Лекция 2. Основы CSS

Основы CSS

- **CSS (Cascading Style Sheets)** – технология для управления отображением документа (веб-страницы)
- Т.е. позволяет задавать **стили** элементов – цвет, цвет фона, размер шрифта, положение элемента на странице и т.д.

- До появления CSS, для этих целей использовались специальные HTML теги и атрибуты, сейчас они deprecated (нежелательны к использованию)

Почему CSS лучше

- **Нет дублирования стилей.**
  При описании отображения при помощи HTML приходится писать много тегов и атрибутов, и если такое же оформление понадобится в другом месте, то эту разметку придется продублировать
- <h2>
  Текст заголовка 1
  </h2>
- В CSS один стиль может применяться сразу ко многим элементам, что избавляет от дублирования и загромождения HTML разметки
- <h2 class="header"></h2>

Почему CSS лучше

- **Разделение ответственности.**
  HTML тогда отвечает только за структуру и содержимое документа, а CSS – только за отображение.
- В итоге HTML и CSS пишутся отдельно, и HTML содержит только теги, отвечающие за структуру, что упрощает модификацию каждой из частей в отдельности
- За счет разделения файлов, можно сделать, чтобы к одному и тому же HTML можно было подключать разные CSS, и оформление страницы будет разным. Так реализуются темы сайтов

Почему CSS лучше

- **Легкость модификации.**

  В CSS каждый стиль может применяться сразу к большому количеству элементов

  То есть меняем стиль в одном месте, и он применяется везде, где нужно

  А в HTML нам пришлось бы вручную искать каждое место, где использовался тег или атрибут, и менять там стили

Почему CSS лучше

- **Единый файл со стилями на все страницы.**
  Один и тот же файл со стилями можно подключать сразу ко многим страницам – не приходится ничего дублировать
- И, как уже говорилось, можно сделать несколько CSS с разными темами

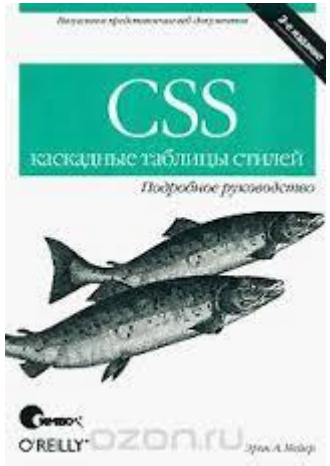
Почему CSS лучше

- **Несравнимо больше возможностей.**
  CSS предоставляет несравнимо большее число возможностей, чем возможности HTML по управлению отображением документа.

- CSS позволяет также делать адаптивный дизайн, который будет разным для разных устройств, в зависимости от ширины экрана

Литература

- Эрик Мейер. CSS – каскадные таблицы стилей. Подробное руководство



- http://htmlbook.ru/

Синтаксис CSS.
Как применить стили

Синтаксис CSS

- Файл стилей CSS представляет собой набор правил следующего вида:

- ```
  селектор {
      свойство1: значение1;
      свойство2: значение2;
  }
  ```

- Например,
  ```
  p {
      color: red;
      font-size: 16px;
  }
  ```

Селектор – это выражение, указывающее, к каким элементам страницы применять правила

Правила задаются парами ключ-значение.
Ключ отделяется от значения при помощи :
В конце пары ставится ;

