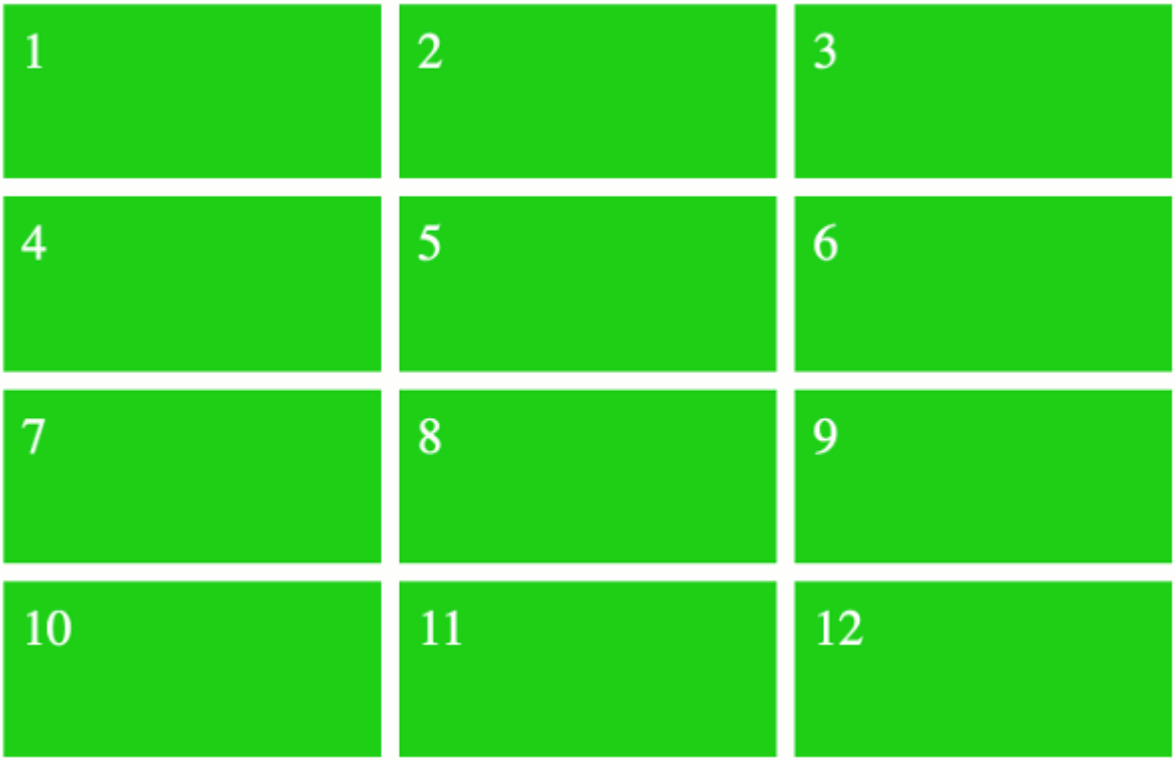
CSS Grid это новая модель для создания шаблонов, оптимизированная для создания двумерных макетов. Она идеально подходит для: *шаблонов сайтов, форм, галерей и всего, что требует точного и отзывчивого позиционирования.*

Хотя flexbox уже позволил разработчикам начать уходить от флоат элементов, но он работает только в одном измерении. Grid CSS же это делает в двух, таким образом лучше подходя для создания сложных, комплексных шаблонов.

Grid шаблон работает по системе сеток. Grid это набор пересекающихся горизонтальных и вертикальных линий, которые создают размерность и позиционируют систему координат для контента в самом grid-контейнере.

Чтобы создать Grid разметку, нужно выставить элементу display: grid. Этот шаг автоматически сделает всех прямых потомков этого элемента — grid элементами. После

этого вы можете смело использовать разнообразные grid свойства для выравнивания размеров и позиционирования элементов должным образом. Обычно первым шагом является определение того, сколько колонок и рядов есть в гриде.



Это пример грида с четырьмя рядами и тремя колонками. Он состоит из *12 grid элементов*. Каждый из этих элементов отмечен зеленым и между ними есть небольшое расстояние.

Контейнер. Во-первых, нужно создать контейнер, внутри которого будут находится элементы сетки . Сетка создается на контейнере и все элементы внутри него выстраиваются по заданной сетке. Просто добавляем к CSS контейнера свойство display: grid; После этого можно указывать прочие свойства Grid. Но пока не заданы другие свойства сетки, элементы выстроены вертикально друг за другом так, как они идут в html разметке.

Колонки. За колонки отвечает свойство grid-template-columns. В значении свойства указывается размер каждой колонки через пробел. Например, три колонки по 100 px каждая: grid-template-columns: 100px 100px 100px; Элементы распределятся в три колонки.

Ряды. За ряды отвечает свойство grid-template-rows. Задать высоту рядам по 100px: grid-template-rows: 100px 100px 100px;

Отступы. Пространство между колонками добавляется при помощи grid-column-gap и пространство между рядами при помощи grid-row-gap. Эти свойства можно объединить в grid-gap: 20px 30px, где первое значение — отступ между рядами, а второе — между колонками. Если нужен одинаковый отступ между колонками и рядами, задаем его одним значением grid-gap: 20px;

Фракция — специальная единица измерения CSS Grid для создания адаптивного макета. Фракция — это часть целого.

Если определить три колонки, шириной 1 фракция каждая (1fr), колонки будут равномерно делить ширину экрана или свободного для них пространства и занимать по 1 части каждая.

grid-template-columns: 1fr 1fr 1fr; вся доступная ширина разделится на 3 части и каждая колонка займет одну из этих частей.

grid-template-columns: 2fr 1fr 1fr; доступная ширина поделится на 4 части, первая колонка займет две части, а остальные колонки — по одной части.

Фракции можно комбинировать с точными единицами измерения.

grid-template-columns: 200px 1fr 1fr; первая колонка займет фиксированную ширину в 200px, а две другие будут делить оставшееся пространство между собой поровну. При изменении ширины экрана, первая колонка будет также занимать 200px, а ширина колонок, заданная во фракциях, будет пересчитываться.

Лекция № 7. Адаптивная верстка

План

1. Понятие адаптивной верстки сайта
2. Адаптация верстки
3. Медиа-запросы

1) Понятие адаптивной верстки сайта

Адаптивная верстка сайта – это верстка, обеспечивающая правильное отображение сайта на различных устройствах, и динамически подстраивающаяся под заданные размеры окна браузера.

Основные принципы адаптивного дизайна:

- адаптивный шаблон сайта, способность шаблона подстраиваться под различные разрешения экранов устройств от монитора компьютера до смартфона;
- адаптация и перемещение блоков контента, способность блоков контента в зависимости от разрешения экрана устройства принимать необходимые размеры, а также способность передвигаться на другую позицию в макете;
- адаптация изображений, способность изображений менять размер в зависимости от разрешения экрана или загружать более адаптированное изображение с меньшим весом и размером;
- использование гибкой сетки, позволяет максимально быстро изменять конструкцию макета;
- скрытие менее важных блоков на небольших экранах;
- переработка элементов навигации – так как на мобильных устройствах элементы навигации становятся менее кликабельными, их перерабатывают, делая удобно-используемыми;
- упрощение ряда элементов на веб-странице для их использования на мобильных устройствах;
- адаптация видео-контента;
- использование медиа-запросов.

