Практическая работа №3 «Приоритет CSS-правил. Правила специфики и каскадирования. Наследование свойств в CSS. Позиционирование элементов в CSS»

3.6 Приоритет CSS-правил

3.6.1 Правила специфики

Когда к одному и тому же элементу применяется более одного набора правил CSS, браузер должен решить, какой конкретный набор будет применяться к элементу. Правила, которым следует браузер, в совокупности называются **спецификой**.

Правила специфичности включают в себя:

- Стиль CSS, примененный путем ссылки на внешнюю таблицу стилей, имеет самый низкий приоритет и переопределяется внутренним и встроенным CSS.
- Внутренний CSS переопределяется встроенным CSS.
- Встроенный CSS имеет наивысший приоритет и переопределяет все остальные селекторы.

Иерархия специфики: у каждого селектора элементов есть позиция в иерархии.

1. **Встроенный стиль:** имеет наивысший приоритет.
2. **Идентификаторы (id):** `id` (например, `#firstitem`) имеют второй наивысший приоритет.
3. **Классы** (`.listitem`)**, псевдоклассы** (`:hover, :nth-child(2)`) **и атрибуты** (`[title]`, `[colspan="2"]`)**.**
4. **Элементы и псевдоэлементы.** Элементы (например, `div`, `li`) и псевдоэлементы (например, `::before`, `::first-letter`) имеют самый низкий приоритет.

Примечания:

Когда два или более селектора имеют одинаковую специфичность, учитывается последний.

Универсальные селекторы, такие как `body` и унаследованные селекторы, обладают наименьшей специфичностью.

Универсальный селектор (`*`) и комбинаторы, такие как `>` и `~`, не имеют специфичности.

Объявление `!important` используется для переопределения обычной специфичности в таблице стилей, присваивая правилу более высокий приоритет. Его использование: `свойство: value !important`.

3.6.2 Правила каскадирования

Каскадирование и специфичность используются вместе для определения окончательного значения свойства стиля CSS. Они также определяют механизмы разрешения конфликтов в наборах правил CSS.

Порядок загрузки CSS

Стили считываются из следующих источников в указанном порядке:

1. Таблица стилей агента пользователя (стили, предоставляемые поставщиком браузера).

2. Пользовательская таблица стилей (дополнительный стиль, который пользователь установил в своем браузере).

3. Авторская таблица стилей (здесь автор является создателем веб-страницы/веб-сайта) – один или несколько файлов .css/в элементе `<style>` HTML-документа.

4. Встроенные стили (в атрибуте `style` HTML-элемента).

Браузер будет искать соответствующий стиль (стили) при рендеринге элемента.

Как разрешаются конфликты? Когда только один набор CSS-правил пытается установить стиль для элемента, тогда конфликта нет, и используется этот набор правил. При обнаружении нескольких наборов правил с конфликтующими настройками **сначала** используются **правила специфики**, а затем **правила каскадирования**, чтобы определить, какой стиль использовать.

Примечание:

- Специфика селектора всегда имеет приоритет.
- Встроенные стили превыше всего.

3.7 Наследование свойств (дополнение к практической работе №2)

В CSS не все свойства могут наследоваться.

3.7.1 Наследуемые свойства

К наследуемым относятся в основном свойства, определяющие параметры отображения текста: `font-size`, `font-family`, `font-style`, `font-weight`, `color`, `text-align`, `text-transform`, `text-indent`, `line-height`, `letter-spacing`, `word-spacing`, `white-space`, `direction` и другие.

Полный список наследуемых свойств можно найти на странице **в стандарте CSS: https://www.w3.org/TR/CSS22/propidx.html** (если в колонке *Inherited?* напротив свойства стоит *yes*, то свойство наследуемое, иначе ненаследуемое).

Также к наследуемым свойствам относятся `list-style`, `cursor`, `visibility`, `border-collapse` и некоторые другие. Но они используются значительно реже.

Наследуемые свойства можно и нужно задавать через предков, следуя структуре документа.

