CREATING WEB-PAGES
Using HTML5 and CSS3

STEP
IT ACADEMY

Lesson 5

Box Model. Pseudo-classes

Contents

Lesson materials are attached to this PDF file. In order to get access to the materials, open the lesson in Adobe Acrobat Reader.

Estimation of Element Real Width. Box-sizing Property

In the last lesson, we looked at the different width options for box elements. However, when assigning an element not only width, but also padding, as well as border, you're bound to face the fact that the actual width of the element is greater than a preset size for width property.

The fact is that a browser determines the width of an element as a complex of properties, in which width is the width of the content itself, to which the sizes of padding, borders and margins are added. Therefore, when placing 2 or more adjacent elements in a row, all of these dimensions must be considered.

Thus, an element with the following properties:

```
.elem {
    width: 500px;
    padding: 15px;
    border: 2px solid #ccc;
    margin: 20px;
}
```

will in the browser have an estimated width of 574px (500px+2*15px+2*2px+2*20px). Total we have 574px instead of 500px specified in the width property.

We can say that the actual width of the element in a browser is calculated by this formula:

```
margin-left + border-left + padding-left + width +
padding-right + border-right + margin-right
```

The same situation relates to the height of an element. It is determined by the formula:

```
margin-top + border-top + padding-top+ height +
padding-bottom + border-bottom + margin-bottom
```

The height of an element should be calculated less often than the width, because in most cases, it is determined by the content and has the auto value, and distributing the columns across the width is a constant task for a layout designer; at the moment column markup is a trend, especially for Landing Pages.

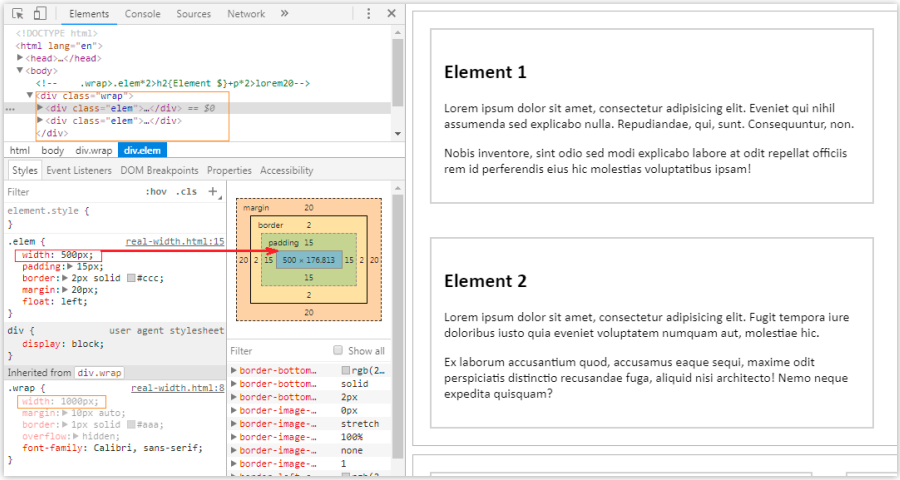


Figure 1

The screenshot (Fig. 1) clearly shows that 2 floating elements with the elem class and the width of 500px do not fit into a container with the *wrap* class with the width of 1000px, although according to the rules of mathematics 500px*2 = 1000px, and they should fit. This is caused by the increase in

the estimated width of the element in a browser due to the addition of the widths of padding, margins and borders.

You can see an example in the *real-width.html* file in the *example* folder of this lesson.

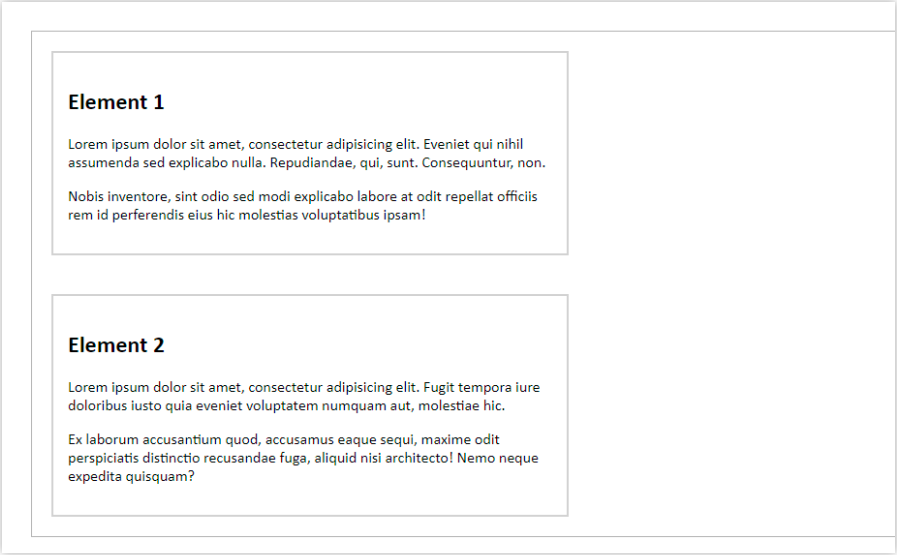


Figure 2

What to do in this case? There are 3 ways to get out of the situation: the first one is based on the calculation of element width taking into account the padding, margins and borders, the second one was often used earlier and implies the presence of a nested element, and the third one is modern and uses the box-sizing property.

Support for this property has appeared since 2006 (Mozilla Firefox v.2) and since 2010 in Chrome 4, but then it was used with vendor prefixes (-moz-, -webkit-), since in the standard form it has not yet received the status of recommended by W3C (Fig. 3).

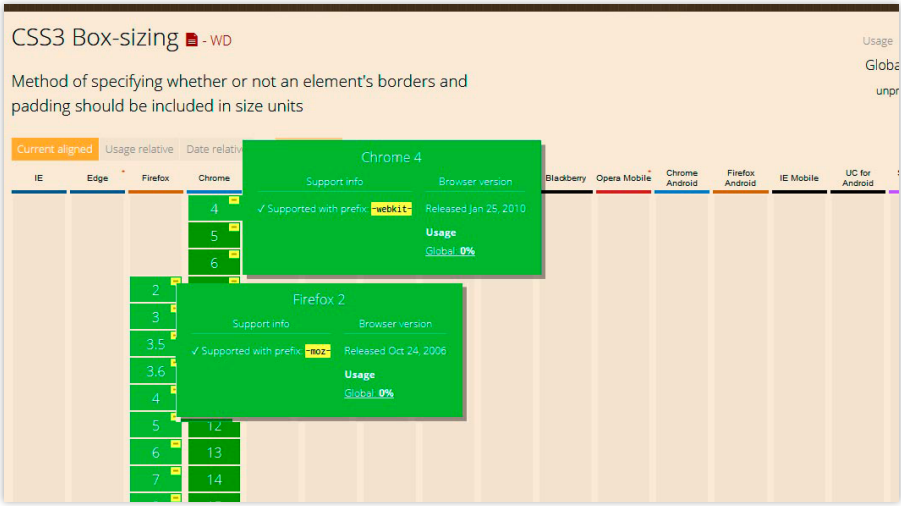


Figure 3

At the moment, all browsers, including Internet Explorer 8 and above, support this property without vendor prefixes. Data on the support of this property can be found on the website caniuse.com (Fig. 4).

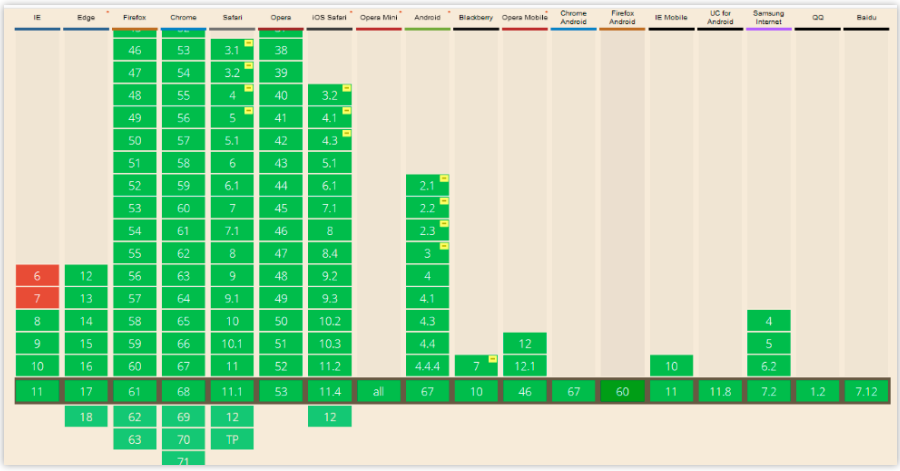


Figure 4

The first method is based on the desired width of an element, but in the width property, you need to specify a value without padding, margins and borders. Thus, again you have to count the values, but not adding, but subtracting the values of the specified properties: 500px – 2*15px (padding) – 2*2px (border) – 2*20px (margin) = 426px

```
.elem-real {
    width: 426px;
    padding: 15px;
    border: 2px solid #ccc;
    margin: 20px;
    float: left;
}
```

In the Property Inspector, you can see how all the box-model properties that we set are distributed (Fig. 5):

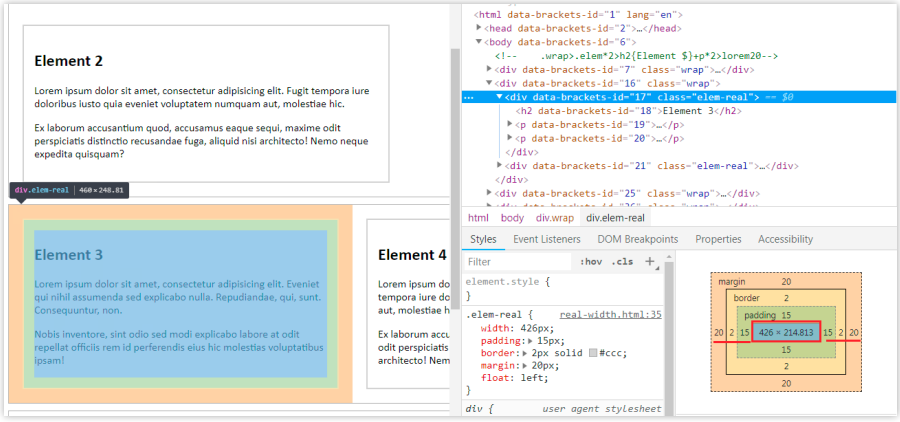


Figure 5

To use the *second method*, namely to use the *nested element*, you have to add to the html-markup 2 elements with different classes instead of one:

```
<div class="elem-parent">
    <div class="inner-elem">
        <h2>Element 1 inner</h2>
        <p>Lorem ipsum dolor sit ...</p>
        <p>Nobis inventore, sint…</p>
    </div>
</div>
```

And in the styles we distribute the properties that were written for the *elem* class between the two classes in this way:

```
.elem-parent{
    width: 500px;
    float: left;
}

.inner-elem {
    padding: 15px;
    border: 2px solid #ccc;
    margin: 20px;
}
```

As a result we get 2 adjacent elements (Fig. 6):

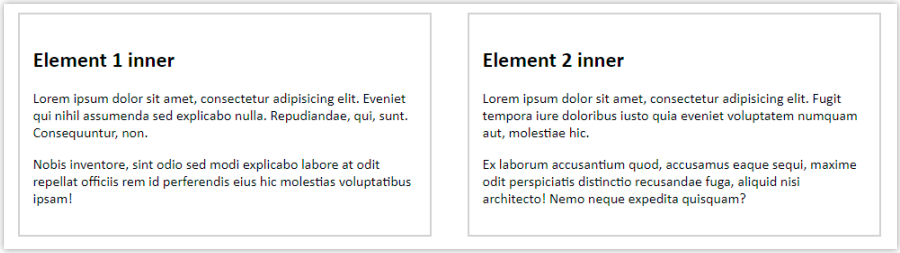


Figure 6

The Property Inspector shows that the dimensions of the nested element with *inner-elem* class coincide with those

that we calculated in the first method, although the width property was not specified for it (Fig. 7).