2) Адаптация верстки

Для того чтобы сообщить браузеру, как отобразить размеры страницы и изменить ее масштаб используется мета-тег **viewport.** Данный мета-тег прописывается в блоке <head> сайта.

<meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0" >

Свойство *width* определяет размер окна просмотра. Он может быть установлен на определенное количество пикселей, например, width=600 или на специальное значение device-width, которое означает ширину экрана в пикселях CSS в масштабе 100%.

Свойство *initial-scale* контролирует уровень масштабирования при первой загрузке страницы. Свойства *maximum-scale, minimum-scale и user-scalable* определяют, как пользователям разрешено увеличивать или уменьшать страницу.

Как сделать адаптивный дизайн сайта из фиксированного макета?

1. Необходимо добавить мета-тег ***viewport.***
2. Ширину основного контейнера необходимо задать с помощью свойства ***max-width.***
3. Необходимо перевести все статические единицы измерения в относительные единицы измерения: ***px необходимо перевести в %, а шрифты задать в em.***

Перевод ширины контейнера или элемента из px в % осуществляется по формуле:

Размер контейнера (px) / размер основного контейнера (родителя) в (px) * 100% = результат **(%)**.

Пример

Размер основного контейнера 960px, в нем имеется контейнер 720px. Для него получим размер в % следующим образом: 720/960*100=75%.

Перевод шрифта из px в em осуществляется по формуле:

Размер шрифта (px) /16px (стандартный размер) = размер шрифта (em)

Пример

Размер шрифта 32px, тогда 32/16=2em.

3) **Медиа-запросы**.

Медиа-запросы включают в себя медиа-тип (принтеры, смартфоны, экраны, телевизоры, проекторы и др.) и условия, которое может принимать в свою очередь истину или ложь (true, false). В зависимости от того верный ли медиа-тип и выполняется ли условие будут применяться различные стили css. Если условие верно, то будут применяться те стили, которые прописаны в этом медиа-запросе, если же будет ложным, то будут применяться обычные стили css.

Благодаря таким запросам и создаются различные отображения сайта для мобильных, планшетов и экранов мониторов. Медиа-запросы поддерживаются всеми современными браузерами.

Медиа-запрос записывается следующим образом:

```
@media screen and (max-width: 1000px) {
.class {
        свойство: значение;
        }
}
```

Здесь:

@media – медиа-запрос;

screen – медиа-тип (также называют тип носителя);

max-width: 1000px – условие, которое должно выполняться (в данном примере стили будут применяться, если ширина окна меньше ширины 1000px);

class – прописываются соответствующие селекторы (классы, id) в которых свойствам задаются новые значения.

Значения, которые используются в медиа-функциях называют также **breakpoints** (переломные или контрольные точки). В этих контрольных точках и меняется дизайн сайта.

Используются **320px** – мобильные; **480px** – мобильные; **768px** – планшеты; **1024px** – планшеты, нетбуки, **1280px** и более-запросы.

Медиа-запросы прописываются в конце файла стилей, после всех основных стилей css.

Лекция № 8. Библиотека Bootstrap. Анимация на сайте

План

1. Библиотека Bootstrap
2. Анимация CSS

1) Библиотека Bootstrap

Bootstrap – это HTML, CSS и JS фреймворк (библиотека) для быстрого создания web-страниц.

Официальная документация Bootstrap на английском языке находится на сайте https://getbootstrap.com/. Документация на русском языке http://bootstrap-4.ru/.

Сетка Bootstrap 4 состоит из следующих элементов:

- Обёрточные контейнеры (container и container-fluid);
- Ряды (row);
- Адаптивные блоки (col).

Обёрточный контейнер – это элемент сетки Bootstrap 4, с которого **начинается создание адаптивного макета** страницы или некоторого блока. Другие элементы сетки (ряды и адаптивные блоки) должны быть размещены внутри него. Используйте класс .container для фиксированной ширины или .container-fluid для 100%-ной ширины.

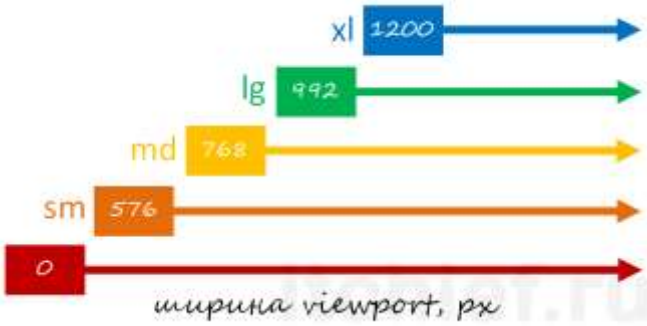
Ряд (row) – это специальный элемент сетки, который используется при создании макета в следующих случаях:

- между контейнером и адаптивными блоками, которые надо в него поместить;
- между одним и другими адаптивными блоками, которые надо поместить в первый адаптивный блок.

Если вы будете использовать адаптивные блоки вне ряда, они работать не будут. В **Bootstrap 4 адаптивные блоки должны обязательно** находиться в блоке с классом row.

Адаптивные блоки – это основные строительные элементы адаптивного макета, именно от них будет зависеть, как будет выглядеть макет веб-страницы на разных контрольных точках.

Т.е. содержимое должно быть расположено в адаптивных блоках: col, col-sm, col-md, col-lg, col-xl.



Фреймворк Bootstrap 4 имеет 5 контрольных точек или названий устройств (без обозначения, sm, md, lg, xl), и, следовательно, позволяет создать макет, который на каждой из них может выглядеть по-разному.

2) Анимация CSS

Все современные браузеры, кроме IE9- поддерживают CSS transitions и CSS animations, которые позволяют реализовать анимацию средствами CSS, без привлечения JavaScript.

1) CSS3-переходы – свойство **transition**

CSS3-переходы позволяют анимировать исходное значение CSS-свойства на новое значение с течением времени, управляя скоростью смены значений свойств.

Смена свойств происходит при наступлении определенного события, которое описывается соответствующим псевдоклассом. Чаще всего используется псевдокласс :hover.

Для задания всех свойств перехода обычно используют краткую запись свойства **transition**. Это свойство включает следующие значения:

transition: transition-property transition-duration transition-timing-function transition-delay;

а) Название свойства transition-property – название CSS-свойств, к которым будет применен эффект перехода

б) Продолжительность перехода transition-duration – промежуток времени, в течение которого должен осуществляться переход

в) Функция перехода transition-timing-function – свойство задаёт временную функцию, которая описывает скорость перехода объекта от одного значения к другому.

г) Задержка перехода transition-delay – необязательное свойство, позволяет сделать так, чтобы изменение свойства происходило не моментально, а с некоторой задержкой.