Пример задает всем параграфам размер шрифта 16px и красный цвет
  ```

Пробелы, ошибки

- селектор {
  свойство1: значение1;
  свойство2: значение2;
  }
- Пробелы и переводы строк можно ставить произвольным образом
- Конечно, за исключением свойств и значений
- Если одно и то же свойство встретится несколько раз в одном правиле, то победит последнее (нижнее) значение
- Перечень свойств (и многих допустимых значений) предопределен. Если допустить опечатку, то свойство не применится. Если допустить синтаксическую ошибку, то может не примениться всё правило

Регистр символов, комментарии

- селектор {
  свойство1: значение1;
  свойство2: значение2;
  }
- Регистр символов важен
- Комментарии в CSS – такие же, как многострочный комментарий в Java:
- /* Это комментарий
  p {
  width: 100px;
  }
  */

Как применить CSS

- Есть 3 варианта
  - Inline-стили (встроенные стили)
  - HTML-элемент <style>
  - **Подключаемый CSS файл**

- Крайне рекомендуется третий вариант

Inline-стили

- Пишутся прямо в HTML разметке
- Параграф

- **Плюсы:**
  - Простота использования
- **Минусы:**
  - Дублирование стилей, если такие стили нужны в нескольких местах
  - Трудность модификации стилей

Использование inline-стилей

- Используются, в основном, в следующих случаях:
  - Верстка HTML-писем – Gmail поддерживает только inline стили
  - Используются JavaScript библиотеками, чтобы «перебить» значения из CSS файла – inline стили имеют более высокий приоритет
- Лучше использовать inline-стили только в указанных выше случаях

Элемент style

- Позволяет описать правила в HTML-файле
- Этот элемент должен быть помещен внутрь элемента head:
-

```
<head>
   <style type="text/css">
      .header {
         font-size: 16px;
         font-weight: bold;
      }

      p {
         color: black;
      }
   </style>
</head>
```

Элементом style пользуются редко

Минус – стили можно использовать только на данной странице

Обычно выносят стили в отдельный подключаемый CSS файл

Подключаемый CSS

- Для CSS стилей пишется отдельный файл, который подключается на всех страницах, где он нужен
- Подключить CSS-файл можно при помощи элемента link, который должен быть помещен в head
- **index.html**

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

style.css

```
.header {
    font-size: 16px;
    font-weight: bold;
 }
```

Атрибут media

- CSS на самом деле может использоваться не только для отображения данных на экране, но и для печати данных принтером, при отображении проектором, для синтезаторов речи и др.
- Это задается при помощи атрибута media
- <link rel="stylesheet" type="text/css" href="style.css" media="screen" />
- Возможные значения:
  - all – все варианты
  - screen – экран
  - print – печать
  - др.

Подключаемый CSS

- Можно подключать много CSS файлов

- ```
  <head>
    <link rel="stylesheet" type="text/css" href="style1.css" />
    <link rel="stylesheet" type="text/css" href="style2.css" />
  </head>
  ```

- При этом стили будут комбинироваться, про это поговорим позже
  ```

Демонстрация

- Chrome Dev Tools (по F12)

Практика

- Возьмите какую-нибудь свою страницу, попробуйте применить какие-нибудь стили всеми тремя способами:
  - Inline-стили
  - Элемент style
  - Подключаемый CSS-файл

Как браузер загружает страницу

Загрузка HTML страницы

1. Браузер запрашивает HTML страницу, сервер отдает её
2. Браузер начинает отрисовывать страницу, встречает элемент link, ссылающийся на CSS или элемент script, ссылающийся на JavaScript
3. Браузер делает ещё один запрос на сервер, чтобы получить CSS файл или JS файл
4. Так делается для всех скриптов и подключаемых CSS
5. Если встречается элемент img, то браузер делает запрос, чтобы получить картинку, так для каждой картинки

- **Итого:** браузер делает огромное число запросов для каждой страницы. Каждый запрос – нагрузка на сервер. А если пользователей много?