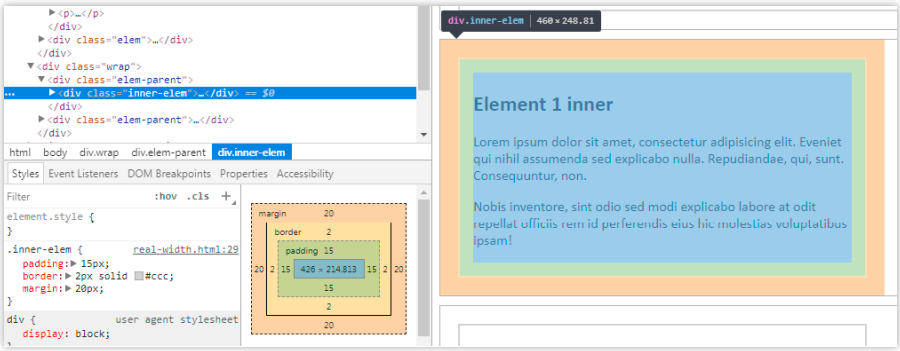


Figure 7

Since adding an additional class is not the best solution, consider the *third method*. It is based on applying the box-sizing property, which can have the following values:

```
box-sizing: content-box | border-box | initial | inherit
```

By default, the box-sizing property is set to content-box, i.e. the width and height properties set the size of the content and do not include padding, margin and border. This option leads to an increase in the actual width of an element in a browser when adding padding, margins and borders.

The border-box value implies that the width and height properties include padding and border, but not margin. That is, the width or height of an element is specified taking into account padding and borders. Margins are not taken into account; they do not always have to be set on both sides, sometimes an margin is only needed just at the right or bottom.

Given this property with the border-box value, the rules for our elements become as follows:

```
.elem-sizing {
    box-sizing: border-box;
    width: 460px;
    padding: 15px;
    border: 2px solid #ccc;
    margin: 20px;
    float: left;
}
```

The width of the element is 460px, because it is necessary to subtract the values of two 20px margins from 500px. The screenshot (Fig. 8) shows that 460px consist of 426px content, 30px (15px*2) padding and 4px (2px*2) border. That is, we again came to the content width value, which was calculated in the first method.

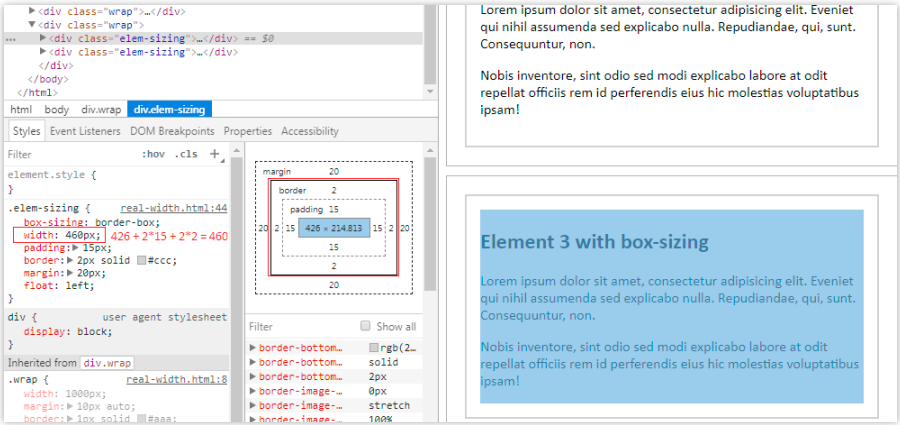


Figure 8

As a result, we get 2 adjacent blocks without special calculations and additional elements (Fig. 9).

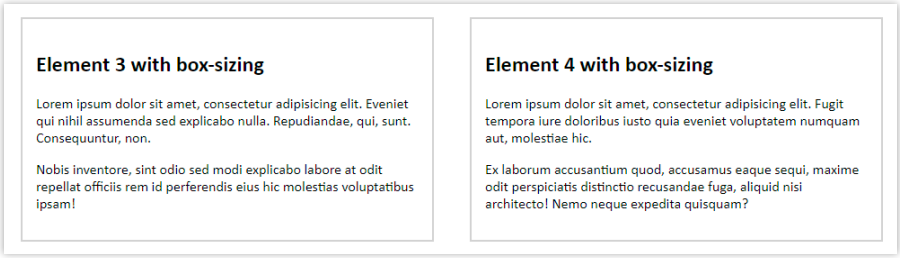


Figure 9

I would like to note that the latter method is now the most common, allows you to specify the width of an element without thinking about the dimensions of its padding or borders and without an additional element in the markup. This is its difference from the first method, in which the width depend on the changing padding and border values, which should be taken into account for the width property, and, accordingly, the width needs to be recalculated all the time, for example, for different screen resolutions. And this is its difference from the second method, in which a nested element is not needed from the point of view of the markup logic (or semantics). Perhaps you do not yet realize this, but this is an excellent way to change the size of the padding and borders: you do not need to recalculate the dimensions of an element. And what is especially valuable, when assigning the width of an element in %, it does not matter what its indents will be — the width of the content will necessarily be calculated taking into account all the values — and this is the work of a browser, not a layout designer.

At the moment, it is so widely used property that it usually starts the markup of any page. Most often it is assigned to all elements at once, using the universal selector:

```
*{ box-sizing: border-box;}
```

In addition, it is desirable to specify this property for the ::before and ::after pseudo-elements, since they are very often used to form various effects on the page, including animations:

```
*, *::before, *::after {
      box-sizing: border-box;
}
```

You can find another writing of the box-sizing property:

```
*, *::before, *::after {
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
}
```

Figure 10

Such an entry implies that you first specify the vendor prefixes for browsers that do not support the standard property, and then the property itself. At the moment, this is not necessary, because according to the website caniuse.com, the global use of browser versions requiring prefixes is almost 0% (Fig. 10).

Related links:

1. https://www.w3schools.com/css/css_boxmodel.asp.
2. https://www.w3schools.com/css/css3_box-sizing.asp.
3. https://developer.mozilla.org/en-US/docs/Web/CSS/box-sizing.
4. https://css-tricks.com/box-sizing/.

Display Property: Inline-block, Inline, and Block. Difference.

One of the important properties of the box-model is the display property, which is responsible for displaying elements on a page and has the following values:

Value	Description
block	block elements
inline	inline elements
inline-block	inline-block elements
none	the elements is not displayed
table, inline-table	table
table-row	table row
table-cell	table cell
table-column	table column created by a <col> element
table-column-group	one or several table columns created by a <colgroup> element
table-header-group	an analog of displaying a table header using a <thead> element
table-footer-group	an analog of displaying a table footer using a <tfoot> element
table-row-group	an analog of displaying a table body using a <tbody> element
list-item	bulleted list item
flex, inline-flex	flex container
grid, inline-grid	grid container
run-in	either block or inline elements depending on the context (incomplete browser support)
inherit	inherits this property from its parent element
initial	sets this property to its default value

By default, the value of the display property is assigned for each item based on the recommendations in the HTML specification, and is taken from the browser style sheet. Like any other css-property, you can override it for each specific element or group of elements using css-selectors.

This property has a number of values, which we will consider in the corresponding topics. In the last lesson, we went into detail on the properties of the box-model, which is most often defined for block elements (the display property has block value). For today's lesson, we need display property values such as inline and inline-block. We will consider them with reference to the box model of elements, since there are a number of features associated with the display of various properties such as width, padding, margin, depending on the display property assigned to them.

Related links:

1. https://www.w3.org/TR/CSS22/visuren.html#display-prop.
2. https://www.w3.org/TR/css-display-3/.
3. https://www.w3.org/wiki/CSS/Properties/display.
4. https://www.w3schools.com/cssref/pr_class_display.asp.
5. https://developer.mozilla.org/en-US/docs/Web/CSS/display.

Inline Elements

Lowercase elements have a display: inline property, so they take up exactly as much space as there are text in them. If these elements have a background color, padding and margins, and you place them in a block with a lot of text, you can see an undesirable picture:

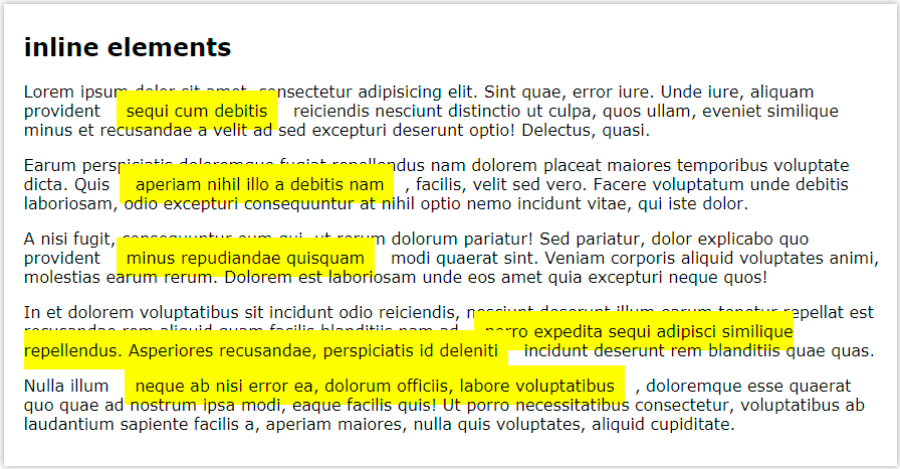


Figure 11

You can see an example in the *inline-and-inline-block-elements.html* file in the *examples* folder of this lesson. CSS-properties for the selected elements are as follows:

```
.inline {
    background-color: yellow;
    padding: 10px;
    margin: 10px;
}
```

The screenshot (Fig. 11) shows that the background of *inline* elements with the inline class is superimposed on the text that follows it, overlapping it and making it unreadable. You can avoid this by either decreasing padding, or by setting a high line height (the line-height property). There is another way to solve the problem: to make these elements inline-block, specifying the display: inline-block property for them.

Note. *in the sample file you will find the commented line of the Emmet abbreviation, which allows you to generate the text of the example:*
*<!-- h1{Inline and inline-block elements}+h2{inline elements}+p*5>lorem40^h2{inline-block elements}+p*5>lorem40 -->*
To remove a comment, place the cursor anywhere in the line and press Ctrl + /, and then move the cursor right after the number 40 (no spaces!) and press the Tab key.

Inline-block Elements

The inline-block elements behave, on the one hand, as inline ones, i.e. they take up exactly as much space as there are text in them, and on the other hand, they interpret padding, margins, width and text alignment in the same way as block elements do.

Figure 12

Compare the appearance of a text in the screenshot (Fig. 12) with the one that was earlier. In the same file *inline-and-inline-block-elements.html* from the *examples* folder, the *inline-block* class was added for elements that were previously set to the *inline* class, and it has the following code:

```
.inline-block {
    background-color: pink;
    padding: 10px;
    margin: 10px;
    display: inline-block;
}
```

Immediately strikes that the inline-block elements are no longer superimposed on the adjacent text, the margins moved them a certain distance from the adjacent words and lines. However, it is difficult to call this text nicely formatted.