Например, параметры текста зачастую не меняются в пределах крупных блоков страницы: меню, основного содержания, информационных панелей. Поэтому общие параметры текста (цвет, размер, гарнитура) обычно указывают в стилях этих крупных блоков.

3.7.2 Ненаследуемые свойства

Основные ненаследуемые свойства — это параметры позиционирования, размеров, отступов, фона, рамок: `background`, `border`, `padding`, `margin`, `width`, `height`, `position` и другие.

Не наследуются они из соображений здравого смысла. Например, если для какого-либо блока установлен внутренний отступ, автоматически выставлять такой же отступ каждому вложенному элементу нет никакой надобности. Эти параметры чаще всего уникальны для каждого отдельного блока.

3.8 Позиционирование элементов в CSS

3.8.1 CSS-свойство `float` (обтекание элемента)

CSS-свойство `float` используется для позиционирования элементов слева и справа от его контейнера, а также позволяет обтекать его текстом и встроенными элементами.

Самое простое использование `float` — это обтекание изображения текстом. Обратите внимание, что в HTML-коде элемент (текст), который обтекает плавающий элемент (изображение), всегда находится после плавающего элемента (листинг 3.8.1, 3.8.2 и рисунок 3.8.1).

Свойство `float` определяет поток содержимого на странице. Остальные элементы будут частью потока, если элемент будет удален из обычного потока содержимого. Это свойство игнорируется абсолютно позиционированными элементами. Он может быть записан в файле CSS или может быть указан непосредственно в стиле элемента.

Синтаксис:

```
float: none|left|right|initial|inherit;
```

Значения свойства:

- `none`: это значение по умолчанию, и элемент не «плавает».
- `left`: элемент «плавает» на левой стороне контейнера.
- `right`: элемент «плавает» с правой стороны контейнера.
- `initial`: значение по умолчанию.
- `inherit`: элемент наследует «плавающее» свойство своего родительского свойства.

Листинг 3.8.1 – Пример создания элементов для применения свойства `float` в HTML

```
<div style="width:30%">
<p>Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Integer nec odio. Praesent libero. Sed cursus
ante dapibus diam. Sed nisi. Nulla quis sem at nibh
elementum imperdiet. Duis sagittis ipsum. Praesent
mauris. Fusce nec tellus sed augue semper porta.
Mauris massa. Vestibulum lacinia arcu eget nulla.
</p>

<img src="адрес картинки" />

<p>Class aptent taciti sociosqu ad litora torquent
per conubia nostra, per inceptos himenaeos. Curabitur
sodales ligula in libero. Sed dignissim lacinia nunc.
Curabitur tortor. Pellentesque nibh. Aenean quam. In
scelerisque sem at dolor. Maecenas mattis. Sed
convallis tristique sem. Proin ut ligula vel nunc
egestas porttitor. Morbi lectus risus, iaculis vel,
suscipit quis, luctus non, massa. Fusce ac turpis
quis ligula lacinia aliquet. </p>

</div>
```

