**Теория к практическому занятию №11.**

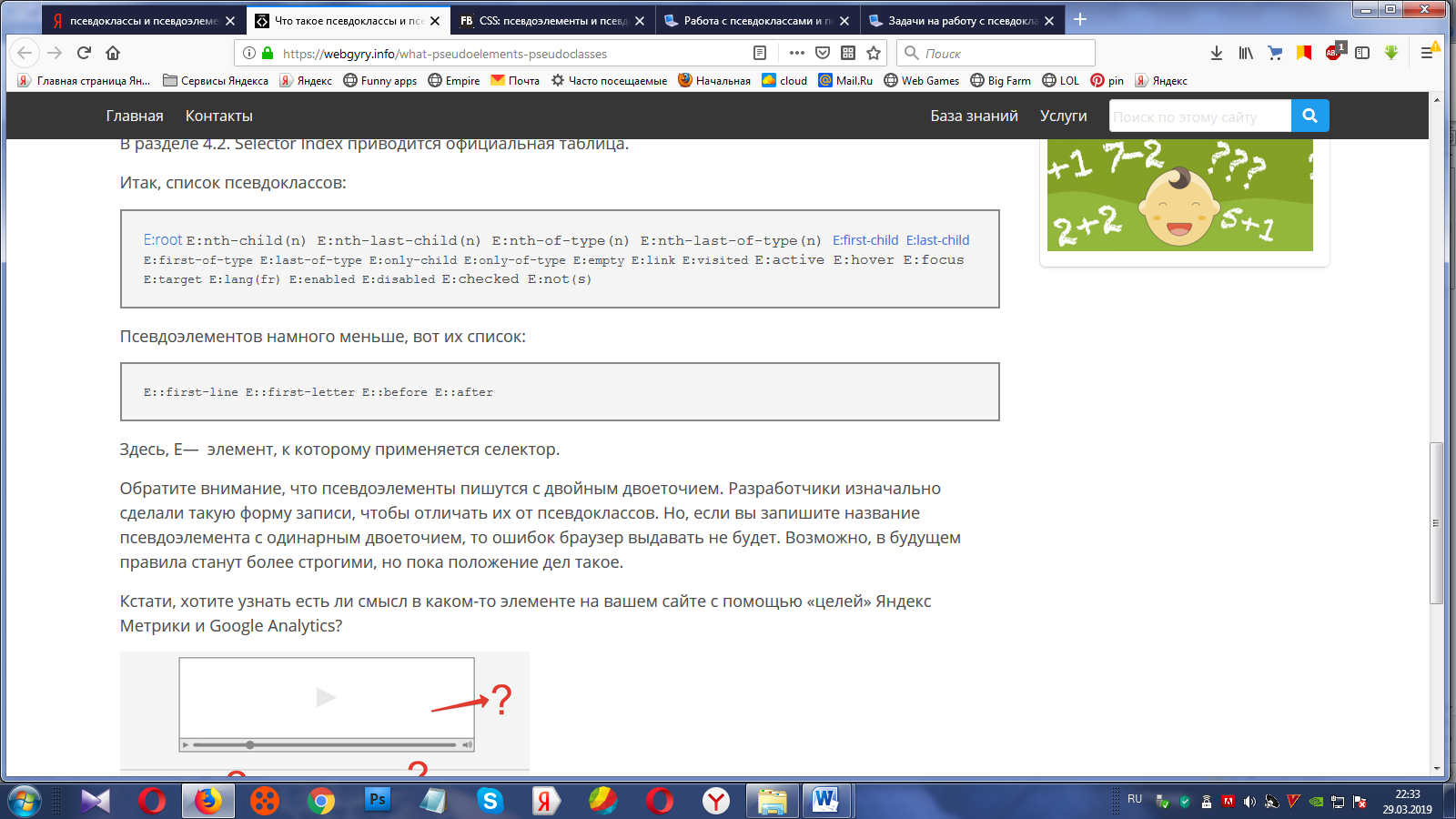
**Работа с псевдоклассами и псевдоэлементами**

**Псевдоклассы** – это селекторы, которые определяют **состояние** уже существующих элементов, которое может меняться при определенных условиях.

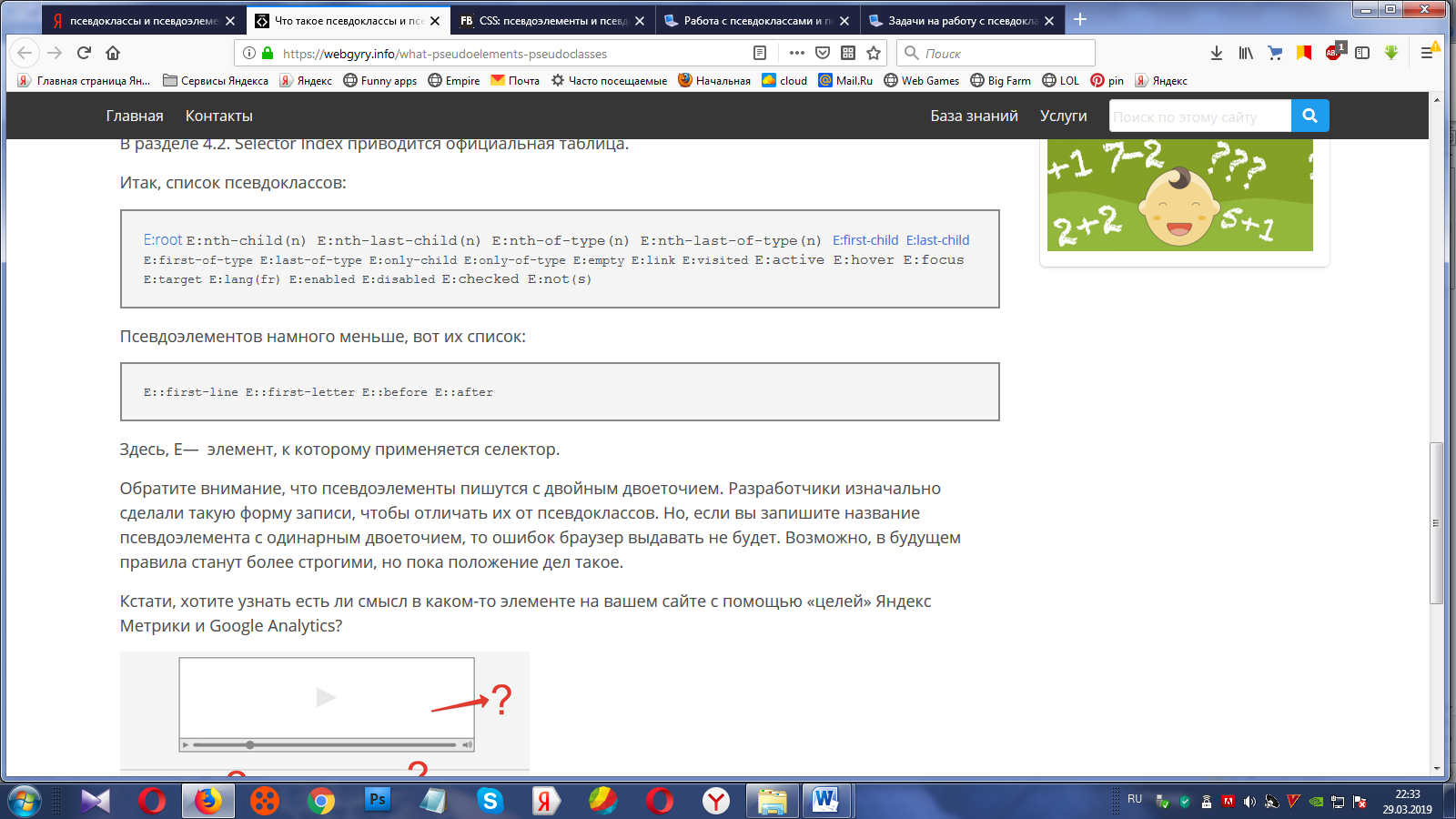
**Псевдоэлементы** – это селекторы, которые определяют **область элементов**, которая изначально отсутствует в дереве документа. Эта область создается искусственно с помощью CSS.

Ключевое отличие между ними в том, что псевдоклассы определяют именно состояние элементов, которые уже существуют на странице, а псевдоэлементы создают области (искусственные элементы), которых изначально на веб-странице не было. Но и те и другие отсутствуют в исходном коде документа.

Список псевдоклассов:



Псевдоэлементов намного меньше, вот их список:



E—  элемент, к которому применяется селектор.

**Блок 1.**На **first-letter**

Псевдоэлемент **first-letter** позволяет задать стили первой букве текста в элементе.

**Синтаксис**

селектор::first-letter{

}

**Пример**

Обратимся к первым буквам каждого абзаца:

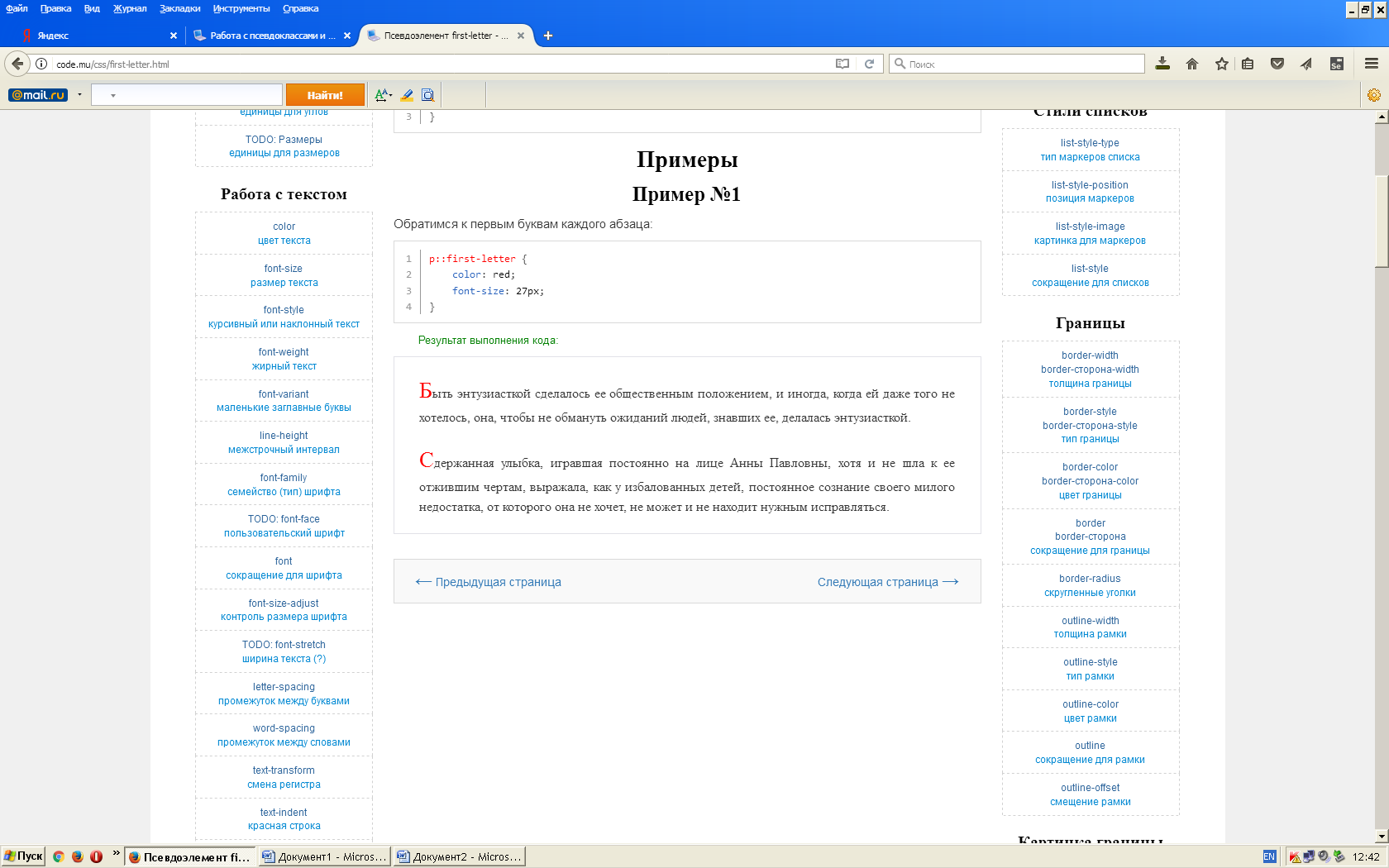
p::first-letter{

color:red;

font-size: 27px;

}

Результат выполнения кода:



**Блок 2. На first-line**

Псевдоэлемент **first-line** позволяет задать стили первой линии текста.