From the above, you can draw the following conclusions: if you need to highlight a text in a sentence in a background color, you can use inline elements with small padding, margins should be set only on the left and right. It also makes sense to increase the line height (css-property line-height). For example, the same text that we discussed above can be formatted using the span elements with the inline-formatting class, placing them in a div with a container class, for which the line height is increased:

```
.inline-formatting {
    background-color: aquamarine;
    padding: 3px;
    margin-left: 2px;
    margin-right: 2px;
}

.container {line-height: 170%;}
```

The appearance of the text changed immediately (*inline-and-inline-block-elements.html* file) (Fig. 13).

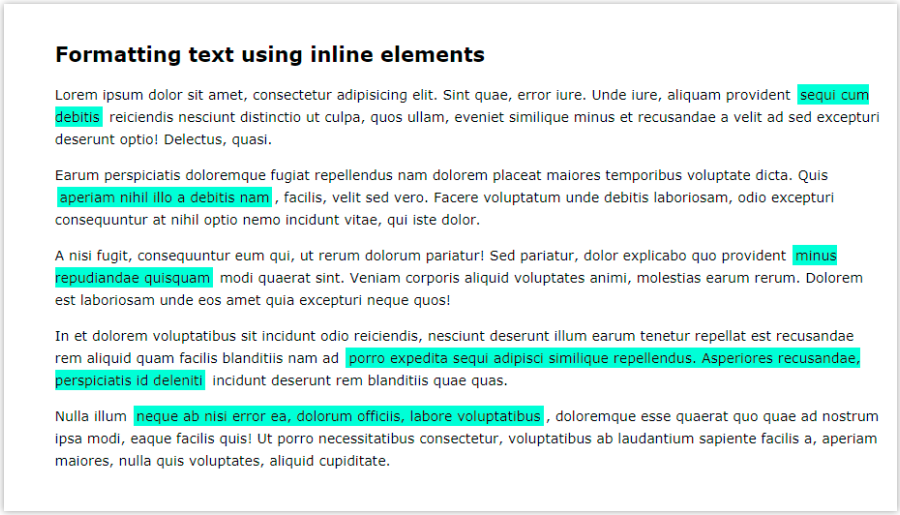


Figure 13

If the purpose of formatting is to visually highlight the element on the background of others — it is better to make it inline-block and use all the properties of the box model in any variations.

Use of Pseudo-elements

We have already looked at the ::before and ::after pseudo-elements. In addition to them, you can use such pseudo-elements as ::first-line, ::first-letter or ::selection. Consider how you can use these pseudo-elements.

Let me remind you that pseudo-elements are written with one (CSS2 and CSS2.1 specification) or two colons (CSS3 specification) before the name. The exception is the pseudo-element ::selection — it is always written with two colons.

::First-line Pseudo-element

By the name of this pseudo-element, you can guess that it is intended for formatting the *first line of block elements*. To format the first line, you can use not all the css properties, but only those listed below:

1. Font group properties (font-family, font-size, font-style, font-weight, font-variant, etc.),
2. color,
3. background group properties,
4. word-spacing,
5. letter-spacing,
6. text-decoration, text-decoration-color, text-decoration-line, text-decoration-style,
7. vertical-align,
8. text-shadow,
9. text-transform,
10. line-height,
11. clear.

For example,

```
p::first-line {
    color: red;
    text-shadow: 1px 1px1px #333;
    font-family: "Segoe Script", cursive;

}
```

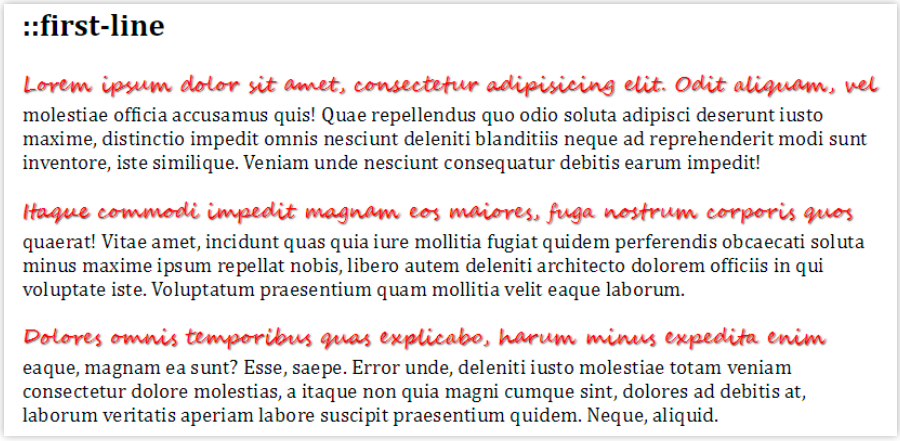


Figure 14

If you use other properties, you may not see any changes. For example, you cannot move text in the first line using a pseudo-element, with the text-indent property, or add margin. But you can do this for the whole element.

```
p {
    text-indent: 20px;
    margin-left: 20px;
}
```

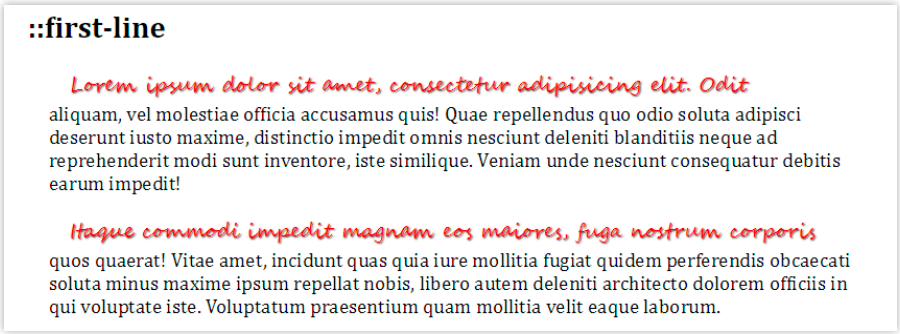


Figure 15

::First-letter Pseudo-element

By the name of this pseudo-element, you can also understand that its purpose is to format the first letter of a block element. If you ever encountered a drop cap, then imagine how you can use this pseudo-element. There are limitations on the properties used. You can apply the same css properties as for ::first-line plus the ones listed below:

1. margin group properties,
2. padding group properties,
3. border group properties,
4. box-shadow,
5. float,
6. vertical-align (only if float equals none).

For example, you can arrange the first letters in paragraphs as follows:

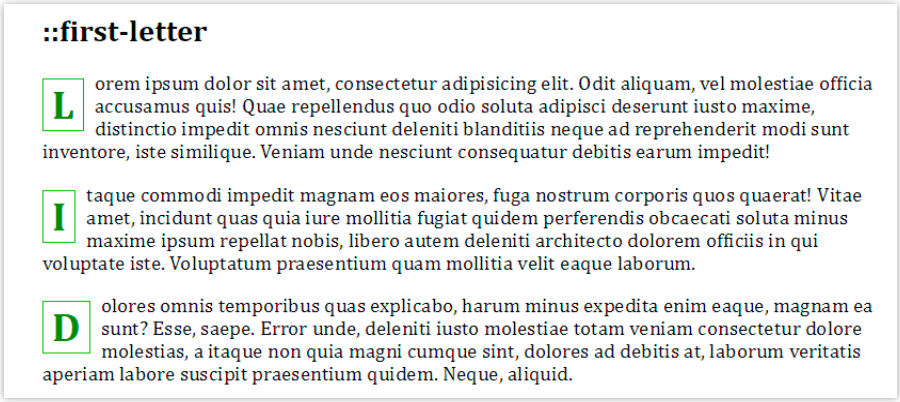


Figure 16

Example code:

```
p::first-letter {
    display: inline-block;
    color: green;
    font-size: 2em;
    font-weight: bold;
    border: 1px solid #1ebc1e;
    padding: 2px 8px;
    margin-right: 10px;
    margin-top: 7px;
    float: left;
}
```

::Selection Pseudo-element

The ::selection pseudo-element is intended to change the appearance of a text when selected. It can be assigned for the entire document, or for certain elements. It should be remembered that with this pseudo-element you can use a small number of css-properties , namely:

1. color,
2. background-color,
3. cursor,
4. outline and its variants,
5. text-decoration property group,
6. text-shadow.

For the Mozilla Firefox browser, a non-standard spelling of this pseudo-element is used ::-moz-selection.

For all elements in a document:

```
::selection {
    background-color: #fcfc6b;
}
```

For a paragraph:

```
p::-moz-selection {
    background-color: #025302;
    color: #fff;
}
p::selection {
    background-color: #025302;
    color: #fff;
}
```

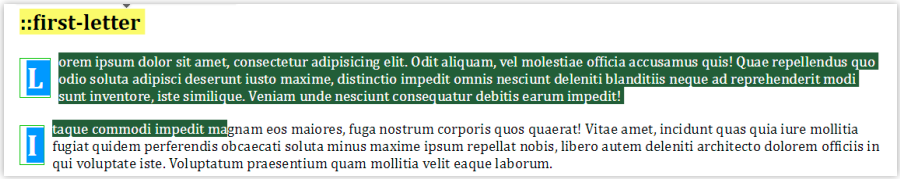


Figure 17

You can find example code in the *pseudoelements.html* file in the examples folder.

Useful links:

1. https://developer.mozilla.org/en-US/docs/Web/CSS/::-first-letter.
2. https://developer.mozilla.org/en-US/docs/Web/CSS/::-first-line.

Pseudo-classes for Elements

In addition to pseudo-elements, there are *pseudo-classes* in CSS. They are always written with one colon after the selector. Pseudo-classes define different states of elements, for example, their appearance when hovering over or clicking on an element. Today we will look at pseudo-classes that are responsible for formatting certain elements in the document structure.

For example, the :first-child pseudo-class is responsible for formatting the first element among similar ones. For example, this code will cause only the first paragraph to be highlighted in a different color and have a different font:

```
p:first-child {
    font-family: 'Segoe Script', cursive;
    color: #2c5dcc;
    font-weight: bold;
    border-bottom: 3px double #2c5dcc;
}
```

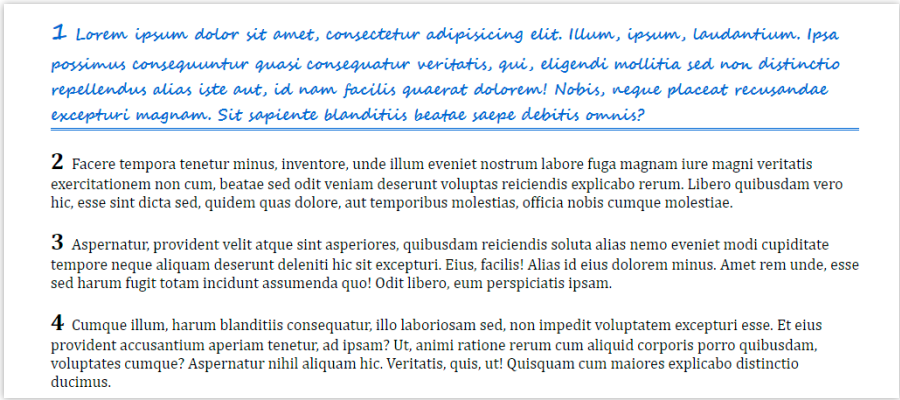
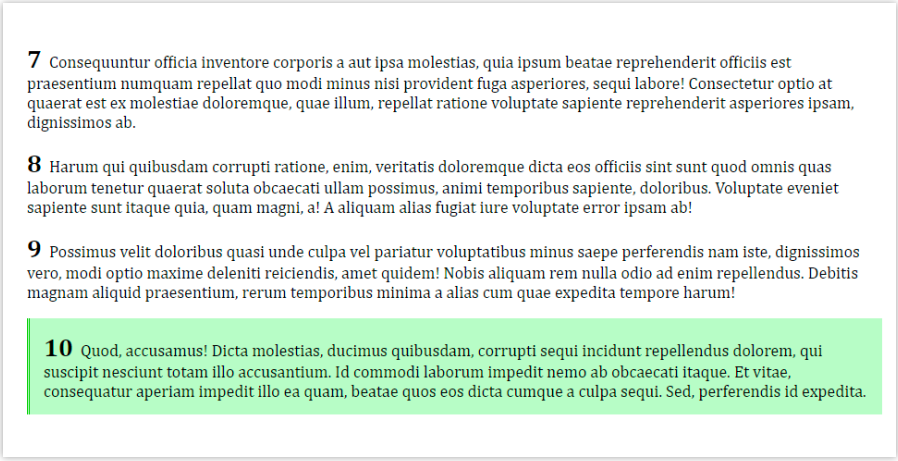


Figure 18

In contrast to :first-child the :last-child pseudo-class is responsible for formatting the last element among similar ones:

```
p:last-child {
    padding: 15px;
    background-color: #c8fdc1;
    border-left: 3px double #32c71e;
}
```



7 Consequuntur officia inventore corporis a aut ipsa molestias, quia ipsum beatae reprehenderit officiis est praesentium numquam repellat quo modi minus nisi provident fuga asperiores, sequi labore! Consectetur optio at quaerat est ex molestiae doloremque, quae illum, repellat ratione voluptate sapiente reprehenderit asperiores ipsam, dignissimos ab.

