Практическая работа №2 «Семантическая разметка HTML5. Введение в CSS. Основные понятия CSS. Блочная модель. Подключение CSS к HTML. Структура и формат CSS-правил. Селекторы CSS»

2 Основные семантические теги HTML5

Семантическая вёрстка — подход к разметке, который опирается не на содержание сайта, а на смысловое предназначение каждого блока и логическую структуру документа.

2.1 Элемент `header` – «шапка сайта»/верхний колонтитул

Значение элемента: вводная часть смыслового раздела или всего сайта, обычно содержит подсказки и навигацию. Чаще всего повторяется на всех страницах сайта.

Особенности: этих элементов может быть несколько на странице.

Типовые ошибки: использовать только как шапку сайта.

Чтобы создать шапку сайта, используется парный тег `<header>`, внутри которого мы и располагаем необходимую информацию.

Элемент `header` представляет собой группу вводных или навигационных средств. Также его можно использовать для обертки оглавления раздела, формы поиска и любых соответствующих логотипов.

Элемент `header` не разделяет содержимое; он не вводит новый раздел.

2.2 Элемент `nav` – главная навигация сайта или страницы

Значение элемента: представляет собой раздел страницы, который ссылается на другие страницы или части внутри страницы: раздел с навигационными ссылками.

Особенности: используется для основной навигации, а не для всех групп ссылок. Основной является навигация или нет — на усмотрение верстальщика. Например, меню в подвале сайта можно не оборачивать в `<nav>`. В подвале обычно появляется краткий список ссылок (например, ссылка на главную, копирайт и условия) — это не является основной навигацией, семантически для такой информации предназначен `<footer>`, но использование тега `<nav>` всё же возможно.

Типовые ошибки: многие считают, что в `<nav>` может быть только список навигационных ссылок, но согласно спецификации там может быть навигация в любой форме.

На листинге 2.1 приведен один из возможных примеров добавления элементов `header` и `nav` в код:

Листинг 2.1 – Пример добавления элементов `header` и `nav`

```
<body>
    <header>
     <h1>Little Green Guys With Guns</h1>
     <nav>
      <ul>
       <li><a href="/games">Games</a>
       <li><a href="/forum">Forum</a>
       <li><a href="/download">Download</a>
      </ul>
     </nav>
     <h2>Important News</h2>
     <p>...</p>
     <h2>Games</h2>
    </header>
    ...

</body>
</html>
```

2.3 Элемент `main` – уникальный контент на странице

Значение элемента: основное, не повторяющееся на других страницах, содержание страницы - **на странице не должно быть более одного элемента `main`, для которого не указан атрибут `hidden`.**

Типовые ошибки: включать в этот тег то, что повторяется на других страницах (навигацию, копирайты и так далее).

Элемент `main` представляет доминирующее содержание документа.

Старайтесь в нем держать только контент. Обычно меню, боковые панели и футер в эту область не входят. Исключением может быть только ситуация, если эти блоки действительно уникальны для данной страницы. Например, меню может вести по разделам страницы. В таком случае оно на полных правах может быть включено в область уникального контента.

Наличие тега `<main>` также очень важно для мобильных браузеров. Вы могли видеть, что многие из них имеют функцию «Режим чтения». При его включении браузер автоматически удалит все оформление и все ненужные блоки, оставив только главный контент. Этим контентом и будет являться область, заключенная в тег `<main>`. Такой режим отлично подходит для людей, у которых в настоящий момент слабое подключение к интернету.

2.4 Элемент `section` – секции (разделы) страницы

Значение элемента: смысловой раздел документа. Неотделяемый, в отличие от `<article>`.

Особенности: желателен заголовок внутри.

Типовые ошибки: путают с тегами `<article>` и `<div>`.

Элемент `section` представляет общий раздел документа или приложения. Раздел в этом контексте представляет собой тематическую группу контента, обычно с заголовком.

Примерами разделов могут быть главы, различные страницы с вкладками в диалоговом окне с вкладками или пронумерованные разделы диссертации. Домашняя страница веб-сайта может быть разделена на разделы для введения, новостей и контактной информации. Также, например, на странице могут присутствовать области с описанием преимуществ, цены, формы и так далее. Их хочется как-то выделить. И на это есть несколько причин:

- С правильно разделенным контентом удобно работать. Мы легко можем перемещать такие области, менять местами или удалять. В коде их будет легко найти.
- Правильная группировка разделов — важная часть при создании доступного *web'a.*

Элемент `section` не является универсальным элементом контейнера. Когда элемент необходим только для целей стилизации или для удобства написания скриптов, авторам рекомендуется вместо этого использовать элемент `div`. Общее правило заключается в том, что элемент `section` подходит только в том случае, если содержимое элемента будет явно указано в структуре документа.

2.5 Элемент `article` – независимые секции страницы

Значение элемента: независимая, отделяемая смысловая единица, например, комментарий, твит, статья, виджет ВК и так далее.

Особенности: желателен заголовок внутри.

Типовые ошибки: путают с тегами `<section>` и `<div>`.

Элемент `article` представляет собой полную или автономную композицию в документе, странице, приложении или сайте, которая, в принципе, может распространяться или повторно использоваться независимо друг от друга. Это может быть сообщение на

форуме, статья в журнале или газете, запись в блоге, пользовательский комментарий, интерактивный виджет или гаджет или любой другой независимый элемент контента.

2.6 Выбор между <section>, <article> и

Между <section> и <article> есть одна существенная разница: <article> является независимой секцией, то есть ее можно перенести на любую страницу сайта или даже на другой сайт, и при этом она не потеряет своего контекста.

Представьте себе блог и отдельную статью в нем. Можем ли мы понять статью, если она вдруг окажется не в блоге, а, например, на странице с услугами? Конечно! Ведь статья — это законченный текст. Следовательно, такую статью можно обернуть в тег <article>.