## Синтаксис

селектор::first-line{

}

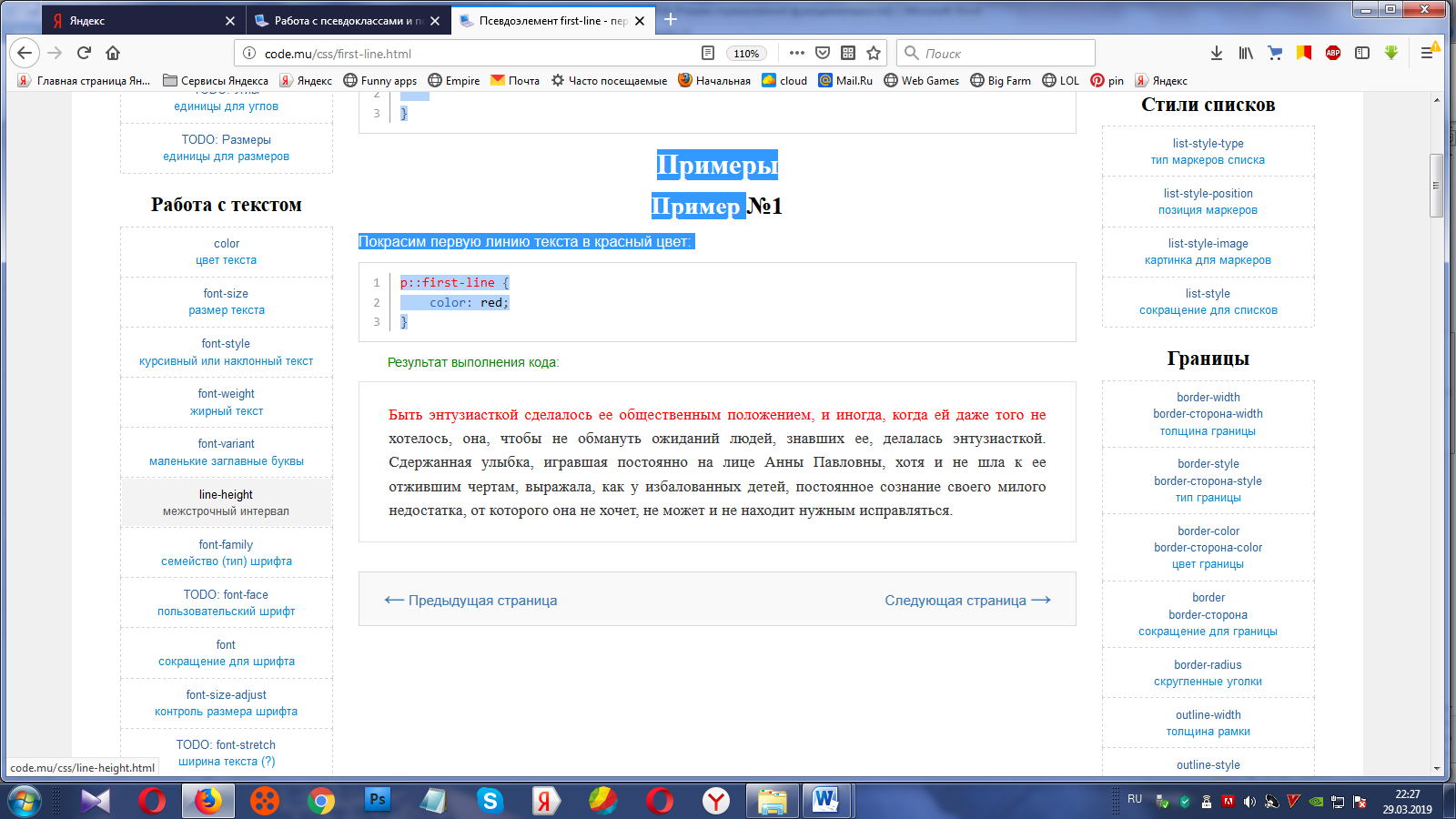
### Пример

Покрасим первую линию текста в красный цвет:

p::first-line{

color:red;

}



**Блок 3. На selection**

Псевдоэлемент **selection** позволяет задать стили выделенному тексту (**только цвет и фон**).

## Синтаксис

селектор::selection{

}

### Пример

Выделите текст и вы увидите, что он будет красный на черном фоне. К сожалению, для кроссбраузерности нужно писать 3 раза (объединить через запятую вместе **нельзя**, не будет работать):

p::-moz-selection{

color:red;

background-color:black;

}

p::-webkit-selection{

color:red;

background-color:black;

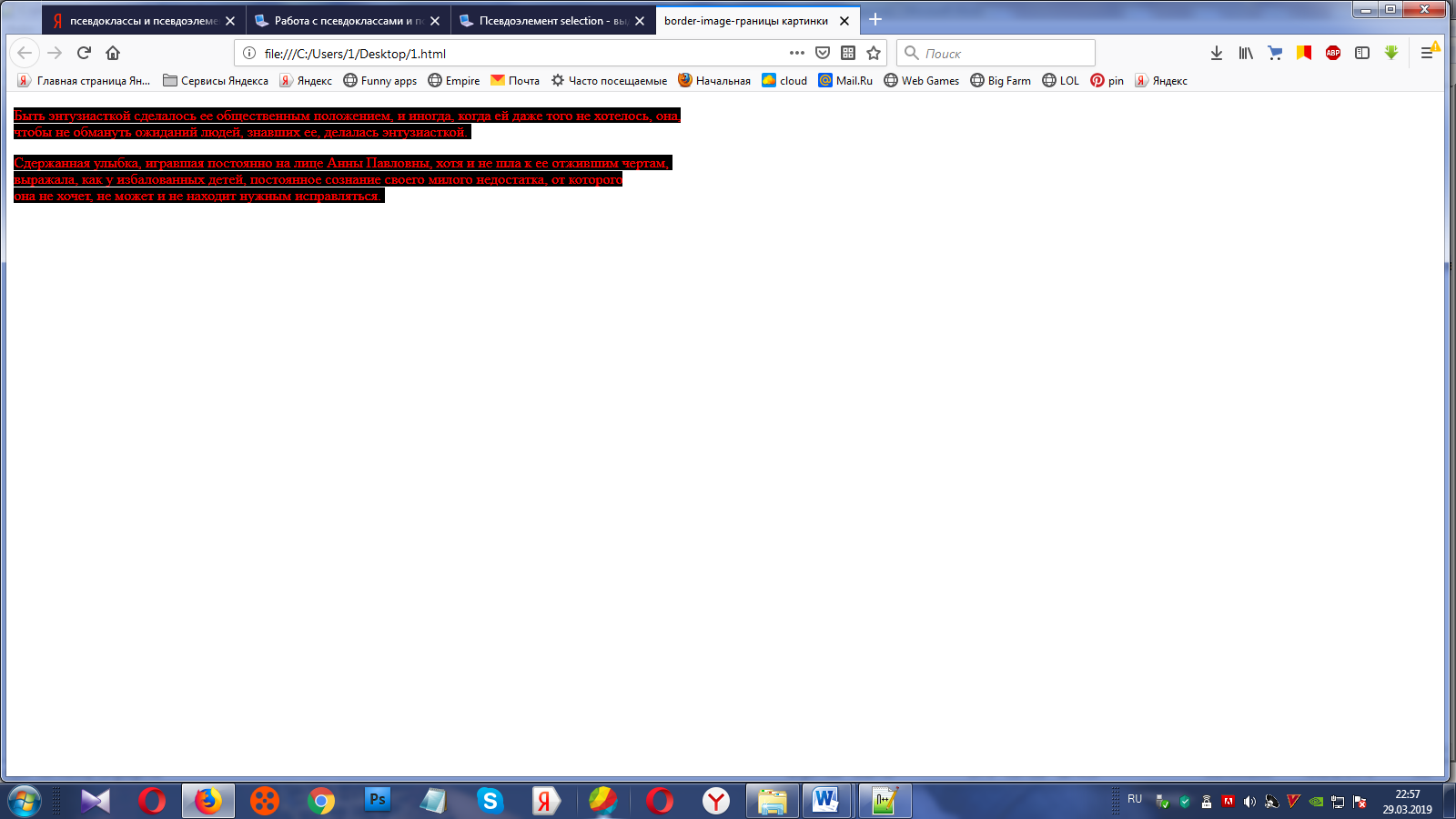
}

p::selection{

color:red;

background-color:black;

}



**Блок 4. На placeholder**

Псевдоэлемент**placeholder** задает стили для подсказки placeholder в тегах input или textarea.

Подсказка **placeholder** нужна для того, чтобы указать пользователю, что именно следует вводить в данное поле формы. Подробнее см. по ссылке placeholder.

Если цвет подсказки **placeholder** не задан через псевдоэлемент**placeholder**, то он будет такой же, какой задан для всего элемента через свойство color, но более бледный, так как **по умолчанию браузер добавляет полупрозрачность** подсказке **placeholder**. **Отменить** ее можно можно через свойство opacity для псевдоэлемента**placeholder** (не для элемента!).

## Синтаксис

input::placeholder{

color:red;

}

### Пример

В данном примере подсказка покрашена в красный цвет. Обратите внимание на то, что **для кроссбраузерности добавлено гораздо больше кода**, чем вы могли бы ожидать. Кроме того, обратите внимание на то, что цвет получился бледным (так как он по умолчанию полупрозрачный):

<input type="text" placeholder="Я подсказка:)">

input:-moz-placeholder{

color:red;

}

input::-moz-placeholder{

color:red;

}

input::-webkit-input-placeholder{

color:red;

}

input:-ms-placeholder{

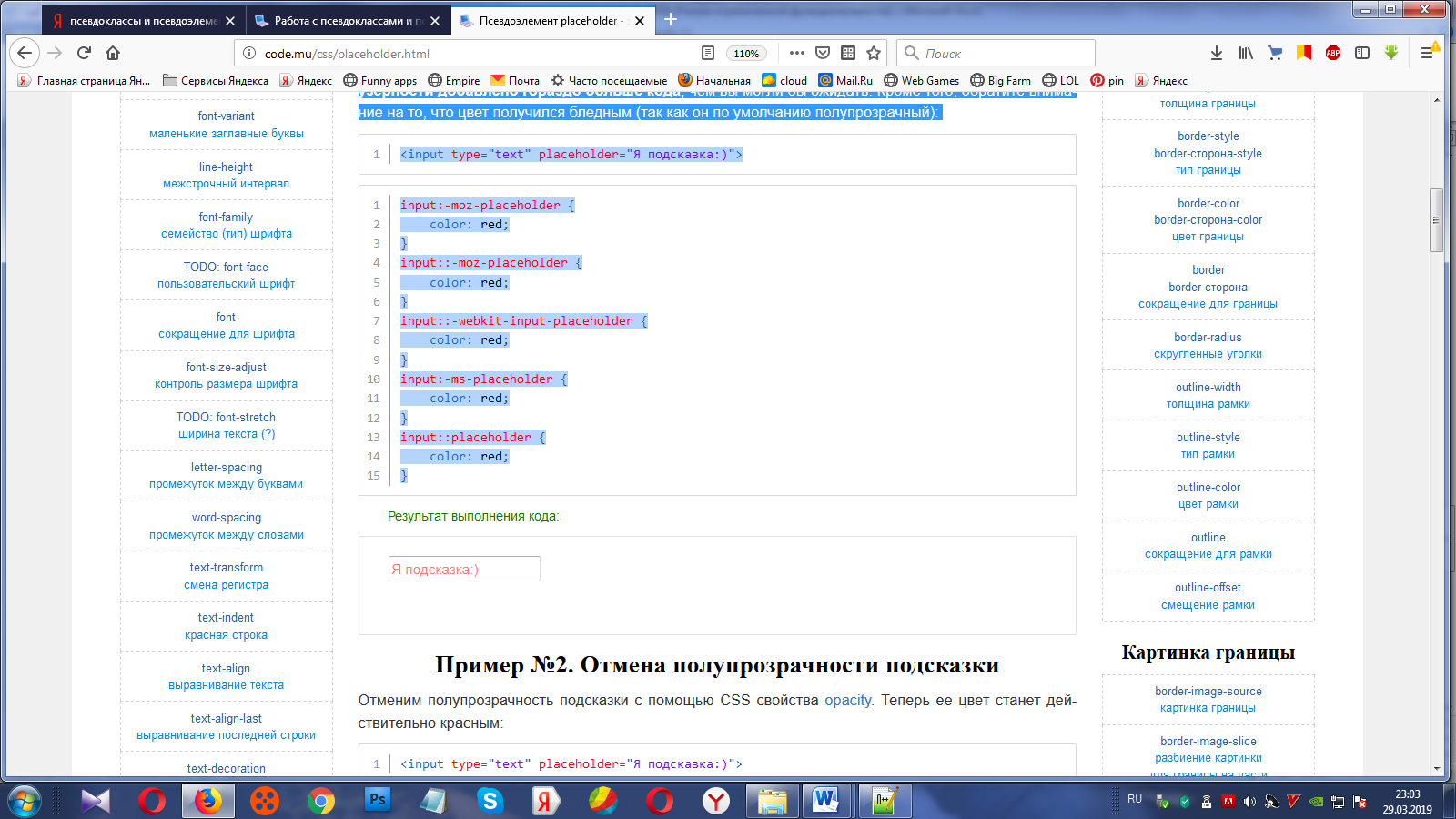
color:red;

}

input::placeholder{

color:red;

}



**Блок 5. На focus**

Псевдокласс**focus** задает стили для инпута, текстареа или ссылки с фокусом ввода. **Фокус ввода** - это когда мы нажали на инпут и там моргает курсор-палочка (можно вводить текст).

