# HTML

1. **HTML** (**HyperText Markup Language**) – язык, на котором описывается структура элементов сайта.
2. Структура состоит из иерархических «блоков» - **узлов** / **тэгов** / **заголовков**.

Структура строится иерархически из элементов (тегов, узлов), таких как <html>, <div>, <p>, <h1>.

Пример:

Так корневой узел в структуре html это <html></html>:  
  
<!DOCTYPE html>

<html>

<head>

<title>Минимальная страница</title>

</head>

<body>

<h1>Привет мир!</h1>

<p>Это самая простая HTML страница</p>

</body>

</html>

# Браузер

## Определения

Страница / сайт / документ / веб-страница –

## Общая механика работы браузера

1. Это обычное десктопное приложение, которое умеет делать запросы в интернет и расшифровывать ответы
2. Визуал сайта это данные которые браузер получил на запрос и расшифровывал.

Когда пользователь делает запрос и сайт визуально открывается – это и есть данные, которые сайт получил в ответ на совершенный запрос.

1. HTTP и прочие протоколы, и многие другие интернет стандарты устроены таким образом, чтобы с помощью сериализованных битовых данных можно было отправлять и принимать любые данные: визуал сайтов и изображения, разные типы файлов.
2. Таким образом браузер:
3. Отправляет запрос …
5. Более полной и корректный механизм работы браузера представлен в Механизм работы браузера

## Детали устройства работы браузера

1. Как и любая другая дескопная программа имеет в основе объектно-ориентированную логику и такие компоненты браузера как (вкладка, ..) – это **объекты** (то есть экземпляры некоторого класса).

## Корневой объект страницы

1. Так такой компонент как **вкладка (window)** – это корневой объект для всего визуала, который будет отображен для пользователя.

Вся информация о сайте, его структуре, его визуале итд содержится в window:

1. У этого класса есть множество свойств и методов, но основные свойства касающиеся сайта:  
   <https://developer.mozilla.org/ru/docs/Web/API/Window>

window.document // DOM документ

window.navigator // Информация о браузере

window.location // URL и навигация

window.history // История браузера

window.screen // Информация об экране

window.frames // iframes

window.parent // Родительское окно

window.top // Верхнее окно

window.self // Текущее окно

## HTML файл -> визуальная страница во вкладке

1. Сайт (или страница / документ) имеет определенную структуру заголовков (или **тэгов** / **узлов**), которая описывается в html файлах.

Заголовок / тэг / узел - в html это <div>, <table>, итд.

1. Однако браузер отображает не полученный html на прямую, а
2. Считывает html файл, разбивая структуру на узлы.
3. В оперативной памяти создаёт объекты (экземпляры классов) для каждого узла, так чтобы получилась точно такая же структура из соответсвующих объектов.
4. Каждый объект на странице вкладки отображает свой визуальный элемент, так что получается структура из визуальных элементов и как результат полная визуальная структура – копия исходного html.

Тк такая структура объектов (узлов) существует в оперативной памяти, то с ней уже можно взаимодействовать с помощью скриптов изменять визуал страницы (изменять DOM).

1. Более полный и детальный процесс рендеринга страницы из html писан в Рендеринг страниц.

## DOM

1. Таким образом вся структура сайта в оперативной памяти – это совокупность объектов узлов, которые объединили в чисто условное понятие **DOM** (**Document Object Model**).

DOM – это только обозначение для множества объектов узлов, DOM не представлен в коде вообще.

1. Можно сказать, что страница целиком представлена в объекте document, который обязан соответствоать DOM спецификации.

А саму структуру страницы можно назвать DOM и она представлена в виде структуры из объектов узлов каждый из которых является свойством другого объекта, кроме корневого document.

Иван иванов отец семьи – document

Семья ивановых - DOM

## Скрипты и изменение DOM

1. Для взаимодействия с каждым узлом DOM существует специальный объект **document** , представляющий текущую страницу на вкладке и являющийся свойством объекта window.  
     
   С таким объектом в оперативной памяти уже можно взаимодействовать и который может изменяться в отличии от html.

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Материалы

### Механизм работы браузера

1. Процесс рендеринга страницы (от получения байтов до отрисовки пикселей) — это сложный конвейер. Упрощенно его можно представить так:
2. Навигация: Браузер отправляет HTTP(S) запрос на сервер по введенному URL.

Получение ответа: Сервер отвечает. Ответ обычно включает:

Статус-код (например, 200 OK или 404 Not Found)

Заголовки (Headers) с мета-информацией (тип содержимого, кодировка и т.д.)

Тело ответа (Body) — сам HTML-документ.

1. Парсинг (Разбор HTML):

Браузер начинает обрабатывать полученные байты HTML.

Он строит Дерево DOM (Document Object Model). Это пошаговый процесс: читаются теги, создаются узлы, строится иерархия.