Листинг 3.8.2 – Пример добавления свойства `float` элементу изображения в CSS

```
img {
  float:left;
  margin-right:1rem;
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet.

Рисунок 3.8.1 – Результат применения свойства `float` к изображению

3.8.2 Методы позиционирования элементов в CSS

Свойство `position` в CSS сообщает о методе позиционирования элемента или объекта HTML. В CSS доступно **пять различных типов свойства `position`**:

— `fixed`

— `static`

— `relative`

— `absolute`

— `sticky`

Позиционирование элемента может быть выполнено с помощью свойств `top`, `right`, `bottom` и `left`. Они определяют расстояние элемента HTML от края области просмотра *(viewport)*. Чтобы установить позицию по этим четырем свойствам, мы должны объявить метод позиционирования. Давайте подробно разберем каждый из этих методов позиции:

1. `Fixed`: любой элемент HTML со свойством `position: fixed` будет позиционироваться относительно окна просмотра *(viewport)*. Элемент с фиксированным позиционированием остается на той же позиции, даже если мы прокручиваем страницу. Мы можем установить `position` элемента, используя `top`, `right`, `bottom` и `left`.
2. `Static`: этот метод позиционирования установлен по умолчанию. Если мы не указываем метод позиционирования для любого элемента, то по умолчанию будет стоять `static`. Определив `static`, `top`, `right`, `bottom`

и left не будут иметь никакого контроля над элементом. Элемент будет позиционироваться в соответствии с обычным потоком страницы.

3. Relative: элемент с position: relative позиционируется относительно других элементов, которые находятся над ним. Если мы установим его top, right, bottom или left другие элементы не будут заполнять пробел, оставленный этим элементом. Элемент с заданным относительным положением и применением top, right, bottom и left будет позиционирован относительно его исходного положения.
4. Absolute: элемент с position: absolute будет позиционироваться относительно своего ближайшего нестатического предка. Позиционирование этого элемента не зависит от его братьев и сестер или элементов, которые находятся на том же уровне.
5. Sticky: элемент с position: sticky и top: 0 играет роль между фиксированным и относительным в зависимости от положения, в котором он находится. Если элемент размещен в середине документа, то, когда пользователь прокручивает документ, «липкий» элемент начинает прокручиваться, пока не коснется верхней части. Когда он коснется вершины, он будет зафиксирован на этом месте, несмотря на дальнейшую прокрутку. Мы можем прикрепить элемент внизу с помощью свойства bottom.

Перекрывающиеся элементы с `z-index`

Чтобы изменить порядок расположения элементов в стеке по умолчанию (для свойства position установлено relative, absolute или fixed значение), используйте свойство z-index. Чем выше z-index, тем выше в контексте стека (по оси z) он расположен (листинги 3.8.3 и 3.8.4, рисунок 3.8.2).

Синтаксис:

```
z-index: [ number ] | auto;
```

number - целочисленное значение. Большее число выше в стеке z-index.

0 - это значение по умолчанию. Допускаются отрицательные значения.

Листинг 3.8.3 – Создание блоков для применения свойства z-index

```HTML
<div id="div1"></div>
<div id="div2"></div>
<div id="div3"></div>
```

Листинг 3.8.4 – Добавление стиля к блокам `div` с `z-index`

```css
div {
  position: absolute;
  height: 200px;
  width: 200px;
}
div#div1 {
  z-index: 1;
  left: 0px;
  top: 0px;
  background-color: blue;
}
div#div2 {
  z-index: 3;
  left: 100px;
  top: 100px;
  background-color: green;
}
div#div3 {
  z-index: 2;
  left: 50px;
  top: 150px;
  background-color: red;
}
```

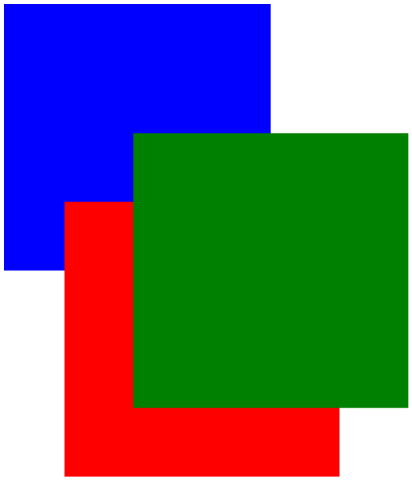

Рисунок 3.8.2 – Отображение элементов с помощью `z-index`

Примечание: все элементы размещаются в 3D-оси в CSS, включая ось глубины, измеряемую свойством `z-index`. Оно работает только с позиционированными элементами. Единственное значение, где оно игнорируется, — это значение по умолчанию, `static`.

Прочтите о свойстве `z-index` и контекстах стекирования в спецификации CSS (https://drafts.csswg.org/css-position/#layered-presentation) по многоуровневому представлению и в Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Web/CSS/z-index.

3.8.3 Управление макетом

Представьте себе процесс строительства дома. Условно его можно разделить на несколько этапов: возведение структуры дома и его отделка. При возведении дома мы заливаем фундамент, возводим стены, устанавливаем крышу. После этого уже переходим к покраске дома, устанавливаем окна и занимаемся декорированием.

Процесс создания верстки можно описать похожими процессами. В ней так же важна роль возведения структуры и отделки. Для этого существуют несколько основных типов элементов:

Блочные элементы (Block level). Эти элементы отвечают за каркас страницы.

Строчные элементы (Inline level). Они помогают нам в процессе стилизации страницы или добавления функциональных частей.

Важно: сами по себе элементы HTML не являются блочными или строчными. HTML — всего лишь разметка. За то, будет элемент блочным или строчным, отвечает CSS.

3.8.3.1 Блочные элементы

Основная особенность блочных элементов — они занимают всю доступную им ширину. Из-за этого другие элементы до и после блочных элементов не встают в один ряд с ними, а располагаются до или после них, но уже на другой строке.

Стоит отдельно отметить понятие «занимает всю доступную ширину». Доступная ширина — вся доступная ширина родителя. То есть, если наш блок лежит внутри `<body>`, то эта ширина будет равна именно ширине `<body>`. Стоит изменить ширину `<body>`, так сразу изменится и ширина блочного элемента внутри.

Блочных элементов очень много. Основными из них являются:

- `<div>`
- `<p>`
- Теги списков: `<ul>/<ol>/<li>`
- Заголовки: `<h1>/<h2>/<h3>/<h4>/<h5>/<h6>`

Пример создания блочных элементов (листинг 3.8.5):

Листинг 3.8.5 – Создание блочных элементов `div`