**Синтаксис**

input:focus {

}

**Пример**

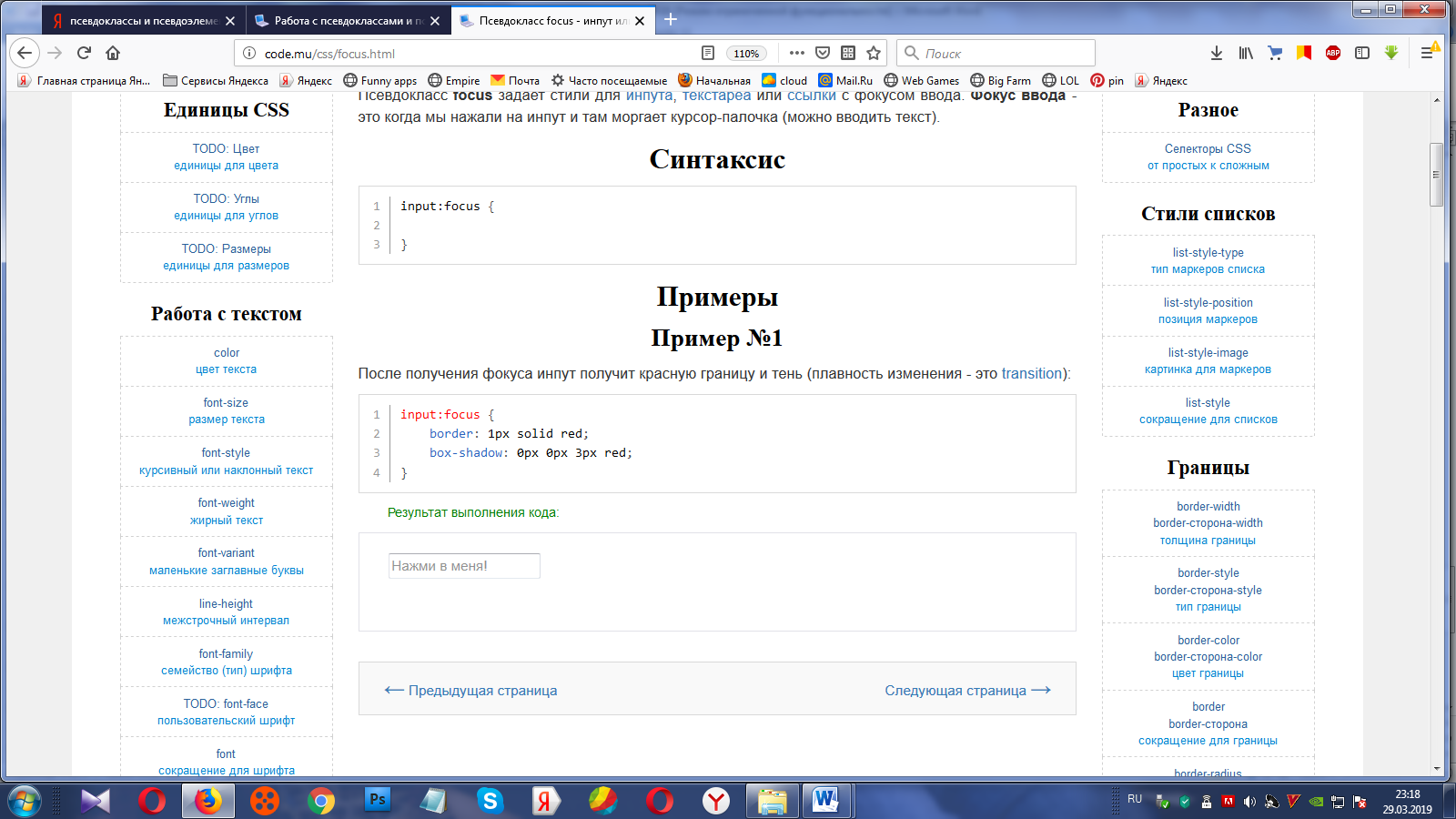
После получения фокуса инпут получит красную границу и тень (плавность изменения - это transition):

input:focus {

border: 1px solid red;

box-shadow: 0px 0px 3px red;

}



**Блок 6. На after, before, content**

Псевдоэлемент **after** вставляет текст после элемента. Данный псевдоэлемент используется **только** совместно со свойством content, которое задает текст, который следует вставить.

**Синтаксис**

селектор::after {

}

**Пример**

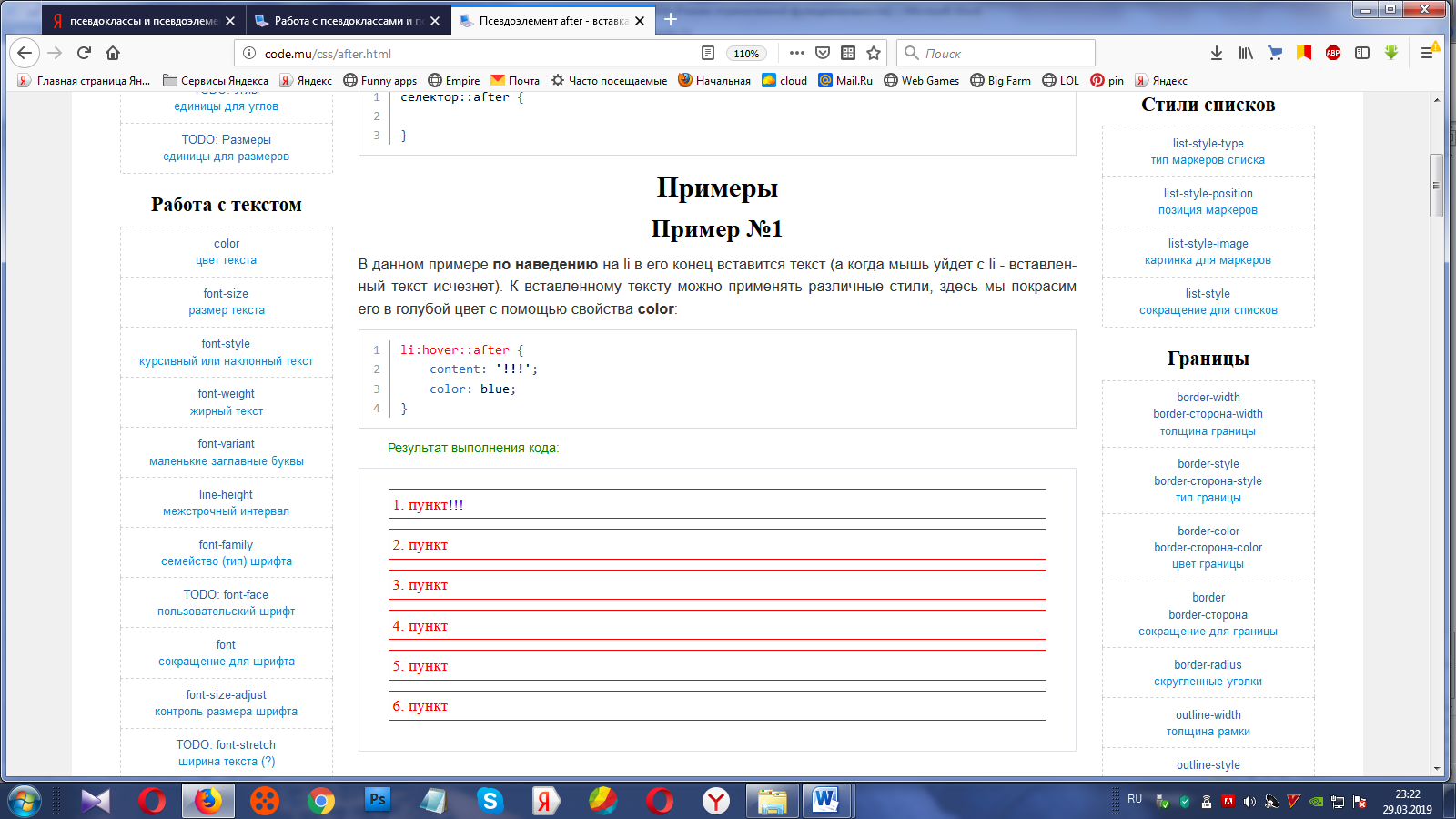
В данном примере **по наведению** на li в его конец вставится текст (а когда мышь уйдет с li - вставленный текст исчезнет). К вставленному тексту можно применять различные стили, здесь мы покрасим его в голубой цвет с помощью свойства **color**:

li:hover::after {

content: '!!!';

color: blue;

}



Псевдоэлемент **before** вставляет текст перед элементом. Данный псевдоэлемент используется **только** совместно со свойством **content**, которое задает текст, который следует вставить.

## Синтаксис

селектор::before{ }

### Пример

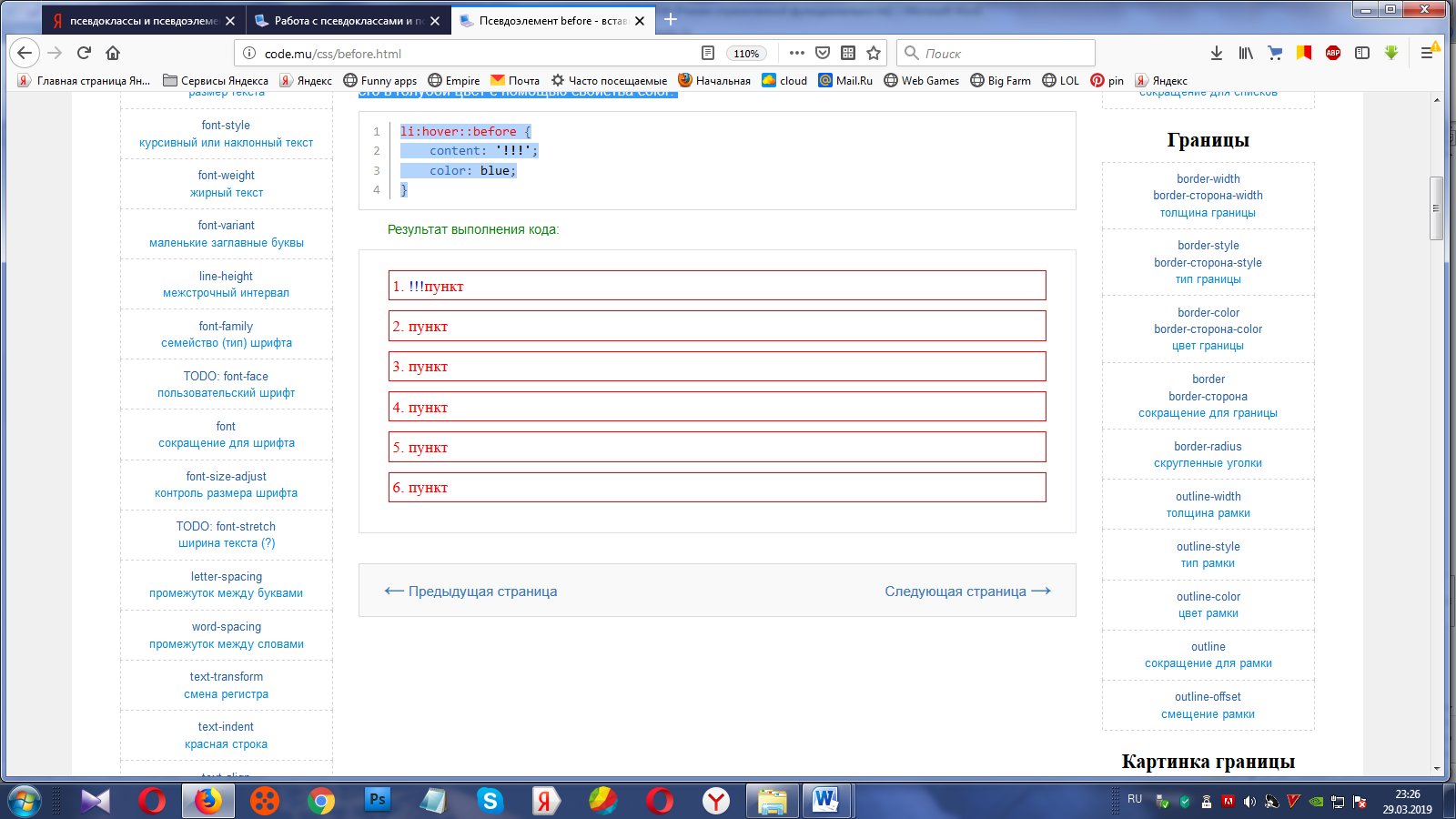
В данном примере **по наведению** на li в его конец вставится текст (а когда мышь уйдет с li - вставленный текст исчезнет). К вставленному тексту можно применять различные стили, здесь мы покрасим его в голубой цвет с помощью свойства color:

li:hover::before{

content:'!!!';

color:blue;

}



**Блок7. На counter-increment**

Свойство **counter-increment** - задает автоматическую нумерацию элементов, например, абзацев или заголовков. Используется совместно со свойством counter-reset, псевдоэлементами::after и ::before, свойством content, внутри которого используется функция counter. Для лучшего понимания рекомендую посмотреть примеры и видео.

