**Электронный вариант дипломной работы**

**История HTML**

Давным давно, практически на заре цивилизации (до 70-х годов XX века), не было интернета. Вообще не было.

С появлением компьютеров появилось желание и потребность как-то их объединить, и 4 крупных университета США взялись за эту задачу. Примерно 7 лет прошло от создания концепции до исторического момента:

**29 октября 1969** года был проведен сеанс связи между двумя первыми узлами сети ARPANET, на расстоянии в 640 км: в Калифорнийском университете и в Стэнфордском исследовательском институте. Ученый из Лос-Анджелеса подключился удаленно к компьютеру в Стэнфорде. Стэнфордский коллега видел, как появляются на экране символы, введенные на расстоянии, и подтверждал по телефону передачу каждой буковки. Таким образом началась эпоха компьютерных сетей.

Долгое время Интернет был предназначен только для специалистов и служил в основном для обмена технической документации и имейлами. Простому пользователю делать там было нечего. И только в начале 90-х прошлого века произошла революция "интернет-народу!"

1991 год

Британец **Тимоти Джон Бернерс-Ли** в Женевском ЦЕРНе изобрел **язык гипертекстовой разметки**, он же **HyperText Markup Language**, он же **HTML**, предназначенный для разметки и оформления документов World Wide Web.



Ах да! Еще Сэр Тим разработал глобальный гипертекстовый проект (сейчас Вы его знаете как **Всемирную паутину**). Собственно, HTML и родился в процессе работы над этим проектом.

Язык HTML

HTML это наследник [SGML](https://ru.wikipedia.org/wiki/SGML), вот только создавался он для того, чтобы им могли пользоваться и люди-неспециалисты в области верстки. Т.е. уже с первых дней у HTML были такие плюсы:

* Простота - за счет небольшого набора структурных элементов – дескрипторов (они же «[теги](https://vertex-academy.com/tutorials/ru/kak-sozdat-pervuyu-veb-stranicu/)»). Все теги пишутся в угловых скобках, напр. <img> и несут какую-то смысловую нагрузку
* Возможность форматировать документ без привязки к средствам отображения (будь то монитор компьютера, экранчик телефона, или ридер)

И как вы уже, наверное, догадались, **первый в мире веб-сайт**создал тоже Тим Бернерс-Ли  И если точная дата изобретения HTML не известна (т.к. это был долгосрочный проект), то день рождения первого сайта очень даже - это было **6 августа 1991**года. Кстати, его можно увидеть и сегодня, правда уже в [архиве](http://info.cern.ch/hypertext/WWW/TheProject.html).

Полагаю, **все веб-разработчики** с полным правом могут считать именно **6 августа своим профессиональным праздником**



Как вам понравилась первая веб-страница?  Думаю, большинству из вас не захотелось её читать и уж тем более оставаться там надолго. Создатели тоже понимали, что это только начало, и пошло поехало:

**Июнь 1993 - HTML 1.2**

В этой версии, из её сорока с копейками тегов уже появилось аж 3 тега, которые намекали на какое-то визуальное оформление документа (например, полужирный курсив). Остальные же теги служили исключительно для логической разметки.

**1994 год - основание W3C**

Сэр Тим Бернерс-Ли основал **Консорциум Всемирной Паутины** (World Wide Web Consortium, **W3C**). Миссией W3C было и остается

Полностью раскрыть потенциал Всемирной паутины, путём создания протоколов и принципов, гарантирующих долгосрочное развитие Сети

Огромная заслуга этих ребят принадлежит в том, что HTML был выпущен с единым базовым набором тегов и атрибутов и веб-страницы стали такими, какими мы их знаем сейчас. Только представьте, в середины девяностых, несколько крупнейших производителей ПО планировали выпустить свои версии HTML со своими названиями тегов. Какая каша была бы сейчас в сфере веб-разработки! Например объявления о работе: *«Ищем верстальщика для создания сайтов в Mozilla Firefox», «Срочно требуется верстальщик для Edge», «Нужен верстальщик для Kindle»…*словом, спасибо W3C

**22 сентября 1995 – Версия 2.0**

Процесс разработки и утверждения новой версии был очень неспешным, а единственным заметным улучшением новой версии стали:

* Запросы: например поиск по ключевым словам.
* Формы для передачи данных с компьютера на сервер: например ввести дату рождения или выбрать один из нескольких вариантов в опроснике.

**Март 1995 – начало работы над HTML 3.0**

Первый вариант стандарта включал в себя много интересностей:

* теги для создания таблиц,
* разметки математических формул,
* обтекание изображений текстом и др.

Авторы вдруг вспомнили, что по своей идее HTML должен только размечать структуру документа и не должен содержать параметры графических стилей отображения элементов в браузере.

При создании HTML 3.0 разработчики ломали голову, как решить нестыковку между идеологией структурной разметки и потребностями пользователей, которых гораздо больше интересовал внешний вид веб-страницы.

И чтобы это противоречие не изменило первоначальные свойства HTML, создатели третьей версии решили добавить к ней поддержку нового средства, которое служило для **оформления** веб-страниц.

**17 декабря 1996 – CSS**

**CSS (Cascading** **Style** **Sheets)**, а по-русски – иерархические стилевые спецификации. Думаю, понятней не стало?)) Если совсем уж по-русски, то это таблицы стилей, которые присоединяются к документу HTML и служат для визуального оформления тех или иных частей документа.

Итак, система CSS:

* формально не зависит от HTML,
* у неё свой, отличный от HTML, синтаксис,
* На неё не влияют идеологические ограничения HTML,
* Она позволяет задавать параметры внешнего представления для любого тега HTML.

С помощью CSS создатель веб-страницы наконец мог спокойно менять шрифт и размер чего угодно, и что важно – это стало возможным не за счет внедрения кучи новых тегов, а с помощью механизма, который влиял на интерпретацию уже существующих тегов HTML.

Чтобы наглядно продемонстрировать чудеса, на которые способен CSS, канадец Дейв Ши создал [сад CSS дзена](http://csszengarden.com/) – место, где сотни веб-разработчиков медитируют верстая одну и ту же HTML-страницу с абсолютно [разными стилями CSS](http://www.mezzoblue.com/zengarden/alldesigns/)

**Влияние Microsoft**

А тем временем, в далекой-далекой галактике… Microsoft заметил стремительный взлет компании Netscape на продаже браузеров Navigator и конечно же, не мог стоять в стороне. В MS слегка переработали браузер Mosaic и начали выпускать свои эксплореры, которые поначалу не очень полюбились пользователям (да и не только поначалу;)).

В августе 1996 была создана версия Internet Explorer 3.0 Браузер предлагал значительные нововведения на то время и обрел популярность, что поделило рынок браузеров пополам между Netscape Communications и Microsoft.

И в это же время Microsoft взял под свою опеку W3C.

**14 января 1997 – HTML 3.2**

Версия 3.2 вышла спустя месяц после утверждения CSS, и была уже полностью приспособлена к взаимодействию с таблицами стилей.

В ней были опущены многие нововведения версии 3.0, но добавлены нестандартные элементы, поддерживаемые браузерами Netscape Navigator и Internet Explorer 3.

**18 декабря 1997 – HTML 4.0**

В этой версии изрядно подчистили элементы из предыдущих версий. Многие теги были отмечены как устаревшие и не рекомендованные к использованию. Вместо них нужно было использовать таблицы стилей CSS**.**

Новая версия включала поддержку фреймов, скриптов, общую процедуру внедрения разных объектов. Также в ней были усовершенствованы таблицы и формы, что кроме прочих плюсов обеспечивало б**о**льшую доступность для людей с физическими недостатками.

Версия HTML 4 была разработана с помощью экспертов в области интернационализации, таким образом стало возможно писать документы на любом языке и легко передавать их по всему миру.

**24 декабря 1999 – HTML 4.01**

В этой версии слегка подправили объекты, формы и изображения, пофиксили баги и в целом создали более стабильную версию, которой пользовались веб-разработчики **более 10 лет**.

**Рабочая группа** **WHAT**

**2004 год**: Швейцарский программист **Ян Хиксон** (на тот момент разработчик в Opera) и несколько представителей таких компаний как Mozilla, Google и Apple, основал рабочую группу под названием **WHATWG** (*Web* *Hypertext* *Application* *Technology* *Working* *Group*).

Поводом для создания подобного сообщества стал тот факт, что W3C на тот момент потерял интерес к HTML и занялся развитием [XML](https://ru.wikipedia.org/wiki/XML), а на его основе - расширяемого языка гипертекстовой разметки **XHTML**. Не будем углубляться в детали, т.к. на данный момент этот язык перестал развиваться.

Два года W3C и WHAT Working Group работали каждый над своим проектом. Но позже стало видно, что WHAT Working Group достигает каких-то результатов, тогда как XHTML 2 так и не реализовался.

И в 2006 году Тим Бернерс-Ли объявил, что W3C и WHATWG будут вместе работать над дальнейшим развитием HTML.

**28 октября 2014 – HTML5**

C этого дня W3C официально рекомендует использовать HTML5.

* Новая версия сделала синтаксис более строгим по сравнению с предыдущей
* Улучшилась поддержка мультимедиа-технологий
* Появились 28 новых структурных элементов, благодаря которым код стал более понятным
* Исключена еще часть устаревших тегов
* Стало больше внимания уделяться поддержке скриптов, например javascript



На данный момент развитие HTML продолжается…

W3C выложил в начале июня рабочий черновик [версии 5.1](https://www.w3.org/TR/html/). Консорциум просит всех писать свои отзывы и комментарии относительно этой версии, т.к. HTML остается как и раньше проектом, над которым работают одновременно множество неравнодушных людей, любящих угловые скобки.

Может, скоро и вы станете одним из них?

И напоследок, минутка юмора:

Оптимист учит английский, Пессимист - китайский, а реалист учит html.

Будьте реалистами

**История CSS**

****

CSS — одна из широкого спектра технологий, одобренных консорциумом W3C и получивших общее название «стандарты Web». В 1990-х годах стала ясна необходимость стандартизировать Web, создать какие-то единые правила, по которым программисты и веб-дизайнеры проектировали бы сайты. Так появились языки HTML 4.01 и XHTML, и стандарт CSS.  
В начале 1990-х различные браузеры имели свои стили для отображения веб страниц. HTML развивался очень быстро и был способен удовлетворить все существовавшие на тот момент потребности по оформлению информации, поэтому CSS не получил тогда широкого признания.  
  
Термин «каскадные таблицы стилей» был предложен Хокон Виум Ли в 1994 году. Совместно с Бертом Босом он стал развивать CSS.  
  
В отличие от многих существовавших на тот момент языков стиля, CSS использует наследование от родителя к потомку, поэтому разработчик может определить разные стили, основываясь на уже определенных ранее стилях.  
  
В середине 1990-х Консорциум Всемирной паутины (W3C) стал проявлять интерес к CSS, и в декабре 1996 года была издана рекомендация CSS1.  
  