Например, разберем колонку новостей. Сами по себе новости являются достаточно уникальным элементом, ведь даже если их перенести на другую страницу, то они не потеряют свою актуальность. В этом случае каждую новость можно обернуть в <article>. А что делать с оберткой блока? Она объединяет по смыслу несколько различных новостей, ее можно спокойно назвать одним словом, и она точно будет иметь свой заголовок. Следовательно, ей подойдет тег <section>.

Тег является элементом разделения контента HTML, т.е. универсальным контейнером для потокового контента. Он не влияет на контент или макет до тех пор, пока не будет стилизован с помощью CSS. Если вы, соответственно, используете элемент только как оболочку стиля, используйте . Негласное правило состоит в том, что <section> должен логически появляться в структуре документа.

2.7 Элемент aside – дополняющие секции на странице

Значение элемента: представляет собой часть страницы, состоящую из содержимого, косвенно связанного с содержимым вокруг элемента aside, и которое можно считать отдельным от этого содержимого. Такие разделы часто представлены в виде боковых панелей в печатной типографике.

Этот элемент можно использовать для типографских эффектов, таких как кавычки или боковые панели, для рекламы, для групп элементов навигации и для другого контента, который считается отдельным от основного контента страницы.

Особенности: может иметь свой заголовок. Может встречаться несколько раз на странице.

Обратите внимание, что <aside> не обязан являться боковой панелью по внешнему виду. Это может быть даже дополнительная информация внутри статьи. Но чаще всего внешнее оформление у такого тега именно в виде боковой панели.

2.8 Элемент `footer` – «подвал» сайта/нижний колонтитул

Значение элемента: заключительная часть смыслового раздела или всего сайта, обычно содержит информацию об авторах, список литературы, копирайт и так далее. Чаще всего повторяется на всех страницах сайта.

Особенности: этих элементов может быть несколько на странице. Тег <footer> не обязан находиться в конце раздела.

Типовые ошибки: использовать только как подвал сайта.

Когда элемент footer содержит целые разделы, они представляют собой приложения, указатели, длинные колофоны, подробные лицензионные соглашения и другое подобное содержимое.

Контактная информация автора или редактора раздела содержится в элементе address внутри <footer>. Авторы и другая информация, которая может подходить как для <header>, так и для <footer>, может быть помещена в любой из них (или ни в один из них). Основная цель этих элементов — просто помочь автору написать не требующую пояснений разметку, которую легко поддерживать и стилизовать; они не предназначены для навязывания авторам определенных структур.

Элемент footer не разделяет содержимое; он не вводит новый раздел.

2.9 Элемент `address` – контактная информация

Элемент address представляет контактную информацию для своего ближайшего предка статьи или элемента тела. Если это дочерний элемент body, то контактная информация применяется ко всему документу.

Тег <address> не должен содержать любой другой информации, кроме контактной, например, дата публикации - относится к тегу <time>.

Обычно тег <address> размещается внутри тега <footer>.

Визуально информация внутри тега <address> выделяется курсивом на странице.

Примечание: для представления адреса, который не является контактной информацией, воспользуйтесь <p>.

На листинге 2.2 приведен один из возможных примеров добавления элементов `footer` и `address` в код:

Листинг 2.2 – Пример добавления элементов `footer` и `address`

```
    ...
    <footer>
        <address>
        For more details, contact
        <a href="mailto:js@example.com">John Smith</a>.
        </address>
        <p><small>© copyright 2038 Example Corp.</small></p>
    </footer>
</body>
</html>
```

2.10 Как разметить страницу с точки зрения семантики

Процесс разметки можно разделить на несколько шагов с разной степенью детализации.

1. Крупные смысловые блоки на каждой странице сайта.

   Теги: `<header>`, `<main>`, `<footer>`.
2. Крупные смысловые разделы в блоках.

   Теги: `<nav>`, `<section>`, `<article>`, `<aside>`.
3. Заголовок всего документа и заголовки смысловых разделов.

   Теги: `<h1>-<h6>`.
4. Мелкие элементы в смысловых разделах. Списки, таблицы, демо-материалы, параграфы и переносы, формы, цитаты, контактная информация и прогресс.
5. Фразовые элементы. Изображения, ссылки, кнопки, видео, время и мелкие текстовые элементы.

Один из возможных примеров развертывания семантической разметки элементов на сайте представлен на рисунке 2.1:

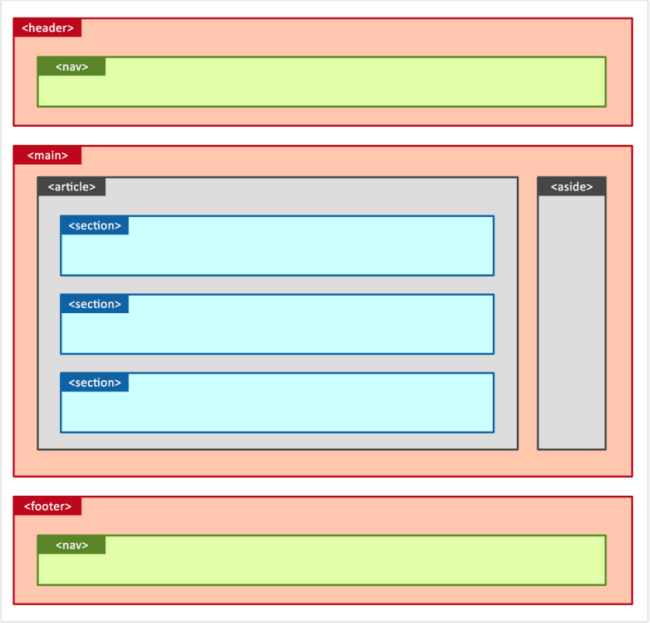

Рисунок 2.1 – Пример структуры сайта с семантической разметкой

Дополнительная информация

1. Основные семантические элементы HTML5 с примерами: https://html5css.ru/html/html5_semantic_elements.php

3 Введение в CSS

CSS (Cascading Style Sheets) — это код, который вы используете для стилизации вашей веб-страницы.