См. также свойство counter-reset, которое сбрасывает значение счетчика в ноль. Данное свойство обязательно нужно использовать при работе с **counter-increment**, при этом оно устанавливается для родителя.

**Синтаксис**

селектор {

counter-increment: название\_счетчика [шаг\_изменения\_счетчика] | none | inherit;

}

**Значения**

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| название\_счетчика | Обычное слово (как название класса или id). Можно задавать несколько названий, разделяя их пробелом. В этом случае одновременно будут меняться несколько счетчиков. |
| шаг\_изменения\_счетчика | Целое положительное или отрицательное число. Необязательный параметр. |
| none | Запрещает увеличение счетчика для текущего селектора. |
| inherit | Наследует значение родителя. |

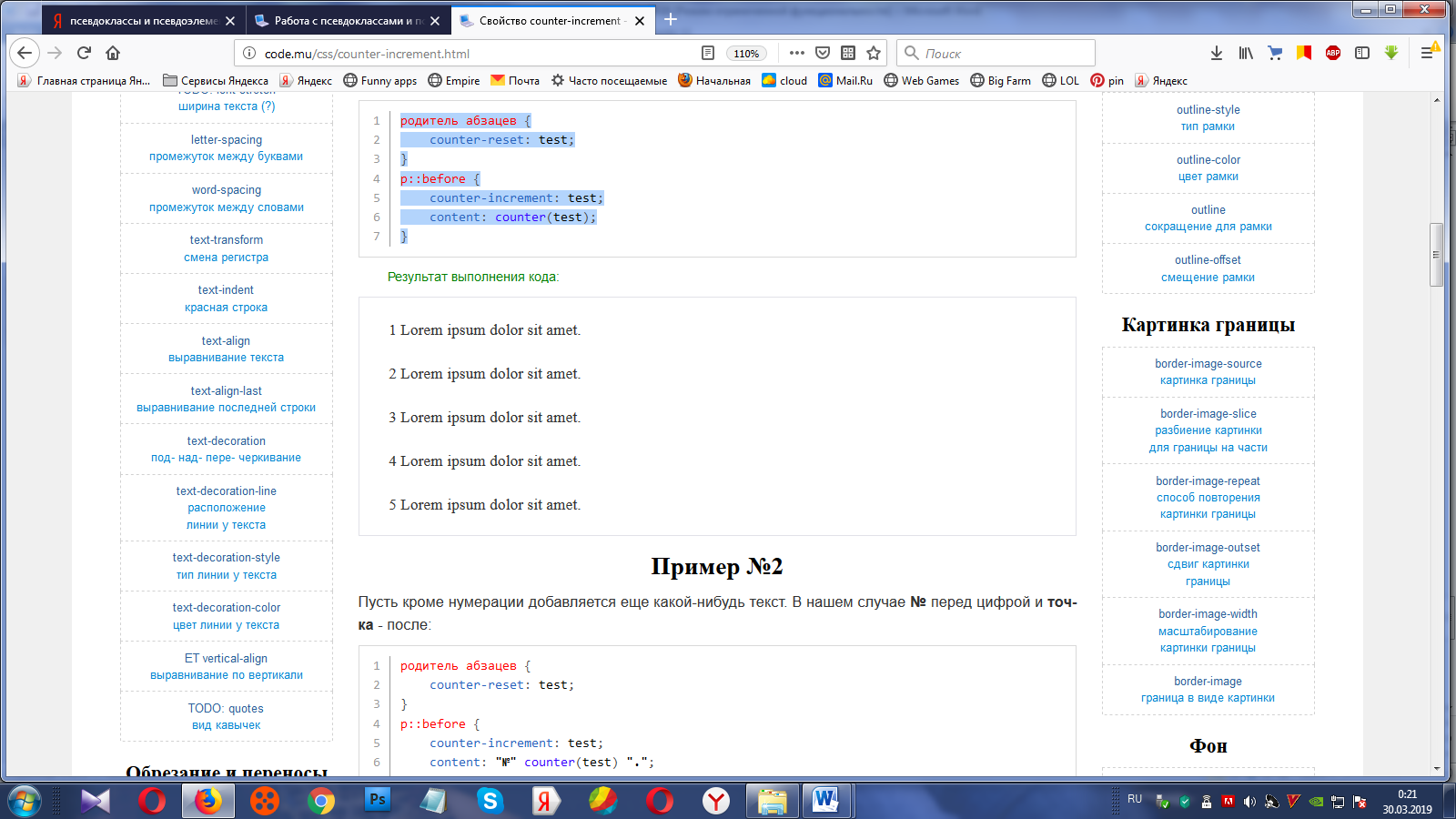
Значение по умолчанию: **none**.

Значение по умолчанию для шага, если он не задан (но при этом задано название счетчика): **1**.

**Пример 1.**

Сделаем так, чтобы абзацы нумеровались автоматически:

родительабзацев {

 counter-reset: test;

}

p::before {

counter-increment: test;

content: counter(test);

}

### Пример 2.

Пусть кроме нумерации добавляется еще какой-нибудь текст. В нашем случае **№** перед цифрой и **точка** - после:

родительабзацев{

counter-reset: test;

}

p::before{

counter-increment: test;

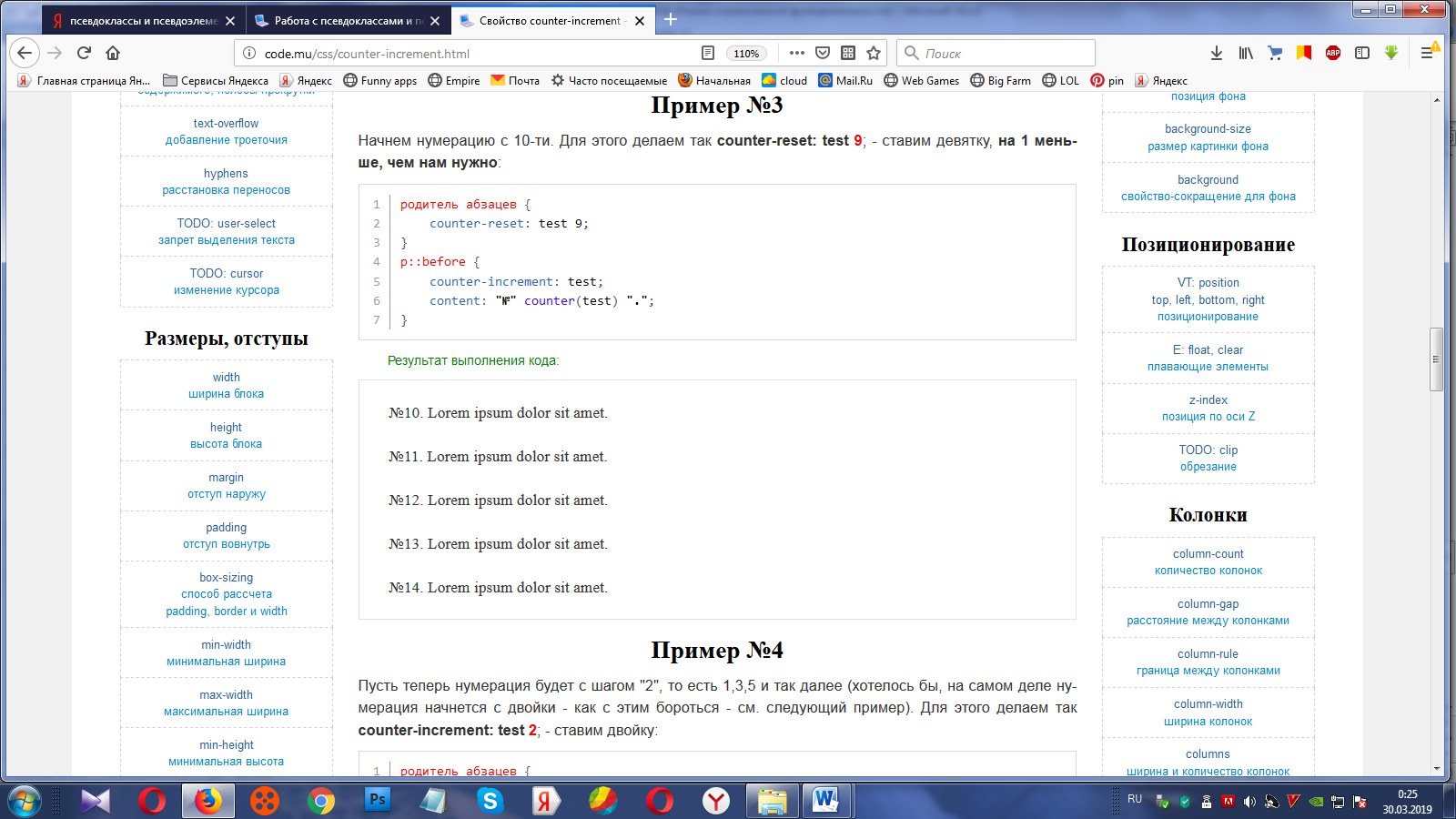
content:"№"counter(test)".";

}

Результат выполнения кода:

### Пример 3.

Начнем нумерацию с 10-ти. Для этого делаем так **counter-reset: test9**; - ставим девятку, **на 1 меньше, чем нам нужно**:

родительабзацев{

counter-reset: test 9;