```
<div id="div1">Block 1</div>
<div id="div2">Block 2</div>
<div id="div3">Block 3</div>
```

Внутрь блочных элементов мы можем вкладывать другие блочные элементы. Это позволяет нам делать сложную верстку, оперируя различными компонентами, которые будут созданы.

Важно: хоть HTML и позволяет бесконечно вкладывать блоки друг в друга, лучше сохранять здравый смысл. Дело в том, что процесс вывода верстки достаточно долгий для браузеров. И чем больше вложенность блоков друг в друга, тем сложнее браузеру и компьютеру все это обработать.

Вложенность блочных элементов друг в друга ограничена только несколькими пунктами:

1. Нельзя вкладывать заголовки в заголовки
2. Нельзя вкладывать параграфы в параграфы

Для создания блочного элемента используется тег `<div>`. Он не имеет никаких дополнительных стилей, кроме блочного отображения. Этим тегом оборачивается связанная информация, создаются каркасы компонентов. Но тег не является семантическим, то есть не несет смысловой нагрузки.

3.8.3.2 Строчные элементы

Строчные элементы являются отделочным материалом, с помощью которого мы можем выделить участки текста, или добавить немного логики, как в случае со ссылками. Строчные элементы противоположны блочным. Они не занимают всю доступную ширину и из-за этого не происходит переноса соседних элементов. Помимо этого, есть важная особенность, связанная с применением свойств `width` и `height` в CSS. **Для строчных элементов эти свойства не работают.**

Наиболее часто используемыми строчными элементами являются:

- `<span>`
- `<a>`
- Теги выделения текста: `<i>/<em>/<b>/<strong>`

Так же, как и блочные элементы, мы можем вкладывать строчные элементы внутрь строчных.

Важно: не вкладывайте внутрь строчных элементов блочные. Это нарушает семантику и усложняет чтение кода

3.8.4 Свойство `display`

Свойство `display` в CSS определяет, как компоненты (div, гиперссылка, заголовок и т.д.) будут размещены на веб-странице. Как следует из названия, это свойство используется для определения отображения различных частей веб-страницы (таблица 3.8.1).

Таблица 3.8.1 – Описание значений свойства `display`

Значение свойства	Описание значения
`inline`	Отображение элемента как встроенного элемента.
`block`	Отображение элемента как блочного элемента.
`contents`	Используется для исчезновения контейнера.
`flex`	Отображение элемента в виде гибкого контейнера на уровне блока.
`grid`	Отображение элемента в виде контейнера сетки на уровне блока.
`inline-block`	Отображение элемента в виде блочного контейнера встроенного уровня.
`inline-flex`	Отображение элемента в виде гибкого контейнера встроенного уровня.
`inline-grid`	Отображение элемента в виде контейнера сетки встроенного уровня.
`inline-table`	Отображение таблицы встроенного уровня.
`list-item`	Отображение всех элементов в элементе `<li>`.
`run-in`	Отображение элемента на встроенном или блочном уровне, в зависимости от контекста.
`table`	Установка поведения как `<table>` для всех элементов.
`table-caption`	Установка поведения как `<caption>` для всех элементов.
`table-column-group`	Установка поведения как `<column>` для всех элементов.
`table-header-group`	Установка поведения как `<header>` для всех элементов.
`table-footer-group`	Установка поведения как `<footer>` для всех элементов.
`table-row-group`	Установка поведения как `<row>` для всех элементов.
`table-cell`	Установка поведения как `<td>` для всех элементов.
`table-column`	Установка поведения как `<col>` для всех элементов.
`table-row`	Установка поведения как `<tr>` для всех элементов.
`none`	Используется для удаления элемента.
`initial`	Установка значения по умолчанию.
`inherit`	Наследование свойства от родительских элементов.

Несколько важных значений описаны ниже с примером.

`block`: это значение используется как значение по умолчанию для `div`. Это значение размещает элементы `div` один за другим по вертикали (листинг 3.8.6 и рисунок 3.8.3).

Листинг 3.8.6 – Применение свойства `display` со значением `block`