8 Harum qui quibusdam corrupti ratione, enim, veritatis doloremque dicta eos officiis sint sunt quod omnis quas laborum tenetur quaerat soluta obcaecati ullam possimus, animi temporibus sapiente, doloribus. Voluptate eveniet sapiente sunt itaque quia, quam magni, a! A aliquam alias fugiat iure voluptate error ipsam ab!

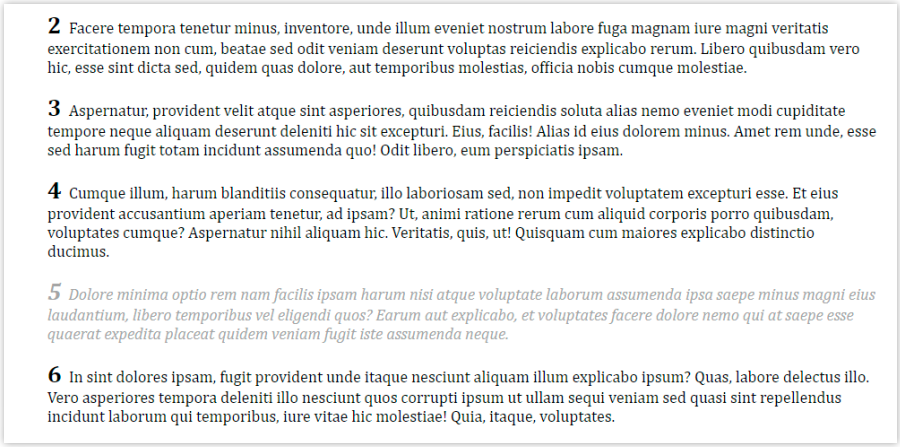
9 Possimus velit doloribus quasi unde culpa vel pariatur voluptatibus minus saepe perferendis nam iste, dignissimos vero, modi optio maxime deleniti reiciendis, amet quidem! Nobis aliquam rem nulla odio ad enim repellendus. Debitis magnam aliquid praesentium, rerum temporibus minima a alias cum quae expedita tempore harum!

10 Quod, accusamus! Dicta molestias, ducimus quibusdam, corrupti sequi incidunt repellendus dolorem, qui suscipit nesciunt totam illo accusantium. Id commodi laborum impedit nemo ab obcaecati itaque. Et vitae, consequatur aperiam impedit illo ea quam, beatae quos eos dicta cumque a culpa sequi. Sed, perferendis id expedita.

Figure 19

To format a specific element, you must use the:nth-child(n) pseudo-class. You need to put a digit instead of n. For example, for the 5th paragraph, the code will look like this:

```
p:nth-child(5) {
    font-style: italic;
    color: #9b9b9b;
}
```



2 Facere tempora tenetur minus, inventore, unde illum eveniet nostrum labore fuga magnam iure magni veritatis exercitationem non cum, beatae sed odit veniam deserunt voluptas reiciendis explicabo rerum. Libero quibusdam vero hic, esse sint dicta sed, quidem quas dolore, aut temporibus molestias, officia nobis cumque molestiae.

3 Aspernatur, provident velit atque sint asperiores, quibusdam reiciendis soluta alias nemo eveniet modi cupiditate tempore neque aliquam deserunt deleniti hic sit excepturi. Eius, facilis! Alias id eius dolorem minus. Amet rem unde, esse sed harum fugit totam incidunt assumenda quo! Odit libero, eum perspiciatis ipsam.

4 Cumque illum, harum blanditiis consequatur, illo laboriosam sed, non impedit voluptatem excepturi esse. Et eius provident accusantium aperiam tenetur, ad ipsam? Ut, animi ratione rerum cum aliquid corporis porro quibusdam, voluptates cumque? Aspernatur nihil aliquam hic. Veritatis, quis, ut! Quisquam cum maiores explicabo distinctio ducimus.

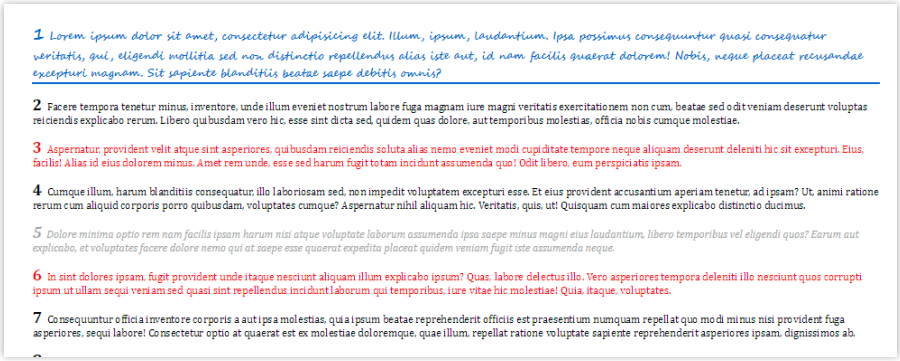
5 *Dolore minima optio rem nam facilis ipsam harum nisi atque voluptate laborum assumenda ipsa saepe minus magni eius laudantium, libero temporibus vel eligendi quos? Earum aut explicabo, et voluptates facere dolore nemo qui at saepe esse quaerat expedita placeat quidem veniam fugit iste assumenda neque.*

6 In sint dolores ipsam, fugit provident unde itaque nesciunt aliquam illum explicabo ipsum? Quas, labore delectus illo. Vero asperiores tempora deleniti illo nesciunt quos corrupti ipsum ut ullam sequi veniam sed quasi sint repellendus incidunt laborum qui temporibus, iure vitae hic molestiae! Quia, itaque, voluptates.

Figure 20

If you need to set the formatting for each 3rd paragraph, you should use the :nth-child(3n) pseudo-class. Instead of the digit 3 there can be any digit or expression, for example, 4n-1, 3n+1, etc.

```
p:nth-child(3n) {
    color: red;
}
```



1 Lorem ipsum dolor sit amet, consectetur adipisicing elit. Illum, ipsum, laudantium. Ipsa possimus consequuntur quasi consequatur veritatis, qui, eligendi mollitia sed non distinctio repellendus alias iste aut, id nam facilis quaerat dolorem! Nobis, neque placeat recusandae excepturi magnam. Sit sapiente blanditiis beatae saepe debitis omnis?

2 Facere tempora tenetur minus, inventore, unde illum eveniet nostrum labore fuga magnam iure magni veritatis exercitationem non cum, beatae sed odit veniam deserunt voluptas reiciendis explicabo rerum. Libero quibusdam vero hic, esse sint dicta sed, quidem quas dolore, aut temporibus molestias, officia nobis cumque molestiae.

3 Aspernatur, provident velit atque sint asperiores, quibusdam reiciendis soluta alias nemo eveniet modi cupiditate tempore neque aliquam deserunt deleniti hic sit excepturi. Eius, facilis! Alias id eius dolorem minus. Amet rem unde, esse sed harum fugit totam incidunt assumenda quo! Odit libero, eum perspiciatis ipsam.

4 Cumque illum, harum blanditiis consequatur, illo laboriosam sed, non impedit voluptatem excepturi esse. Et eius provident accusantium aperiam tenetur, ad ipsam? Ut, animi ratione rerum cum aliquid corporis porro quibusdam, voluptates cumque? Aspernatur nihil aliquam hic. Veritatis, quis, ut! Quisquam cum maiores explicabo distinctio ducimus.

5 *Dolore minima optio rem nam facilis ipsam harum nisi atque voluptate laborum assumenda ipsa saepe minus magni eius laudantium, libero temporibus vel eligendi quos? Earum aut explicabo, et voluptates facere dolore nemo qui at saepe esse quaerat expedita placeat quidem veniam fugit iste assumenda neque.*

6 In sint dolores ipsam, fugit provident unde itaque nesciunt aliquam illum explicabo ipsum? Quas, labore delectus illo. Vero asperiores tempora deleniti illo nesciunt quos corrupti ipsum ut ullam sequi veniam sed quasi sint repellendus incidunt laborum qui temporibus, iure vitae hic molestiae! Quia, itaque, voluptates.

7 Consequuntur officia inventore corporis a aut ipsa molestias, quia ipsum beatae reprehenderit officiis est praesentium numquam repellat quo modi minus nisi provident fuga asperiores, sequi labore! Consectetur optio at quaerat est ex molestiae doloremque, quae illum, repellat ratione voluptate sapiente reprehenderit asperiores ipsam, dignissimos ab.

Figure 21

You can find all examples in the *pseudoclasses-1.html* file.

Quite often there are need to format even or odd elements. For this purpose, you can use, for example, the :nth-child(2n) or :nth-child(2n-1) pseudo-class. In addition, there are special keywords for even and odd elements. In the examples from the *pseudoclasses-2.html* file, pseudo-classes are defined for elements with the box class:

```
.box:nth-child(odd) {background-color: #f99;}
```

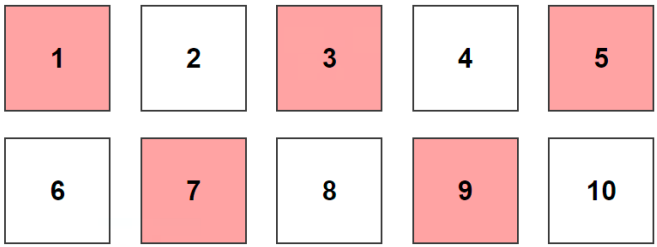


Figure 22

```
.box:nth-child(even) {background-color: #f8f48c;}
```

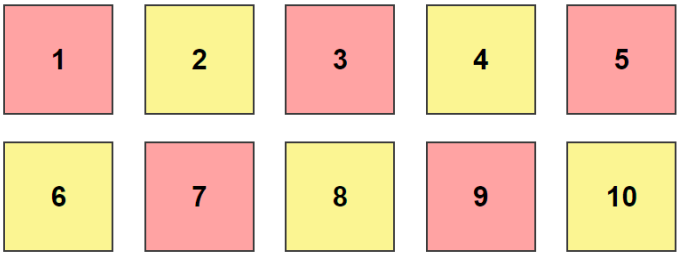


Figure 23

When writing a css-code, Brackets will prompt you when selecting pseudo-classes. Please pay attention to such

prompts — this speeds up the writing of code and prevents errors (Fig. 24, 25).