**Уровень 1 (CSS1)**  
  
Рекомендация W3C, принята 17 декабря 1996 года, откорректирована 11 января 1999 года. Среди возможностей, предоставляемых этой рекомендацией:  
  
    Параметры шрифтов. Возможности по заданию гарнитуры и размера шрифта, а также его стиля — обычного, курсивного или полужирного.  
    Цвета. Спецификация позволяет определять цвета текста, фона, рамок и других элементов страницы.  
    Атрибуты текста. Возможность задавать межсимвольный интервал, расстояние между словами и высоту строки (то есть межстрочные отступы)  
    Выравнивание для текста, изображений, таблиц и других элементов.  
    Свойства блоков, такие как высота, ширина, внутренние (padding) и внешние (margin) отступы и рамки. Так же в спецификацию входили ограниченные средства по позиционированию элементов, такие как float и clear.  
  
**Уровень 2 (CSS2)**  
  
Рекомендация W3C, принята 12 мая 1998 года. Основана на CSS1 с сохранением обратной совместимости за несколькими исключениями. Добавление к функциональности:  
  
    Блочная вёрстка. Появились относительное, абсолютное и фиксированное позиционирование. Позволяет управлять размещением элементов по странице без табличной вёрстки.  
    Типы носителей. Позволяет устанавливать разные стили для разных носителей (например монитор, принтер, КПК).  
    Звуковые таблицы стилей. Определяет голос, громкость и т. д. для звуковых носителей (например для слепых посетителей сайта).  
    Страничные носители. Позволяет, например, установить разные стили для элементов на чётных и нечётных страницах при печати.  
    Расширенный механизм селекторов.  
    Указатели.  
    Генерируемое содержимое. Позволяет добавлять содержимое, которого нет в исходном документе, до или после нужного элемента.  
  
В настоящее время W3C больше не поддерживает CSS2 и рекомендует использовать CSS2.1  
  
**Уровень 2, ревизия 1 (CSS2.1)**  
  
Рекомендация W3C, принята 7 июня 2011 года.  
  
CSS2.1 основана на CSS2. Кроме исправления ошибок, в новой ревизии изменены некоторые части спецификации, а некоторые и вовсе удалены. Удаленные части могут в будущем быть добавлены в CSS3.  
  
**Уровень 3 (CSS3)**  
  
Разрабатываемая версия.  
Сильно расширена по сравнению с предыдущими версиями. Нововведения, начиная с малых, вроде закругленных углов блоков, заканчивая трансформацией (анимацией) и, возможно, введением переменных.  
  
**Уровень 4 (CSS4)**  
  
CSS4 разрабатывается консорциумом W3C с 29 сентября 2011 года. Модули CSS4 построены на основе CSS3 и дополняют их новыми свойствами и значениями. Все они существуют пока в виде черновиков (working draft).

**История Photoshop**

****

Дата создания **Adobe Photoshop** не отмечена ни на одном календаре. В 2005 году состоялась годовщина выпуска программы, которой вы и многие другие пользуетесь больше всего. Всё произошло [17 лет назад\*](https://akvis.com/ru/articles/photoshop-history/index.php#comm), в феврале, когда компания Adobe «спустила на воду» первую версию Photoshop — 1.0 – графического редактора, который до сих пор является самым популярным среди художников, фотографов и дизайнеров. **Photoshop** на сегодняшний день занимает позиции лидера среди редакторов растровой графики, а также является единственной программой, которая обзавелась собственной формой глагола.



Своё начало Photoshop берёт гораздо раньше. Программа, на экранной заставке которой сегодня список разработчиков – их 41, ранее принадлежала двум братьям – **Томасу** и **Джону Нолл**. С детства отец привил сыновьям любовь к искусству и компьютерной технике. В фотолаборатории отца, которая была расположена в подвале, Томас изучал основы цветокоррекции и контраста, а Джон с интересом ковырялся в стареньком Apple. В 1984 году отец купил один из первых Макинтошей, функциональные возможности которого очень разочаровали братьев – именно это и положило начало созданию программы, которая в будущем принесёт прибыль в миллионы долларов.

**Начало**

До 1987 г. Джон Нолл работал в Industrial Light and Magic (ILM) – в подразделении фирмы Lucasfilm, которое специализировалось на создании спецэффектов для проекта «Звёздные войны» («Stars Wars»). Томас в это время писал диссертацию о процессе обработки изображений – получение степени кандидата наук для него было самым главным в то время. Окончательно разочаровавшись в новеньком Apple Mac Plus – монитор только что купленного компьютера не отображал полутоновые изображения (Grayscale images), Томас приступил к написанию программы, которая поможет исправить ситуацию.

Как ни странно, в ILM Джон также работал над процессом обработки изображений. Поражённый успехами брата, Джон предложил Тому объединить усилия. В своей книге „Основы компьютерной графики“ Джон писал: „Как только увидел результаты работы Томаса, я вспомнил программное обеспечение по обработке изображений в студии Pixar, сходства были поразительными“. С того момента мы начали совместную разработку более сложной и совершенной программы, которую в последствии нарекли Display.

Немного времени спустя Джон приобрёл новый Макинтош II с цветным экраном и убедил Томаса переписать «Display» для работы с цветными изображениями. Более того, чем больше Джон работал с „Display” тем больше функций он требовал от программы: коррекцию цветовой гаммы, импорт и сохранение файлов в различных форматах и т.д.

Работа над программой отвлекала Томаса от написания диссертации, но всё же он был этому рад. Некоторое время спустя Том разработал инновационный метод выбора отдельного участка изображения и последующей работы с ним, а также наборы приложений для обработки изображений, которые позже будут известны как плагины – “Plug-ins”. Затем были разработаны: настройки тонов (Levels), функция управления балансом, управление оттенками и насыщением (Hue and Saturation). Эти возможности стали ключевыми в Photoshop, ведь в то время такие функции присутствовали лишь в специализированном программном обеспечении, которое находилось в лабораториях – или в ILM.

В 1988 «Display» был переименован в «ImagePro». Программа была современной на то время, Джон надеялся, что у них есть шанс продать её в виде коммерческого приложения. Томас был против этого, кроме того, он ещё не закончил свою диссертацию и для разработки полностью завершённого программного продукта потребуется много времени и труда. После проведения анализа конкурентов братья поняли, что созданный ими ImagePro по многим параметрам опережал аналоги.

**От ImagePro к Photoshop**

Начались поиски инвесторов. Никто точно не знает, откуда произошло название «Photoshop», говорят, имя было предложено одним из потенциальных издателей, во время демонстрации программного обеспечения – название, как говорится, прилипло. В самых ранних версиях, на экранной заставке красовалось название “PhotoShop” – в наше время наблюдается такая же тенденция, очень часто встречаются названия на подобии «ExTraneous CapitaliSation».

На удивление, большинство производителей программного обеспечения повернули свои корпоративные носы в направлении Photoshop или пытались разработать подобные программы своими силами. Всего лишь одной компании было по силам купить Photoshop – это была Adobe, но до взаимовыгодного решения было ещё далеко. Производитель сканеров Barneyscan предложил братьям поставлять **Photoshop** в комплекте со своей продукцией, в итоге чего около 200 копий программы разошлось под торговой маркой Barneyscan XP.

К счастью для будущего цифровой графики, переговоры с Adobe длились не долго — вскоре Джон пришёл в компанию, чтобы привлечь больше внимания к своему продукту. Там он встретился с Расселом Брауном и продемонстрировал ему возможности программы, затем **Photoshop** был продемонстрирован арт-директору, который остался доволен увиденным и настоял на немедленной покупке программного обеспечения. То ли по наивности со стороны Adobe, то ли по осторожности со стороны братьев, Photoshop не был продан полностью, а только лицензирован и гонорары от продаж программы принадлежали братьям.

Всё шло превосходно, но это не означало, что братья могут расслабиться и остановиться на достигнутом. Теперь им предстояла работа куда тяжелее – нужно подготовить **Photoshop** для выпуска официальной версии – версии 1.0

Томас продолжал работу над исходным кодом программы. В это время Джон отдельно вёл разработку расширений (Plug-ins), что сильно волновало некоторых сотрудников Adobe — они видели во всём этом хитрость и обман.

Странности на этом не закончились. Некоторые сотрудники – сторонники строгого соблюдения правил, заявляли, что расширения для Photoshop тем или иным образом не соответствуют стандартам и не будут восприняты положительно, в то время как другие клялись в том, что расширения помогут сделать программу гибкой, настраиваемой и более производительной, если использовать их правильно.

Даже во время окончательного этапа работы над финальной версией вносились изменения и усовершенствования – любым способом Томас находил время и силы, чтобы делать это. При поддержке Джона и Рассела Брауна – который вскоре стал ярым сторонником Photoshop – как и остальных креативных решений в Adobe, программа постепенно стала обретать форму. Официальная версия **Adobe Photoshop**была выпущена в феврале 1990 года.

**Цифровая графика для всех**

Безусловно, выход первой версии стал успехом, несмотря даже на большое количество ошибок. Как и у сегодняшнего Apple, ключевым маркетинговым шагом Adobe было показать, что **Photoshop** — программа для широкого рынка пользователей – простой инструмент в использовании, которым сможет пользоваться каждый в отличие от большинства графических программных продуктов того времени, которые были разработаны для специалистов.

Adobe обещала, что при помощи **Photoshop** вы сможете достичь таких же результатов, которых раньше можно было получить с помощью профессионального программного обеспечения стоимостью несколько тысяч долларов. Главным конкурентом **Photoshop** на то время была программа под названием **Letraset’s ColorStudio**стоимостью $1,995, в то время как **Photoshop** можно было приобрести менее чем за $1,000. В процессе разработки версии 2.0, Adobe расширил штат программистов. Для внедрения в программу кривых Безье в команду был приглашён Марк Гамбург. В это время для **Photoshop** были разработаны новые возможности: перо (Pen tool), работа с двухтоновыми изображениями, импорт и растеризация векторных изображений из **Illustrator**, вдобавок ко всему появилась поддержка CMYK. Теперь для **Photoshop** был открыт путь на профессиональный рынок полиграфических услуг, что является очень важным тактическим ходом Adobe. Первый менеджер по продажам, Стивен Гутман, предложил давать кодовые названия к бета-версиям программы — эта традиция соблюдается по сегодняшний день. ‘Fast Eddy’ – кодовое название бета-версии 2.0, которая была запущена в 1990 году.

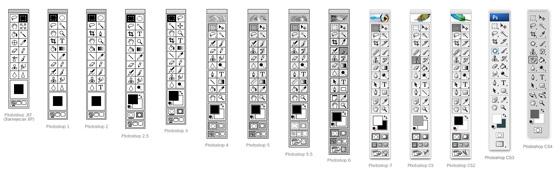
Продолжительное время **Photoshop** работал исключительно на платформе Макинтош, но его успех гарантировал появление версии для развивающегося на то время рынка графических приложений для Windows. Разработка версии для новой платформы являлась заданием не из лёгких – для этого было основано новое подразделение во главе с Брайаном Лемпкином. Не смотря на то, что было разработано большое количество нововведений, как, например, поддержка 16-битных файлов, это появилось лишь в версии 2.5.1 в 1993 году.

Так же как издание третьего альбома для музыкальной группы – разработка версии 3.0 могла принести успех или потерпеть полное фиаско. К счастью удача была на стороне команды разработчиков – был сделан огромный шаг в развитии программы — разработана поддержка слоёв (Layers).