Пример плавного изменения цвета текста и заливки блока при наведении на него мыши:

```
div{
    color: #0d6efd;
    transition: all 1s ease-in .5s;
}
div:hover{
    background: #0d6efd;
    color: #fff;
    transition: all 1s ease-in .5s;
}
```

2) **CSS3-трансформации – свойство transform**

CSS3-трансформации позволяют сдвигать, поворачивать и масштабировать элементы. Свойство transform задаёт вид преобразования элемента. Оно может принимать следующие значения:

а) translate(x,y) Сдвигает элемент на новое место, перемещая относительно обычного положения вправо и вниз, используя координаты X и Y, не затрагивая при этом соседние элементы.

б) scale(x,y) Масштабирует элементы, делая их больше или меньше. Значения от 0 до 1 уменьшают элемент.

в) rotate(угол) Поворачивает элементы на заданное количество градусов, отрицательные значения от -1deg до -360deg поворачивают элемент против часовой стрелки, положительные — по часовой стрелке.

3) CSS3-анимация.

CSS3-анимация придаёт сайтам динамичность. Она оживляет веб-страницы, улучшая взаимодействие с пользователем. В отличие от CSS3-переходов, создание анимации базируется на ключевых кадрах, которые позволяют автоматически воспроизводить и повторять эффекты на протяжении заданного времени, а также останавливать анимацию внутри цикла.

Ключевые кадры используются для указания значений свойств анимации в различных точках анимации.

Ключевые кадры указываются с помощью правила @keyframes, определяемого следующим образом:

```
@keyframes имя анимации { список правил }
```

После объявления правила @keyframes, можно на него ссылаться в свойстве animation у элемента, который мы хотим анимировать.

Пример создания анимированной тени у текста:

```
@keyframes shadow {
from {text-shadow: 0 0 3px black;}
50% {text-shadow: 0 0 30px black;}
to {text-shadow: 0 0 3px black;}
}
```

```
h1 {
font-size: 3.5em;
color: darkmagenta;
animation: shadow 2s infinite ease-in-out;
}
```

animation-name

После того, как объявлены ключевые кадры для анимации, они должны быть назначены для элемента. Для этого используется свойство animation-name с именем

анимации из правила @keyframes, как значение свойства. Декларация animation-name применяется к элементу, для которого должна быть задана анимация.

Использования одного свойства animation-name при этом недостаточно. Кроме того, необходимо объявить свойство animation-duration и значение, чтобы браузер знал, как долго должна длиться анимация до завершения.

animation-duration, функция времени и animation-delay

После того, как вы объявили свойство animation-name для элемента, анимация ведёт себя подобно переходам. Она включают в себя длительность, функцию времени и задержку при желании. Сперва анимации требуется длительность, объявленная с помощью свойства animation-duration. Как и в случае с переходами, длительность может быть задана в секундах или миллисекундах.

Лекция № 9. Основы языка JavaScript

План

1. Общие сведения о языке Java Script. Способы подключения JS
2. Переменные и константы
3. Вывод данных
4. Типы данных
5. Операторы JS
6. Реализация линейного, разветвляющегося и циклического алгоритмов в JS

1) Общие сведения о языке Java Script. Способы подключения JS

Язык программирования JavaScript был разработан фирмой Netscape в сотрудничестве с SunMicrosystems и представлен в 1995 году. JavaScript предназначен для создания интерактивных html-документов.

JavaScript (JS) - это объектно-ориентированный язык программирования, который используется в веб-разработке. JS является языком сценариев, которые работают на стороне клиента браузера. Его интерпретатор установлен в браузере и исполнение скрипта (программы на JS) выполняется без отправки запроса на сервер, по мере загрузки веб-документа в браузере.

Основные области использования JavaScript:

- Создание динамических страниц, т.е. страниц, содержимое которых может меняться после загрузки.
- Проверка правильности заполнения пользовательских форм.
- Обработка событий на странице
- Передача отдельных данных на сервер.

Способы подключения:

1) Скрипт добавляется в раздел <body> или раздел <head> html-документа
```
<script>
  // операторы
</script>
```
2) Подключение внешних файлов с расширением .js
```
<script src='scripts.js'></script>
```

В атрибуте src тега <script> указывается путь к файлу. Путь может быть относительным или же абсолютным. Например, мы можем написать набор своих скриптов или подключать готовые внешние библиотеки.

```
<script src='some_scripts.js'></script>
<script src='http://apis.com/2.4/some_scripts.js'></script>
```

В js документе слова script уже не пишутся, записываются только команды языка.

Рекомендуется подключать js перед закрывающимся тегом body. Этот фактор учитывает google при ранжировании страниц и позволяет загрузить верстку сайта до js.

```
<script>
        alert("Hello World);
</script>
```

2) **Переменные и константы**

Инструкцией называют строку кода, которая эквивалентна некоторому действию.

```
a = 1;
alert(a);
```

Каждая инструкция в JS должна быть отделена точкой с запятой ;

Определение переменных. Переменная - это символьное представление для некоторого значения, которое сохраняется в оперативной памяти.

Переменные в JS не являются типизированными. Это означает, что при создании переменной не указывается, какого типа информация будет в ней находиться. Для того, чтобы использовать переменную, ее необходимо объявить или инициализировать. Объявление переменной реализуется при помощи ключевого слова var. Инициализация переменной - это присвоение переменной некоторого начального значения. Инициализация также может иметь место при объявлении переменных.

```
var a, b;
var x = 12;
t = 10;
```

В JS, как и в других языках программирования, названия переменных должны соответствовать некоторым правилам.

Название переменной в JS может содержать символы английского алфавита, цифры, подчеркивания и символ $, однако не может начинаться с цифры. Также, названия переменных регистро-зависимы.

Примеры правильных названий переменных:

```
a = 1;
hello = 123;
```

```
get_element = 121;
```

Примеры неправильных названий переменных:

```
12a = 1;
%hello = 123;
@get_element = 121;
```

Определение констант. Константы - это символьные названия некоторых значений, которые не будут изменены в программе. Правила для составления названий констант такие же, как и для переменных. Константы объявляются при помощи ключевого слова const:

```
const pi = 3.1415;
```

Комментарии в JS:

```
// Однострочные комментарии.
/*
И многострочные.
Они могут отключить сразу несколько строк кода.
*/
```

3) Вывод данных

1) alert('Текст сообщения') – выводит диалоговое окно с сообщением
2) confirm() - выводит сообщение с подтверждением
3) console.log()
4) document.write()

4) Типы данных

Числовой тип

Для представления целых чисел и чисел с плавающей точкой используется тип данных Number. В случае чисел с плавающей точкой дробная часть отделяется точкой. Также, допустимо использовать запись числа в т.н. экспоненциальной форме.

```
<script>
var x = 10;
var y = 1.567;
z = x + y; // 11.567

var a = 1.2E2; // x = 1.2 * 10 * 10 = 120
var b = 2E-5; // x = 2 * 10^(-5) = 0.00005;
</script>
```

Может содержать значения NaN (не число), infinity и -infinity. Для проверки на эти значения используются функции isNaN() и isFinite(). Приведение к числовому типу данных **Number() (или поставить+)**.