Загрузка HTML страницы

- **Итого:** браузер делает огромное число запросов для каждой страницы. Каждый запрос – нагрузка на сервер. А если пользователей много?
- Чтобы избавиться от этой проблемы, все CSS файлы минифицируют (сжимают) и конкатенируют (объединяют)
- То же самое делают для JS файлов
- За счет этого получается всего 1 запрос за JS и 1 запрос за CSS, что сокращает нагрузку на сервер
- Бывает что и картинки объединяют в одну (спрайт)

Селекторы CSS

Селекторы

- Как мы уже увидели, чтобы применить стили, надо:
  1. При помощи селектора указать, к каким элементам их применять
  2. Указать сами стили

- Язык селекторов очень гибкий и позволяет легко выбрать нужные нам элементы

- **Полезная шпаргалка по селекторам CSS:**
- http://code.tutsplus.com/ru/tutorials/the-30-css-selectors-you-must-memorize--net-16048

Универсальный селектор

- Позволяет выбрать все элементы

- Указать всем элементам красный цвет текста
- ```
  * {
      color: red;
  }
  ```

- Пользоваться им следует только в очень редких случаях, он сильно ухудшает производительность работы страницы
  ```

Селектор по типу элемента. Запятая

- Самый простой селектор – по типу элемента
- ```
  p {
      color: red;
  }
  ```

Сделать текст всех абзацев красным

- В целях сокращения дублирования стилей, для одного набора правил в фигурных скобках можно указать несколько селекторов через запятую:
- ```
  p, h1 {
      color: red;
  }
  ```

Сделать текст всех абзацев и заголовков первого уровня красным

Запятая применима ко всем селекторам – она просто позволяет применить одни стили к разным селекторам

HTML-атрибуты class и id

- Для «точечного» выбора элементов, CSS может пользоваться двумя стандартными HTML атрибутами, которые могут быть применены почти к любому элементу: id и class
- id – уникальный идентификатор элемента. Обязательно должен быть уникален во всем документе, иначе будут ошибки
- Должен начинаться с латинского символа, а дальше может идти латиница, цифра, тире или подчеркивание
- Примеры:
- <h1 id="page-header"></h1>
  <h2 id="page_subtitle"></h2>
  <h3 id="pageTitle"></h3>

Выбирайте единый стиль именования id и class'ов

HTML-атрибуты class и id

- class – уже не обязан быть уникальным. Одним классом следует помечать элементы, у которых должны быть одинаковые стили
- Правило именования такое же
- **Важное отличие**: в одном атрибуте class можно через пробел указывать много классов
- Примеры:
- <h2 class="header"></h1>
  <h2 class="header"></h2>
  <h3 class="header small"></h3>

К h3 тут применено 2 класса – header и small

- Элемент может иметь и id, и class одновременно
- Оба атрибута - **регистрозависимы**

HTML-атрибуты class и id

- В CSS чаще рекомендуется пользоваться атрибутом class:
  - У class нет ограничения на уникальность. Если сейчас у вас всего 1 элемент со стилями, и обращение идет через id, то в будущем на странице может появиться второй такой же элемент, тогда стили придется менять
  - class более рекомендуем, чем селектор по типу элемента, т.к. разметка (и типы элементов) могут поменяться, а атрибут class можно навесить почти на любой элемент, и существующие стили сохранятся
- id чаще используется в JavaScript, чтобы обращаться к конкретным элементам
- Также id полезен для автотестирования – элемент легко найти

Селектор по id

- <h1 id="page-header"></h1>

- Селектор по id:
- #page-header {
  font-size: 20px;
  }

 # ставится без пробела

- Селектору соответствует только 1 элемент, с этим id

Селектор по class

- <h2 class="header"></h2>
  <h3 class="header small"></h3>

- Селектор по class:
- .header {
  font-weight: bold;
  }

  .small {
  font-size: 10px;
  }

. ставится без пробела

Результат – оба элемента, у них есть класс header

Результат – 1 элемент, у которого есть класс small

Селекторы по атрибуту

- Есть возможность выбирать элементы, у которых есть заданный атрибут
- Text

- ```
  [myAttr] {
      color: red;
  }
  ```

- Но есть и другие варианты этого селектора, которые позволяют проверять значение атрибута
  ```

Селекторы по атрибуту

- Проверка, что у элемента есть такой атрибут, и его значение совпадает с указанным
- [myAttr=“myValue”] {
  color: red;
  }