```
<!DOCTYPE html>
<html>
    <head>
        <title>CSS | Display property</title>
        <style>
            #div1{
                height: 100px;
                width: 200px;
                background: yellow ;
                display: block;
            }
            #div2{
                height: 100px;
                width: 200px;
                background: cyan;
                display: block;
            }
            #div3{
                height: 100px;
                width: 200px;
                background: orange;
                display: block;
            }
            .divblock {
                font-size:25px;
                margin-top: 70px;
                margin-left: 70px;
            }
            .main {
                margin:50px;
                text-align:center;
            }
        </style>
    </head>

    <body>
        <div class = "divblock">display: block</div>
        <div class = "main">
            <div id="div1">Block 1</div>
            <div id="div2">Block 2</div>
            <div id="div3">Block 3</div>
        </div>
    </body>
</html>
```

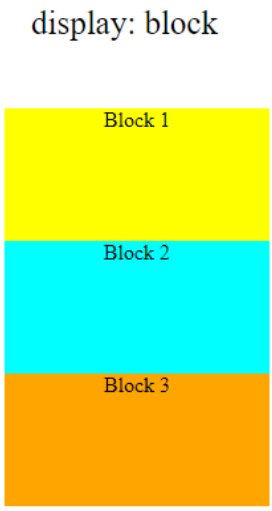

Рисунок 3.8.3 – Результат применения к `div`-элементу свойства `display` со значением `block`

`inline`: используется для размещения `div` в строке, то есть горизонтально. Значение встроенного отображения игнорирует высоту и ширину, заданные пользователем (листинг 3.8.7 и рисунок 3.8.4).

Листинг 3.8.7 – Применение свойства `display` со значением `inline`

```
    <style>
        #div1{
            height: 200px;
            width: 200px;
            background: yellow;
            display: inline;
        }
        #div2{
            height: 200px;
            width: 200px;
            background: cyan;
            display: inline;
        }
        #div3{
            height: 200px;
            width: 200px;
            background: orange;
            display: inline;
        }
        .divblock {
            font-size:25px;
            margin-left:60px;
        }
        .main {
            margin:50px;
        }
    </style>
</head>