Значение Infinity будет возникать в результате арифметической операции, которая приводит к результату вне допустимого диапазона значений или в результате деления на ноль:

```
var x = 243 / 0;
```

NaN (Not a number) будет возникать в результате некоторой ошибки вычислений. Например, при попытке использовать не числовой тип данных в арифметической операции (если привести тип переменных не удается). При попытке использовать значение NaN в других выражениях, результат также будет NaN

```
var x = 2 / 'two';
```

Строка в JS - это последовательность символов, которая заключена в одинарные или двойные кавычки. Тип данных, который соответствует строкам называется String. Рассмотрим несколько простых примеров объявления строковых переменных:

```
var s = "Hello, world";
var a = 'Single quotes delimiters example';
```

При использовании строк важно следить, чтобы открывающая и закрывающая кавычки были одного типа, иначе это будет считаться ошибкой.
Для создания строки, которая содержит сам символ кавычки, необходимо его экранировать, т.е. писать символ \ перед символом кавычки. Экранирование необязательно, если тип кавычки, которую необходимо включить в строку не совпадает с открывающими/закрывающими кавычками. Например,

```
var s = "Let's rock";
var a = "What does word \"alias\" mean?";
```

Доступ к конкретному символу строки можно получить, указав номер символа в квадратных скобках при переменной, которая содержит строку. Нумерация символов начинается с нуля.

```
var s = "Hello, world";
var c = s[0]; // c is 'H' now
c = s[3]; // c is 'l' now
c = s[200]; // c is undefined
```

Логический тип данных

Рассмотрим программу.
<**script**>
Var *a* = 2 * 2 == 4;
alert (***a***);
</ **script**>

Это программа при запуске выведет true, что в переводе с английского означает «истина». Здесь мы с вами знакомимся с еще одним типом данных - логическим. В переменную **a** поместился результат операции **отношения** равенство (==). Сначала посчиталась левая часть, операция 2*2, и сравнилась справой частью 4. Так как результаты равны, результат операции отношения «истина». Результат операции присвоился в переменную a.

Логический тип данных может принимать всего два значения: истина и ложь. В JavaScript используется специальные значения для их обозначения: true и false.

5) Операторы JS

Оператор присваивания

Наберите и запустите следующую программу

<**script**>

Var *a* = 2 ;
Var *b* = 2 ;
Var *c* = ***a*** + ***b***;
alert (***c***);

</ **script**>

Программа вывела на экран сумму двух чисел. Для хранения чисел используются переменные. Переменные нужны для хранения изменяющихся данных.

Арифметические операторы. Для арифметических вычислений используют следующие бинарные операторы:

+ сложение
- вычитание
* умножение
/ деление
% взятие остатка от деления

Приоритет этих операций совпадает с оригинальным приоритетом в математике. К унарным операторам относятся следующие:

++ инкремент - увеличить значение переменной на 1
-- декремент - уменьшить значение переменной на 1
- изменение знака на противоположный

Примеры использования

```
a = 1;
x = 2.75;
a++;  / 2
b = -a;      // -2
y = 2 + 5; // 7
z = y % 2; // 1
x = x + 10; // 12.75
x--; // 11.75
```

Операторы сравнения. Для сравнения значений переменных используются следующие операторы сравнения:

- == равно
- != не равно
- === равно и тип данных операндов совпадает
- !== не равно или не совпадает тип данных операндов
- > строго больше
- >= больше или равно
- < строго меньше
- <= меньше или равно

Логические операторы. К логическим операторам относят:

- || - логическое ИЛИ
- && - логическое И

! – отрицание

Конкатенация строк. Конкатенация (склеивание) - это строковая операция, которая позволяет соединять строки в одну. Эту операцию можно применять как к непосредственно строкам, так и к переменным, которые содержат строки. В JS конкатенация обозначена символом +

Рассмотрим следующий пример:

```
<script>
    Var a;
    a = prompt ();
    alert ( a + " is a good number!" );
</ script>
```

Здесь демонстрируется операция склеивания строк. Для строк знак плюс означает, что должна получится новая строка, состоящая из нескольких строк, стоящих слева и справа от знака +.

```
<script>
var s = "Hello, " + "world!";
var s1 = ' Lets code some JS!';
var str = s + s1; // "Hello, world! Lets code some JS!"
</script>
```

Вывод значений на экран. Для отображения значений на экране можно воспользоваться функцией alert(), которая выводит модальное окно с указанным значением, либо воспользоваться console.log(), которая выводит значения в консоль браузера.

```
document.write( "<strong>Hello!</strong><br>" );
document.write ( "I am <em>HTML!</em>" );
```

Ввод данных

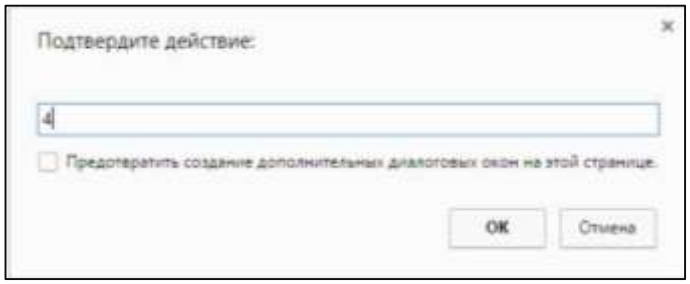
Ввод данных в программу может производиться разными способами. Это может быть клавиатура, мышь, касание экрана, считывание данных из файла или из базы данных. Но в каждом языке есть команда, с которой начинают изучать ввод данных.

В JavaScript это команда **prompt – команда ввода данных**.

Наберите следующую программу и запустите её на выполнение:

```
<script>
    Var a;
    a = prompt ();
    alert ( a);
</ script>
```

Эта программа выводит окно со строкой, в которую пользователь может вводить данные. Команда prompt возвращает строковое значение.



Введенные данные сохраняются в переменной, которую можно вывести на экран.

Преобразование из строки в число

При необходимости, JavaScript переводит число в строку. Что делать, если нужно сделать наоборот? Рассмотрим пример:

```
<script>
Var a = prompt ( "a:" );
Var b = prompt ( "b:" );
Var c = a + b;
alert ( c);
</ script>
```

Запустите программу и введите числа 10 и 20. Вместо того, чтобы сложить два числа, программа склеит две строки. В этом нет ничего удивительного, если знать, что результатом выполнения команды prompt является строка.

Чтобы преобразовать строку в число можно поставить перед prompt символ +. Такой способ позволяет переводить как целые, так и дробные числа.

```
<script>
        Var a = + prompt ( "a:" );
        Var b = + prompt ( "b:" );
        Var c = a + b;
        alert( c);
</ script>
```

Другой способ преобразования из строки в число - использование функции **parseInt**

```
<script>
        Var a = parseInt ( prompt ( "a:" ));
        Var b = parseInt ( prompt ( "b:" ));
        Va rc = a + b;
        alert ( c);
</ script>
```

6) Реализация линейного, разветвляющегося и циклического алгоритмов в JS

Линейный алгоритм

Линейный алгоритм описывает последовательность команд, выполняющихся строго друг за другом.

```
<script>
        vara = + prompt ( "a:" );
        varb = + prompt ( "b:" );
        var c = a + b;
        alert( c);
</ script>
```

Разветвляющиеся алгоритмы

Решим задачу нахождения наибольшего из двух чисел. Вводим два числа. Сравниваем числа, если первое число больше второго, то запоминаем первое число, иначе, запоминаем второе число.