- Проверка, что у элемента есть такой атрибут, и его значение содержит указанную строку
- [myAttr*=“myValue”] {
  color: red;
  }

Селекторы по атрибуту

- Проверка, что у элемента есть такой атрибут, и его значение начинается с указанной строки
- [myAttr^=“myValue”] {
  color: red;
  }

- Проверка, что у элемента есть такой атрибут, и его значение заканчивается на указанную строку
- [myAttr$=“myValue”] {
  color: red;
  }

Селекторы по атрибуту

- Проверка, что у элемента есть такой атрибут, и его значение содержит указанное слово
- Предполагается, что атрибут имеет значение, состоящее из одного или нескольких слов, разделенных пробелами
-
- [source~="external"] {
  color: red;
  }
  [source~="image"] {
  border: 1px solid black;
  }

Комбинирование селекторов

- Все селекторы можно комбинировать (если это имеет смысл)
- Для этого надо написать условия **подряд, без пробела**

Если поставить пробел, то смысл селектора изменится

- Например, можно делать так:
- ```
  p.wide {
      …   // выбрать параграфы с классом wide
  }
  ```
- ```
  p[myAttr="value"].wide {
      …  // выбрать параграфы с классом wide,
         // у которых при этом есть атрибут myAttr, и его
         // значение равно "value"
  }
  ```

Комбинирование селекторов

- #some-id.wide {
  … // выбрать элемент с id some-id и классом wide
  }
- .wide#some-id {
  … // выбрать элемент с id some-id и классом wide
  }
- Эти селекторы равносильны, порядок можно менять
- Порядок нельзя поменять для случаев, когда мы используем селектор по элементу – CSS просто не поймет где кончается класс и начинается тип элемента
- a.link – хорошо, .linka – искать элементы с классом linka
- Обычно, нет смысла комбинировать селектор по id с другими

Селекторы с учетом вложенности

- Можно применять селекторы, основываясь на вложенности элементов друг в друга
- Для этого нужно поставить пробел между селекторами
-
  Параграф 1
  Параграф 2

- .wide p {
  … // выбрать параграфы, которые лежат на любом
  // уровне вложенности внутри элементов с классом wide
  }
- Селектор выберет оба параграфа

Селектор выбора детей

- Есть селектор, позволяющий выбрать только непосредственных детей
- ```

      Параграф 1
      Параграф 2

  ```
- ```
  .wide > p {
      … // выбрать параграфы, которые лежат
       // непосредственно внутри элементов с классом wide
  }
  ```
- Селектор выберет только параграф 2

Примеры

- Еще примеры:
- .wide > p > a
- .wide p a
- .long.high [src]
- #someId p.wide
- form.form input[type=“text”]

Селектор +

- Div
  Paragraph 1
  Paragraph 2
- div + p {

  }
- Такой селектор выберет все параграфы, для которых предыдущий элемент на этом же уровне является div’ом
- Т.е. этот селектор – выбор непосредственных соседей
- В данном примере будет выбран только первый параграф, потому что для второго предыдущим элементом является первый параграф, а не div

Селектор ~

- Div
  Paragraph 1
  Paragraph 2
- div ~ p {

  }
- Такой селектор выберет все параграфы, для которых некоторый предыдущий элемент на этом же уровне является div’ом
- Т.е. этот селектор – выбор всех соседей, которые идут после указанного элемента
- В данном примере будут выбраны оба параграфа

Селекторы + и ~ используются редко, но в некоторых случаях очень выручают

Демонстрация

- Chrome Dev Tools (по F12)

Практика

1. Напишите селектор, чтобы выбрать указанные элементы и применить к ним стили, не меняя разметку

2. Измените разметку и напишите селекторы, чтобы можно было применить стили к указанным элементам

- На практике часто встречаются обе эти задачи. С одной стороны, при необходимости менять стили, хочется как можно меньше менять разметку
- Но если при этом получаются негибкие селекторы (id, тип элемента и т.д.), то лучше поменять разметку и навесить классы на некоторые элементы, чтобы селекторы были более простые и гибкие (устойчивые к изменениям)

Некоторые CSS свойства

CSS свойства

- В CSS есть просто огромное количество свойств. Их можно разбить на группы:
  - Оформление текста
  - Работа с размерами и отступами
  - Позиционирование элементов
  - Создание сеток
  - Декоративные: цвета, фон, тени
  - Др.