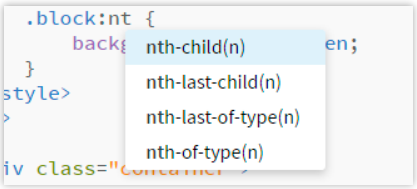


Figure 24

Figure 25

:Nth-last-child(n) Pseudo-class

The:nth-last-child(n) pseudo-class is intended for formatting elements that are not counted from the beginning of their appearance in the code, i.e. from the first element, and vice versa — from the end, i.e. from the last element.

In parentheses, instead of n, you can write a number, an expression (2n, 3n-1, 4n+1, etc.), as well as the keywords odd or even:

```
selector:nth-last-child( number | expression |
                         odd | even) {
    properties;
}
```

Related links:

1. https://www.w3schools.com/cssref/sel_nth-last-child.asp.
2. https://css-tricks.com/almanac/selectors/n/nth-last-child/.
3. https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-last-child.

:Only-child Pseudo-class

The :only-child pseudo-class is intended for formatting elements that are the only children of their parent tag. That is, it will be applied only if there are no other elements of the same type in the parent element for the specified selector.

In the *only-child.html* file in the *examples* folder of this lesson, you can find examples that illustrate the application of this pseudo-class.

The first two paragraphs contain the <mark> tags inside, designed to highlight the text. By default, it appears in the browser with a yellow background color. If such a tag occurs once, we change its appearance using the pseudo-class:

```
mark:only-child {
    background-color: #45abf5;
    color: #fff;
    padding: 3px;
    font-family: serif;
    font-style: italic;
}
```

The text in this element now has an italic style, white color, a blue background and a serif font.

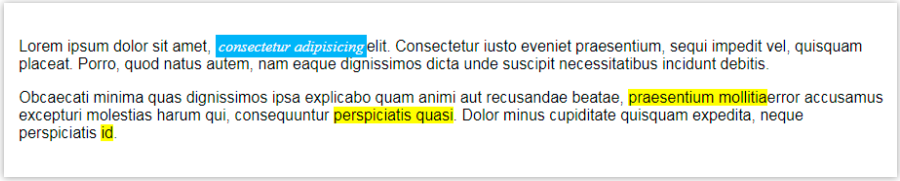


Figure 26

In the last two paragraphs of the *only-child.html* file you can find the <*spanclass=”elem”*> elements, for which the text color is changed to red and it is displayed in capital letters:

```
.elem {
    color: red;
    text-transform: uppercase;
}
```

But only in the paragraph, where there is one such element, it is also highlighted in bold:

```
.elem:only-child {
    font-weight: bold;
}
```

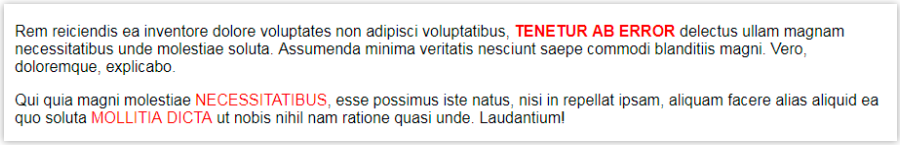


Figure 27

To some extent, the :only-child pseudo-class is similar to the :first-child or :last-child pseudo-classes, but they always apply, no matter how many similar elements exist, and rules written for :only-child will only work when the element, for which this pseudo-class is used, occurs only once in its container element. That is, it will also be the first (:first-child) and the last (: last-child) child for the parent element.

Related links:

1. https://developer.mozilla.org/en-US/docs/Web/CSS/:only-child.
2. https://www.w3schools.com/cssref/sel_only-child.asp.

:Nth-of-type Pseudo-classes

The :nth-of-type(n) pseudo-class is used to format elements of the same type within one parent element. This pseudo-class can be written with a number, an expression or *odd* or *even* keywords in parentheses:

```
selector:nth-of-type( number | expression | odd | even) {
    properties;
}
```

For example, you can write as follows:

```
.block:nth-of-type(3) { properties; }
p:nth-of-type(3n-1) {properties; }
.elem:nth-of-type(odd) {properties; }
```

In addition, you can use the:first-of-type or :last-of-type pseudo-classes for the first and last element of the same type inside the parent container.

There is also a :nth-last-of-type(n) pseudo-class, which allows you to specify formatting for the same type of elements, counting them from the bottom of the markup, i.e. from the last element inside one container.

Perhaps you are wondering — why do we need pseudo-classes that do almost the same thing? Why is it impossible to be limited only to pseudo-classes of :nth-child type?

The point is that pseudo-classes of :nth-child(n) type do not always behave as you expect, especially when changing the html-markup. Their incorrect behavior, from your point

of view, but not from the browser's point of view, is that when you add new elements in the markup to those for which you have set certain properties using :nth-child(n) and its varieties (:first-child, :last-child, :only-child), you will notice that the situation with the formatting of the elements has changed.

Let's look at a practical example, so that it becomes clearer what we are talking about.

Suppose that we need to create 4 blocks with the same formatting, but with different background colors. In the *blocks-nth-child.html* file from the *examples* folder, you will find the formatting of 4 blocks of the following form:

```
.block {
    display: inline-block;
    width: 21%;
    height: 150px;
    line-height: 170%;
    font-size: 2rem;
    font-weight: bold;
    text-align: center;
    margin: 2%;
    border: 2px solid;
}
```

We will set the background color using :first-child, nth-child(n), :last-child pseudo-classes:

```
.block:first-child {
    background-color: green;
}
.block:nth-child(2) {
    background-color: #32c132;
}
```

```
.block:nth-child(3) {
    background-color: #aff062;
}
.block:last-child {
    background-color: #dee863;
}
```

The appearance of the file is shown in the screenshot (Fig. 28):

Figure 28

All elements with the *block* class are currently located in the parent element with the *container* class:

```
<div class="container">
    <div class="block"><h2>Block 1</h2></div>
    <div class="block"><h2>Block 2</h2></div>
    <div class="block"><h2>Block 3</h2></div>
    <div class="block"><h2>Block 4</h2></div>
</div>
```

Let's say we need to add a header (a fairly common situation) to them, which will also be placed inside the element with the *container* class. Code will change as follows:

```
<div class="container">
    <h1>Our blocks</h1>
    <div class="block"><h2>Block 1</h2></div>
    <div class="block"><h2>Block 2</h2></div>
```

```
    <div class="block"><h2>Block 3</h2></div>
    <div class="block"><h2>Block 4</h2></div>
</div>
```

The appearance of the blocks in the browser will change too (Fig. 29).

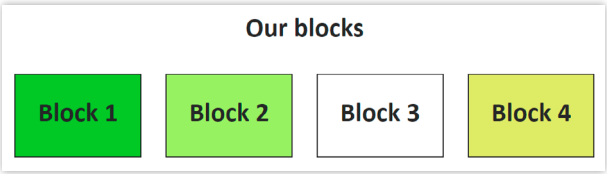


Figure 29

What happened? It will be easier to figure out with the help of the picture (Fig. 30).

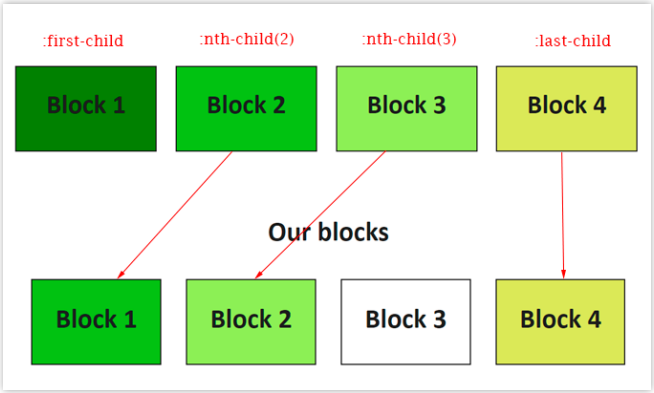


Figure 30

Arrows show “moved” css-rules. You can see that there was a displacement of one block to the left. That is, the element that obeyed the rules for :first-child now obeys the rules for the element with the :nth-child(2) pseudo-class, and the element for

which the properties described the :nth-child(2) pseudo-class now has the properties of the element with the :nth-child(3) pseudo-class. The third block generally lost the background color. Only formatting for :last-child remains unchanged.

Why did the "children" changed in the parent element with the *container* class? The answer is simple — precisely because there are more "children". When adding a header in an element with the *container* class, the number of child, or nested, elements increased, with the header becoming the first in the hierarchy of children, i.e. the :first-child pseudo-class should now refer to it. However, the css- rules are written for .block:first-child, so they do not apply to the header. If the rules were written for .container>:first-child, then we would see a dark green background in the header.

For the same reason, Block 1 became controlled by rules for *.block:nth-child(2)*, and Block 2 — by rules for *.block:nth-child(3)*. Since after the last block we did not add any new elements, formatting in the form of rules for *.block:last-child is* still actual for it. Let's change this by adding the <footer> element to the element with the *container* class at the bottom:

```
<div class="container">
    <h1>Our blocks</h1>
    <div class="block"><h2>Block 1</h2></div>
    <div class="block"><h2>Block 2</h2></div>
    <div class="block"><h2>Block 3</h2></div>
    <div class="block"><h2>Block 4</h2></div>
    <footer>&copy; nth-child blocks</footer>
</div>
```

Now the 4th block also "lost" the background color, since the last child (:last-child) for an element with the *container*

class is now the <footer> element, and the formatting was assigned for *.block:last-child*.

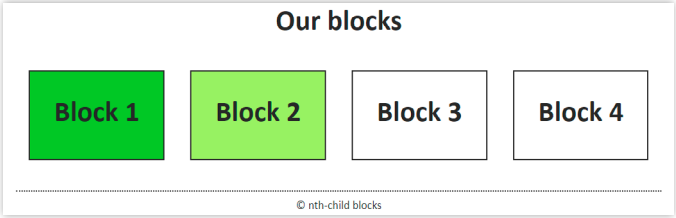


Figure 31

The situation may be bettered by changing the formatting set by :nth-child and similar pseudo-classes to :nth-of-type. A similar example, but with other pseudo-classes can be found in the *blocks-nth-of-type.html* file. The style code with pseudo-classes is as follows:

```
.block:first-of-type {
    background-color: green;
}
.block:nth-of-type(2) {
    background-color: #32c132;
}
.block:nth-of-type(3) {
    background-color: #aff062;
}
.block:last-of-type {
    background-color: #dee863;
}
```

In addition, both the header and the footer element remained in the html-markup. However, all the blocks in this file have a different background color, and in the form in which it was specified in the first file BEFORE adding a header and a footer.

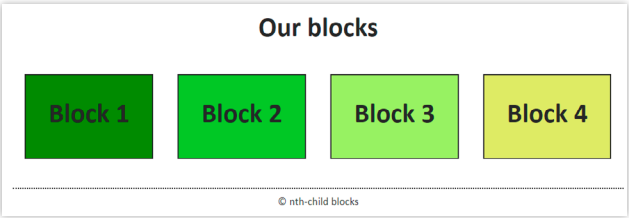


Figure 32

From this example, we can conclude that formatting child elements with pseudo-classes of the :nth-child type is suitable if the parent element contains *elements of the same type* — with the same tags or classes, for example. If there are *elements of different types* in the parent container markup, you need to use the :nth-of-type pseudo-classes.