Поддержка слоёв получила всеобщее одобрение и являлась одной из основных функций программы, которая в скором будущем склонила множество художников и дизайнеров на свою сторону. Всё же поддержка слоёв на то время уже не была уникальной. HSC – позже известная как MetaCreations – разрабатывала программу для обработки изображений под названием Live Picture, которая поддерживала такую же технологию. MetaCreations допустили единственную и роковую ошибку – достойная по своим возможностям Live Picture, была выпущена по очень завышенной цене, что и оставило **Photoshop** 3.0 вне конкуренции на рынке.

В последующих версиях таких глобальных изменений, как например поддержка слоёв, не было. Несмотря на это совершенствование программы продолжалось. В версии 5.0 были введены возможности управления цветом и панель истории (History Palette), с помощью которой можно было влиять на произведённые пользователем действия – что позволило расширить творческие возможности программы. Значительные изменения произошли вместе с выходом версии 5.5 (именно с этой версии началось моё знакомство с **Photoshop**) – в комплекте с **Photoshop** поставлялась ещё одна программа – **ImageReady**, которая добавила к функционалу ещё и богатые средства разработки графики для WEB. В следующей версии – 6.0 появились стили слоя (Layer Styles) и расширенные возможности управления текстом. Восстанавливающая кисть (Healing Brush) появилась в версии 7.0.

**Интерфейсы Photoshop в хронологическом порядке**

Поддерживая позиции лидера на рынке столь продолжительное время, на удивление, Adobe продолжает совершенствовать программу. На сегодняшний день **Photoshop** входит в состав пакетов **Creative Suite 2** (CS2) и **Creative Suite 3** (CS3), **Creative Suite 4** (CS4), вдобавок ко всему Adobe подчеркнула функциональную совместимость программ входящих в пакеты при помощи **Adobe Bridge**.

К счастью, разработка программы не прекращается и скорее всего не прекратится. Во-первых, из-за высокой конкуренции огромного количества производителей программного обеспечения, многие из которых предлагают функциональные возможности **Photoshop** бесплатно. Также конкуренцию составляют более дешёвые программы-аналоги, которые предназначены для любительского и домашнего использования, к примеру, **Paint Shop Pro**, который ориентировался на **Photoshop** многие годы. В ответ Adobe выпустили бюджетную версию программы, хоть и с урезанными возможностями, но всё же достаточно функциональную — **Photoshop Elements** – последняя версия 4.0.

Что же будет дальше? К сожалению, Adobe не анонсирует. **Photoshop** – это драгоценный камень в короне Adobe и аспекты его разработки тщательно скрываются, но изредка компания всё-таки делает намёки. Брайан Лемкин, занимающий пост старшего вице-президента отдела по разработке цифровых изображений и видео, подтвердил предположения о выпуске 64-битной версии приложения (выпущен **Adobe Photoshop CS4 х64**), а также возможную поддержку технологии **Apple’s CoreImage**, которая существенно увеличит производительность программы. Также разносятся слухи о возможном объединении **Illustrator** и **Photoshop** в единое целое приложение. Что бы ни произошло, Томас Нолл продолжит принимать участие в проекте. Хоть он непосредственно разработкой Photoshop не занимается, но постоянно держит руку на пульсе событий, продолжая разрабатывать новые версии плагинов **Adobe Camera Raw** и участвуя в дискуссиях на форумах компании Adobe.

Описание по порядку

**doctype html**

Элемент **<!DOCTYPE>** предназначен для указания типа текущего документа — DTD (document type definition, описание типа документа). Это необходимо, чтобы браузер понимал, как следует интерпретировать текущую веб-страницу, поскольку HTML существует в нескольких версиях, кроме того, имеется XHTML (EXtensible HyperText Markup Language, расширенный язык разметки гипертекста), похожий на HTML, но различающийся с ним по синтаксису. Чтобы браузер «не путался» и понимал, согласно какому стандарту отображать веб-страницу и необходимо в первой строке кода задавать **<!DOCTYPE>**.

**Html lang**

Текст документа может быть набран как на одном языке, так и содержать вставки на других языках, которые могут различаться по своим правилам оформления текста. Например, для русского, немецкого и английского языка характерны разные кавычки, в которые берется цитата. Чтобы указать язык, на котором написан текст внутри текущего элемента и применяется атрибут lang. Браузер использует его значение для правильного отображения некоторых символов.

**head**

Тег **<head>** предназначен для хранения других элементов, цель которых — помочь браузеру в работе с данными. Также внутри контейнера **<head>** находятся метатеги, которые используются для хранения информации предназначенной для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных.

Содержимое тега **<head>** не отображается напрямую на веб-странице, за исключением тега **<title>** устанавливающего заголовок окна веб-страницы.

Внутри контейнера **<head>** допускается размещать следующие элементы: **<base>**, **<basefont>**, **<bgsound>**, **<link>**, **<meta>**, **<script>**, **<style>**, **<title>**.

**title**

Определяет заголовок документа. Элемент **<title>** не является частью документа и не показывается напрямую на веб-странице. В операционной системе Windows текст заголовка отображается в левом верхнем углу окна браузера (рис. 1). Допускается использовать только один тег **<title>** на документ и размещать его в контейнере **<head>**.

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

Указывает кодировку документа. Атрибут введен в HTML5 и предназначен для сокращения формы тега **<meta>**, которая задавала кодировку в предыдущих версиях HTML и XHTML.

**Синтаксис**

|  |  |
| --- | --- |
| HTML | **<meta** charset="кодировка"**>** |
| XHTML | **<meta** charset="кодировка" **/>** |

**Значения**

Название кодировки, например UTF-8.

**<meta name="viewport">**

Так уж сложилось, что мобильные браузеры появились сравнительно недавно, а сайтов в сети к тому времени уже было огромное количество. Естественно, все эти сайты совсем не были оптимизированы под маленькие экраны смартфонов, а смартфоны в свою очередь, вынуждены были полагать, что все сайты в ширину занимают где-то около 1000px (980px в сафари). Нужно было как-то решать сложившуюся ситуацию и в apple придумали метатег <meta name="viewport">, который по традиции потом сперли все остальные производители браузеров

**Link**

Устанавливает связь с внешним документом вроде файла со стилями или со шрифтами. В отличие от тега **<a>**, тег **<link>** размещается всегда внутри контейнера **<head>** и не создает ссылку.

**Синтаксис**

|  |  |
| --- | --- |
| HTML | **<head>**  **<link атрибуты>**  **</head>** |
| XHTML | **<head>**  **<link атрибуты />**  **</head>** |

**Атрибуты**

[charset](http://htmlbook.ru/html/link/charset)

Кодировка связываемого документа.

[href](http://htmlbook.ru/html/link/href)

Путь к связываемому файлу.

[media](http://htmlbook.ru/html/link/media)

Определяет устройство, для которого следует применять стилевое оформление.

[rel](http://htmlbook.ru/html/link/rel)

Определяет отношения между текущим документом и файлом, на который делается ссылка.

[sizes](http://htmlbook.ru/html/link/sizes)

Указывает размер иконок для визуального отображения.

[type](http://htmlbook.ru/html/link/type)

MIME-тип данных подключаемого файла.

Body

Элемент **<body>** предназначен для хранения содержания веб-страницы (контента), отображаемого в окне браузера. Информацию, которую следует выводить в документе, следует располагать именно внутри контейнера **<body>**. К такой информации относится текст, изображения, теги, скрипты JavaScript и т.д.

Тег **<body>** также применяется для определения цветов ссылок и текста на веб-странице. Подобная практика в HTML 4 осуждается и взамен для указания цветовой схемы рекомендуется использовать стили, применяя их к селектору **BODY**. Тем не менее, большинство атрибутов до сих пор поддерживается разными браузерами.

Часто тег **<body>** используется для размещения обработчика событий, например, [onload](http://htmlbook.ru/html/attr/onload), которое выполняется после того, как документ завершил загрузку в текущее окно или фрейм.

Открывающий и закрывающий теги **<body>** на веб-странице не являются обязательными, однако хорошим стилем считается их использование, чтобы определить начало и конец HTML-документа.

**Синтаксис**

**<body>**

...

**</body>**

**Атрибуты**

[alink](http://htmlbook.ru/html/body/alink)

Устанавливает цвет активной ссылки.

[background](http://htmlbook.ru/html/body/background)

Задает фоновый рисунок на веб-странице.

[bgcolor](http://htmlbook.ru/html/body/bgcolor)

Цвет фона веб-страницы.

[bgproperties](http://htmlbook.ru/html/body/bgproperties)

Определяет, прокручивать фон совместно с текстом или нет.

[bottommargin](http://htmlbook.ru/html/body/bottommargin)

Отступ от нижнего края окна браузера до контента.

[leftmargin](http://htmlbook.ru/html/body/leftmargin)

Отступ по горизонтали от левого края окна браузера до контента.

[link](http://htmlbook.ru/html/body/link)

Цвет ссылок на веб-странице.

Отступ от правого края окна браузера до контента.

Устанавливает, отображать полосы прокрутки или нет.

Цвет текста в документе.

Отступ от верхнего края окна браузера до контента.

Цвет посещенных ссылок. Также для этого тега доступны [универсальные атрибуты](http://htmlbook.ru/html/attr/common) и [события](http://htmlbook.ru/html/attr/event).

**Закрывающий тег**

Открывающий и закрывающий теги не обязательны.

**nav**

Тег **<nav>** задает навигацию по сайту. Если на странице несколько блоков ссылок, то в **<nav>** обычно помещают приоритетные ссылки. Также допустимо использовать несколько тегов **<nav>** в документе. Запрещается вкладывать **<nav>** внутрь **<address>**.

**div**

Элемент **<div>**является блочным элементом и предназначен для выделения фрагмента документа с целью изменения вида содержимого. Как правило, вид блока управляется с помощью стилей. Чтобы не описывать каждый раз стиль внутри тега, можно выделить стиль во внешнюю таблицу стилей, а для тега добавить атрибут class или id с именем селектора.

Как и при использовании других блочных элементов, содержимое тега **<div>**всегда начинается с новой строки. После него также добавляется перенос строки.

**A href**

Тег **<a>** является одним из важных элементов HTML и предназначен для создания ссылок. В зависимости от присутствия атрибутов name или href тег **<a>** устанавливает ссылку или якорь. Якорем называется закладка внутри страницы, которую можно указать в качестве цели ссылки. При использовании ссылки, которая указывает на якорь, происходит переход к закладке внутри веб-страницы.

Для создания ссылки необходимо сообщить браузеру, что является ссылкой, а также указать адрес документа, на который следует сделать ссылку. В качестве значения атрибута href используется адрес документа (URL, Universal Resource Locator, универсальный указатель ресурсов), на который происходит переход. Адрес ссылки может быть абсолютным и относительным. Абсолютные адреса работают везде и всюду независимо от имени сайта или веб-страницы, где прописана ссылка. Относительные ссылки, как следует из их названия, построены относительно текущего документа или корня сайта.

I

Устанавливает курсивное начертание шрифта. Допустимо использовать этот тег совместно с другими тегами, которые определяют начертание текста.