Поддержка CSS браузерами

- Не все свойства поддерживаются всеми браузерами
- Со всеми браузерами обычно всё хорошо, кроме старых IE. Например, до IE 9 не поддерживается свойство border-radius, которое позволяет скруглить края элемента
- Поэтому если ваш проект должен поддерживать такие браузеры, то для каждого селектора и свойства нужно проверять, поддерживается ли оно

- На этом сайте можно посмотреть информацию о поддержке свойства разными браузерами:
- http://htmlbook.ru/css/
- https://webref.ru/css

Полезные свойства - цвет

- color – цвет текста, background-color – цвет фона
-
```
p {
    color: red;
    color: #ff0000;
    color: #f00;
    color: rgb(255, 0, 0);
    background-color: #000000;
}
```

Это всё разные варианты применения одного и того же цвета

- **Варианты значений для цветов:**
  1. Название цвета – red, blue, green и т.д.
  2. RGB HEX - #rrggbb. По две шестнадцатеричной цифры для каждой компоненты цвета
  3. RGB HEX - #rgb. Это краткая запись для #rrggbb
  4. rgb(r, g, b) – числа от 0 до 255 включительно

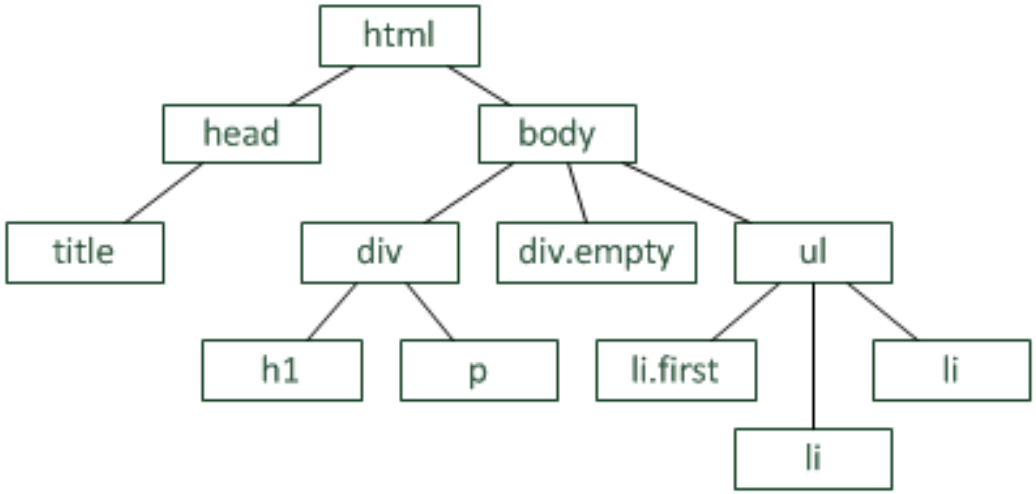
Единицы измерения - длина

- width – ширина, height – высота
- ```
  p {
    width: 100px;
    width: 50%;
    width: 10em;
  }
  ```