3.1 Блочная модель (Box Model)

Блочная модель — правила, по которым браузер определяет размер элемента на странице, его ширину и высоту.

Каждый элемент в CSS заключён в блок, и понимание поведения этих блоков — это ключ к умению задавать раскладку с помощью CSS, то есть выстраивать одни элементы относительно других элементов.

Блочную модель можно использовать как набор инструментов для настройки макета различных элементов. Веб-браузер отображает каждый элемент в виде прямоугольного блока в соответствии с блочной моделью CSS (рисунок 3.1).

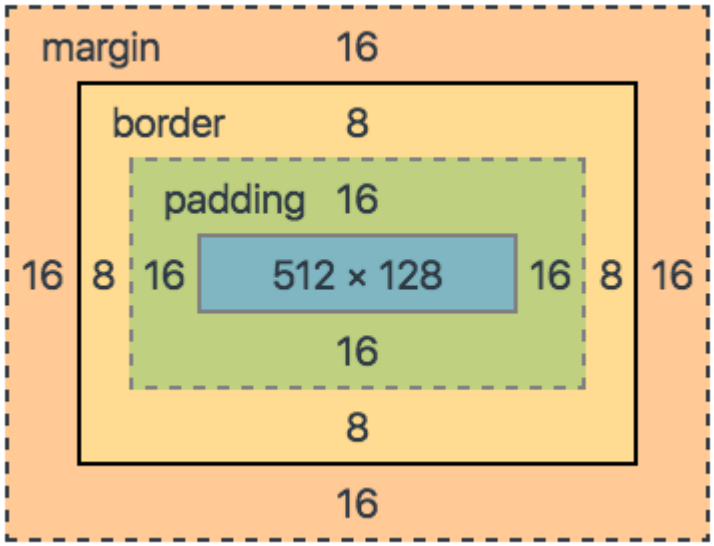

Рисунок 3.1 – Блочная модель из DevTools Google Chrome

Блочная модель состоит из нескольких CSS-свойств, влияющих на размеры элемента (рисунок 3.2):

`width` — ширина элемента;

`height` — высота элемента;

`padding` — внутренние отступы от контента до краёв элемента;

`border` — рамка, идущая по краю элемента;

`margin` — внешние отступы вокруг элемента.

Реальный размер элемента получится при сложении значений всех этих свойств.

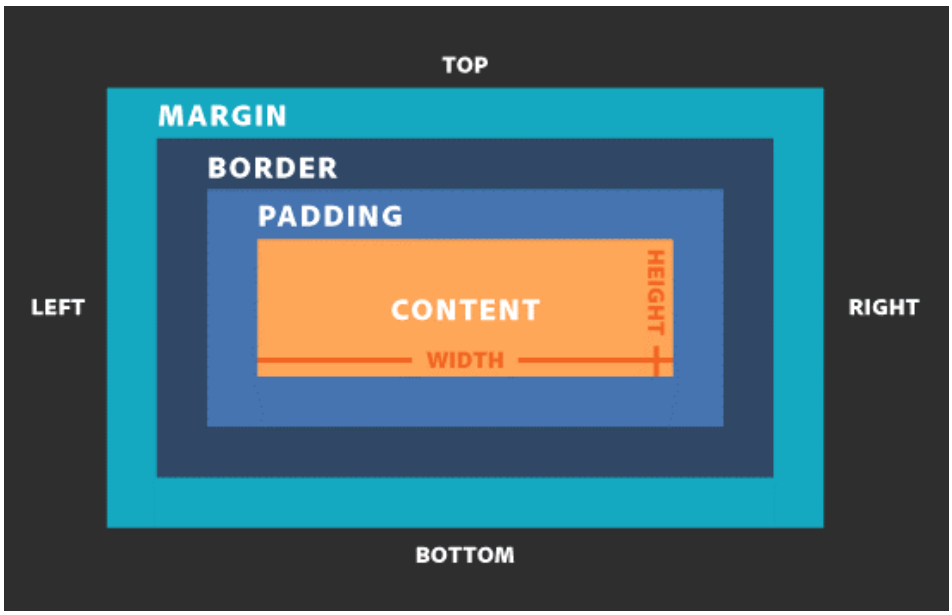

Рисунок 3.2 – Блочная модель

3.1.1 Внутренние (`padding`) и внешние (`margin`) отступы

По своим значениям, свойства `margin` и `padding` достаточно похожи, за исключением того, «куда» устанавливаются отступы. `padding` делает отступы внутри элемента, тем самым увеличивая его в размерах

В качестве значения свойство `padding` принимает четыре числа в следующей последовательности (листинг 3.1):

- **отступ сверху.** Можно установить отдельно свойством `padding-top`.
- **отступ справа.** Можно установить отдельно свойством `padding-right`.
- **отступ снизу.** Можно установить отдельно свойством `padding-bottom`.
- **отступ слева.** Можно установить отдельно свойством `padding-left`.

Листинг 3.1 – Создание внутренних отступов `padding`

```
.element {
    padding: 10px 20px 30px 40px;
}

/* Эквивалентно записи */

.element {
    padding-top: 10px;
    padding-right: 20px;
    padding-bottom: 30px;
    padding-left: 40px;
}
```

Свойство padding может принимать несколько разных вариаций сокращенных записей:

- **Одно значение** — устанавливает одинаковый отступ по всем сторонам сразу. Например, padding: 20px установит внутренний отступ в 20px сверху/справа/снизу/слева.
- **Два значения** — устанавливает отступы по вертикали и горизонтали. Например, padding: 20px 30px установит внутренний отступ в 20px сверху/снизу и 30px справа/слева.
- **Три значения** — устанавливает отступы сверху, по горизонтали и снизу. Например, padding: 20px 30px 40px установит внутренний отступ в 20px сверху, 30px справа/слева и 40px снизу.