<body>
    <div class = "divblock">display: inline</div>
    <div class = "main">
        <div id="div1">Block 1</div>
        <div id="div2">Block 2</div>
        <div id="div3">Block 3</div>
    </div>
</body>
```

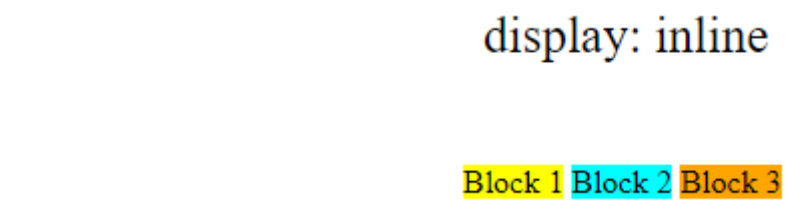

Рисунок 3.8.4 - Результат применения к `div`-элементу свойства `display` со значением `inline`

`inline-block`: эта функция использует оба значения, упомянутые выше, `block` и `inline`. Это значение выравнивает `div` внутри строки, но разница в том, что оно может редактировать высоту и ширину блока. По сути, это выровняет `div` как в блочном, так и во встроенном виде (листинг 3.8.8 и рисунок 3.8.5).

Листинг 3.8.8 – Применение свойства `display` со значением `inline-block`

```
    <style>
        #div1{
            height: 100px;
            width: 200px;
            background: yellow;
            display: inline-block;
        }
        #div2{
            height: 100px;
            width: 200px;
            background: cyan;
            display: inline-block;
        }
        #div3{
            height: 100px;
            width: 200px;
            background: orange;
            display: inline-block;
        }
        .divblock {
            font-size:25px;
            margin-left:250px;
        }
        .main {
            margin:50px;
        }
    </style>
</head>

<body>
    <div class = "divblock">display: inline-block</div>
    <div class = "main">
        <div id="div1">Block 1</div>
        <div id="div2">Block 2</div>
        <div id="div3">Block 3</div>
    </div>
</body>
```

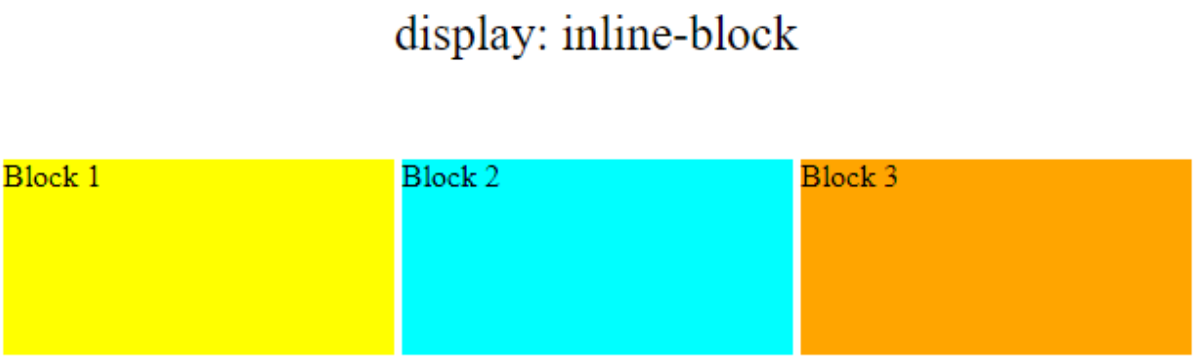

Рисунок 3.8.5 - Результат применения к `div`-элементу свойства `display` со значением `inline-block`

`none`: это значение скрывает `div` или контейнер, которые используют это значение.

Практическое задание

Практическую работу №3 можно также продолжать выполнять на основе Практических работ №1-2 (**можно добавлять контент на страницу, который непосредственно относится к Вашей курсовой работе, и удалять наполнение из предыдущих работ**).

Требования к выполнению практической работы:

1. В работе необходимо **задействовать 4 значения свойства** `position` (`relative, absolute, fixed, sticky`) для различных элементов веб-страницы с установлением параметров ширины, высоты и отступов (там, где возможно применение данных свойств).
2. Задать `position` для `navigation` (меню сайта) и логотипа/названия веб-страницы.
3. Использовать свойство `display` в меню сайта с заданием для самого меню таких параметров, как ширина, высота, отступы (`padding`, `margin`), границы – если ранее не было добавлено; `text-decoration`, `list-style` и т.д. (если ранее не использовались).
4. Использовать свойство `display` для блоков, созданных в области контента страницы.
5. Применить свойство `float` с разными значениями к нескольким созданным блокам на сайте (для обтекания текста).