Using Pseudo-classes for Page Block Markup

Let's look at another example in which we will need to format 4 such blocks, taking into account the use of different colors for text, background and frames. In addition, the blocks will be located offset from each other vertically.

The appearance of the blocks is presented in the screenshot (Fig. 33), the text for the blocks could be found in the *pseudo-classes-example-text.txt* file, and the finished example is in the *pseudoclasses-example.html* file.

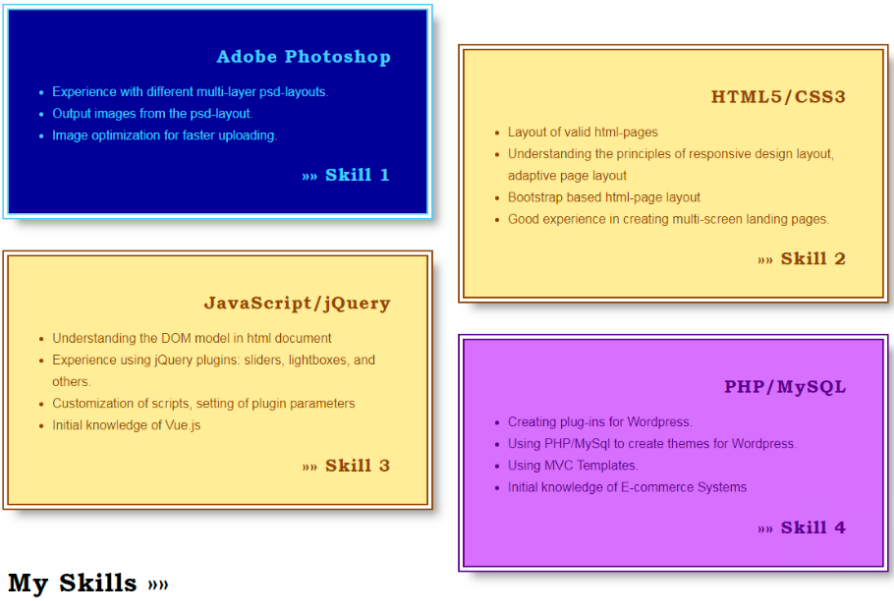


Figure 33

To begin with, change the file extension from txt to html. This can be done in the Brackets by pressing the F2 key. Only after this, select all the text (CTRL+A) and press the key shortcut CTRL+SHIFT+A.

In the field below with the *Enter Abbriviation* header, enter the exclamation mark (!) and get the basic structure of the html-document. Change the text in the <title> tag and continue using the CTRL+SHIFT+A key shortcut for other text elements (Fig. 34).

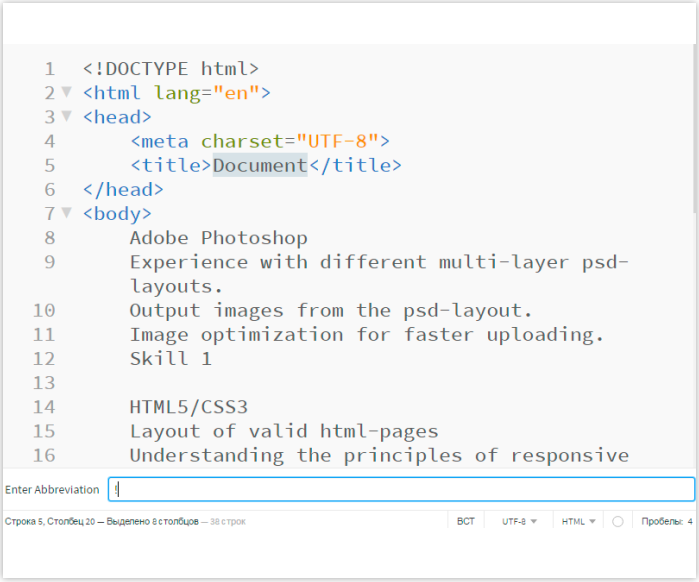


Figure 34

Since the text of each of the 4 blocks contains 2 headers (at the beginning and the end) and a bulleted list of 3-4 items, we will use wrapping for the same type of elements in each block, selecting the corresponding text with the CTRL key held down (Fig. 35).

```
<body>
    Adobe Photoshop
    Experience with different multi-layer psd-layouts.
    Output images from the psd-layout.
    Image optimization for faster uploading.
    Skill 1

    HTML5/CSS3
    Layout of valid html-pages
    Understanding the principles of responsive design layout, adaptive page layout
    Bootstrap based html-page layout
    Good experience in creating multi-screen landing pages.
    Skill 2

    JavaScript/jQuery
    Understanding the DOM model in html document
    Experience using jQuery plugins: sliders, lightboxes, and others.
    Customization of scripts, setting of plugin parameters
    Initial knowledge of Vue.js
    Skill 3

    PHP/MySQL
    Creating plug-ins for Wordpress.
    Using PHP/MySql to create themes for Wordpress.
    Using MVC Templates.
    Initial knowledge of E-commerce Systems
    Skill 4
```

Figure 35

After pressing the CTRL+SHIFT+A keys enter h2 in the field with *Enter Abbriviation* and get 4 headers of the second level (Fig. 36):

```
    <title>Document</title>
</head>
<body>
    <h2>Adobe Photoshop</h2>
    Experience with different multi-layer psd-layouts.
    Output images from the psd-layout.
    Image optimization for faster uploading.
    Skill 1

    <h2>HTML5/CSS3</h2>
    Layout of valid html-pages
    Understanding the principles of responsive design layout, adaptive page layout
    Bootstrap based html-page layout
    Good experience in creating multi-screen landing pages.
    Skill 2
```

Figure 36.1

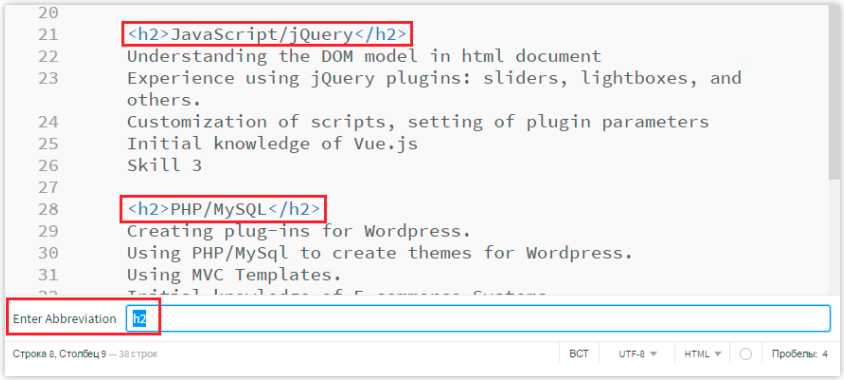


Figure 36.2

Similarly, select the text 'Skill' with a digit and wrap it in the header tags of the 3rd level h3 (Fig. 37).

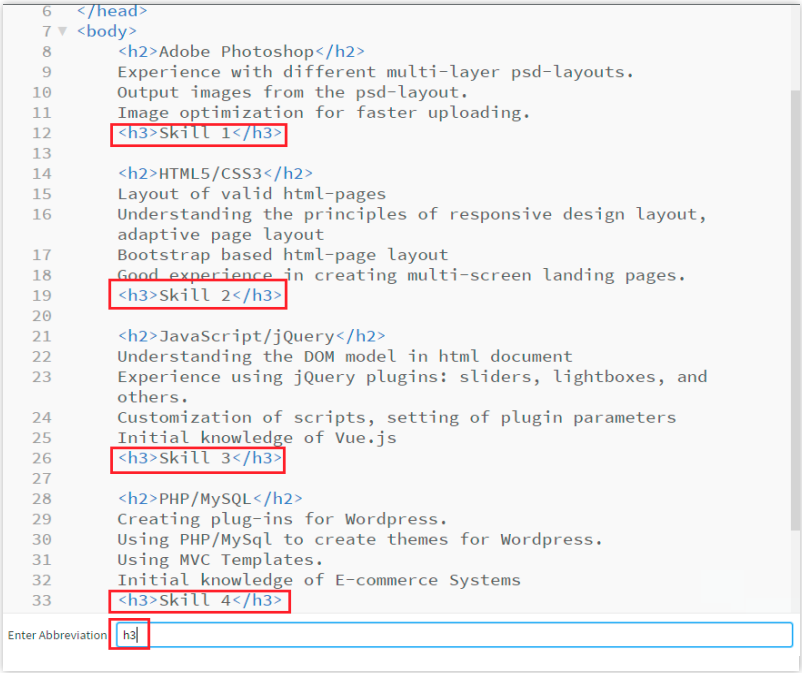


Figure 37

You can still improve our abbreviation by adding the html-specchar », which denotes the right double-quote character and is used in our blocks (Fig. 38).

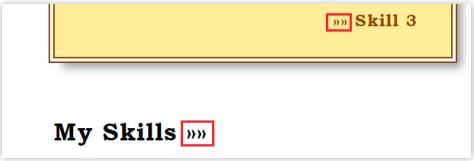


Figure 38

In this case, the abbreviation for wrapping tags is changed to h3>{» }.

Note that there is a space inside curly braces — this is the text that will be added before the word 'Skill' (Fig. 39).

```
7  <body>
8      <h2>Adobe Photoshop</h2>
9      Experience with different multi-layer psd-layouts.
10     Output images from the psd-layout.
11     Image optimization for faster uploading.
12     <h3>&raquo; Skill 1</h3>
13
14     <h2>HTML5/CSS3</h2>
15     Layout of valid html-pages
16     Understanding the principles of responsive design layout,
       adaptive page layout
17     Bootstrap based html-page layout
18     Good experience in creating multi-screen landing pages.
19     <h3>&raquo; Skill 2</h3>
20
21     <h2>JavaScript/jQuery</h2>
22     Understanding the DOM model in html document
23     Experience using jQuery plugins: sliders, lightboxes, and
       others.
24     Customization of scripts, setting of plugin parameters
25     Initial knowledge of Vue.js
26     <h3>&raquo; Skill 3</h3>
27
28     <h2>PHP/MySQL</h2>
29     Creating plug-ins for Wordpress.
30     Using PHP/MySql to create themes for Wordpress.
31     Using MVC Templates.
32     Initial knowledge of E-commerce Systems
33     <h3>&raquo; Skill 4</h3>
Enter Abbreviation  h3>{&raquo; }
```

Figure 39

When using the special character, we get only one quotation mark. The second one will be added using the ::before pseudo-element to get an idea of using one more method (Fig. 40).

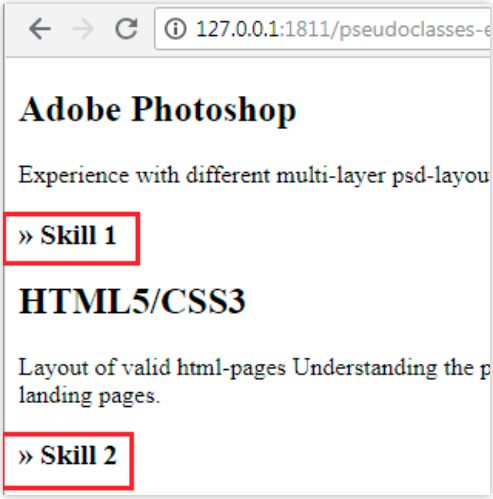


Figure 40

Another header, this time of the first level, is at the very bottom. First we add the special character » at the end, and then wrap it in the <h1> tags (Fig. 41).

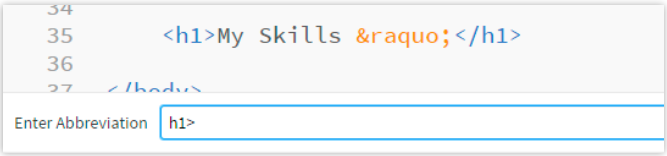


Figure 41

Now you need to wrap all the remaining lines of text in the list tags.