Ветвление в JavaScript реализовано условным оператором

```
if(условие)
{
        //что делать, если условие верно
} else
{
        //что делать, если условие неверно
}
```

Особенности:

- вторая часть (else) может отсутствовать (неполная форма условного оператора);
- если в блоке один оператор, можно скобки { и } не ставить.

Реализация алгоритма на JavaScript

```
<script>
        Var a = + prompt ( "a:" );
        Var b = + prompt ( "b:" );
        Var max;
        if( a >b) {
                max= a;
        } else {
                max= b;
        }
        alert ( max);
</ script>
```

Решим задачу нахождения наибольшего числа, используя неполную форму условного оператора

```
<script>
        Var a = + prompt ( "a:" );
        Var b = + prompt ( "b:" );
        varmax = a;
        if( b >a) max= b;
        alert ( max);
</ script>
```

Сложные условия

В языках программирования условия, которые состоят из нескольких отношений, объединенных логическими операциями, называются *сложными условиями*.

Рассмотрим логические операции в порядке приоритетов выполнения:

! - Отрицание - если отношение истинно, то отрицание делает его ложным и наоборот;

&&- И - сложное условие истинно, когда оба отношения истины;

|| - ИЛИ - условие истинно, когда истинно хотя бы одно из условий;

Тогда программа, использующая сложное условие, будет выглядеть так:

```
<script>
        Var a = + prompt ( "a:" );
        if( a >= 25 &&a <= 50 ) {
                alert ( "YES" );
```

```
        } else {
                alert ( "NO" );
        }
</ script >
```

Примеры

```
<script>
// some code here with x and y init
var message;
var result;

if (y != 0) {
        message = "We can divide x by y";
        result = x / y;
} else {
        message = "We cannot divide x by y";
}

// some code here with message and result
</script>
```

Сравнение строк

Часто нужно получить от пользователя ответ на вопрос. В примере показано, как можно сравнить строковые значения.

```
<script>
        Var answer = prompt ( "Вы уверены?" );
        if( answer == "Да" || answer == "да" || answer == "ДА" ) {
                alert( "Хорошо, сделайте это!" );
        } else{
                alert ( "Возможно позже..." );
        }
</ script >
```

Циклы

Цикл while

Решим задачу. Вывести на экран числа от 1 до 5. Вот ее решение:

```
<script>
        document .write ( "1<br>" );
        document .write ( "2<br>" );
        document .write ( "3<br>" );
        document .write ( "4<br>" );
        document .write ( "5<br>" );
</ script>
```

Вот вывод:

Мы добились результата, но как быть, если нам нужно вывести 100 чисел? Воспользуемся циклом.

```
<script>
Var i= 1 ;
while( i<= 5 ) {
document .write ( i+ "<br>" );
```

i = ***i*** + 1 ;
}
</ **script**>

Формат записи цикла while
while(условие)
{
//тело цикла
}
Особенности:

- Условие может быть сложным;
- Операторные скобки могут отсутствовать, если тело цикла состоит из одного оператора

Цикл for

Программистам весьма часто приходится решать задачи, в которых что-то подсчитывается, а для подсчета используется специальная переменная - счётчик.

Для таких задач удобно использовать специальный цикл – цикл со счётчиком
Рассмотрим программу:
<**script**>
for(**var** ***i***= 1 ; ***i***<= 5 ; ***i***++) {
document .write (***i***+ **"
"**);
}
</ **script** >
Здесь мы выводим числа от 1 до 5 с использованием цикла for. В этом цикле условие инициализации начальным значением и изменения счетчикаописывается непосредственно в заголовке цикла. Для решения многих задач такое описание может быть более удобным, поэтому этот цикл довольно часто заменяет цикл while.

Формат записи цикла for
for(переменная = начальное значение; условие; изменение переменной)
{
//тело цикла
}

Особенности:
● Условие может быть сложным;
● Операторные скобки могут отсутствовать, если тело цикла состоит из одного оператора.

Цикл dowhile

Предположим нам нужно попросить пользователя ввести число, но при этом ограничить ввод только положительными значениями. Программа с использованием цикла while будет выглядеть вот так:
Это можно решить с помощью цикла do while:
<**script**>
Var ***number***;
do{
number= + prompt (**"Введите пожалуйста число больше нуля:"**);
} **while** (***number***<= 0);
</ **script**>

Формат записи цикла dowhile
do{
//тело цикла
} **while**(условие);
Особенности:
● Условие может быть сложным;
● Операторные скобки{ и } обязательны

Лекция № 10. Объектно-ориентированное программирование в Java Script

План
1. Массивы JS
2. Объекты в JS
3. Функции в JS

1) Массивы JS

Синтаксис для создания нового массива – квадратные скобки со списком элементов внутри.

Пустой массив:

```
var arr = [];
```

Массив fruits с тремя элементами:

```
var fruits = ["Яблоко", "Апельсин", "Слива"];
```

Элементы нумеруются, начиная с нуля.

Чтобы получить нужный элемент из массива – указывается его номер в квадратных скобках:

```
var fruits = ["Яблоко", "Апельсин", "Слива"];

alert( fruits[0] ); // Яблоко
alert( fruits[1] ); // Апельсин
alert( fruits[2] ); // Слива
```

Элемент можно заменить:

```
fruits[2] = 'Груша'; // теперь ["Яблоко", "Апельсин", "Груша"]
```

Или добавить:

```
fruits[3] = 'Лимон'; // теперь ["Яблоко", "Апельсин", "Груша", "Лимон"]
```

Общее число элементов, хранимых в массиве, содержится в его свойстве length:

```
var fruits = ["Яблоко", "Апельсин", "Груша"];

alert(fruits.length ); // 3
```

Через alert можно вывести и массив целиком.

При этом его элементы будут перечислены через запятую:

```
var fruits = ["Яблоко", "Апельсин", "Груша"];

alert(fruits ); // Яблоко,Апельсин,Груша
```

В массиве может храниться любое число элементов любого типа.

В том числе, строки, числа, объекты, вот, например:

```
// микс значений
var arr = [ 55, 'Имя', { name: 'Петя' }, true ];

// получить объект из массива и тут же -- его свойство
alert( arr[2].name ); // Петя
```

Методы:

1) *arr*.push(*element1*, ..., *elementN*) – добавляет новый элемент в конец массива и возвращает новую длину массива.

```
var sports = ['футбол', 'бейсбол'];
var total = sports.push('американский футбол', 'плавание');

console.log(sports); // ['футбол', 'бейсбол', 'американский футбол', 'плавание']
console.log(total);  // 4
```

2) *arr*.unshift(*element1*, ..., *elementN*) – добавляет новый элемент в начало массива и возвращает новую длину массива.