}

p::before{

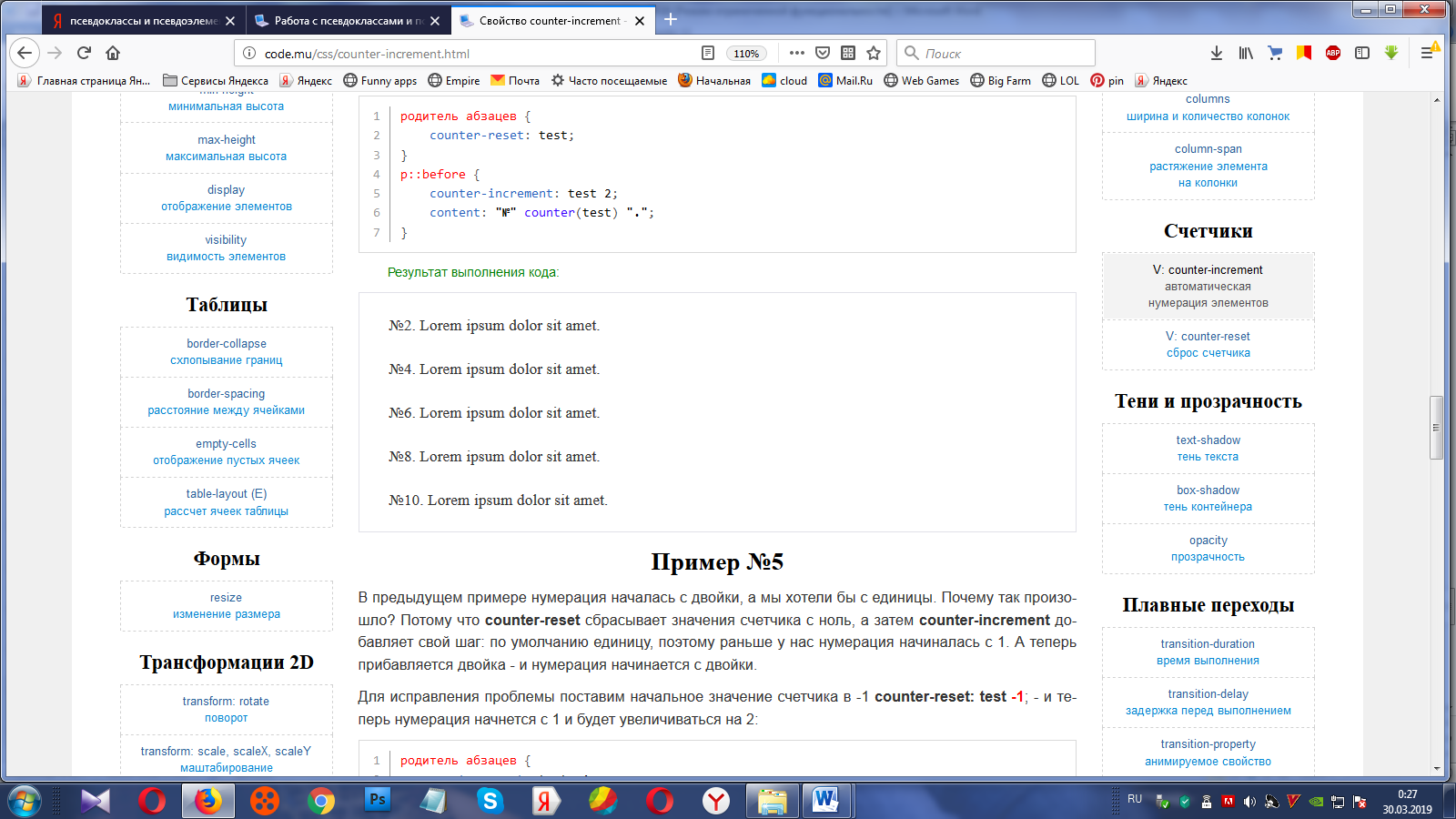
counter-increment: test;

content:"№"counter(test)".";

}

Результат выполнения кода:

### Пример 4.

Пусть теперь нумерация будет с шагом "2", то есть 1,3,5 и так далее (хотелось бы, на самом деле нумерация начнется с двойки - как с этим бороться - см. следующий пример). Для этого делаем так **counter-increment: test2**; - ставим двойку:

родительабзацев{

counter-reset: test;

}

p::before{

counter-increment: test 2;

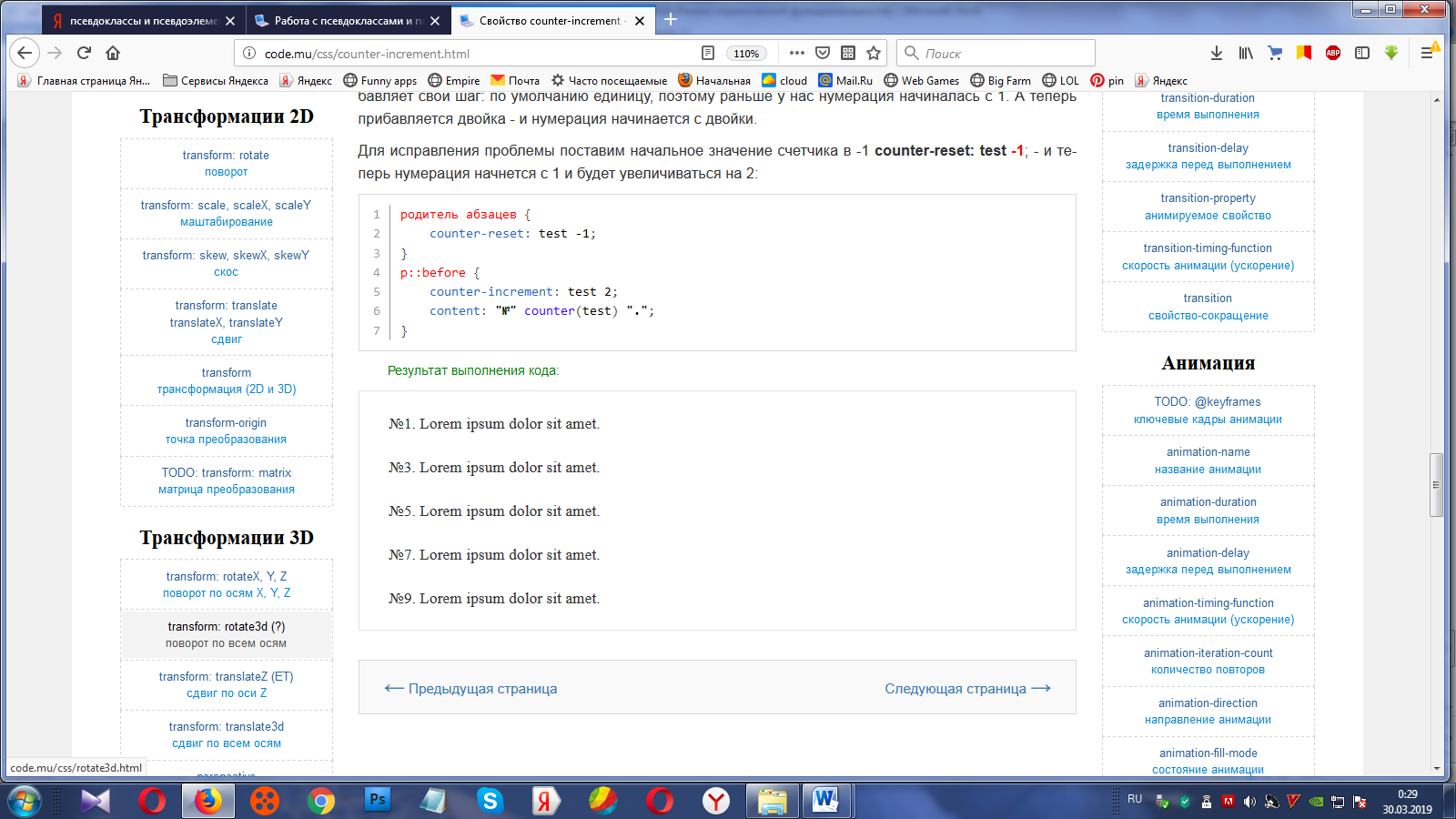
content:"№"counter(test)".";

}

Результат выполнения кода:

### Пример 5.

В предыдущем примере нумерация началась с двойки, а мы хотели бы с единицы. Почему так произошло? Потому что **counter-reset** сбрасывает значения счетчика с ноль, а затем **counter-increment** добавляет свой шаг: по умолчанию единицу, поэтому раньше у нас нумерация начиналась с 1. А теперь прибавляется двойка - и нумерация начинается с двойки.

Для исправления проблемы поставим начальное значение счетчика в -1 **counter-reset: test-1**; - и теперь нумерация начнется с 1 и будет увеличиваться на 2:

родительабзацев{

counter-reset: test -1;

}

p::before{

counter-increment: test 2;

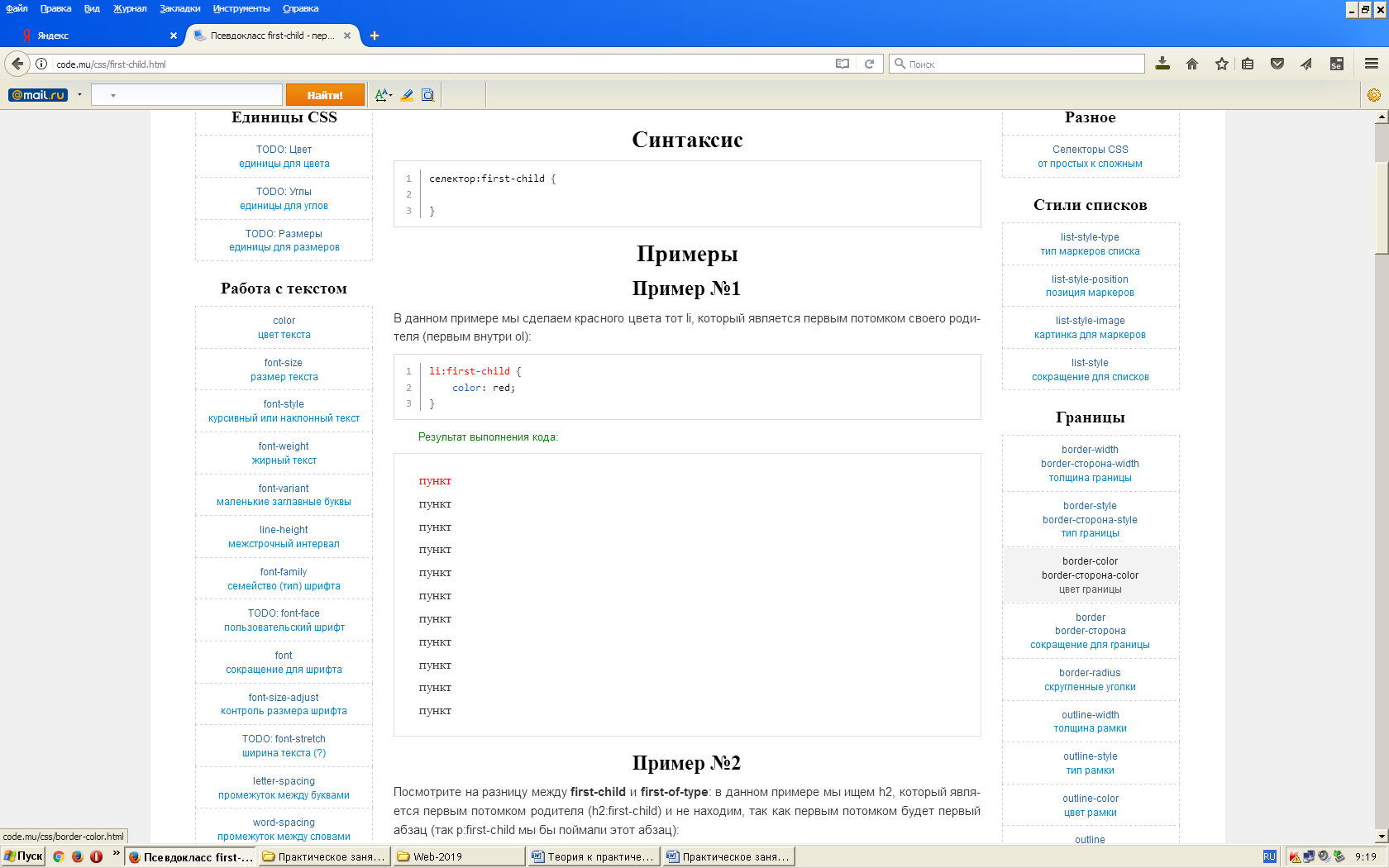
content:"№"counter(test)".";

}

Результат выполнения кода:

**Блок8. На nth**

Длярешениязадачданногоблокавампонадобятсяследующиепсевдоклассыипсевдоэлементы: first-child, last-child, nth-child, nth-last-child, only-child, first-of-type, last-of-type, nth-of-type, nth-last-of-type, only-of-type.

Псевдокласс **first-child** выбирает элемент, который является первым потомком родителя.