Select them using the CTRL key.

```
    <h2>Adobe Photoshop</h2>
    Experience with different multi-layer psd-layouts.
    Output images from the psd-layout.
    Image optimization for faster uploading.
    <h3>&raquo; Skill 1</h3>

    <h2>HTML5/CSS3</h2>
    Layout of valid html-pages
    Understanding the principles of responsive design layout, adaptive page layout
    Bootstrap based html-page layout
    Good experience in creating multi-screen landing pages.
    <h3>&raquo; Skill 2</h3>

    <h2>JavaScript/jQuery</h2>
    Understanding the DOM model in html document
    Experience using jQuery plugins: sliders, lightboxes, and others.
    Customization of scripts, setting of plugin parameters
    Initial knowledge of Vue.js
    <h3>&raquo; Skill 3</h3>

    <h2>PHP/MySQL</h2>
    Creating plug-ins for Wordpress.
    Using PHP/MySql to create themes for Wordpress.
    Using MVC Templates.
    Initial knowledge of E-commerce Systems
    <h3>&raquo; Skill 4</h3>
```

Figure 42

In the *Enter Abbriviation* field, we enter the abbreviation with the list tags and the * character at the end, which will allow us to add the <li> tags to all the lines that are formed in the text document using the line break.

```
ul>li*
```

```
    <h2>Adobe Photoshop</h2>
    <ul>
        <li>Experience with different multi-layer psd-layouts.</li>
        <li>Output images from the psd-layout.</li>
        <li>Image optimization for faster uploading.</li>
    </ul>
    <h3>&raquo; Skill 1</h3>

    <h2>HTML5/CSS3</h2>
```

Figure 43.1

```
    <ul>
        <li>Layout of valid html-pages</li>
        <li>Understanding the principles of responsive design
        layout, adaptive page layout</li>
        <li>Bootstrap based html-page layout</li>
        <li>Good experience in creating multi-screen landing pages.
        </li>
    </ul>
    <h3>&raquo; Skill 2</h3>

    <h2>JavaScript/jQuery</h2>
    <ul>
        <li>Understanding the DOM model in html document</li>
        <li>Experience using jQuery plugins: sliders, lightboxes,
        and others.</li>
        <li>Customization of scripts, setting of plugin
        parameters</li>
        <li>Initial knowledge of Vue.js</li>
```

Enter Abbreviation ul>li*

Figure 43.2

Now wrap the part of the already formed markup from the <h2> tag to the closing tag </h3> inclusive in the div with the *box* class:

```
.box
```

The procedure is repeated 4 times.

```
<body>
    <div class="box">
        <h2>Adobe Photoshop</h2>
        <ul>
            <li>Experience with different multi-layer psd-layouts.
            </li>
            <li>Output images from the psd-layout.</li>
            <li>Image optimization for faster uploading.</li>
        </ul>
        <h3>&raquo; Skill 1</h3>
    </div>

    <div class="box">
        <h2>HTML5/CSS3</h2>
        <ul>
            <li>Layout of valid html-pages</li>
            <li>Understanding the principles of responsive design
            layout, adaptive page layout</li>
            <li>Bootstrap based html-page layout</li>
```

Figure 44. 1

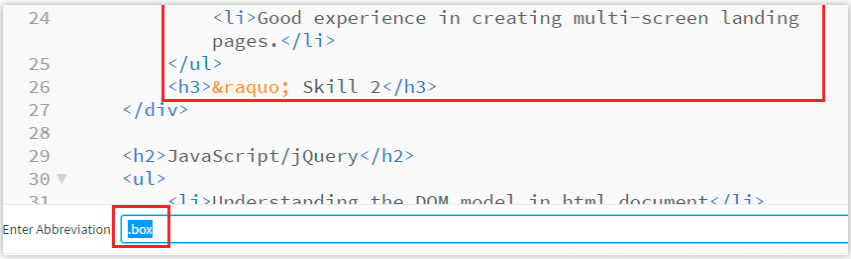


Figure 44.2

Last our work with wrapping text and tags in the abbreviation is that you need to select all the elements between the <body> </body> tags and enclose them in the div with the *container* class:

```
.container
// Emmet-abbreviation
```

Figure 45

The appearance of the page after adding all the tags is as follows (Fig. 46)

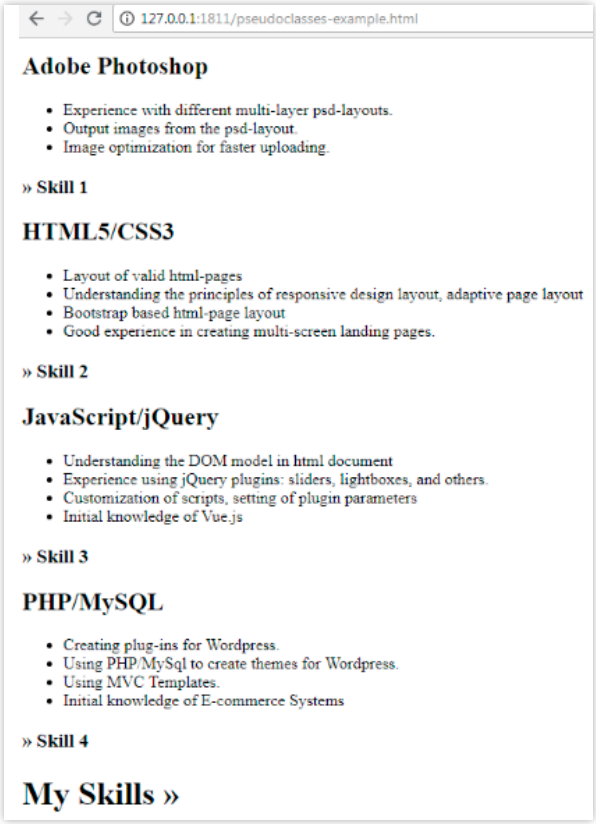


Figure 46

Now it's time to apply css-formatting. In the <style> tags in the <head> ... </head> block, place the following code:

```
*, *::before, *::after {
    box-sizing: border-box;
}
body { font-family: sans-serif; }
.container {
    width: 90%;
    max-width: 1200px;
    min-width: 768px;
```

```
    margin: 20px auto;
}
h1, h2, h3 {
    font-family: 'Bookman Old Style', serif;
    letter-spacing: 1px;
}
.box {
    width: 48%;
    border: 2px solid;
    padding: 4%;
}
```

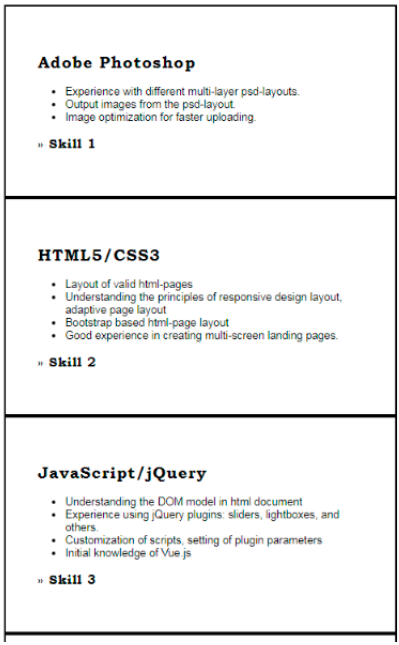


Figure 47

At the moment, we told the browser that the sizes of all elements, as well as the ::before and ::after pseudo-elements, will have padding and borders, specified the font family for the body,

and the minimum and maximum width for the div.container, and changed the font and space between letters (letter-spacing) for headers. For div.box, we specified the dimensions, style, and thickness of the frame, as well as padding. We received 4 blocks closely adjacent to each other in the browser (Fig. 47).

Now we need to distribute them as 2 on the left and 2 on the right and assign a background and text color. We will do this using pseudo-classes of the :nth-child type. All the odd blocks (*.box:nth-child(odd)*) will be placed to the left with the float: left css-property, and all even blocks (*.box:nth-child(even)*) — to right with float: right:

```
.box:nth-child(odd) {
    float: left;
}
.box:nth-child(even) {
    float: right;
}
```

We get this appearance of file in a browser (Fig. 48):

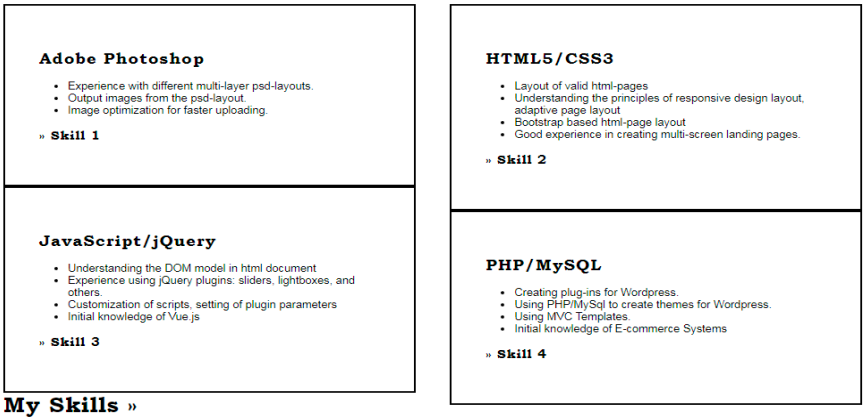


Figure 48

Add margins to our odd and even blocks with the *box* class:

```
.box:nth-child(odd) {
    float: left;
    margin-bottom: 50px;
}

.box:nth-child(even) {
    margin-top: 50px;
    float: right;
}
```

Here's what happened (Fig. 49):

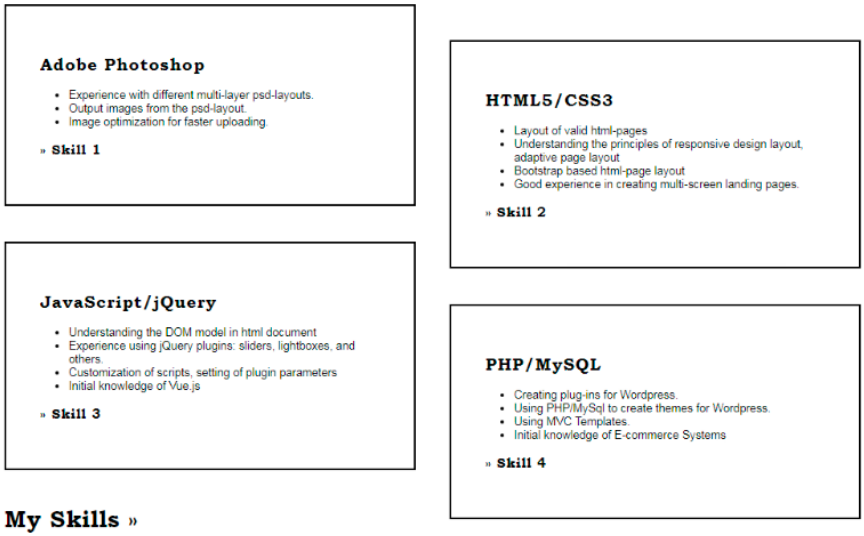


Figure 49

Notice that we used 2 selectors in order to place 4 blocks.

Now you need to add a color design. We will also set it using pseudo-classes:

```
.box:first-child {
    color: #79c6ff;
    background-color: darkblue;
}
.box:nth-child(2), .box:nth-child(3) {
    background-color: #ffea91;
    color: #773b08;
}
.box:last-child {
    color: #450080;
    background-color: #b864ff;
}
```

Note that when you specify the same rules for different blocks, enumeration is not as nth-child(2,3), but as .box:nth-child(2), .box:nth-child(3).