3) *arr*.indexOf(*searchElement*) – поиск элемента в массиве, возвращает позицию элемента или -1, если элемент не найден

4) **includes()** определяет, содержит ли массив определённый элемент, возвращая в зависимости от этого true или false

5) Метод **pop()** удаляет **последний** элемент из массива и возвращает его значение.

6) Метод **shift()** удаляет **первый** элемент из массива и возвращает его значение. Этот метод изменяет длину массива.

7) Метод **map()** создаёт новый массив с результатом вызова указанной функции для каждого элемента массива.

```
let new_array = arr.map(function callback( currentValue[, index[, array]]) {
    // Возвращает элемент для new_array
}[, thisArg])
```

Следующий код берёт массив чисел и создаёт новый массив, содержащий квадратные корни чисел из первого массива.

```
var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);
// теперь roots равен [1, 2, 3], а numbers всё ещё равен [1, 4, 9]
```

Фильтрация

Метод **filter() создаёт новый массив со всеми элементами**, прошедшими проверку, задаваемую в передаваемой функции.

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
const result = words.filter(word => word.length > 6);
console.log(result);
// expected output: Array ["exuberant", "destruction", "present"]
```

Перебор элементов в массиве

1) Цикл For

```
for(let i=0; i<arr.length; i++){
  console.log('item:', arr[i]);
}
```

2) Цикл For of – автоматически перебирает все что можно перебрать

```
for(let item of arr){
  console.log('item:', item);
}
```

3) forEarch

```
arr.forEarch(item =>{
  console.log('item:', item);
});
```

2) Объекты в JS

Язык сценариев JS является объектно-ориентированным.

Объекты JS представляют собой наборы свойств и методов.

Свойства объектов – это данные, связанные с объектом, а методы - функции для обработки данных объекта.

Адресация свойств в сценариях JS возможна либо по именам свойств, либо по их номеру. Последнее возможно благодаря тому, что все свойства объекта хранятся как элементы массива и потому каждое свойство имеет свой номер.

В JS объекты, свойства и методы разделяются с помощью точек.

Пример, объект – визитная карточка с именем card.

Этот объект содержит такие свойства, как имя — name, адрес — address, номер телефона — phone и т. д. В синтаксисе JS эти свойства будут иметь следующий вид:

card.name
card.phone
card.address

Его методами будут функции, занимающиеся извлечением, изменением и другими действиями со свойствами. Например, для вызова метода, отображающего свойства объекта card, можно воспользоваться следующим синтаксисом:

card.display()

Вспомните представленные ранее примеры и обратите внимание на те из них, в которых применяется инструкция document.write. Уяснив, что JavaScript основан на работе с объектами, вы поймете, что write — это метод объекта document.

В языке JavaScript имеется три типа объектов: встроенные объекты, объекты браузера и объекты, которые программист создает самостоятельно. Каждый из этих типов имеет свое назначение и свои особенности.

Встроенные объекты – объекты, реализующие основную функциональность языка.

Ниже перечислены встроенные объекты, свойства и методы которых доступны в сценариях JS без предварительного определения этих объектов *(записать в тетрадь 2-3 примера встроенных объектов JS).*

Объект	Описание
Array	Массив
Boolean	Логические данные
Date	Календарная дата
Function	Функция
Global	Глобальные методы
Math	Математические константы и функции
Number	Числа
Object	Объект

String	Строки

\

Встроенные объекты удобны для выполнения различных операций со строками, календарными датами, массивами, числами и так далее. Они освобождают программиста от выполнения различных рутинных операций вроде преобразования строк или вычисления математических функций.

Инициализация объекта

let ob1 = {}; - пустой объект без свойств

Однако, преимущество *литеральной* или *инициирующей* нотации это возможность быстро создавать объекты со свойствами внутри фигурных скобок. Создаётся простой список пар ключ: значение, разделённых запятой. Следующий код создаёт объект с тремя парами значений и ключи это "foo", "age" и "baz". Значения этих ключей строка "bar", число 42 и другой объект.

```
var object = {
  foo: 'bar',
  age: 42,
  baz: {myProp: 12}
}
```

let ob2 = new Object();

Для обращения к свойствам используется запись «через точку»:

```
// получаем свойства объекта:
alert( user.name ); // John
alert( user.age ); // 30
```

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true  // имя свойства из нескольких слов должно быть в кавычках
};
```

Для свойств, имена которых состоят из нескольких слов, доступ к значению «через точку» не работает.

Для таких случаев существует альтернативный способ доступа к свойствам через квадратные скобки. Такой способ сработает с любым именем свойства:

```
let user = {};

// присваивание значения свойству
user["age"] = 40;

// получение значения свойства
alert(user["name"]); // John

// удаление свойства
delete user["age"];
```

Для доступа к информации внутри объекта метод может использовать ключевое слово this.

Значение this – это объект «перед точкой», который использовался для вызова метода.

```
let user = {
  name: "Джон",
  age: 30,

  sayHi() {
    // this - это "текущий объект"
    alert(this.name);
  }

};

user.sayHi(); // Джон
```

3) Функции в JS

Функция – это подпрограмма

Программисты при написании своих программ постоянно используют функции. Одной из главных целей использования функций является упрощение программирования. Действительно, ведь при написании большой программы часто возникают случаи, когда один и тот же код нужно выполнять много раз. В этом случае повторяющийся код может быть оформлен в виде подпрограммы внутри программы только со своим собственным именем.

Такие программы внутри программы в общем случае называются подпрограммами, а в разных языках программирования эти подпрограммы могут носить разное название: функции, процедуры, методы в зависимости от назначения подпрограммы.

Функции используются, когда какой-то кусок кода нужно вызвать в любом месте программы.

Функция пишется так:

```
function Имя функции(параметры) {
  тело функции
}
```

```
<meta charset ="utf-8">
<script type="text/javascript">

  function sayHello(){
            alert("Привет");
            alert("Мир");
  }

  sayHello(); /*вызов функции*/
  sayHello(); /*вызов функции*/

</script>
```

Если мы хотим, чтобы была возможность передать данные из основной программы в функцию, то можно при описании функции в скобках перечислить названия параметров функции. Теперь при вызове функции мы можем в скобках писать данные, которые будут переданы через параметры внутрь функции. Можно воспринимать параметры как специальные переменные, необходимые для передачи данных внутрь функции. Можно описать сколько угодно много параметров, перечисляя их через запятую.

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

```
<script type="text/javascript">

  function sayHello(name){
        alert('Привет ' + name);
  }

  sayHello("Вася"); /*выведет Привет Вася*/
  sayHello("Паша"); /*Выведет Привет Паша*/

</script>
```

Функции вычисляют и возвращают значения. Чтобы вернуть значение, используется ключевое слово return.

Пример: вычисление среднеарифметического трех чисел.
Можно сделать так:

```
var avg1 = (1+2+3)/3;
  var avg2 = (4+5+6)/3;

<meta charset ="utf-8">
<script type="text/javascript">
  function average(a, b, c){
        var av = (a + b + c) / 3;
        return av; /*возвращает значение в avg1 и avg2*/
  }

  var avg1 = average(1, 2, 3);
  var avg2 = average(4, 5, 6);

</script>
```

Здесь вместо average(1, 2, 3) подставляется значение, которое получится при подстановке 1, 2, 3 в функцию, стоит после return. Все что после return не выполняется.