В таблице 3.1 приведены значения для свойства margin, которые можно использовать. Для margin, как и для padding, могут использоваться отдельные свойства для каждого отступа: margin-top, margin-right, margin-bottom, margin-left.

Таблица 3.1 – Значения свойства margin

Значение	Описание
0	Установка свойства margin на значение none
auto	Используется для центрирования путем равномерной установки значений с каждой стороны
units	Единицы (например, px)
inherit	Наследование значения margin от родительского элемента
initial	Восстановление исходного значения

Примечание: если внешние отступы (margin) двух соседних элементов накладываются, то будет выбран тот, который больше, - этот эффект называется *схлопыванием внешних отступов*.

Горизонтальное центрирование элементов на странице с использованием margin

Пока элемент является блоком и имеет явно установленное значение ширины, отступы (margin) можно использовать для центрирования блочных элементов на странице по горизонтали.

Для этого можно добавить значение ширины, которое меньше, чем ширина окна, и значение `auto` свойства `margin` распределит оставшееся пространство слева и справа (листинг 3.2):

Листинг 3.2 – Центрирование элементов с `margin`

```
#exampleDiv {
    width:80%;
    margin:0 auto;
}
```

В приведенном выше примере используется сокращенное объявление `margin`, чтобы сначала установить `0` для значений верхнего и нижнего `margin` (хотя это может быть любое значение), а затем `auto`, чтобы позволить браузеру автоматически выделять пространство для значений левого и правого `margin`.

В приведенном выше примере для элемента `#exampleDiv` задана ширина `80%`, поэтому остается использовать 20% остатка. Браузер распределяет это значение на оставшиеся стороны так:

$$(100\% - 80\%) / 2 = 10\%$$

3.1.2 Границы `border` (и `outline`)

Видимые границы элемента можно задать одним из двух свойств:

- `border` - прямо влияет на блочную модель и размеры элемента.
- `outline` - рисует границу «поверх» элемента и не влияет на его размеры.

Их синтаксис похож, поэтому разберем только свойство `border`, которое является обобщенным для трех свойств (листинг 3.3):

- `border-width` - ширина границы
- `border-style` - тип границы
- `border-color` - цвет границы

Листинг 3.3 – Свойство `border`

```
#exampleDiv {
    border: 1px solid #0ff;
}
```

Свойство `border-width` тоже сокращенное. Если указано одно значение, то ширина устанавливается для всех сторон одновременно, чего в большинстве случаев достаточно. Сокращения же повторяют сокращения свойств `margin` и `padding` (листинг 3.4, 3.5).

Листинг 3.4 – Свойства для `border` по отдельности

```
#exampleDiv {
    border-width: 1px;
    border-style: solid;
    border-color: #0ff;
}
```

Листинг 3.5 – Свойство `border-top` (пример отдельного свойства)

```
#exampleDiv {
    border-top: 2px dashed #0ff;
}
```

3.1.2.1 Свойство `border-radius`

Свойство `border-radius` позволяет изменить форму базовой блочной модели.

Каждый угол элемента может иметь до двух значений вертикального и горизонтального радиуса этого угла (максимум 8 значений) (рисунок 3.3).

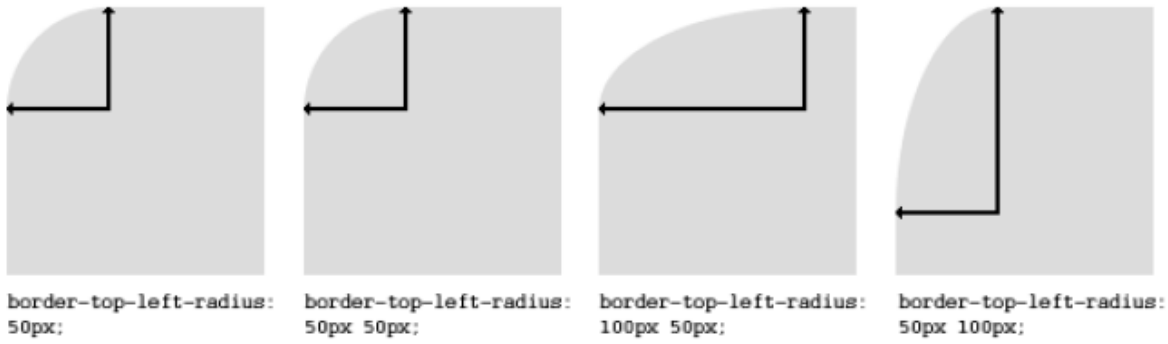

Рисунок 3.3 – Свойство `border-radius`

Первый набор значений определяет горизонтальный радиус. Необязательный второй набор значений, которому предшествует «/», определяет вертикальный радиус. Если указан только один набор значений, он используется как для вертикального, так и для горизонтального радиуса (листинг 3.6).

Листинг 3.6 – Пример набора значений для горизонтального и вертикального радиусов

```
#exampleDiv {
    border-radius: 10px 5% / 20px 25em 30px 35em;
}
```

- `10px` – это горизонтальный радиус верхнего левого и нижнего правого углов (`top-left` и `bottom-right`).
- `5%` – это горизонтальный радиус верхнего правого угла и нижнего левого (`top-right` и `bottom-left`).

Остальные четыре значения после «/» — это вертикальные радиусы для верхнего левого (`top-left`), верхнего правого (`top-right`), нижнего правого (`bottom-right`) и нижнего левого (`bottom-left`) углов.