**Синтаксис**

селектор:first-child {

}

**Пример 1.**

В данном примере мы сделаем красного цвета тот li, который является первым потомком своего родителя (первым внутри ol):

li:first-child {

color: red;

}

Результат выполнения кода:

**Пример 2.**

Посмотрите на разницу между **first-child** и **first-of-type**: в данном примере мы ищем h2, который является первым потомком родителя (h2:first-child) и не находим, так как первым потомком будет первый абзац (так p:first-child мы бы поймали этот абзац):

<section>

<p>

Это абзац (первый потомок)

</p>

<h2>Это h2</h2>

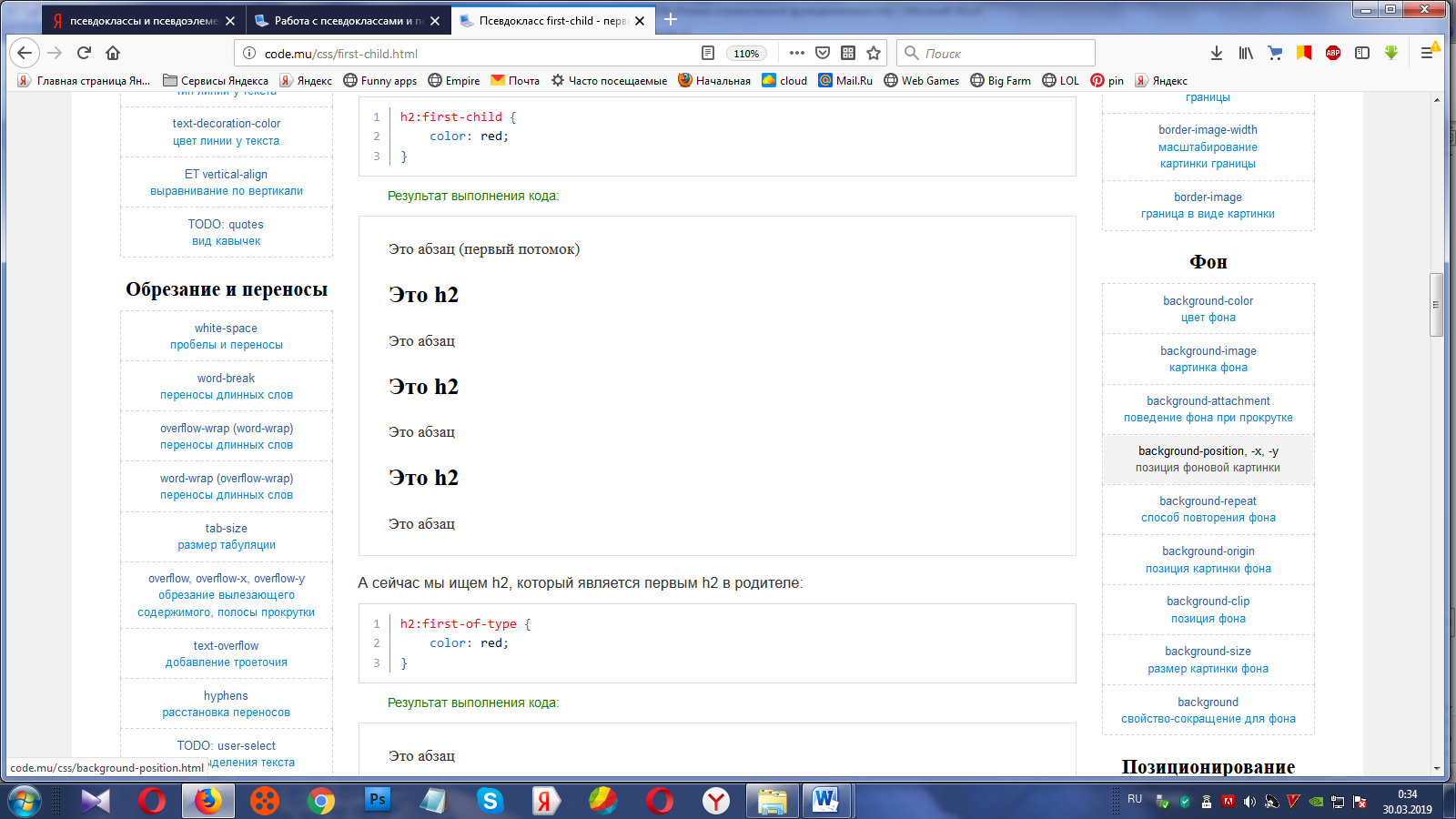
<p>

Это абзац

</p>

<h2>Это h2</h2>

<p>

 Это абзац

</p>

<h2>Это h2</h2>

<p>

Это абзац

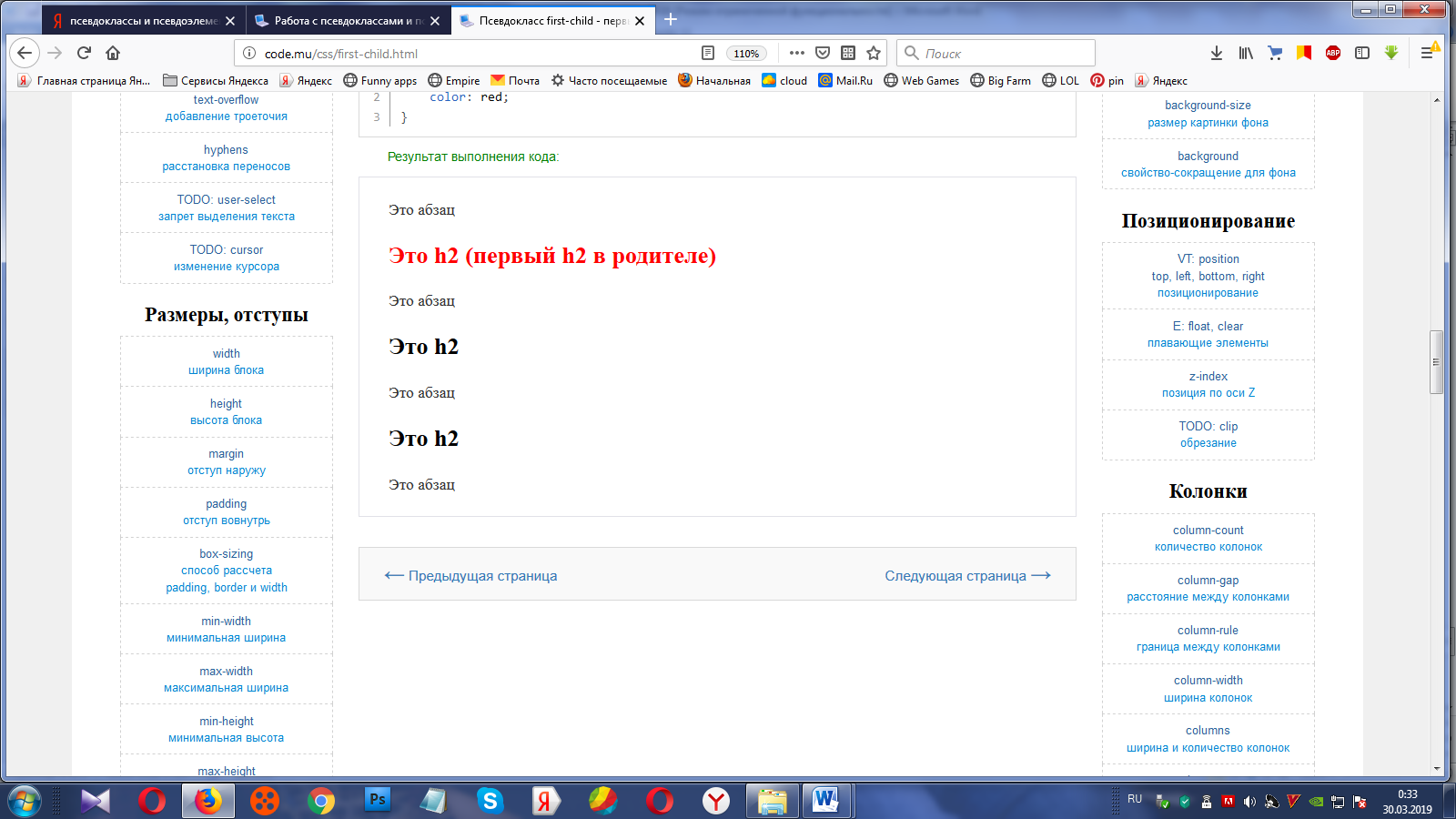
</p>

</section>

h2:first-child {

color: red;

}

А сейчас мы ищем h2, который является первым h2 в родителе:

h2:first-of-type {

color: red;

}

Результат выполнения кода:

**Блок9. На checked**

Для решения задач данного блока вам понадобятся следующие псевдоклассы и псевдоэлементы: checked.

Псевдокласс **checked** позволяет поймать отмеченный чекбокс или radio. К сожалению, стили для их украшения задать **нельзя**. К примеру нельзя задать фон или цвет границы (можно задавать, к примеру, ширину и высоту, но они окажут не такое влияние, как можно было бы ожидать).

Несмотря на это, можно задать стили для соседних элементов, например, так: **input:checked+p** - зададим стили для абзаца, который находится непосредственно под отмеченным инпутом.

## Синтаксис

input:checked {

}

### Пример

Поотмечайте чекбоксы и radio и вы увидите как загораются красным цветом метки label, которые находятся непосредственно после отмеченных элементов:

<input type="checkbox" checked id="checkbox1"><label for="checkbox1">Первый чекбокс</label><br>

<input type="checkbox" id="checkbox2"><label for="checkbox2">Второй чекбокс</label><br>

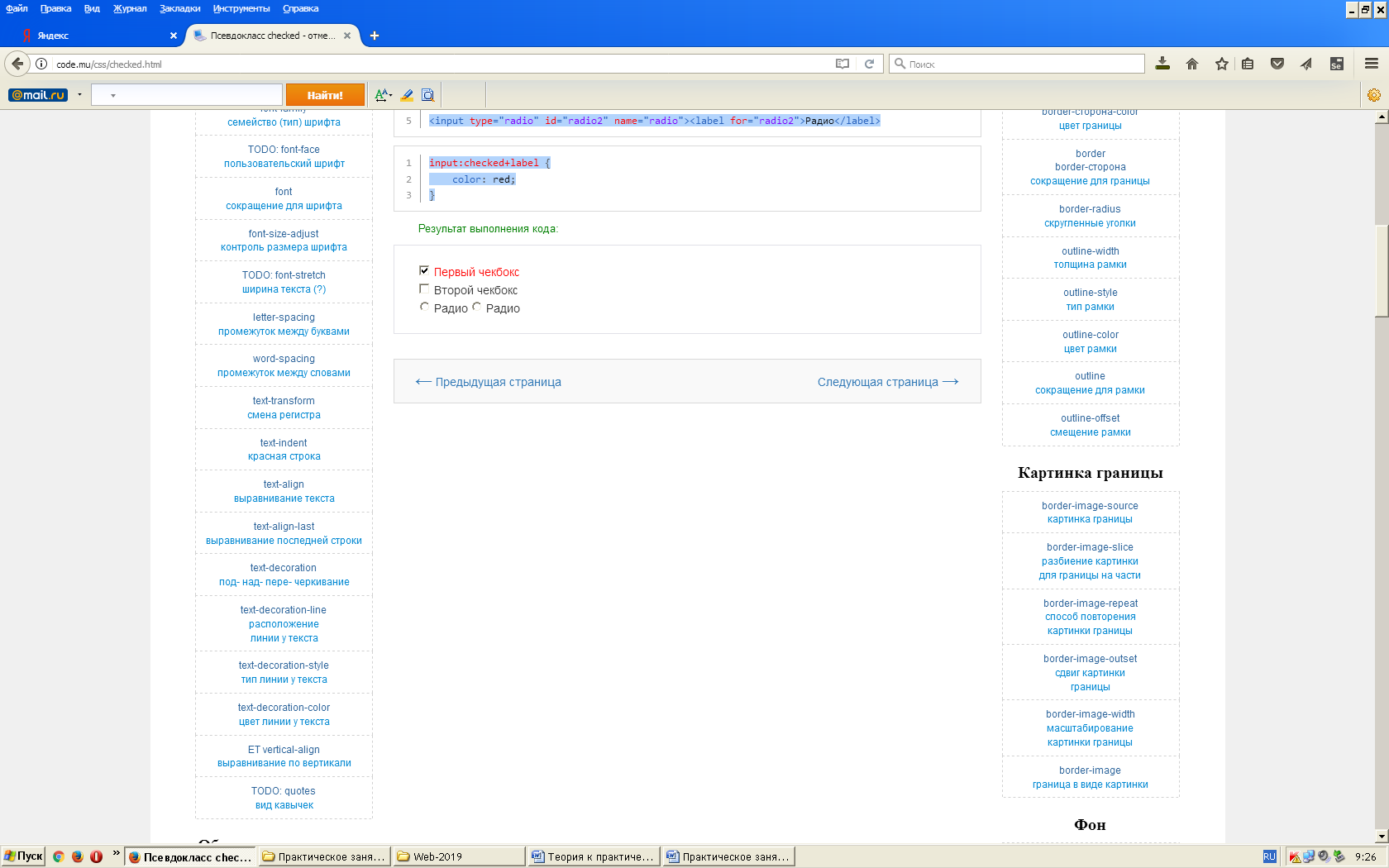
<input type="radio" id="radio1" name="radio"><label for="radio1">Радио</label>

<input type="radio" id="radio2" name="radio"><label for="radio2">Радио</label>

input:checked+label {

color: red;

}



**Блок10. На disabled, enabled**

Псевдокласс **disabled** задает стили для неактивного инпута (которому задан атрибут disabled), а **enabled** - для активного (у которого нет атрибута disabled).