Типы объявлений функций
1) Function Declaration

```
function sum(a, b){
  return a + b;
}
```

2) Function Expression

```
Var sum = function sum(a, b){
  return a + b;
}
```

Фактически создается переменная sum и в нее возвращается результат.
3) Стрелочные функции – не нужно слово function, перед фигурной скобкой знак =>

```
Var sum = (a, b)=>{
  return a + b;
}
Var result = sum(10, 20);
```

Если внутри стрелочной функции только одно выражение, то можно убрать и фигурные скобки и return

Var sum(a, b)=>a + b

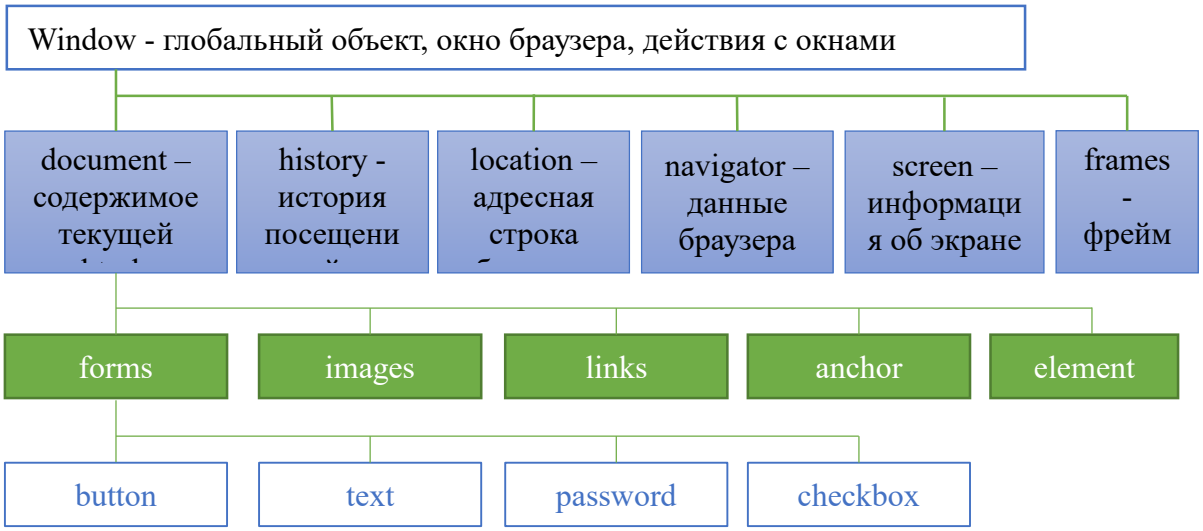
Лекция № 11. Объектная модель документа DOM. Обработка событий

План

1. Объекты браузера. Document Object Model (DOM)
2. Методы объекта Window
3. События объекта Window
4. Методы объекта Document
5. DOM-события

1) Объекты браузера. Document Object Model (DOM)

С точки зрения JS браузер представляется иерархически организованным набором объектов. На рисунке схематически показана **иерархия объектов браузера** ***(перерисовать схему)***.



На самом верху этой модели находится глобальный объект **window**. Он представляет собой одно из окон или вкладку браузера с его панелями инструментов, меню, строкой состояния, HTML страницей и другими объектами. Доступ к этим различным объектам окна браузера осуществляется с помощью следующих основных объектов: **navigator, history, location, screen, document** и т.д. Так как данные объекты являются дочерними по отношению к объекту window, то обращение к ним происходит как к свойствам объекта window.

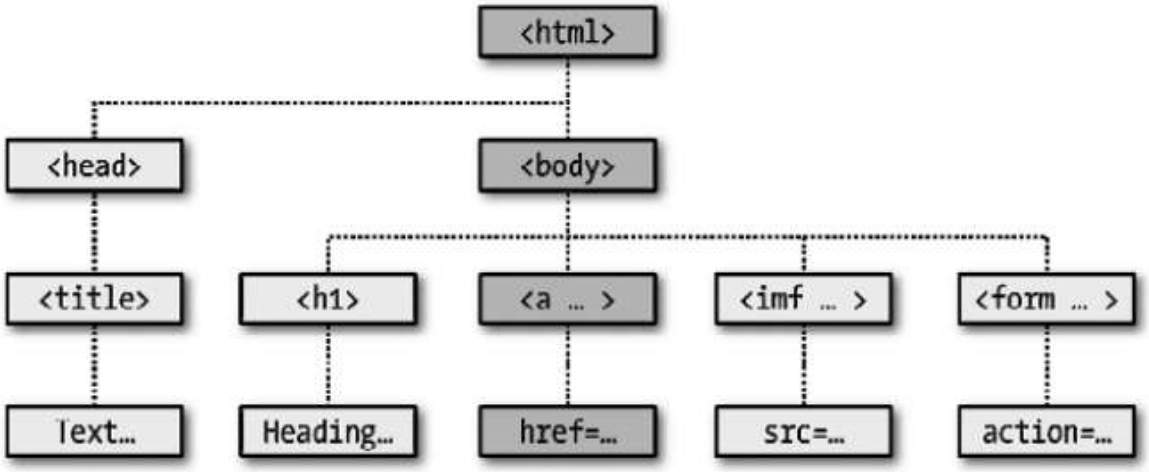
Например, для того чтобы обратиться к объекту screen, необходимо использовать следующую конструкцию: window.screen. Но если мы работаем с текущим окном, то "window." можно опустить. Например, вместо window.screen можно использовать просто screen.

Из всех этих объектов, наибольший интерес и значимость для разработчика представляет **объект document – является корнем DOM**. Данная модель в отличие от объектной модели браузера стандартизована в спецификации и поддерживается всеми браузерами.

Объект document представляет собой HTML документ, загруженный в окно (вкладку) браузера. С помощью свойств и методов данного объекта можно получить доступ к содержимому HTML-документа, а также изменить его содержимое, структуру и оформление.

Document Object Model (DOM) – объектная модель документа, предоставляющая браузеру доступ к элементам web-страницы, описанным с помощью HTML-кода.

Эта модель разбивает части HTML-документа на отдельные *объекты*, у каждого из которых есть собственные *свойства* и *методы* и каждым из которых можно управлять с помощью JavaScript.



На этом рисунке используются уже знакомые вам HTML-теги, иллюстрирующие родительско-дочерние взаимоотношения между различными объектами.