Загрузка дополнительных ресурсов: Парся HTML, браузер находит ссылки на CSS, JavaScript, изображения, шрифты. Для каждого из этих ресурсов он отправляет новые запросы.

Обработка CSS и построение CSSOM: Пока грузятся CSS-файлы, браузер парсит их и строит CSSOM (CSS Object Model) — дерево стилей.

1. Компоновка (Layout / Reflow): Браузер комбинирует DOM и CSSOM, чтобы вычислить геометрию каждого элемента на странице: его размеры, положение относительно viewport (области просмотра) и друг друга.
2. Отрисовка (Paint): На этом этапе браузер преобразует вычисленную геометрию в пиксели на экране. Он "рисует" элементы в их слоях.
3. Композиция (Composition): Слои объединяются в окончательное изображение, которое пользователь видит на экране.

1. Любое взаимодействие с JavaScript, изменяющее DOM или стили, может запустить часть этого конвейера заново (например, только перерисовку или, что дороже, перекомпоновку).

### Рендеринг страниц

1. Шаги рендера.
2. Парсинг HTML: Браузер читает HTML-код и строит из него DOM-дерево — иерархическую структуру объектов в памяти.
3. Парсинг CSS: Строится CSSOM — дерево стилей.
4. Формирование дерева рендеринга: DOM и CSSOM объединяются.
5. Компоновка (Layout): Вычисляется точное положение и размер каждого элемента.
6. Отрисовка (Paint): Браузер "рисует" пиксели на экране.

# Написание кода веб-страниц

## Доставка веб-страниц

1. Архитектура доставки сайтов конечным пользователям основана на передаче файлов по интернету через протокол HTTP.  
   Это значит, что каждый раз при клике пользователя на сайт (или на ссылку на сайт) браузер запрашивает файлы у другого компьютера.

Такая модель называется **клиент – серверная архитектура**.

1. Вся эта модель и в частности HTTP протокол накладывают некоторые ограничения на формат данных, которыми можно описывать веб-страницы.

Сайт стоит описывать таким образом, чтобы его можно было легко запрашивать, передавать через HTTP протокол по сети, и расшифровывать при получении.

Так веб-страницы должны быть описаны в формате, который легко адресовать по URL, передавать как байтовый поток по HTTP, распознавать по Content-Type, кэшировать, парсить по мере получения и безопасно исполнять в браузере.   
  
Более полный и детальный список ограничений представлен в главе Список ограничений модели «клиент–сервер + HTTP».

## Формат описания веб-страниц

1. Эти требования привели к появлению самого популярного формата для описания сайтов – **HTML**, удовлетворяющий этим требованям.

|  |
| --- |
| Note!!!  Стоит подчеркнуть, что любые сайты так или иначе передаются конечным клиентам в виде HTML.  Существуют более продвинутые способы разработки сайтов, но стоит понимать, чтобы даже разрабатывая сайты на другом языке они в конечном итоге всё равно компилируются в HTML и передаются по HTTP и доставляются клиентам именно в этом формате. |

1. Технически всегда нужен HTML-файл как точка входа. Даже если он пустой.

Минимум:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>JS-only</title>

</head>

<body>

<script src="main.js"></script>

</body>

</html>

Браузер не умеет напрямую открыть .js и показать его как страницу. Он всегда интерпретирует JS только в контексте HTML.

Исключение — среды вне браузера (Node.js, Deno). Там можно запускать .js без HTML, но это уже не «страница», а просто программа.

## Общая механика создания и доставки веб-страниц

1. Сайт разрабатывается на языке HTML, позволяющим легко доставлять по протоколу HTTP структуру сайта и данные внтури него конечному пользователю.
2. Готовый HTML помещается на сервер и настраивается прослушивание портов, чтобы по запросам правильно возвращать клиентам этот файл HTML по интернету.
3. Клиенты получают HTML файл(ы) и клиентский браузер расшифровывает их так, чтобы была визуальная страница.

## \_\_\_\_\_\_\_\_\_\_\_\_\_

## Виды веб-страниц

1. Полностью статичный сайт – HTML  
     
   Никаких переменных нет, числа нельзя ихиенить переменную нельзя изменить.  
     
   Как написано в HTML так и будет на сайте.
2. Динамика на стороне клиента   
   Например HTML + JS / React
3. Динамичный сайт.   
   Сайт поддерживают и изменяют на стороне сервера

## Разработка разных видов веб-страниц

1. !!! Смотри Note!!! в Формат описания веб-старниц

### Простой HTML

Статичный сайт

1. Можно писать прямо на HTML и помещать на сервер.
2. Сайт получится полностью статичным.

### Сайт с динамикой на стороне клиента

1. Разработка такого вида сайтов основана внедрение скриптов в HTML, которые запускаются при действиях пользователя и меняют HTML сайта, чтобы он в реальном времени изменялся для пользователя.