## Синтаксис

input:disabled {

}

### Пример

Применим разные стили (границу border и тень блоку box-shadow) для активного и неактивного инпута:

input:disabled {

border: 1px solid red;

box-shadow: 0px 0px 3px red;

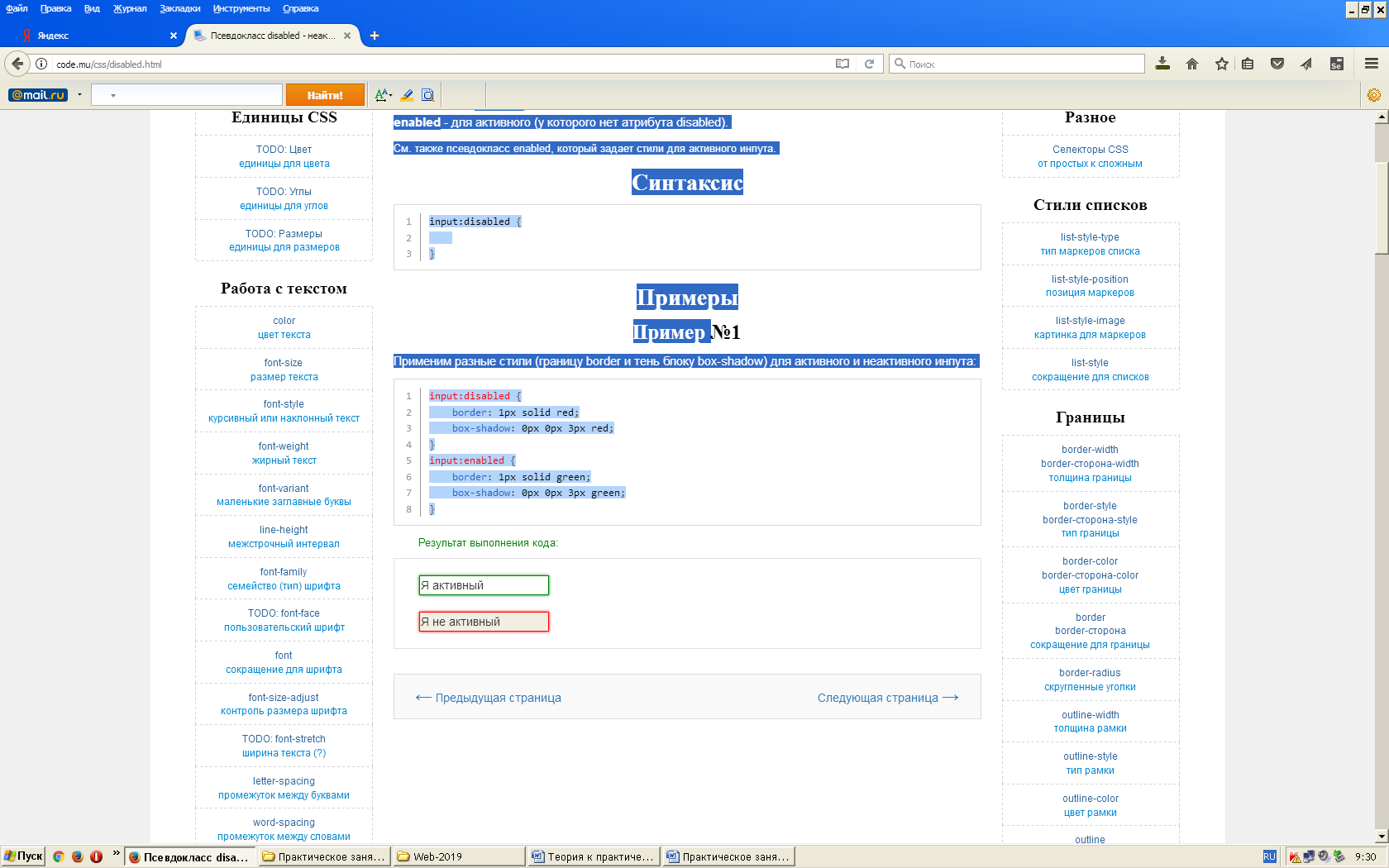
}

input:enabled {

border: 1px solid green;

box-shadow: 0px 0px 3px green;

}



**Блок11. На empty**

Псевдокласс **empty** задает то, как будет выглядеть пустой элемент (не содержащий текста).

**Синтаксис**

селектор:empty {

}

**Пример 1.**

В данном примере последняя li пустая, но все равно будет обведена границей и иметь маркер (зависит от браузера):

<ul>

<li>пункт</li>

<li>пункт</li>

<li>пункт</li>

<li>пункт</li>

<li></li>

</ul>

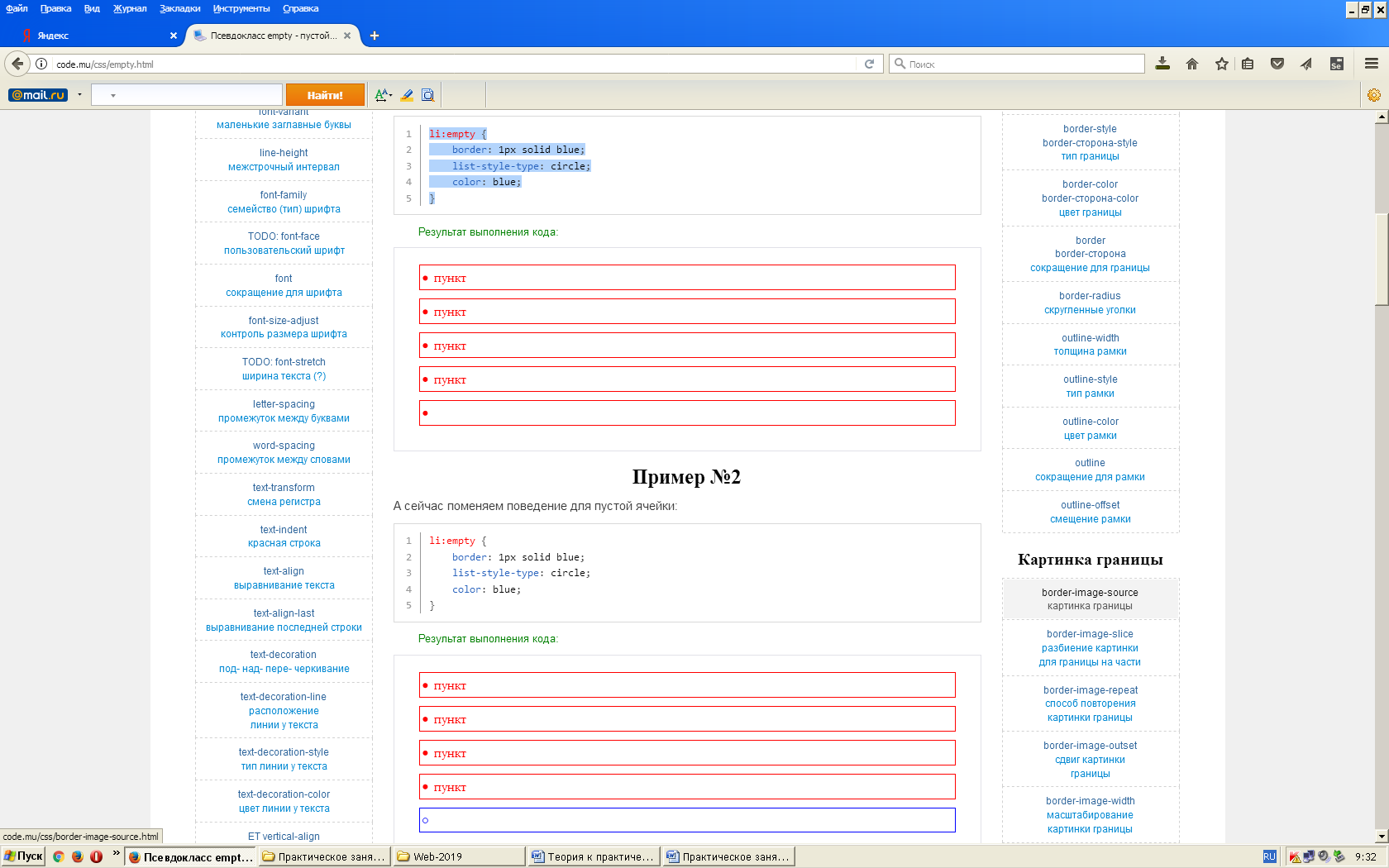
li:empty {

border: 1px solid blue;

list-style-type: circle;

color: blue;

}



**Пример 2.**

А сейчас поменяем поведение для пустой ячейки:

li:empty {

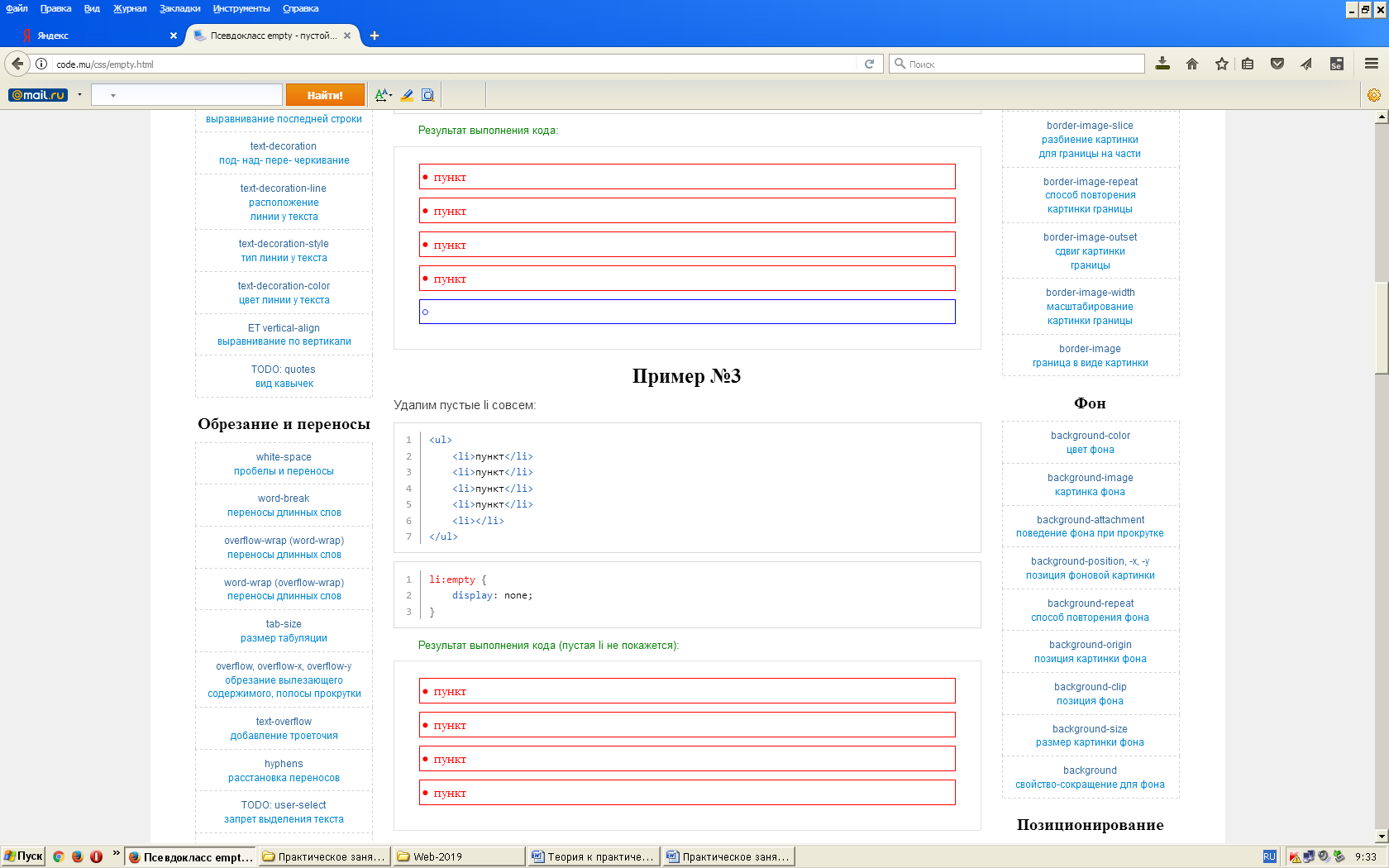
border: 1px solid blue;

list-style-type: circle;

color: blue;

}

Результат выполнения кода:



**Пример 3.**

Удалим пустые li совсем:

<ul>

<li>пункт</li>

<li>пункт</li>

<li>пункт</li>

<li>пункт</li>

<li></li>

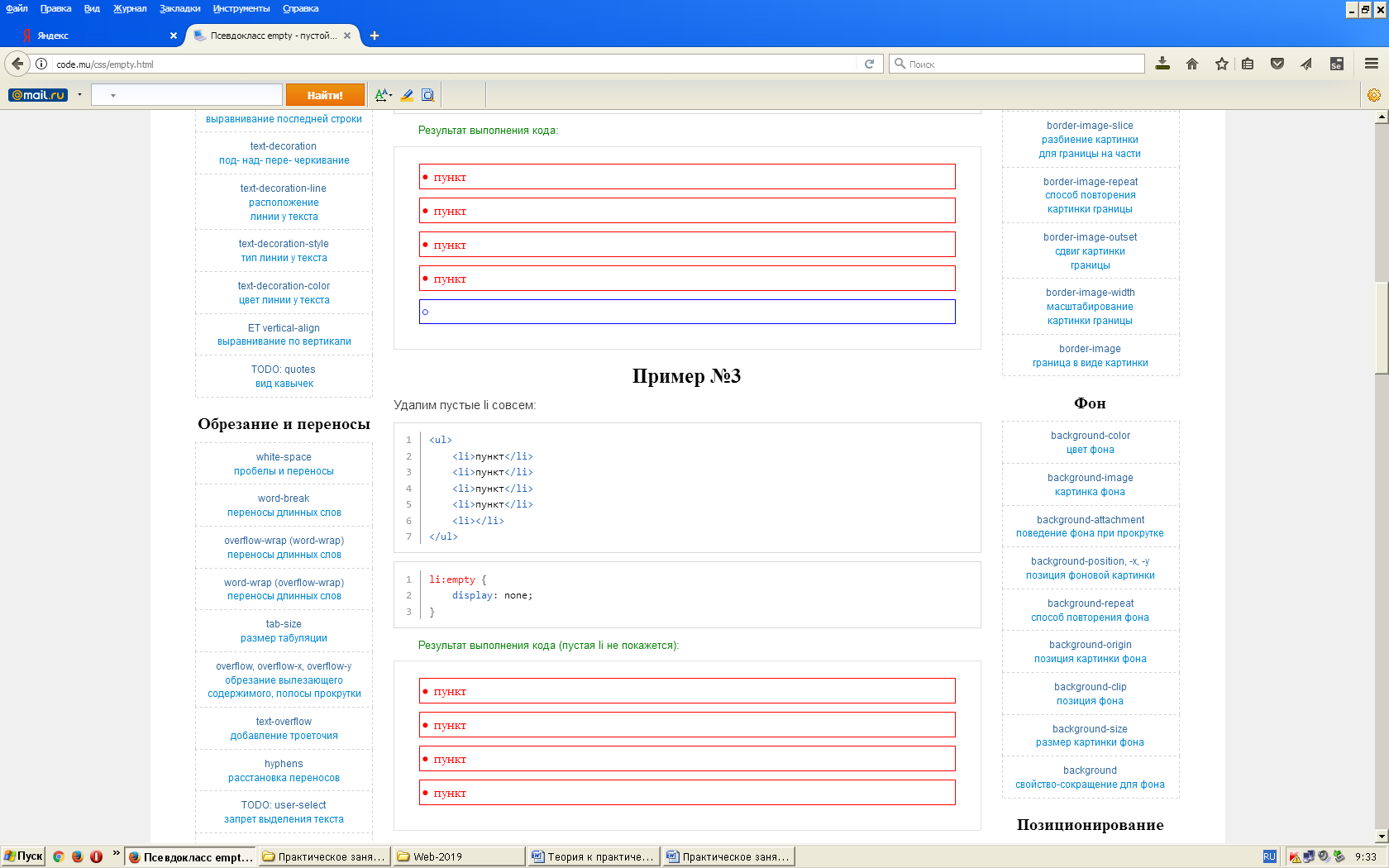
</ul>

li:empty {

display: none;

}

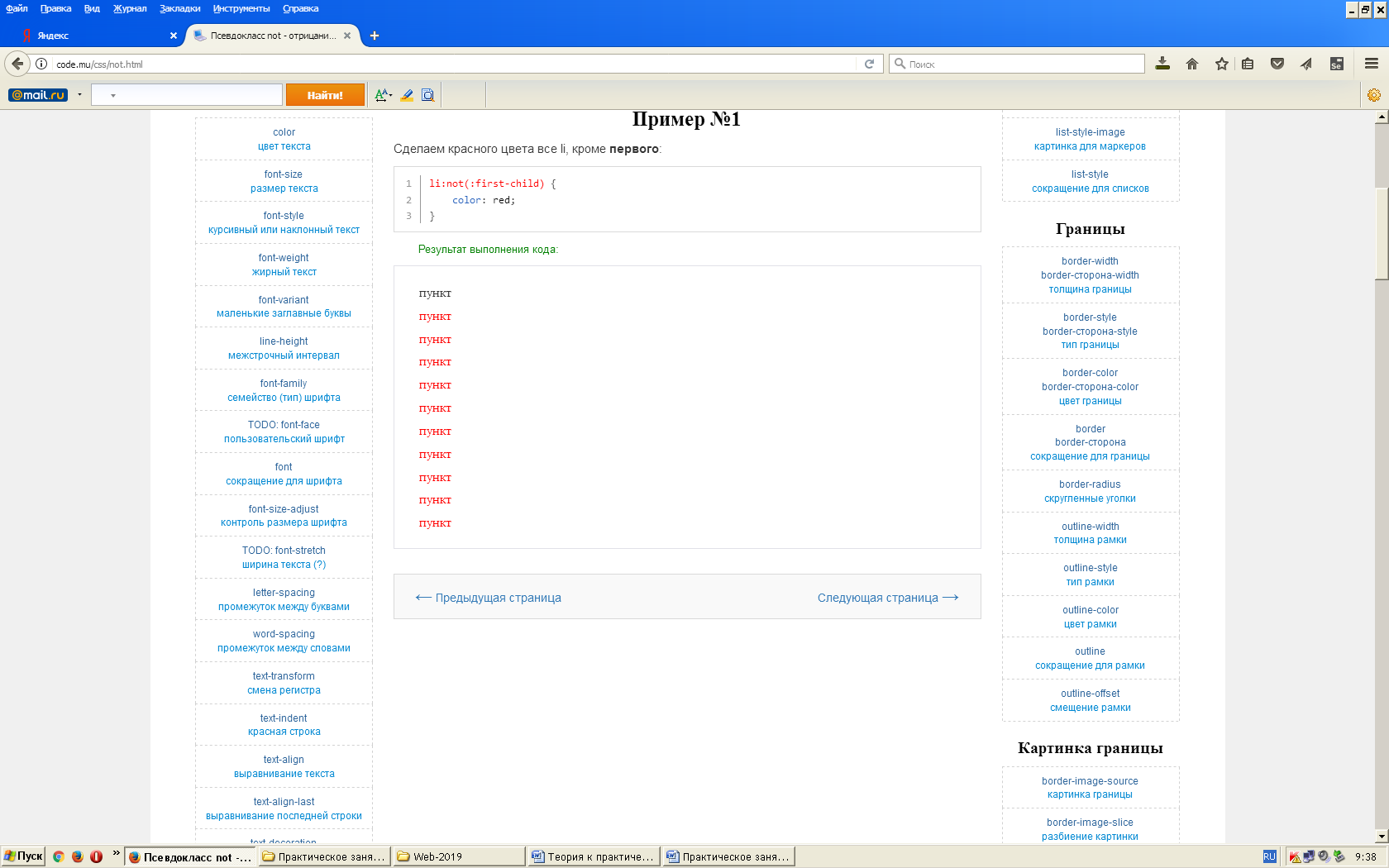
Результат выполнения кода (пустая li не покажется):



**Блок12. На not**

Псевдокласс **not** задает отрицание, к примеру, **p:not(.last)** означает *выбрать все абзацы, у которых нет класса last*. Вложенные **not** не работают.

**Синтаксис**

селектор:not(отрицание) {

}

**Пример**

Сделаем красного цвета все li, кроме **первого**:

li:not(:first-child) {

color: red;

}

**Блок13. На lang**

Псевдокласс **lang** выбирает текст по языку, заданному в атрибуте lang. В качестве языка могут выступать следующие значения: ru — русский, en — английский, de — немецкий, fr — французский, it — итальянский и так далее.

## Синтаксис

селектор:lang(язык) {

}

### Пример

Поставим разные кавычки для различных языков (используем quotes):

<p>

Цитата на английском: То be or not to be.

</p>

<p>

Цитата на немецком: Versuche die Gotter nicht.

</p>

q:lang(en) {

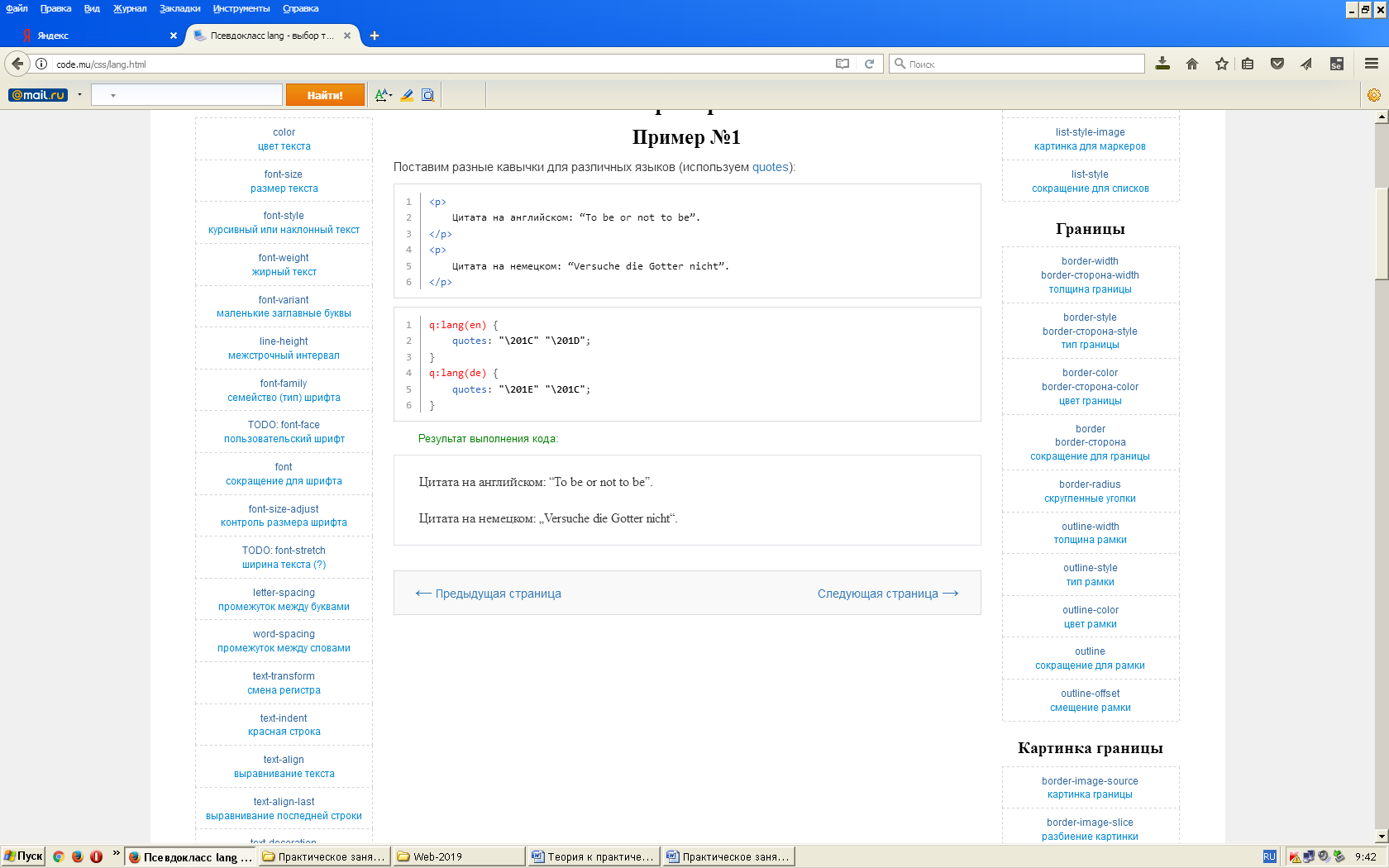
quotes: "\201C" "\201D";

}

q:lang(de) {

quotes: "\201E" "\201C";

}



**Блок14. На target**

Псевдокласс **target** задает то, как будет выглядеть целевой элемент. **Целевой элемент** - это тот, на id которого мы ссылаемся, указывая в конце урла его id после решетки # (пример: **/css/target.html#my-id**).

**Синтаксис**

селектор:target {

}