**span**

Тег **<span>** предназначен для определения строчных элементов документа. В отличие от блочных элементов, таких как **<table>**, **<p>** или **<div>**, с помощью тега **<span>** можно выделить часть информации внутри других тегов и установить для нее свой стиль. Например, внутри абзаца (тега **<p>**) можно изменить цвет и размер первой буквы, если добавить начальный и конечный тег **<span>** и определить для него стиль текста. Чтобы не описывать каждый раз стиль внутри тега, можно выделить стиль во внешнюю таблицу стилей, а для тега добавить атрибут class или id с именем селектора.

**button**

Тег **<button>** создает на веб-странице кнопки и по своему действию напоминает результат, получаемый с помощью тега **<input>** (с атрибутом type="button | reset | submit"). В отличие от этого тега, **<button>** предлагает расширенные возможности по созданию кнопок. Например, на подобной кнопке можно размещать любые элементы HTML, в том числе изображения. Используя стили можно определить вид кнопки путем изменения шрифта, цвета фона, размеров и других параметров.

Теоретически, тег **<button>** должен располагаться внутри формы, устанавливаемой элементом **<form>**. Тем не менее, браузеры не выводят сообщение об ошибке и корректно работают с тегом **<button>**, если он встречается самостоятельно. Однако, если необходимо результат нажатия на кнопку отправить на сервер, помещать **<button>** в контейнер **<form>** обязательно.

**div**

Элемент **<div>**является блочным элементом и предназначен для выделения фрагмента документа с целью изменения вида содержимого. Как правило, вид блока управляется с помощью стилей. Чтобы не описывать каждый раз стиль внутри тега, можно выделить стиль во внешнюю таблицу стилей, а для тега добавить атрибут class или id с именем селектора.

Как и при использовании других блочных элементов, содержимое тега **<div>**всегда начинается с новой строки. После него также добавляется перенос строки.

**ul li**

Тег **<ul>** устанавливает маркированный список. Каждый элемент списка должен начинаться с тега **<li>**. Если к тегу **<ul>** применяется таблица стилей, то элементы **<li>** наследуют эти свойства.

h1

HTML предлагает шесть заголовков разного уровня, которые показывают относительную важность секции, расположенной после заголовка. Так, тег **<h1>** представляет собой наиболее важный заголовок первого уровня, а тег **<h6>** служит для обозначения заголовка шестого уровня и является наименее значительным. По умолчанию, заголовок первого уровня отображается самым крупным шрифтом жирного начертания, заголовки последующего уровня по размеру меньше. Теги **<h1>**,...,**<h6>** относятся к блочным элементам, они всегда начинаются с новой строки, а после них другие элементы отображаются на следующей строке. Кроме того, перед заголовком и после него добавляется пустое пространство.

**P**

Определяет текстовый абзац. Тег **<p>** является блочным элементом, всегда начинается с новой строки, абзацы текста идущие друг за другом разделяются между собой отбивкой. Величиной отбивки можно управлять с помощью стилей. Если закрывающего тега нет, считается, что конец абзаца совпадает с началом следующего блочного элемента.

**Section**

Задаёт раздел документа, может применяться для блока новостей, контактной информации, глав текста, вкладок в диалоговом окне и др. Раздел обычно содержит заголовок. Допускается вкладывать один тег **<section>** внутрь другого.

**Blockquote**

Тег **<blockquote>** предназначен для выделения длинных цитат внутри документа. Текст, обозначенный этим тегом, традиционно отображается как выровненный блок с отступами слева и справа (примерно по 40 пикселов), а также с отбивкой сверху и снизу.

**Strong**

Тег **<strong>** предназначен для акцентирования текста. Браузеры отображают такой текст жирным начертанием.

**Form**

Тег **<form>** устанавливает форму на веб-странице. Форма предназначена для обмена данными между пользователем и сервером. Область применения форм не ограничена отправкой данных на сервер, с помощью клиентских скриптов можно получить доступ к любому элементу формы, изменять его и применять по своему усмотрению.

Документ может содержать любое количество форм, но одновременно на сервер может быть отправлена только одна форма. По этой причине данные форм должны быть независимы друг от друга.

Для отправки формы на сервер используется кнопка Submit, того же можно добиться, если нажать клавишу Enter в пределах формы. Если кнопка Submit отсутствует в форме, клавиша Enter имитирует ее использование.

Когда форма отправляется на сервер, управление данными передается программе, заданной атрибутом action тега **<form>**. Предварительно браузер подготавливает информацию в виде пары «имя=значение», где имя определяется атрибутом name тега **<input>**, а значение введено пользователем или установлено в поле формы по умолчанию. Если для отправки данных используется метод GET, то адресная строка может принимать следующий вид.

**Input**

Тег **<input>** является одним из разносторонних элементов формы и позволяет создавать разные элементы интерфейса и обеспечить взаимодействие с пользователем. Главным образом **<input>** предназначен для создания текстовых полей, различных кнопок, переключателей и флажков. Хотя элемент **<input>** не требуется помещать внутрь контейнера **<form>**, определяющего форму, но если введенные пользователем данные должны быть отправлены на сервер, где их обрабатывает серверная программа, то указывать **<form>** обязательно. То же самое обстоит и в случае обработки данных с помощью клиентских приложений, например, скриптов на языке JavaScript.

Основной атрибут тега **<input>**, определяющий вид элемента — type. Он позволяет задавать следующие элементы формы: текстовое поле (text), поле с паролем (password), переключатель (radio), флажок (checkbox), скрытое поле (hidden), кнопка (button), кнопка для отправки формы (submit), кнопка для очистки формы (reset), поле для отправки файла (file) и кнопка с изображением (image). Для каждого элемента существует свой список атрибутов, которые определяют его вид и характеристики. Кроме того, в HTML5 добавлено еще более десятка новых элементов.

**Footer**

Тег **<footer>** задаёт «подвал» сайта или раздела, в нём может располагаться имя автора, дата документа, контактная и правовая информация.

**Label**

Тег **<label>** устанавливает связь между определенной меткой, в качестве которой обычно выступает текст, и элементом формы (**<input>**, **<select>**, **<textarea>**). Такая связь необходима, чтобы изменять значения элементов формы при нажатии курсором мыши на текст. Кроме того, с помощью **<label>** можно устанавливать горячие клавиши на клавиатуре и переходить на активный элемент подобно ссылкам.

Существует два способа связывания объекта и метки. Первый заключается в использовании идентификатора id внутри элемента формы и указании его имени в качестве атрибута for тега **<label>**. При втором способе элемент формы помещается внутрь контейнера **<label>**.

**Script**

Тег **<script>** предназначен для описания скриптов, может содержать ссылку на программу или ее текст на определенном языке. Скрипты могут располагаться во внешнем файле и связываться с любым HTML-документом. Такой подход позволяет использовать одни и те же общие функции на многих веб-страницах и ускоряет их загрузку, т.к. внешний файл кэшируется при первой загрузке, и скрипт вызывается быстрее при последующих вызовах.

**<script>** может располагаться в заголовке или теле HTML-документа в неограниченном количестве. В большинстве случаев местоположение скрипта никак не сказывается на работу программы. Однако скрипты, которые должны выполняться в первую очередь, обычно помещают в заголовок документа**.**

**Style css**

**:root**

В этой статье мы с вами узнаем, что такое псевдокласс *:root* и разберемся в том, как можно его использовать в своих проектах.

CSS селектор псевдокласса *:root* используется для выбора родителя самого высокого уровня. Например в HTML *:root* по сути эквивалентен тегу html.  
В приведенном ниже фрагменте CSS стили *:root* и *html* делают одно и то же:

:root {

background-color: gray;

}

html {

background-color: gray;

}

Если вы обратили внимание, я сказал, что *:root* **по сути эквивалентен** тегу *html*. На самом деле *:root* имеет больший “вес” чем тег *html*. Так получается, потому что он считается селектором псевдокласса (например как *:first-child* или *:hover*) и соответственно обладает большей специфичностью (весомостью) чем просто селектор тега.

:root {

background-color: blue;

color: white;

}

html {

background-color: red;

color: white;

}

Несмотря на то, что стили для тега *html* были заданы позднее, будут применены стили заданные для *:root* из-за его более высокой специфичности.

Поскольку CSS разработан также для SVG и XML, вы можете использовать *:root* и там, только он будет соответствовать другому элементу. Например в SVG он будет эквивалентен тегу svg.

:root {

fill: gold;

}

svg {

fill: gold;

}

Как и в HTML, селекторы *:root* и *svg* выбирают один и тот же элемент, однако селектор *:root* будет иметь более высокую специфичность.

## **Практическое применение**

Как же использовать *:root* на практике? Как мы уже говорили ранее - это безопасная замена для тега *html* и вы можете работать с *:root* как с обычным тегом *html*.

:root {

margin: 0;

padding: 0;

color: #0000FF;

font-family: "Helvetica", "Arial", sans-serif;

line-height: 1.5;

}

Если вы хотите, то можете изменить этот код чтобы использовать кастомные CSS-свойства для создания переменных на глобальном уровне!

:root {

margin: 0;

padding: 0;

--primary-color: #0000FF;

--body-fonts: "Helvetica", "Arial", sans-serif;

--line-height: 1.5;

}

p {

color: var(--primary-color);

font-family: var(--body-fonts);

line-height: var(--line-height);

}

Дополнительным преимуществом использования *:root* вместо *html* является то, что вы можете стилизовать вашу SVG графику!

:root {

margin: 0;

padding: 0;

--primary-color: #0000FF;

--body-fonts: "Helvetica", "Arial", sans-serif;

--line-height: 1.5;

}

svg {

font-family: var(--body-fonts);

}

svg circle {

fill: var(--primary-color);

}

**::before**

Псевдоэлемент **::before** добавляет определённое содержимое перед каждым указанным элементом. Псевдоэлемент **::before** используется вместе со свойством **[content](https://basicweb.ru/css/css_pr_content.php)**, которое необходимо для вставки сгенерированного контента.

Обратите внимание, что в настоящее время используется синтаксис, который предусматривает двойное двоеточие перед псевдоэлементом. Не смотря на то, что браузеры поддерживают оба варианта, рекомендуется использовать синтаксис CSS 3:

**/\* синтаксис CSS 3 \*/**

**селектор::псевдоэлемент** **{** **/\* двойное двоеточие \*/**

**CSS свойство**: **значение**;

**}**

**/\* синтаксис CSS 2 \*/**

**селектор:псевдоэлемент** **{** **/\* одинарное двоеточие \*/**

**CSS свойство**: **значение**;

**}**

**::after**

Псевдоэлемент, который используется для вывода желаемого текста после содержимого элемента, к которому он добавляется. Псевдоэлемент :after работает совместно со свойством [content](http://htmlbook.ru/css/content).

Для :after характерны следующие особенности.

При добавлении :after к блочному элементу, значение свойства display может быть только: block, inline, none, list-item. Все остальные значения будут трактоваться как block.

При добавлении :after к встроенному элементу, display ограничен значениями inline и none. Все остальные будут восприниматься как inline.