- Есть абсолютные и относительные единицы измерения:
  - **Абсолютные:** пиксели 100px, пункты 10pt, мм 100mm. Обычно используют пиксели
  - **Относительные:** проценты от размеров родителя, em – количество размеров шрифта для элемента
  - Т.е. 1em равно длине размера шрифта
  ```

Каскадность

Каскадность

- Почему CSS – каскадные?
- Дело в том, что разные стили для одного и того же элемента могут быть объявлены в нескольких местах одновременно:
  - Один и тот же элемент может соответствовать нескольким селекторам из одного или даже нескольких CSS файлов
  - Дополнительно, у элемента могут быть inline-стили
  - Кроме того, в каждом браузере есть стили по умолчанию для элементов
- Браузер находит для каждого элемента все селекторы, которые его затрагивают, и по четким правилам составляет итоговые стили
- При этом часть стилей объединяется, а часть – переопределяется
- Слово «каскадные» хорошо описывает этот процесс построения итоговых правил

Каскадность – объединение стилей

- Это крайне важная тема. Некоторые разработчики даже не в курсе, что есть четкие правила **приоритета стилей** и удивляются, почему их стили не применяются, думают, что это какой-то баг или магия. И в итоге пишут очень плохие CSS стили, с которыми сложно работать
- ```
  Параграф 1
  Параграф 2
  ```
- ```
  p {
      color: red;
  }
  .info {
      background-color: green;
  }
  ```

Второй параграф соответствует обоим селекторам. Поэтому к нему применятся все эти стили

Каскадность – переопределение стилей

- Что если в разных правилах определим одинаковые CSS свойства?
- Параграф 1
  Параграф 2
- p {
  color: red;
  }
  .info {
  color: green;
  }

Цвет текста второго параграфа будет зеленым

Почему?

Специфичность

- Упрощенно выглядит так:
  - Самый высокий приоритет у inline-стилей
  - Селектор по id
  - Селектор по class
  - Селектор по имени тега

- Если селекторы одного типа, то побеждает тот, что объявлен ниже

- На самом деле всё ещё сложнее, см. курс «Наследование и каскадирование» https://htmlacademy.ru/courses/66

Ключевое слово important

- Самый высокий приоритет у inline-стилей
- Но есть ключевое слово important, которое позволяет перебить даже их
- Текст
- ```
  p {
      color: blue !important;
  }
  ```
- Эту возможность следует избегать
  ```

Наследование

- **Наследование стилей** – еще один важный принцип
- HTML-документ представляет из себя дерево элементов
- У каждого элемента есть 1 родитель (элемент, внутри которого находится данный элемент)



- **Наследование в CSS** – это механизм, при котором стили элемента-родителя передаются элементам-потомкам

Наследование

- Например, если указать размер шрифта для одного элемента, то он унаследуется для всех его дочерних элементов
- Но наследуются далеко не все свойства CSS

-
```
<div style="font-size: 14px;">
    Просто текст.
    <p>Параграф</p>
</div>
```

У самого div и вложенных в него элементов будет размер шрифта 14px.
Если для самих элементов явно не указано иное при помощи CSS

- Если элемент явно переопределяет унаследованные стили, то наследование прекращается
- Наследование позволяет сократить размеры CSS файлов

Какие свойства наследуются

- В основном наследуются свойства, которые относятся к стилям текста:
  font-size, font-family, font-style, color, text-align, line-height и др.
- Также наследуются: list-style, cursor, visibility, border-collapse
- Не наследуются, например, границы border, отступы margin, padding

Процесс верстки. Полезные программы при верстке

Как выглядит процесс верстки

- В зависимости от того, является ли дизайн новым, есть 2 варианта
  - **Новый дизайн.**
  - **Новая страница, но дизайн уже существующий.**

Процесс верстки – новый дизайн

- Есть **макет** – просто картинка, на которой нарисован требуемый интерфейс **(ДЕМОНСТРАЦИЯ gabriel.png)**
- Макет делает дизайнер, либо приходит от заказчика. Макет отдают верстальщику или разработчику
- Задача верстальщика или разработчика – максимально точно воспроизвести этот дизайн при помощи HTML и CSS.
  Это достаточно сложно, обычно эту задачу дают верстальщикам – они сделают оптимальнее
- Обычно, такую верстку делают не в самом проекте, а отдельно от него – в верстке нет логики JavaScript, в качестве контента используются статичные данные и т.д.
- Когда все готово, то эти стили и HTML уже переносят в проект, а дальше разработчики подключают JavaScript и реальные данные