Как и многие свойства CSS, сокращения могут использоваться для любых или всех возможных значений. Таким образом, вы можете указать от одного до восьми значений. Следующее сокращение позволяет установить для горизонтального и вертикального радиусов каждого угла одно и то же значение (листинг 3.7):

Листинг 3.7 – Пример свойства `border-radius` для каждого угла

```
.box {
    width: 250px;
    height: 250px;
    background-color: green;
    border-radius: 10px;
}
```

Свойство `border-radius` чаще всего используется для преобразования элементов блока в круги. Установив радиус границы на половину длины квадратного элемента, создается круглый элемент (листинг 3.8):

Листинг 3.8 – Пример создания круга со свойством `border-radius`

```
.circle {
    width: 200px;
    height: 200px;
    border-radius: 100px;
}
```

Поскольку `border-radius` принимает проценты, обычно используется `50%`, чтобы избежать ручного вычисления значения `border-radius` (листинг 3.9):

Листинг 3.9 – Пример создания круга со свойством `border-radius` с процентным значением

```
.circle {
    width: 150px;
    height: 150px;
    border-radius: 50%;
}
```

2.1.2.2 Свойство `border-style`

Границы могут быть разного типа: сплошные, пунктирные и т.д. За определение типа границы отвечает свойство `border-style`, которое может принимать одно из значений: `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`, `hidden`, `none` - отсутствие типа границ, можно назвать это удалением границы (рисунок 3.4).

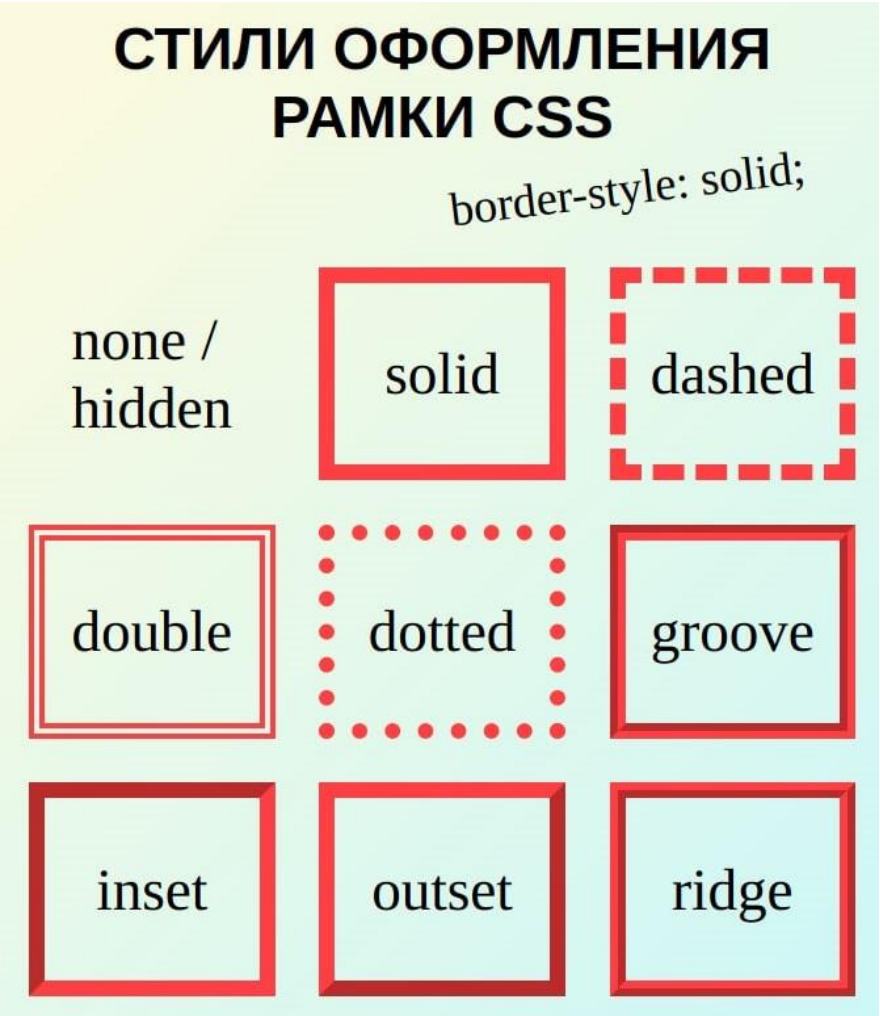

Рисунок 3.4 – Стили оформления рамки CSS

3.1.3 Свойство box-sizing

Блочная модель по умолчанию - content-box - может быть нелогичной, поскольку ширина/высота элемента не будут представлять его фактическую ширину или высоту на экране, как только вы начнете добавлять к элементу стили padding и border.

В следующем примере демонстрируется эта потенциальная проблема с content-box (листинг 3.10):

Листинг 3.10 – Блочная модель по умолчанию

```
textarea {
    width: 100%;
    padding: 3px;
    box-sizing: content-box; /* Значение по умолчанию */
}
```

Поскольку отступ (padding) будет добавлен к ширине текстовой области (textarea), результирующий элемент представляет собой текстовую область, ширина (width) которой превышает 100%.

К счастью, CSS позволяет нам изменить блочную модель с помощью свойства box-sizing для элемента. Доступны три различных значения свойства (рисунок 3.5):

- content-box: обычная блочная модель — ширина (width) и высота (height) включают только содержимое, а не отступы (padding) или границы (border).
- padding-box: ширина и высота включают содержимое и отступы, но не границу.
- border-box: ширина и высота включают содержимое, отступы и границу.

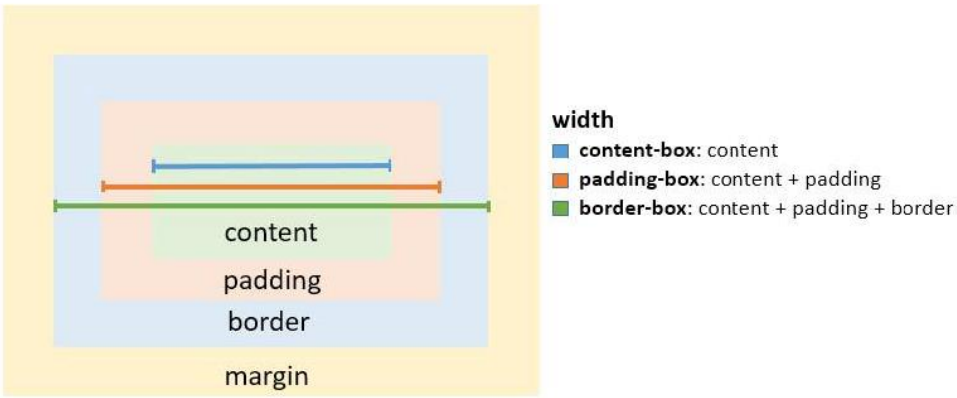

Рисунок 3.5 – Значения для свойства box-sizing

Чтобы решить вышеописанную проблему с текстовой областью, вы можете просто изменить свойство box-sizing на padding-box или border-box.