**breakpoint-xs**

**Breakpoints** (контрольная точка) **Breakpoints** (переломные/контрольные точки) — изменение условий с одного на другое, при котором внешний вид сайта (раскладка) изменяется. Часто встречается в адаптивной вёрстке с использованием медиавыражений.

**font-family-sans-serif**

[Игнорирует мой шрифт в CSS и берет шрифт](https://ru.stackoverflow.com/questions/1201065/%D0%98%D0%B3%D0%BD%D0%BE%D1%80%D0%B8%D1%80%D1%83%D0%B5%D1%82-%D0%BC%D0%BE%D0%B9-%D1%88%D1%80%D0%B8%D1%84%D1%82-%D0%B2-css-%D0%B8-%D0%B1%D0%B5%D1%80%D0%B5%D1%82-%D1%88%D1%80%D0%B8%D1%84%D1%82-bootstapp)

**Width**

Устанавливает ширину блочных или заменяемых элементов (к ним, например, относится тег **<img>**). Ширина не включает толщину границ вокруг элемента, значение отступов и полей.

**Display**

Многоцелевое свойство, которое определяет, как элемент должен быть показан в документе

**Margin**

Устанавливает величину отступа от каждого края элемента. Отступом является пространство от границы текущего элемента до внутренней границы его родительского элемента

**font-size**

Свойство **font-size** определяет размер шрифта. Это свойство также используется для вычисления размера em, ex и других относительных единиц

**font-weight**

Устанавливает насыщенность шрифта. Значение устанавливается от 100 до 900 с шагом 100. Сверхсветлое начертание, которое может отобразить браузер, имеет значение 100, а сверхжирное — 900. Нормальное начертание шрифта (которое установлено по умолчанию) эквивалентно 400, стандартный полужирный текст — значению 700.

**line-height**

Устанавливает интерлиньяж (межстрочный интервал) текста, отсчет ведется от базовой линии шрифта. При обычных обстоятельствах расстояние между строками зависит от вида и размера шрифта и определяется браузером автоматически. Отрицательное значение межстрочного расстояния не допускается.

**Color**

Определяет цвет текста элемента.

**text-align**

Определяет горизонтальное выравнивание текста в пределах элемента.

**background-color**

Определяет цвет фона элемента. Хотя это свойство не наследует свойства своего родителя, из-за того, что начальное значение устанавливается прозрачным, цвет фона дочерних элементов совпадает с цветом фона родительского элемента.

**Outline**

Универсальное свойство, одновременно устанавливающее цвет, стиль и толщину внешней границы на всех четырех сторонах элемента. В отличие от линии, задаваемой через border, свойство outline не влияет на положение блока и его ширину. Также нельзя задать параметры линии на отдельных сторонах элемента, outline применяется сразу ко всем четырём сторонам.

**Height**

Устанавливает высоту блочных или заменяемых элементов (к ним, например, относится тег **<img>**). Высота не включает толщину границ вокруг элемента, значение отступов и полей.

Если содержимое блока превышает указанную высоту, то высота элемента останется неизменной, а содержимое будет отображаться поверх него. Из-за этой особенности может получиться наложение содержимого элементов друг на друга, когда элементы в коде HTML идут последовательно. Чтобы этого не произошло, добавьте overflow: auto к стилю элемента.

**Overflow**

Свойство overflow управляет отображением содержания блочного элемента, если оно целиком не помещается и выходит за область заданных размеров.

**margin-top**

Устанавливает величину отступа от верхнего края элемента. Отступом является расстояние от внешнего края верхней границы текущего элемента до внутренней границы его родительского элемента

**text-decoration**

Добавляет оформление текста в виде его подчеркивания, перечеркивания, линии над текстом и мигания. Одновременно можно применить более одного стиля, перечисляя значения через пробел.

### Синтаксис

text-decoration: [ blink || line-through || overline || underline ] | none | inherit

### Значения

blink

Устанавливает мигающий текст. Такой текст периодически, примерно раз в секунду исчезает, потом вновь появляется на прежнем месте. Это значение в настоящее время не поддерживается браузерами и осуждается в CSS3, взамен рекомендуется использовать анимацию.

line-through

Создает перечеркнутый текст (~~пример~~).

overline

Линия проходит над текстом (пример).

underline

Устанавливает подчеркнутый текст (пример).

none

Отменяет все эффекты, в том числе и подчеркивания у ссылок, которое задано по умолчанию.

inherit

Значение наследуется у родителя.

**Cursor**

Устанавливает форму курсора, когда он находится в пределах элемента. Вид курсора зависит от операционной системы и установленных параметров.

Прежде чем воспользоваться возможностью переделать вид курсора, решите, а будет ли он использоваться к месту. Многих пользователей подобные изменения могут ввести в заблуждение, когда, например, вместо традиционной руки, появляющейся при наведении на ссылку, возникает нечто другое. В большинстве случаев, лучше оставить все как есть.

**border-bottom**

Свойство позволяет одновременно установить толщину, стиль и цвет границы внизу элемента. Значения могут идти в любом порядке, разделяясь пробелом, браузер сам определит, какое из них соответствует нужному свойству.

### Синтаксис

border-bottom: [[border-width](http://htmlbook.ru/css/border-bottom-width) || [border-style](http://htmlbook.ru/css/border-bottom-style) || [border-color](http://htmlbook.ru/css/border-bottom-color)] | inherit

### Значения

Значение border-width определяет толщину границы. Для управления ее видом предоставляется несколько значений свойства border-style. Их названия и результат действия представлен на рис. 1.

Стили рамок

border-color устанавливает цвет границы, значение может быть в любом допустимом для CSS формате.

**Position**

Устанавливает способ позиционирования элемента относительно окна браузера или других объектов на веб-странице.

**Padding**

Устанавливает значение полей вокруг содержимого элемента. Полем называется расстояние от внутреннего края рамки элемента до воображаемого прямоугольника, ограничивающего его содержимое

**caption-side**

Определяет положение заголовка таблицы, который задается с помощью тега **<caption>**, относительно самой таблицы. Свойство caption-side выводит заголовок до или после таблицы, а выравнивание текста по правому или левому краю устанавливается через свойство [text-align](http://htmlbook.ru/css/text-align). Браузер Firefox также поддерживает расположение заголовка слева или справа от таблицы, но эти значения не входят в спецификацию CSS.

**text-transform**

Управляет преобразованием текста элемента в заглавные или прописные символы. Когда значение отлично от none, регистр исходного текста будет изменен.

**border-style**

Устанавливает стиль границы вокруг элемента. Допустимо задавать индивидуальные стили для разных сторон элемента.

### Синтаксис

border-style: [none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset] {1,4} | inherit

### Значения

Для управления видом границы предоставляется несколько значений свойства border-style. Вид зависит от используемого браузера и заданной толщины границы. В табл. 1 приведены названия стилей и получаемая рамка при разных значениях толщины — 1, 3, 5 и 7 пикселов.

**Resize**

Указывает, можно ли пользователю изменять размеры текстового поля.

### Синтаксис

resize: none | both | horizontal | vertical | inherit

### Значения

none

Размеры элемента не изменяются.

both

Можно изменять размеры элемента по горизонтали и вертикали.

horizontal

Можно изменять размеры элемента только по горизонтали.

vertical

Можно изменять размеры элемента только по вертикали.

inherit

Наследует значение родителя.

**Display**

Многоцелевое свойство, которое определяет, как элемент должен быть показан в документе.

### Синтаксис

display: block | inline | inline-block | inline-table | list-item | none | run-in | table | table-caption | table-cell | table-column-group | table-column | table-footer-group | table-header-group | table-row | table-row-group

### Значения

Список возможных значений этого свойства, понимаемый разными браузерами очень короткий — block, inline, list-item и none. Все остальные допустимые значения поддерживаются браузерами выборочно. В табл. 1 приведены некоторые популярные браузеры и их поддержка различных значений.

**line-height**

Устанавливает интерлиньяж (межстрочный интервал) текста, отсчет ведется от базовой линии шрифта. При обычных обстоятельствах расстояние между строками зависит от вида и размера шрифта и определяется браузером автоматически. Отрицательное значение межстрочного расстояния не допускается.

### Синтаксис

line-height: множитель | значение | проценты | normal | inherit

### Значения

Любое число больше нуля воспринимается как множитель от размера шрифта текущего текста. Например, значение 1.5 устанавливает полуторный межстрочный интервал. В качестве значений принимаются также любые единицы длины, принятые в CSS — пикселы (px), дюймы (in), пункты (pt) и др. Разрешается использовать процентную запись, в этом случае за 100% берется высота шрифта.

**white-space**

Свойство white-space устанавливает, как отображать пробелы между словами. В обычных условиях любое количество пробелов в коде HTML показывается на веб-странице как один. Исключением является тег **<pre>**, помещенный в этот контейнер текст выводится со всеми пробелами, как он был отформатирован пользователем. Таким образом, white-space имитирует работу тега **<pre>**, но в отличие от него не меняет шрифт на моноширинный.

### Синтаксис

white-space: normal | nowrap | pre | pre-line | pre-wrap | inherit

### Значения

normal

Текст в окне браузера выводится как обычно, переносы строк устанавливаются автоматически.

nowrap

Пробелы не учитываются, переносы строк в коде HTML игнорируются, весь текст отображается одной строкой; вместе с тем, добавление тега **<br>** переносит текст на новую строку.

pre

Текст показывается с учетом всех пробелов и переносов, как они были добавлены разработчиком в коде HTML. Если строка получается слишком длинной и не помещается в окне браузера, то будет добавлена горизонтальная полоса прокрутки.

pre-line

В тексте пробелы не учитываются, текст автоматически переносится на следующую строку, если он не помещается в заданную область.

pre-wrap

В тексте сохраняются все пробелы и переносы, однако если строка по ширине не помещается в заданную область, то текст автоматически будет перенесен на следующую строку.

inherit

Наследует значение родителя.

**list-style**

Универсальное свойство, позволяющее одновременно задать стиль маркера, его положение, а также изображение, которое будет использоваться в качестве маркера. Для подробного ознакомления смотрите информацию о каждом свойстве [list-style-type](http://htmlbook.ru/css/list-style-type), [list-style-position](http://htmlbook.ru/css/list-style-position) и [list-style-image](http://htmlbook.ru/css/list-style-image) отдельно.

### Синтаксис

list-style: list-style-type || list-style-position || list-style-image | inherit

### Значения

Любые комбинации трех значений определяющих стиль маркеров, они разделяются между собой пробелом. Комбинации значений должны следовать в указанном порядке: вначале идет тип маркера, затем положение и картинка. Ни одно значение не является обязательным, поэтому неиспользуемые можно опустить.

Inherit

Наследует значение родителя.

**word-break**

Свойство word-break указывает, как делать перенос строк внутри слов, которые не помещаются по ширине в заданную область.

### Синтаксис

word-break: normal | break-all | keep-all

### Значения

normal

Используются правила переноса строк по умолчанию. Как правило, в этом случае строки не переносятся или переносятся в тех местах, где явно задан перенос (например, с помощью тега **<br>**).

break-all

Перенос строк добавляется автоматически, чтобы слово поместилось в заданную ширину блока. Значение не работает для текста на китайском, корейском или японском языке.

keep-all

Не разрешает перенос строк в словах на китайском, корейском или японском языке. Для остальных языков действует как normal.