Так можно создавать более сложные и продвинутые дизайны сайтов с более глубоким взаимодействием с пользователем

1. В настоящее время абсолютное большинство таких скриптов написано на JavaScript.

А также очень сложные и продвинутые фреймворки для написания таких скриптов со сложной логикой (будут рассмотрены ниже) также написаны на JS.

Более подробно почему так и про то как добавить динамику для сайта рассмотрено в Добавление динамики в сайтах

#### HTML + JS

1. В этом варинте в html добавляются ссылки на скрипты (или прямые вставки) на языке JavaScript.
2. Основная задача JS скриптов — добавить динамику: менять DOM (структуру HTML в памяти браузера), обрабатывать события (клики, ввод), подгружать данные.
3. Скрипты JS использует браузер, который создал DOM и с помощью этих скриптов может изменять DOM и соотвественно визуал веб-страницы для клиента.
4. JS-файл не может быть «альтернативой HTML». Самостоятельно он не отображается. Но внутри него можно создать и вставить HTML-элементы через DOM-методы (document.createElement, innerHTML и т.п.).
5. Поэтому JS может «собрать страницу на лету» с помощью вставок document.createElement, но всё равно браузер ждёт какой-то HTML как точку входа (хотя бы пустой <html><body><script src="..."></script></body></html>).
6. Браузер не умеет напрямую открыть .js и показать его как страницу. Он всегда интерпретирует JS только в контексте HTML.

Исключение — среды вне браузера (Node.js, Deno). Там можно запускать .js без HTML, но это уже не «страница», а просто программа.

##### JS фреймворки

Продвинутая динамика на стороне клиента

1. Что делают фреймвоки:  
     
   1. Клиент загрузил сайт ( html + js скрипты + библиотеки возможно)
2. Браузер открывает html и запускает js скрипт.   
     
   если я не буду использовать библиотеки, то мне любое изменение DOM, прослушивание кликов и прочее нужно реализовывать самостоятельно. А react предлагает готовые инструменты для добавления интерактивности и динамики веб-страницы (обновление DOM по определенным правилам, если я хочу чтобы моя веб-страница была интерактивной).  
     
   а если я использую реакт, то:
3. JS скрипты запускают множественные процессы прослушивания, создания виртуального DOM , для отслеживания изменений, а также логика для кнопок и прочего, описанная мною же в jsx .
4. Так JS-фреймворки (например React, Vue, Angular) — это библиотеки/среды, которые реализуют:

* **Прослушивание событий**: клики, наведение, ввод текста, прокрутка.
* **Манипуляцию DOM**: динамическое добавление и удаление элементов, изменение атрибутов, классов и стилей.
* **Реактивное обновление интерфейса**: автоматический ререндер при изменении состояния (например, изменение значения в форме сразу отображается на экране).
* **Управление состоянием**: хранение и передача данных между компонентами (например, Redux, Vuex).
* **Маршрутизацию**: переключение между страницами без перезагрузки (SPA).
* **Работу с сервером**: асинхронные запросы (AJAX, fetch) и реакция на полученные данные.
* **Компонентный подход**: переиспользуемые блоки интерфейса с собственной логикой и стилями.

###### React

## \_\_\_\_\_\_\_\_\_\_\_\_\_

## Материалы

### Список ограничений модели «клиент–сервер + HTTP»

1. Список ограничений модели «клиент–сервер + HTTP», которые диктуют формат представления сайтов:

* Адресуемость: всё должно быть доступно по URL как ресурс.
* Запрос–ответ: данные передаются как байтовый поток с явным Content-Type.
* Без состояния: HTTP статeless → состояние либо в клиенте, либо в явных механизмах (URL, формы, cookies).
* Кешируемость: формат должен поддерживать эффективный кеш и валидацию (ETag, Last-Modified, Cache-Control).
* Идемпотентность и безопасность методов: навигация обычно через GET (без побочных эффектов), изменение — через POST/PUT/DELETE.
* Контент-неготиация и локализация: вариации по Accept, Accept-Language, Accept-Encoding.
* Инкрементальная отдача: желательно рендериться по мере получения потока (стриминг, ранний First Paint).
* Встраивание подресурсов: документ должен ссылаться на изображения, стили, скрипты как отдельные URL.
* Ссылкоцентричность: навигация должна строиться через гиперссылки между ресурсами.
* Детерминируемость парсинга: чёткие правила разбора, устойчивые к частичным ошибкам сети.
* Сжатие и диффы: формат должен хорошо ужиматься (gzip, br) и передаваться частями (Range).
* Безопасность на уровне типа: корректный MIME исключает «content sniffing» и даёт верный контур песочницы браузера.
* Кросс-оригин контроль: ресурсы должны корректно работать в модели источников (CORS, SameSite у cookies).
* Семантика для машин: желательно иметь структуру, которую можно индексировать и аннотировать.