3.2 Подключение CSS к HTML

3.2.1 Внешняя таблица стилей

Внешнюю таблицу стилей (CSS) можно применить к любому количеству HTML-документов, поместив элемент `<link>` в каждый HTML-документ. Такие стили ещё называют связанными.

Атрибут `rel` тега `<link>` должен быть со значением `stylesheet,` а атрибут `href` с относительным или абсолютным путем к таблице стилей (листинг 3.11). В HTML5 атрибут `type` может быть опущен.

Листинг 3.11 – Подключение внешней таблицы стилей

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

</body>
</html>
```

Рекомендуется размещать тег `<link>` в теге `<head>` HTML-файла, чтобы стили загружались перед элементами, к которым они применяются. В противном случае пользователи увидят вспышку нестилизованного контента.

Ещё таким образом можно подключать файл стилей, который находится на другом сайте. Например, когда подключаете шрифты с Google Fonts.

Вы можете загрузить столько файлов CSS на свою HTML-страницу, сколько необходимо (листинг 3.12).

Листинг 3.12 – Пример подключения нескольких файлов CSS

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

Преимущества:

- Можно использовать один CSS-файл для нескольких страниц. Изменение стилей в таком файле автоматически применится ко всем страницам, к которым он подключён.
- При первой загрузке страницы файл со стилями кэшируется, и в следующие разы она открывается быстрее.
- Во внешних стилях можно свободно использовать *псевдоклассы* и *псевдоэлементы*. Например, задавать интерактивные состояния отдельно выбранным кнопкам, стилизовать каждый чётный элемент списка и тому подобное.

3.2.2 Внутренняя таблица стилей

CSS, заключенный в теги `<style></style>` в HTML-документе, функционирует как внешняя таблица стилей, за исключением того, что он живет в HTML-документе, который он стилизует, а не в отдельном файле, и поэтому может применяться только к документу, в котором он содержится. Обратите внимание, что этот элемент должен находиться внутри элемента `<head>` для валидности HTML (хотя он будет работать во всех текущих браузерах, если помещен в тело) (листинг 3.13).

Листинг 3.13 – Пример подключения внутренней таблицы стилей

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        h1 {
        color: blue;
        text-decoration: underline;
        }
        p {
        font-size: 25px;
        font-family: 'Georgia', 'Times New Roman', Times, serif;
        }
    </style>
</head>
<body>
    <h1>Hello world!</h1>
    <p>I ♥ CSS</p>
</body>
</html>
```

Преимущества:

- Поскольку код находится непосредственно в HTML-файле, браузер не загружает сторонние файлы. Это позволяет отрисовать страницу быстрее.
- Внутренние стили работают изолированно и применяются непосредственно к странице, на которой прописаны.
- Можно использовать псевдоклассы и псевдоэлементы.

Недостатки:

- С каждым новым правилом вес HTML-файла будет увеличиваться, и страница будет загружаться медленнее.
- Со временем такие же стили могут понадобиться на других страницах, и CSS придётся дублировать.

3.2.3 Импорт CSS

Правило `@import` используется для импорта одной таблицы стилей в другую таблицу стилей. Это правило также поддерживает запросы мультимедиа, чтобы пользователь мог импортировать таблицу стилей, зависящую от мультимедиа (листинг 3.14, 3.15). Правило `@import` должно быть объявлено в начале документа после любого объявления `@charset`.

Этот вариант тоже требует подключения внешнего файла стилей. Отличие в том, что этот файл содержит не весь CSS-код разом, а стили разбиты на отдельные файлы.

Листинг 3.14 – Подключение внешних стилей к HTML-странице

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

</body>
</html>
```

Затем в style.css импортируем несколько других CSS-файлов, header.css для шапки сайта, navbar.css для меню, и так далее:

Листинг 3.15 – Импорт отдельных CSS-файлов в style.css

```
@import url("header.css");
@import url("navbar.css");
@import url("footer.css");
```

Преимущества:

По мере того как ваш проект растёт в размерах и сложности, поддержка CSS-файлов тоже усложняется. `@import` помогает разбить массивную таблицу стилей на более мелкие и понятные части.

Недостатки:

Подключение стилей при помощи CSS-файла, в котором правила `@import` подключают другие файлы, может значительно увеличить время отрисовки страницы.

Примечание: если вам очень нужно подключить несколько файлов, лучше добавить несколько элементов `<link>`.

3.2.4 Встроенные стили

Используйте встроенные стили, чтобы применить стиль к определенному элементу. Обратите внимание, что это не оптимально. Рекомендуется размещать правила стиля в теге `<style>` или во внешнем файле CSS, чтобы сохранить различие между содержимым и представлением. Встроенные стили переопределяют любой CSS в теге `<style>` или во внешней таблице стилей. Хотя это может быть полезно в некоторых обстоятельствах, этот факт чаще всего снижает удобство сопровождения проекта. Стили в следующем примере применяются непосредственно к элементам, к которым они присоединены (листинг 3.16).

Листинг 3.16 – Пример применения встроенных стилей