**flex-basis**

[CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) свойство **flex-basis** задаёт базовый размер флекс элемента по **основной оси**. Это свойство определяет размер контент-бокса, если не задано иначе через [box-sizing](https://developer.mozilla.org/ru/docs/Web/CSS/box-sizing). **Внимание:** в случае, если для элемента одновременно заданы flex-basis (отличное от auto) и width ( или height в случае flex-direction: column), flex-basis имеет приоритет.

**flex-grow**

[CSS](https://developer.mozilla.org/en-US/docs/Web/CSS)-свойство **flex-grow** определяет как много свободного пространства во flex-контейнере должно быть назначено текущему элементу (flex grow factor — «коэффициент увеличения flex»). Свободное пространство — разница между размером flex-контейнера и размером всех его flex-элементов вместе. Если все sibling-элементы (sibling items — элементы одного уровня вложенности, состоящие по отношению друг к другу в родственной связи как брат или сестра), имеют одинаковый коэффициент flex-grow, то все они получат одинаковую долю свободного пространства, в противном случае оно распределяется в соответствии с соотношением, определённым различными коэффициентами flex-grow.

На практике flex-grow используется вместе с другими flex-свойствами [flex-shrink](https://developer.mozilla.org/ru/docs/Web/CSS/flex-shrink) и [flex-basis](https://developer.mozilla.org/ru/docs/Web/CSS/flex-basis), и обычно определяется с помощью сокращения (shorthand) [flex](https://developer.mozilla.org/ru/docs/Web/CSS/flex), чтобы убедиться, что все значения заданы.

**Order**

[CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) свойство  **order** определяет порядок, используемый для размещения flex элементов в их flex контейнере. Элементы располагаются в восходящем порядке по значению order. Элементы с одинаковым значением order  располагаются в том порядке, в каком они находятся в исходном коде. **Важно**: order воздействует только на **визуальный порядок** элементов, но не логический порядок или табуляцию. **order** не должен применяться к невизуальным медиаданным, таким как речь.

**flex-basis**

[CSS](https://developer.mozilla.org/en-US/docs/Web/CSS) свойство **flex-basis** задаёт базовый размер флекс элемента по **основной оси**. Это свойство определяет размер контент-бокса, если не задано иначе через [box-sizing](https://developer.mozilla.org/ru/docs/Web/CSS/box-sizing).

**Внимание:** в случае, если для элемента одновременно заданы flex-basis (отличное от auto) и width ( или height в случае flex-direction: column), flex-basis имеет приоритет.

**z-index**

Любые позиционированные элементы на веб-странице могут накладываться друг на друга в определенном порядке, имитируя тем самым третье измерение, перпендикулярное экрану. Каждый элемент может находиться как ниже, так и выше других объектов веб-страницы, их размещением по z-оси и управляет z-index. Это свойство работает только для элементов, у которых значение position задано как absolute, fixed или relative.

### Синтаксис

z-index: число | auto | inherit

### Значения

В качестве значения используются целые числа (положительные, отрицательные и ноль). Чем больше значение, тем выше находится элемент по сравнению с теми элементами, у которых оно меньше. При равном значении z-index, на переднем плане находится тот элемент, который в коде HTML описан ниже. Хотя спецификация и разрешает использовать отрицательные значения z-index, но такие элементы не отображаются в браузере Firefox до версии 2.0 включительно.

Кроме числовых значений применяется auto — порядок элементов в этом случае строится автоматически, исходя из их положения в коде HTML и принадлежности к родителю, поскольку дочерние элементы имеют тот же номер, что их родительский элемент. Значение inherit указывает, что оно наследуется у родителя.

**box-shadow**

Добавляет тень к элементу. Допускается использовать несколько теней, указывая их параметры через запятую, при наложении теней первая тень в списке будет выше, последняя ниже. Если для элемента задается радиус скругления через свойство border-radius, то тень также получится с закругленными уголками. Добавление тени увеличивает ширину элемента, поэтому возможно появление горизонтальной полосы прокрутки в браузере.

### Синтаксис

box-shadow: none | <тень> [,<тень>]\*  
где <тень>:  
inset <сдвиг по x> <сдвиг по y> <радиус размытия> <растяжение> <цвет>

### Значения

none

Отменяет добавление тени.

inset

Тень выводится внутри элемента. Необязательный параметр.

сдвиг по x

Смещение тени по горизонтали относительно элемента. Положительное значение этого параметра задает сдвиг тени вправо, отрицательное — влево. Обязательный параметр.

сдвиг по y

Смещение тени по вертикали относительно элемента. Положительное значение задает сдвиг тени вниз, отрицательное — вверх. Обязательный параметр.

размытие

Задает радиус размытия тени. Чем больше это значение, тем сильнее тень сглаживается, становится шире и светлее. Если этот параметр не задан, по умолчанию устанавливается равным 0, тень при этом будет четкой, а не размытой.

растяжение

Положительное значение растягивает тень, отрицательное, наоборот, ее сжимает. Если этот параметр не задан, по умолчанию устанавливается 0, при этом тень будет того же размера, что и элемент.

цвет

Цвет тени в любом доступном CSS формате, по умолчанию тень черная. Необязательный параметр.

Допускается указывать несколько теней, разделяя их параметры между собой запятой. Учитывается следующий порядок: первая тень в списке размещается на самом верху, последняя в списке — в самом низу.

**Top**

Для позиционированного элемента определяет расстояние от верхнего края родительского элемента до верхнего края дочернего элемента. Отсчет координат зависит от значения свойства position. Если оно равно absolute, в качестве родителя выступает окно браузера и положение элемента определяется от его верхнего края (рис. 1). В случае значения relative, top отсчитывается от верхнего края исходного положения элемента. Если для родительского элемента задано position: relative, то абсолютное позиционирование дочерних элементов определяет их положение от верхнего края родителя

**flex-flow**

CSS свойство flex-flow позволяет задать значения свойств [flex-direction](https://puzzleweb.ru/css/pr_flex-direction.php" \t "_blank) и [flex-wrap](https://puzzleweb.ru/css/pr_flex-wrap.php" \t "_blank) в одном месте. То есть свойство flex-flow контролирует сразу и направление расположения flex-элементов и перенос.

**Примечание:** если элемент не является flex-контейнером ([display](https://puzzleweb.ru/css/pr_display.php" \t "_blank): flex), свойство flex-flow будет проигнорировано.

**white-space**

Свойство white-space устанавливает, как отображать пробелы между словами. В обычных условиях любое количество пробелов в коде HTML показывается на веб-странице как один. Исключением является тег **<pre>**, помещенный в этот контейнер текст выводится со всеми пробелами, как он был отформатирован пользователем. Таким образом, white-space имитирует работу тега **<pre>**, но в отличие от него не меняет шрифт на моноширинный.

### Синтаксис

white-space: normal | nowrap | pre | pre-line | pre-wrap | inherit

### Значения

normal

Текст в окне браузера выводится как обычно, переносы строк устанавливаются автоматически.

nowrap

Пробелы не учитываются, переносы строк в коде HTML игнорируются, весь текст отображается одной строкой; вместе с тем, добавление тега **<br>** переносит текст на новую строку.

pre

Текст показывается с учетом всех пробелов и переносов, как они были добавлены разработчиком в коде HTML. Если строка получается слишком длинной и не помещается в окне браузера, то будет добавлена горизонтальная полоса прокрутки.

pre-line

В тексте пробелы не учитываются, текст автоматически переносится на следующую строку, если он не помещается в заданную область.

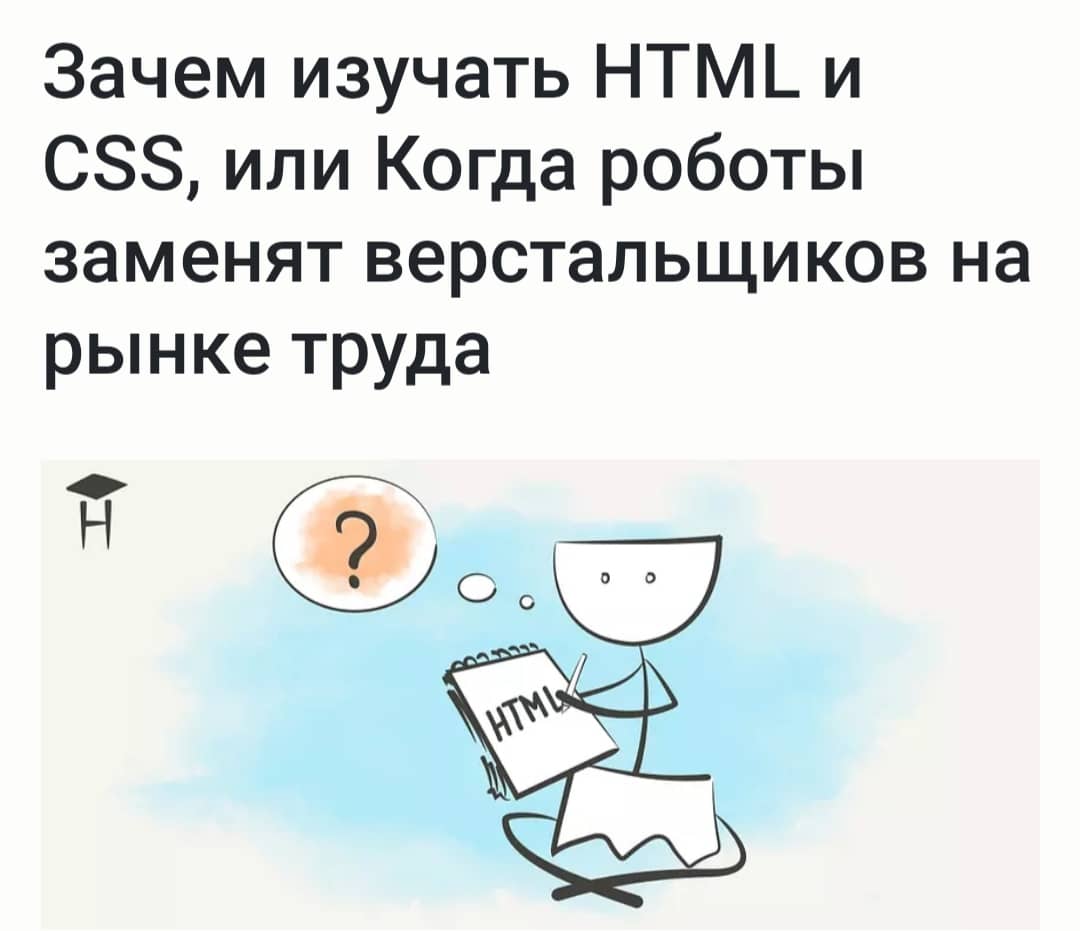
pre-wrap

В тексте сохраняются все пробелы и переносы, однако если строка по ширине не помещается в заданную область, то текст автоматически будет перенесен на следующую строку.

inherit

Наследует значение родителя.