Процесс верстки – CSS reset

- Каждый браузер имеет свой набор стилей по умолчанию для разных элементов
- Поэтому внешний вид страницы будет немного отличаться в разных браузерах
- Это сильно усложняет разработку и проверку страниц
- Чтобы избавиться от этого, есть специальные CSS файлы, в которых определены стили, которые сбрасывают стандартные стили браузеров
- Если подключить такой CSS, а дальше уже писать свои стили, то все браузеры будут вести себя одинаково
- Поэтому при разработке нового проекта, нужно подключить такой CSS. Их есть несколько, самые известные: Normalize и Reset

Стили для старых IE

- Старые IE ведут себя не так, как нормальные браузеры
- Чтобы исправить проблемы с ними, для таких браузеров следует писать отдельные CSS стили и подключать их только для IE
- Это возможно при помощи **условных комментариев (conditional comments)** в HTML
-
```
<head>
   <link rel="stylesheet" type="text/css" href="style.css" />
   <!--[if IE 8]>
   <link rel="stylesheet" type="text/css" href="style-ie8.css" />
   <![endif]-->
</head>
```

Для всех браузеров этот link – комментарий. Но IE проверит условие, и подключит разметку, если условие выполняется

- http://www.quirksmode.org/css/condcom.html
- https://msdn.microsoft.com/ru-ru/library/ms537512(v=vs.85).aspx

Процесс верстки – полезные инструменты

- При работе с макетом могут быть полезные следующие инструменты:
  - **ColorPic** – программа-пипетка для считывания цвета с экрана
  - **A Ruler for Windows** – экранная линейка, полезна для измерения расстояний на макете (отступов, длины и ширины и т.д.)
  - **Photoshop / Gimp / др.** – графический редактор, нужен, чтобы вырезать из макета картинки (например, иконки и т.д.) По-хорошему, все эти картинки по отдельности уже должен предоставить дизайнер – не царская это работа картинки вырезать ☺

Процесс верстки – существующий дизайн

- **Новая страница, но дизайн уже существующий.** Часть проекта уже сделана, многие стили уже существуют.
- У нас есть описание, что должно быть на странице. Мы просто добавляем нужные HTML элементы, пока что ничего не стилизуем
- После этого смотрим аналогичные страницы, берем стили и разметку оттуда (какие классы надо навесить, как расположить элементы друг относительно друга)
- После этого самостоятельно дописываем недостающие стили, если такие есть

Что не упомянуто

- **Псевдоэлементы**: before, after
- **Псевдоклассы**: hover, visited, focus, active, link, first-child, last-child и др.

- **Flex** и **Grid** – новые режимы позиционирования элементов, которые пока не везде работают. Но flex уже практически везде (даже в Bootstrap 4), и его можно применять

- **Clearfix** – исправление проблемы с float’ом
- box-sizing: border-box; - более удобный способ задания размеров и отступов элементов

Домашняя работа «TutorialCss»

- **Пройдите курсы по CSS:**
- Знакомство с CSS: https://htmlacademy.ru/courses/41
- Селекторы: https://htmlacademy.ru/courses/42
- Блочная модель документа: https://htmlacademy.ru/courses/44
- Позиционирование:
- https://htmlacademy.ru/courses/45
- Наследование и каскадирование:
- https://htmlacademy.ru/courses/66

Домашняя работа «Gabriel»

- Сверстать страницу по макету (gabriel.png)
- Слайдер и полосу прокрутки пока можно не делать, это уже JavaScript

- Это была моя первая задача на работе, когда стал веб-разработчиком ☺
- В последующих задачах по JavaScript мы вернемся к этой странице и доделаем её