```
<h1 style="color: blueviolet; text-decoration: underline;">Hello world!</h1>
<p style="font-size: 25px; font-family: 'Georgia';">I ♥ CSS</p>
```

Преимущества:

- Как и со внутренними стилями, браузеру не нужно запрашивать файл, что ускоряет рендеринг.

- Не нужно писать селекторы, потому что стили применяются сразу к нужному элементу.

Недостатки:

- У встроенных стилей наивысший приоритет, их нельзя переопределить с помощью селекторов по идентификатору, классу или тегу во внешнем CSS. Единственный способ это сделать — добавить `!important` к значению свойства. Но этот приём тоже считается плохой практикой.
- При использовании встроенных стилей необходимо добавлять их к каждому элементу, который необходимо оформить.
- Оформление при помощи встроенных стилей нарушает принцип разделения содержимого и оформления.
- Во встроенных стилях нельзя использовать *псевдоклассы* и *псевдоэлементы*.
- Если писать стили внутри атрибута `style`, то HTML становится трудно читать. Логическая структура перестаёт быть видна, стили размазываются по всему коду.

3.3 Структура и формат CSS-правил

3.3.1 Объединение значений свойств

Некоторые свойства могут принимать несколько значений (листинг 3.17).

Листинг 3.17 – Пример свойств CSS с несколькими значениями

```css
/* Два значения у свойства - список свойств */
span {
    text-shadow: yellow 0 0 3px, green 4px 4px 10px;
}

/* Альтернативный формат */
span {
    text-shadow:
        yellow 0 0 3px,
        green 4px 4px 10px;
}
```

3.3.2 Множественные селекторы (список селекторов)

Когда вы группируете селекторы CSS, вы применяете одни и те же стили к нескольким различным элементам, не повторяя стили в таблице стилей. Используйте запятую для разделения нескольких сгруппированных селекторов (листинг 3.18).

Листинг 3.18 – Группировка селекторов для единого стиля

```
div, p { color: blue }
```

Таким образом, синий цвет применяется ко всем элементам и всем элементам <p>. Без запятой только элементы <p>, являющиеся дочерними элементами , будут синими.

Это также относится ко всем типам селекторов (листинг 3.19).

Листинг 3.19 – Группировка разных типов селекторов для единого стиля

```
p, .blue, #first, div span { color : blue }
```

Справочник по CSS-свойствам: https://html5book.ru/css-spravochnik.html

---

3.4 Комментарии в CSS

В CSS комментарии в код добавляются в формате, представленном в листинге 3.20:

Листинг 3.20 – Добавление комментариев в CSS

```
.circle {
    width: 200px; /* Это CSS-комментарий */
    height: 200px;
    border-radius: 100px;
}
```

3.5 Селекторы CSS

Селекторы CSS определяют определенные элементы HTML как цели для стилей CSS. Селекторы используют более 50 методов выбора, предлагаемых языком CSS, включая элементы (element), классы (class), идентификаторы (id), псевдоэлементы и псевдоклассы, а также шаблоны.

3.5.1 Простые селекторы

Таблица 3.2 – Описание простых селекторов CSS

Селектор	Описание
`*`	Универсальный селектор (все элементы)
`div`	Селектор тегов (все элементы `<div>`)
`.blue`	Селектор класса (все элементы с классом `blue`)
`.blue.red`	Все элементы с классами `blue` и `red` (тип составного селектора)
`#headline`	Селектор идентификатора (элемент с атрибутом `id`, установленным в значении `headline`)
`:pseudo-class`	Все элементы с псевдоклассом
`::pseudo-element`	Элемент, соответствующий псевдоэлементу
`:lang(en)`	Элемент, соответствующий объявлению `:lang`, например, `<span lang="en">`
`div > p`	Дочерний селектор
`[attr=value]`	Выбирает все элементы, имеющие заданный атрибут. *Синтаксис:* `[attr]` `[attr=value]` `[attr~=value]` `[attr|=value]` `[attr^=value]` `[attr$=value]` `[attr*=value]` *Пример:* `[autoplay]` будет соответствовать всем элементам, в которых установлен атрибут `autoplay` (любое значение).

Псевдокласс — это селектор, который выбирает элементы, находящиеся в специфическом состоянии, например, они являются первым элементом своего типа, или на них наведён указатель мыши. Они обычно действуют так, как если бы вы применили класс к какой-то части вашего документа, что часто помогает сократить избыточные классы в разметке и даёт более гибкий, удобный в поддержке код.

Псевдоклассы — это ключевые слова, которые начинаются с двоеточия.

Псевдоэлементы ведут себя сходным образом, однако они действуют так, как если бы вы добавили в разметку целый новый HTML-элемент, а не применили класс к существующим элементам. Псевдоэлементы начинаются с двойного двоеточия ::.

3.5.1.1 Селектор имени класса

Селектор имени класса выбирает все элементы с именем целевого класса. Например, класс с именем `.selector` выберет следующий элемент `<div>` (листинг 3.21):

Листинг 3.21 – Добавление класса элементу (код HTML)

```html
<div class="selector">
    <p>Пример добавления класса.</p>
</div>
```

Вы также можете комбинировать имена классов для более точного определения элементов (листинг 3.22, 3.23):

Листинг 3.22 – Комбинирование имен класса (код HTML)

```html
<div class="selector">
    <p>Пример добавления класса.</p>
</div>

<div class="example selector">
    <p>Пример добавления класса.</p>
</div>
```

Листинг 3.23 – Свойства для элементов с заданными селекторами (код CSS)

```css
.example {
    color: orange;
}
.selector {
    color: blue;
}
.selector.example {
    color: green;
}
```

В этом примере все элементы с классом .selector будут иметь синий цвет текста, элементы с классом .example имеют оранжевый цвет текста, а все элементы, которые имеют имена классов .selector и .example, будут иметь зеленый цвет текста.

Обратите внимание, что в CSS объявление .selector.example не содержит пробелов между двумя именами класса. Это означает, что он найдет только те элементы, которые содержат как selector, так и example в имени класса.

Эти имена классов могут быть в любом порядке в элементе. Если бы между двумя классами в объявлении CSS был пробел, тогда были бы выбраны только те элементы, у которых есть родительские элементы с именами классов .selector и дочерние элементы с именами классов .example.

3.5.1.2 Селектор идентификатора

Идентификатор определяет уникальное имя элемента, которое используется для изменения его стиля и обращения к нему через скрипты.

Добавление идентификатора в HTML-коде происходит с помощью атрибута `id`, значением которого выступает имя идентификатора (`id="header"`). Пример показан в листинге 3.24.

Листинг 3.24 – Добавление идентификатора к элементу в HTML-коде