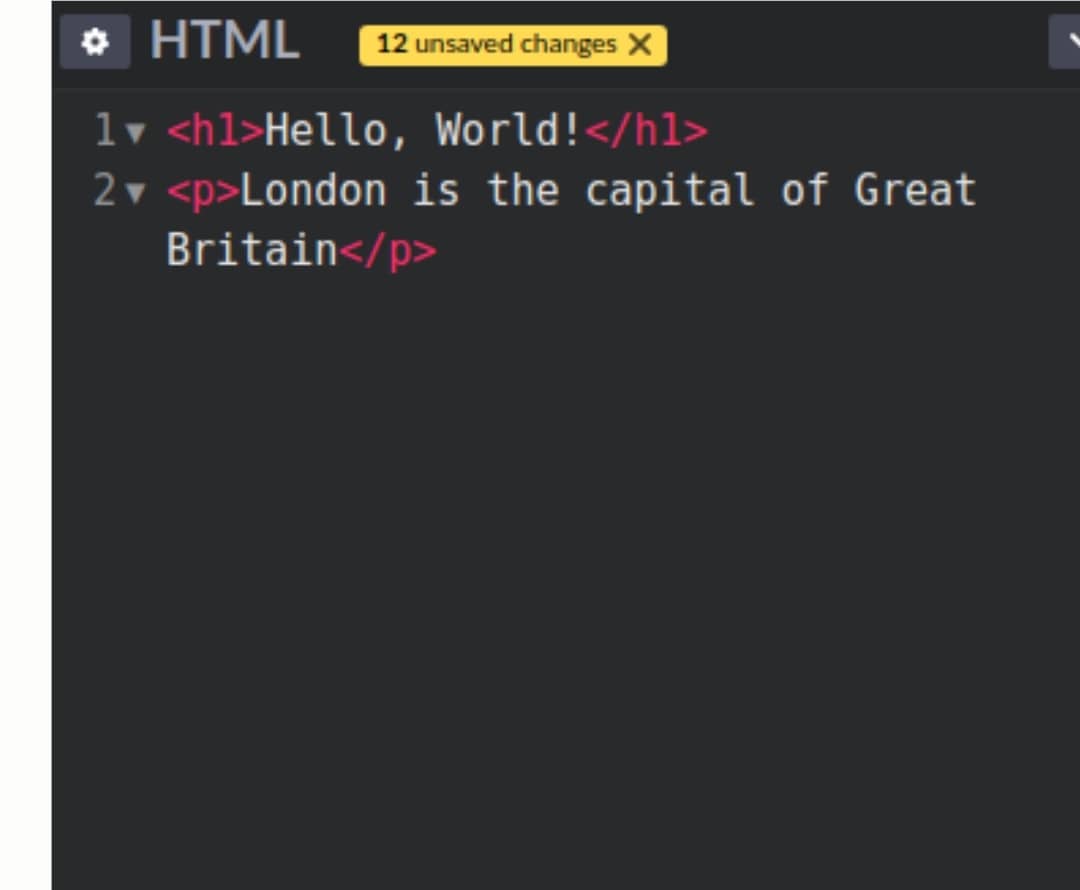
Заключение



Поговорим об актуальности изучения языков HTML и CSS. Узнаем о перспективах «чистых» верстальщиков на рынке труда, поймём, могут ли программисты обойтись без знания вёрстки. Посмотрим на наличие вакансий и уровень зарплат, поинтересуемся мнением экспертов. Поехали! Что такое HTML, CSS, вёрстка, кто такой верстальщик и как называют этого специалиста в англоязычной среде: краткая теория

HTML — язык гипертекстовой разметки. Аббревиатура образовалась от первых букв английских слов HyperText Markup Languge. HTML применяется для разметки веб-страниц. Она нужна браузерам, которые преобразуют гипертекст и выводят на экран страницу в удобном для человека формате.

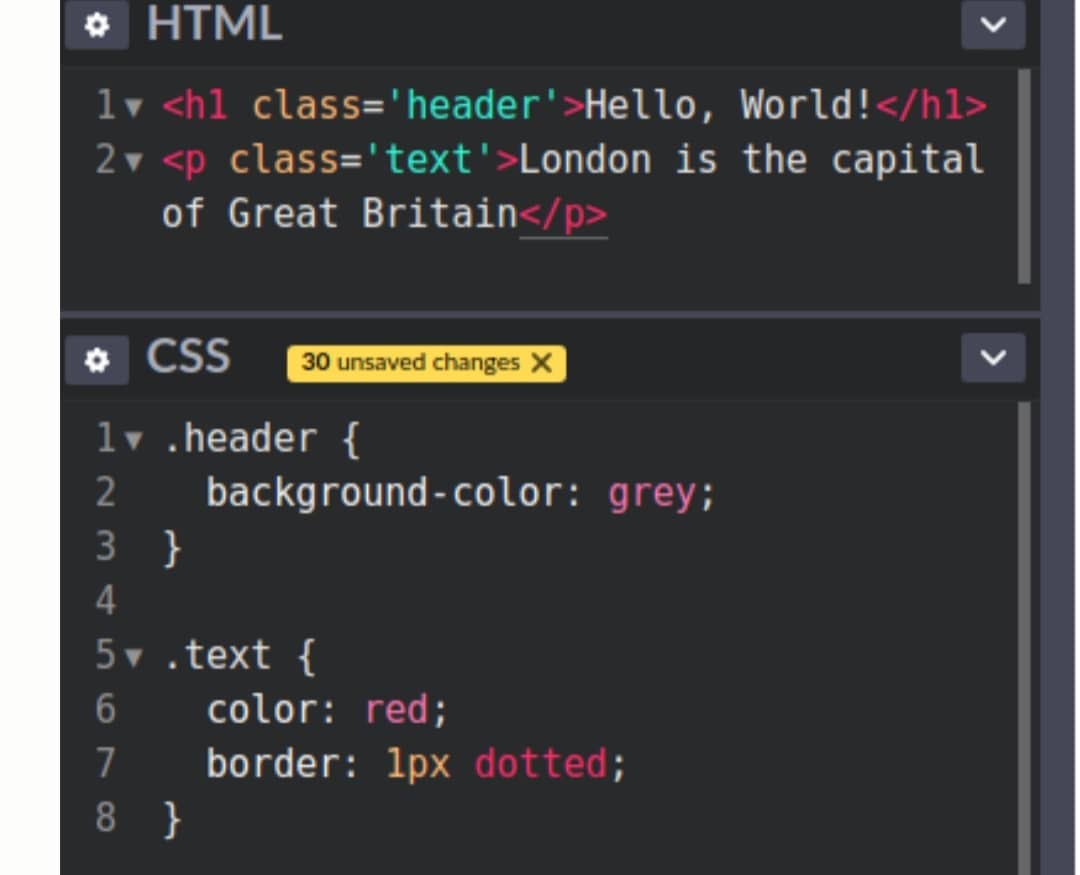
На иллюстрации в верхнем блоке можно увидеть гипертекстовую разметку, а в нижнем вывод на экран в удобном для чтения формате. В данном случае HTML показывает браузеру, где заголовок, а где параграф. Реальные страницы намного сложнее, они содержат больше элементов.



*HTML и вывод страницы в удобном для людей формате*

CSS — язык описания стилей. Аббревиатура образовалась из первых букв английских слов Cascading Style Sheets — каскадные таблицы стилей. CSS описывает внешний вид HTML-элементов. То есть разработчики с помощью каскадных таблиц стилей определяют, как должен выглядеть тот или иной элемент на странице.

На иллюстрации ниже видно, как с помощью CSS добавляется фон к заголовку, а также меняется цвет и устанавливается граница для параграфа.



**Задаём фон и меняем цвет с помощью**

В реальности возможности CSS гораздо шире: каскадные таблицы стилей определяют расположение элементов друг относительно друга, создают полноценную анимацию, позволяют адаптировать страницу к условиям просмотра, например, к яркому свету.

**Если создатель веб-страницы не использует CSS, браузеры отображают элементы HTML с помощью стилей по умолчанию. Эти дефолтные стили определяют разработчики браузеров.**

В русскоязычной среде специалиста по HTML и CSS часто называют верстальщиком, а создание веб­страниц с помощью этих языков — вёрсткой. В англоязычной среде таких специалистов называют веб­дизайнерами. Подробнее об этом мы рассказываем в бесплатном курсе по основам HTML, CSS и веб-дизайна.

На рынке труда есть люди, которых можно назвать «дизайнер- верстальщик». Эти специалисты занимаются дизайном: проектируют интерфейсы, создают визуальные решения. Также эти люди верстают: превращают макеты в веб-страницы с помощью HTML и CSS.

Можно встретить термины «фронтенд-разработчик» и «веб­разработчик». Их вряд ли корректно применять в отношении к «чистым» верстальщикам. Фронтенд- разработчик — это скорее специалист, который не только верстает, но и программирует веб­страницы или создаёт полноценные фронтенд-приложения. Он должен владеть не только HTML и CSS, но ещё и JavaScript.

Веб-разработчик — ещё более широкое понятие, так как в веб­разработке задействованы фронтенд- и бэкенд-разработчики. Специалисты по бэкенду работают с РНР, Python, Java, JavaScript и другими языками программирования. Людей, которые занимаются фронт- и бэкендом одновременно, называют fullstack- разработчиками.

Сфера применения HTML и CSS - веб-разработка, или О востребованности вёрстки

Как отмечалось выше, верстальщики используют HTML и CSS для создания веб-страниц, то есть страниц в интернете. По данным We Are Social, в январе 2020 года в мире насчитывалось 4,54 млрд пользователей интернета. Это 59 % от общего населения планеты. То есть потенциал роста интернет- пользователей исчисляется миллиардами человек.

Эти данные позволяют предположить, что интернет в обозримом будущем будет расти. То есть будут появляться новые сайты, а для создания этих сайтов понадобится HTML и CSS. Очевидно, эти языки будут развиваться, а их возможности будут расти.

Веб-разработчики, как фронтендеры, так и бэкэндеры, так или иначе сталкиваются с вёрсткой во время работы. Не каждому веб­программисту, особенно бэкенд- разработчику, нужно уметь верстать страницы «с точностью до пикселя». Но без понимания принципов HTML и CSS работать в веб-разработке практически невозможно.

То есть вёрстка сама по себе никуда не денется, пока существует и развивается интернет. Значит ли это, что верстальщикам можно не беспокоиться о перспективах на рынке труда? Нет.

На востребованность вёрстки можно посмотреть под другим углом. Порог входа в профессию верстальщика ниже по сравнению с программированием. Поэтому уровень конкуренции среди «чистых» верстальщиков высокий. Это причина задуматься.

Вот вторая причина: верстальщики сталкиваются с достаточно однотипными задачами. Это делает работу верстальщика механистической и скучной. Заниматься нелюбимым делом — не лучший выход, правда? Есть риск выгореть и потерять интерес к профессии.

Третья причина: есть большая вероятность, что в обозримой перспективе «чистые» верстальщики будут сильнее конкурировать не друг с другом, а с программами, которые конвертируют условные макеты psd в HTML. Это не вопрос сегодняшнего дня, но его нужно держать в голове при оценке перспектив рынка труда.