2) Методы объекта Window

Метод	Действие	Код
alert()	Выводит сообщение в окно с кнопкой ОК	window.alert('Сообщение');
confirm()	Выводит сообщение в окно с кнопками ОК и Cancel. Если будет нажато "ОК", то возвратится ИСТИНА, а если Cancel, то ЛОЖЬ.	window.confirm('Сообщение');
prompt()	Просит у пользователя ввести строку и возвращает ее	window.prompt('Введите строку', 'Строка');
open()	Открывает новое окно браузера	myWin = window.open("URL", "имя_окна", "параметр=значение, параметр=значение,...");
c	Закрывает окно браузера	w
setTimeout()	Выполнения определённых действий через определённые промежутки времени	window.setTimeout('TimerAlert()', 3000);
location.href	Переходит к указанной странице	window.location.href = 'somedoc.html';

Поскольку *объект* window является самым старшим, то в большинстве случаев при обращении к его свойствам и методам приставку "window." можно опускать.

Например: можно писать *alert* ('Привет') вместо window.*alert* ('Привет').

Исключениями из этого правила являются вызовы методов open() и close(), у которых нужно указывать *имя окна*, с которым работаем (родительское в первом случае и дочернее во втором).

3) События объекта Window

Рассмотрим события, связанные с объектом window. Обработчики этих событий обычно помещают как *атрибут* контейнера <body> или у тега <form>.

• ***Load*** **- событие происходит в момент, когда загрузка документа в данном окне полностью закончилась.**

<BODY onLoad="alert('Документ полностью загружен.');">

• **Unload - событие происходит в момент выгрузки страницы из окна.**

Например, когда пользователь закрывает окно, либо переходит с данной Web-страницы на другую, кликнув ссылку или набрав адрес в адресной строке, либо при изменении адреса страницы (свойства window.*location*) скриптом.

Например, при уходе пользователя с нашей страницы мы можем позаботиться о его удобстве и закрыть открытое ранее нашим скриптом окно:

<BODY onUnload="myWin.close();">

• **Error - событие происходит при возникновении ошибки в процессе загрузки страницы.**

Если это событие произошло, можно, например, вывести сообщение пользователю с помощью *alert*() или попытаться перезагрузить страницу с помощью window.*location.reload*().

В следующем примере назначена обработчиком события Error функция ff(), которая будет выдавать сообщение. В тексте программы мы допустили ошибку: слово *Alert* написано с заглавной буквы (помните, что в JavaScript это недопустимо). Поэтому при открытии этого примера возникнет ошибка и пользователь получит об этом "дружественное" сообщение.

```
<SCRIPT>
    function ff()
    { alert('Произошла ошибка. Свяжитесь с Web-мастером.'); }

    window.onerror = ff;

    Alert('Привет');
</SCRIPT>
```

• ***Focus*** **- событие происходит в момент, когда окну передается фокус**.

Например, когда пользователь "раскрывает" свернутое ранее окно, либо (в Windows) выбирает это окно браузера с помощью Alt+*Tab* среди окон других приложений. Это событие происходит также при программной передаче фокуса данному окну путем вызова метода window.*focus*(). Пример использования:

<BODY onFocus="alert('Спасибо, что снова вернулись!');">

4) Методы объекта Document

Объект **document является важнейшим свойством объекта window.**

Методы объекта Document:

- ***document.getElementById()*** – обеспечивает доступ к элементу через id, возвращает элемент с указанным id
- ***document.getElementsByTagName()*** – обеспечивает доступ к массиву элементов через название тега name, возвращает массив элементов с указанным именем тега
- ***document.getElementsByClassName()*** – обеспечивает доступ к элементам массива через название класса
- ***document.getElementsByName()*** - обеспечивает доступ к массиву элементов по значению атрибута name
- ***document.querySelector()*** – возвращает первый элемент в документе, соответствующий указанному селектору.
- **document.createElement()** - позволяет создать и вернуть новый элемент с указанным именем тега.
- **document.createAttribute()** - создает новый атрибут с указанным именем и возвращает его.

5) DOM-события

Обработчик события – функция, которая добавляется к элементу, и вызывается, когда произойдет нужное действие.

События мыши:

• click – происходит, когда кликнули на элемент левой кнопкой мыши.
• contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши.
• mouseover / mouseout – когда мышь наводится на / покидает элемент.
• mousedown / mouseup – когда нажали / отжали кнопку мыши на элементе.
mousemove – при движении мыши

События клавиатуры:

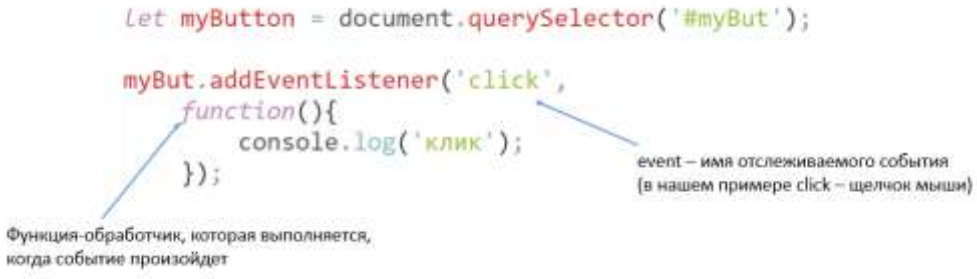
• keydown и keyup – когда пользователь нажимает / отпускает клавишу

События на элементах управления:

• submit – пользователь отправил форму <form>.
• focus – пользователь фокусируется на элементе, например нажимает на <input>.

Обработка событий

• Метод **addEventListener** – добавить обработчик (слушатель) событий – универсальный метод, позволяющий назначить несколько обработчиков.



• Использование **атрибута HTML – on<событие>**

```
<input value="Нажми меня" onclick="alert('Клик!')" type="button">
```

При клике мышкой на кнопке выполнится код, указанный в атрибуте onclick

• Использование **свойства DOM-объекта** – on<событие> применятся в JS к элементу (по сути аналогичен предыдущему способу)

```
<input id="elem" type="button" value="Нажми меня!">
<script>
  elem.onclick = function() {
    alert('Спасибо');
  };
</script>
```

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ

Основная литература:

1. Полуэктова, Н. Р. Разработка веб-приложений: учебное пособие для среднего профессионального образования / Н. Р. Полуэктова. — Москва: Издательство Юрайт, 2022. — 204 с. *(образовательная платформа Юрайт https://urait.ru/)*

Интернет-ресурсы:

1. Колисниченко Д.Н. Разработка веб-приложений. – Спб.: БХВ-Петербург, 2017. – 640 с. [Электронный ресурс]. Форма доступа: https://books.google.ru/books?id=BjExDwAAQBAJ&printsec=frontcover&hl=ru#v=onepage&q&f=false
2. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. – СПб.: Питер, 2019. – 688 с. [Электронный ресурс]. Форма доступа: https://booster.by/files/oeu.pdf
3. Учебники по HTML и CSS [Электронный ресурс]. Форма доступа: https://html5css.ru/
4. Учебники, задачники, справочники по web языкам [Электронный ресурс]. – Форма доступа: http://code.mu/
5. Уроки PHP. [Электронный ресурс]. Форма доступа: http://myblaze.ru/php_lessons/
6. <MyRusakov.ru/>. Уроки и статьи по созданию сайтов [Электронный ресурс]. Форма доступа: http://MyRusakov.ru.
7. Ruseller.com. Частная коллекция качественных материалов для тех, кто делает сайты [Электронный ресурс]. Форма доступа: http://ruseller.com/