```
<div id="element">
    <p>Пример добавления идентификатора.</p>
</div>
```

Для изменения стиля элемента по идентификатору в CSS-коде необходимо применить следующий синтаксис – перед именем идентификатора используется символ # (листинг 3.25):

Листинг 3.25 – Обращение к идентификатору элемента в CSS-коде

```
#element {
    padding: 10px 20px 30px 40px;
}
```

Примечание: значение идентификатора (`id`) должно быть уникальным на веб-странице. Использование значения идентификатора более одного раза в одном и том же дереве документов является нарушением стандарта HTML.

3.5.2 Комбинаторы

Комбинаторы так называются, потому что они комбинируют другие селекторы таким образом, что они имеют полезную связь друг с другом и расположением содержимого в документе (таблица 3.3).

Таблица 3.3 – Основные комбинаторы селекторов

Комбинатор	Описание
`div span`	Комбинатор потомков (все `<span>`, являющиеся потомками `<div>`)
`div > span`	Комбинатор дочерних элементов (все `<span>`, которые являются прямыми дочерними элементами `<div>`)

Продолжение таблицы 3.3

`a ~ span`	Общий комбинатор родственных элементов (все `<span>`, являющиеся одноуровневыми после `<a>`)
`a + span`	Комбинатор с соседним братом (все `<span>` сразу после `<a>`)

3.5.3 Дочерние псевдоклассы

Псевдокласс CSS `:nth-child()` сопоставляет элементы на основе их положения среди группы дочерних элементов (рисунок 3.6).

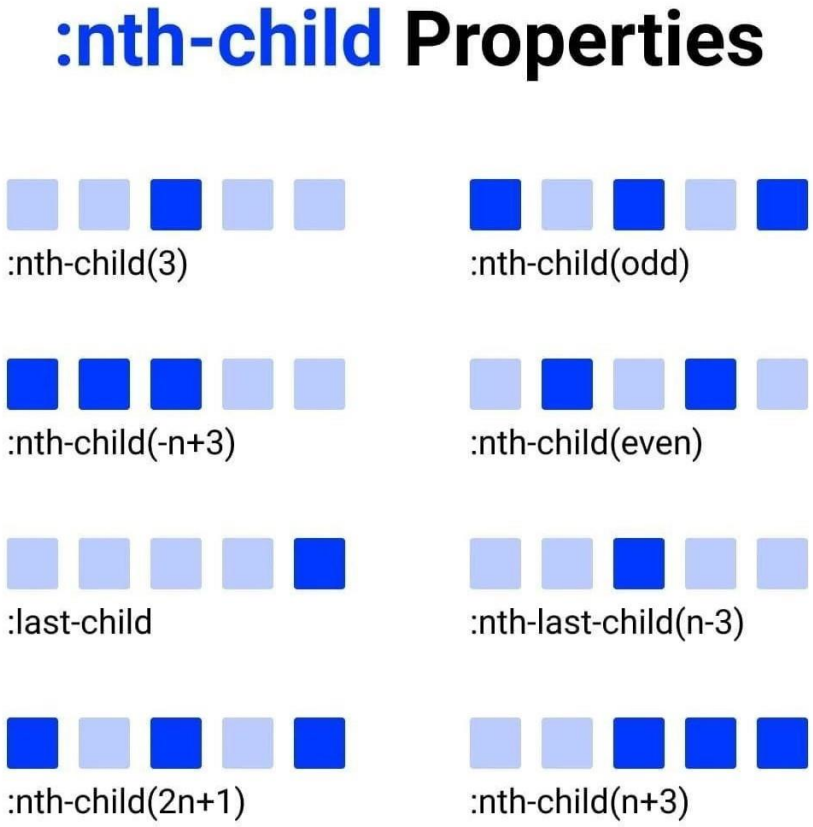

Рисунок 3.6 – Дочерние псевдоклассы

Дополнительная информация

1. Справочник по селекторам, псевдоклассам и псевдоэлементам CSS: https://basicweb.ru/css/css_selectors.php
2. Список псевдоклассов CSS: https://developer.mozilla.org/ru/docs/Web/CSS/Pseudo-classes
3. Основные виды селекторов CSS: https://learn.javascript.ru/css-selectors#osnovnye-vidy-selektorov

Практическое задание

1. На основе Практической работы №1 выполнить задание по Практической работе №2 (в работе должен быть задействован добавленный контент из Практической работы №1, который может быть дополнен):

    а) Создать «шапку» сайта с меню `<nav>` как минимум из трех пунктов.
    б) Добавить логотип сайта в «шапку».
    в) Применить тег `<main>` для разметки уникального контента на странице.
    г) Создать области секций с помощью тега `<section>`, задать секциям заголовки.
    д) Добавить независимые секции при помощи элемента `<article>`.
    е) Добавить дополняющие секции (боковую панель) при помощи тега `<aside>`.
    ж) Создать «подвал» сайта с информацией о разработчике сайта через `<footer>` (использовать в том числе тег `<address>`).

2. При помощи различных видов селекторов CSS задать оформление (шрифты, цвета, границы, отступы) созданным объектам сайта во внешнем файле style.css; также применить, как минимум, три вида отношений селекторов (обращений к дочерним и соседним элементам) при создании стилей. При помощи псевдоклассов (минимальные требования: `hover`, `active`, `visited`) задать оформление для пунктов меню сайта.

3. Стилизовать заголовки на странице с помощью Google Fonts: https://fonts.google.com/