Значит ли это, что вёрстку изучать не стоит из-за отсутствия перспектив? Ни в коем случае. Профессию «верстальщик сайтов» можно рассматривать как способ войти в веб-разработку. Многие веб­программисты вырастают из «чистых» верстальщиков. То есть HTML и CSS — это отличный способ войти в отрасль, испытать себя, а затем развиваться дальше: изучать программирование.

Есть ли работа для верстальщика: вакансии и зарплаты

Чтобы проанализировать ситуацию на рынке труда, в первую очередь заглянем на hh.ru и посмотрим вакансии для верстальщиков. Также изучим предложения на русскоязычной и англоязычной бирже фриланса.

По состоянию на середину марта 2020 года по запросу «верстальщик» на hh.ru есть 678 вакансий по всей России. По запросу «фронтенд- разработчик» есть более 2000 вакансий. Такая же картина по запросу «веб-разработчик».

Уже на этом этапе анализа можно сделать важный вывод: «чистые» верстальщики требуются реже по сравнению с фронтенд- разработчиками. То есть работодателям чаще нужны специалисты, которые умеют верстать страницы и программировать фронтенд- приложения.

При детальном анализе вакансий по запросу «верстальщик» этот вывод подтверждается. Практически в каждом объявлении в требованиях работодатели указывают как умение верстать, так и знание JavaScript.

Программист

Программистами обобщенно называют категорию людей, занимающихся разработкой алгоритмов и программ на основе математических моделей. Условно программистов можно разделить на три категории:

Прикладные программисты. Такие специалисты занимаются разработкой конкретных программ, необходимых для работы организации. Например, сюда можно отнести программистов 1С.

Системные программисты программируют операционные системы, интерфейсы к распределенным базам данных, работают с сетями.

Web-программисты имеют дело с сетями, но, как правило, с глобальными, такими, как Internet. Они пишут web-интерфейсы к базам данных, создают динамические web- страницы и т.п.

Специфика профессии:

Плюсы профессии:

Постоянное профессиональное самосовершенствование,

Высокий спрос на рынке,

Высокая заработная плата,

Работать можно не имея диплома,

Преимущественно творческая профессия.

Личные качества:

Прежде всего, программист должен обладать терпением и выдержкой. Это совершенно незаменимые качества в его работе.

Программирование, бурно развивающаяся область, поэтому нужно уметь быстро адаптироваться и постоянно изучать что-то новое. Иначе через несколько лет ваша ценность как специалиста может заметно снизиться.

Умение объективно оценивать возможности технологий и их использование в каждом конкретном случае. Чтобы не получалось так, что дрова пилятся с применением новейших лазерных разработок.

Карьера и зарплата:

Руководитель группы программистов, 1Т- директор, менеджер проекта, можно перемещаться в рамках специальности, совершенствуясь профессионально.

Программист: гений поневоле В его голове цифры, алгоритмы и оригинальные идеи складываются в полезные, простые и доступные программы, способные облегчить жизнь миллионам пользователей персональных компьютеров.

Программист - одна из самых востребованных и высокооплачиваемых профессий в России. Даже не самый продвинутый специалист может найти работу в соответствии со своим уровнем знаний, а затем постепенно учиться и набираться опыта.

Хорошим стартом карьеры может стать присоединение к группе программистов при разработке какого-либо проекта. Крупные проекты часто привлекают внимание западных компаний, которые «перекупают» российских программистов. К примеру, когда то группа наших молодых ученых разрабатывала процессор «Эльбрус» для Минобороны, а в итоге их всех перекупила корпорация Intel, и теперь наши ученые и программисты работают за рубежом, а сам проект «Эльбрус» потихоньку закрылся. Проблема «утечки мозгов» в этой профессии одна из самых острых. Впрочем, и в России сегодня у программистов работы предостаточно - они нужны как отечественным компаниям-разработчикам программного обеспечения, так и любой фирме, где существует компьютерная сеть и необходимо адаптировать и настраивать ПО для нужд организации.

Спрос на рынке труда на программистов высокий, но и конкуренция соответствующая. Найти интересную и высокооплачиваемую работу вам удастся только в том случае, если вы знаете свое дело «на отлично». Одной из самых сложных в программировании считается разработка системного программного обеспечения - сервисов, которые управляют компонентами вычислительной системы (процессором, коммуникационными и периферийными устройствами, а также тех, которые предназначены для обеспечения функционирования и работоспособности всей системы (загрузчики, операционные системы, драйверы устройств и т. д.). Такие специалисты в России сегодня не слишком востребованы, поскольку практически все продукты подобного рода производятся за рубежом. Поэтому есть шанс найти высокооплачиваемую работу в западных компаниях.

Творцы и «кодировщики»

Другой фронт работ для программистов - разработка и создание программного обеспечения (ПО), предлагающего определенные услуги пользователям (текстовые редакторы, бухгалтерские программы, игры, ПО для систем видеонаблюдения, базы данных и др.).

Требования к таким программистам зависят от потребностей работодателя: от базового знания самых популярных языков программирования (Delphi, Java, C++ и др.) и умения устранить неисправность любой оргтехники до таких редких навыков, как разработка Lotus Notes, знание WinSock, Winlnet, ATL, STL и т. д.

Востребованы сегодня и специалисты, адаптирующие уже готовые программы (в основном «1С: Бухгалтерия», «1С: Склад» и т. д.) под особенности конкретного предприятия.

Web-программеры

Интернет с каждым годом развивается все более активно, и не иметь своего web-сайта для организации уже просто несерьезно. Соответственно, неуклонно растет потребность в профессионалах, способных создать сайт и обеспечивать его техническую поддержку. Традиционные требования к веб­программистам: знание РНР, MySQL, JavaScript, HTML, DHTML, CSS, XML и т. д. Зачастую программистам приходится заниматься и web-дизайном.

В этой работе есть один минус - через некоторое время обязанности штатного web- специалиста сводятся к обновлению контента, устранению ошибок и написанию несложных скриптов. Если вы творческий человек и любите работать самостоятельно, то лучше обсудить условия работы и обязанности заранее.

Другой вариант - найти место в компании, профессионально занимающейся разработкой web-сайтов по заказу.

Карьера & Зарплата

Первой ступенькой карьеры может стать должность программиста-стажера. Крупные 1Т-компании часто публикуют подобные вакансии. Необходимо иметь хорошую теоретическую подготовку, желательно знать языки высокого уровня (С++, УВ, УГР, С#, УВ. МеГ.). Зарплата стажера - около $ 1000. Штатный программист в компании среднего уровня (не 1Т) получает до $ 1500-1800, чуть больше - в организации, связанной с массовой разработкой ПО. Зарплата ведущего программиста - $ 2500-3000. Следующая ступень - руководитель 1Т-отдела. К необходимым знаниям добавляется обязательный опыт работы, владение иностранным языком, навыки управления персоналом и др., а заработок может достигать $ 4000. Хороший программист может стать руководителем крупного проекта по разработке ПО, и тут уровень доходов достигает $ 5000 и выше.

Многие программисты предпочитают не работать в штате компании, а быть фрилансерами (т. е. «свободными художниками») и выполнять разовые заказы, не выходя из дома. Фрилансер получает вознаграждение, напрямую зависящее от сложности проекта. За типовое задание, выполнение которого занимает два-три дня, можно получить $ 300-500. Верхняя граница оплаты может быть абсолютно любой ($ 3000 и выше). Плюс этого способа работы в том, что можно одновременно выполнять несколько заданий разных работодателей, зарабатывая, таким образом, намного больше.

Прикладная информатикаЧто такое быть выпускником специальности «Прикладная информатика»? Это значит быть стопроцентно уверенным, что с помощью информационных технологий, автоматизации рутинных процессов, технологий сбора и обработки информации жизнь современного человека можно сделать проще и удобнее.

Должностные обязанности

Аналитик в основном общается с командой разработки и с заказчиком (проект может быть и сторонним, и внутренним). Он собирает сведения и у заказчиков, и у предполагаемых пользователей продукта и описывает это для команды разработки в понятных для нее терминах. Отвечает на вопрос: что надо сделать? Именно аналитик занимается наиболее соответствующей специальности деятельностью.

Программист - член команды разработки. Отвечает на вопрос: как надо сделать? Получает информацию либо напрямую от заказчика и пользователей (если проект совсем маленький и не предполагает аналитиков), либо от аналитика и из документов, которые тот подготовил. После этого он создает программу.

Тестировщик - инженер по тестированию. Увы, роль этого специалиста в проекте недооценивают. Однако именно тестировщик, или тестер, отвечает за качество продукта: он составляет сценарии проверки программы, чтобы покрыть максимально возможные варианты ее использования и выявить максимально возможное число сбоев в ее работе, а также максимально возможное количество ошибок программиста.

Менеджер проектов зачастую совмещает роль аналитика проекта и обязанности по управлению проектом - распределение задач, контроль над их выполнением, помощь команде в сложных ситуациях и обеспечение дружественной инфраструктуры (комфортных условий и наличия всех необходимых программ).

Дополнительные требования

Специалисту с самого начала придется привыкать оценивать свой труд в трудозатратах. Нужно будет предугадывать, сколько времени отнимет та или иная задача, чтобы начальник проекта или клиент далее смогли оценить эти трудозатраты по установленным для них тарифам.

Для этой работы нужны и хорошо развитые коммуникативные способности. Одно из основных умений - умение доказать и показать, что всё хорошо придумано. Очень сложно доказать клиенту, что он неправ. А еще сложнее признать потом, что именно клиент был прав, потому что «идеальный» сценарий не работает.

Но есть и еще более важное умение - умение признать клиента основным носителем знаний в его предметной области. Увы, очень часто специалист додумывает за клиента, что тому якобы нужно, а потом клиент удивляется: сделано даже то, о чем не просили. Впрочем, бывает, что клиент и сам не знает, чего он хочет. Либо формулирует свои пожелания неточно и невнимательно, поскольку просто не понимает, что же еще надо объяснить: для него-то всё очевидно.

Работая на стыке областей знаний, вы зачастую будете посредником между узкими специалистами выбранной предметной области и специалистами техническими. И именно от вас зависит, найдут ли они общий язык, обретут ли они общее видение ситуации и единую точку зрения на решаемую задачу. Если вы создаете систему для коммерческой компании, то больше полугода вы будете объяснять остальным прикладникам, «а что такое дебет и кредит», «а как мне рассчитать вот этот коэффициент, который указан в документе». И объяснять как можно проще. А потом будете многократно растолковывать пользователям, «а когда нажимать на эту кнопку», и возражать на выпады типа «разве это сложно - добавить новый отчет? Он ведь почти такой же, как другой отчет».

Про себя

Меня зовут Алимова Тахмина Якубовна. Я 1996 года. В 2014 году окончила школу. В свое время не было возможности учиться и я пошла работать. Работала кассиром в кафе, менеджером в игровых заведениях. Так как весь мир уже перешел на цифровизацию, я тоже решила получше освоить компьютер и поступила в лицей. За этот год я очень многому научилась и я очень рада, что сделала такой выбор. Хочу поблагодарить наших преподавателей, которые несмотря на карантин, в любое время отвечали на любые вопросы, помогали, обучали нас…