The appearance of the blocks has changed in a browser (Fig. 50):

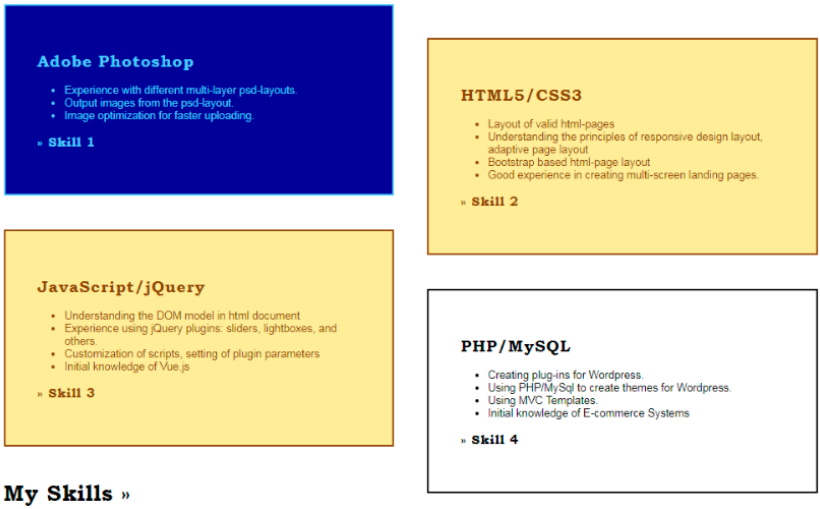


Figure 50

Three blocks of four really changed color, and only the last one remained without formatting. What's the matter?

The screenshot (Fig. 51) shows all the colors that were used to indicate the color of the text and the background color for each of the blocks. Please note that we do not specify the color of frames. By default, they are the same as the text color for a block.

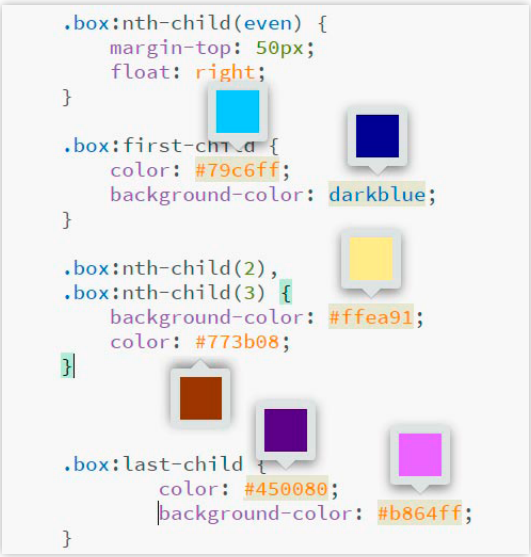


Figure 51

From this screenshot you can see that the colors of the first, second and third blocks coincide, and the last block should have a lilac background color and a purple text color. So, it's not about the wrongly set colors. The problem lies in the wrong selector. Since in our markup, the last div with the *box* class is followed by a first level header, then div.box IS NOT the last child for the parent div element with the *container* class, because the header is:

```
100 ▶          <div class="box"> ··· </div>
110
111            <h1>My Skills &raquo;</h1>
112        </div>
```

Figure 52

Therefore, we change the selector from .box:last-child to .box:last-of-type, because we need to format the last element of .box: type

```
.box:last-of-type {
    color: #450080;
    background-color: #b864ff;
}
```

Now the page in the browser looks as follows (Fig. 53):

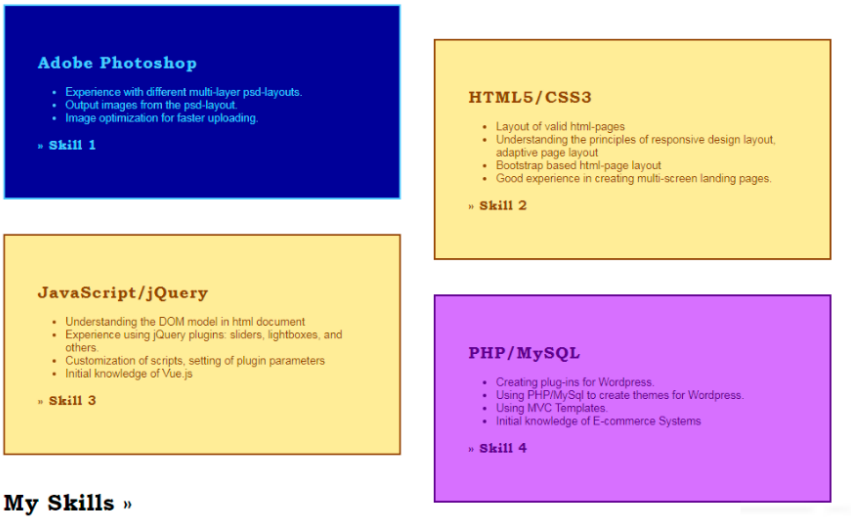


Figure 53

We'll add a bit of beauty to our blocks with the *box* class.

```
.box {
    width: 48%;
    border: 2px solid;
    padding: 4%;
    outline: 2px solid;
    outline-offset: 4px;
}
```

The outline property is similar to the border property, but it specifies the outer border for the element outside of border, but it does not affect the width of the element. In terms of writting, this property is universal, or composite, i.e., consists of 3 properties:

```
outline: outline-color || outline-style ||
         outline-width | inherit
```

All component properties are the same for the border composite property, i.e. for border-color, border-style and border-width, which we considered in the last lesson. Unlike border, outline can be set at once for all sides, it is impossible to set it for each one separately. This property, because it does not add additional pixels to the element's dimensions (width, height), is used to highlight shape elements and links when the focus is shifted to them.

We also use the outline-offset property, which sets the distance between the frame from the outline property and the borders of an element. The downside of this property is that it is not fully supported by browsers.

In this case, we use it to get acquainted with this property (Fig. 54).

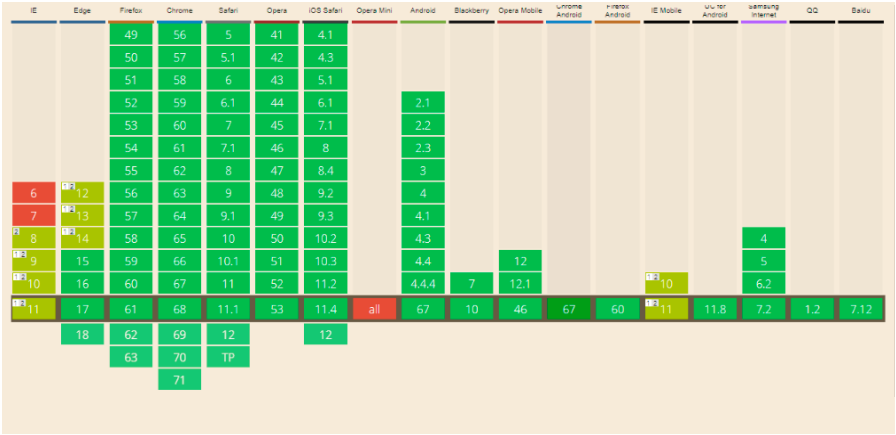


Figure 54

So, we got the following appearance of elements with the *box* class:

Figure 55

Now add a shadow:

```
.box {
    …;
    box-shadow: 6px 6px 16px rgba(0, 0, 0, 0.4);
}
```

Figure 56

This kind of shadow is very weakly visible against the 2 frames formed by the border, outline and outline-offset properties. So make 2 shadows: one white, without offset, which will take 6px (2px outline-width + 4px ouline-offset), and also increase the shift of the dark shadow:

```
.box {
    width: 48%;
    border: 2px solid;
```

```
    padding: 4%;
    outline: 2px solid;
    outline-offset: 4px;
    box-shadow: 0 0 0 6px #fff,
                12px 12px 16px
                rgba(0, 0, 0, 0.4);
}
```

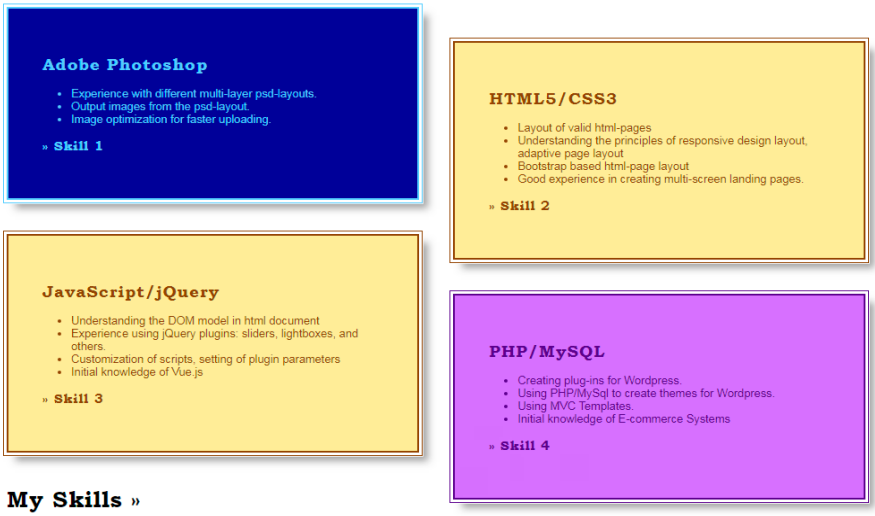


Figure 57

The next task is to design the headers. It is now clear that the indents to the headers of the second and third level are too far from the frame.

```
.box h2, .box h3 {
    font-size: 1.4em;
    text-align: right;
}
```

```
.box h2 { margin-top: 0; }
.box h3 { margin-bottom: -2%; }
```

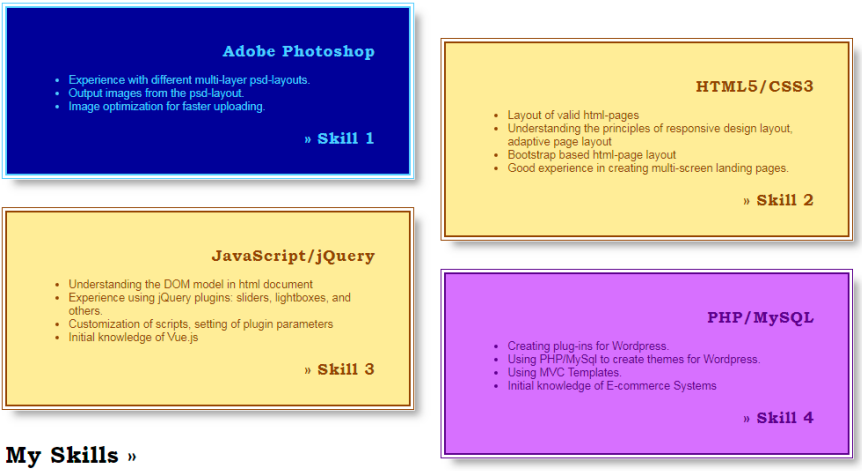


Figure 58

We add one more character of double right quotes. To do this, you need to convert the description of the special character into hexadecimal code. You can match the » on css-tricks.com pressing CTRL+F.

»	»	»	angle quotation mark, right	%BB	p:before { content:"\00bb"; }	alert("\273");

Figure 59

In the end we get the following code:

```
h3:before, h1:after {
    content: '\bb ';
}
```

In each header h3 and h1, the second character of the double right quotation mark is added either before the text (h3:before), or after (h1:after).

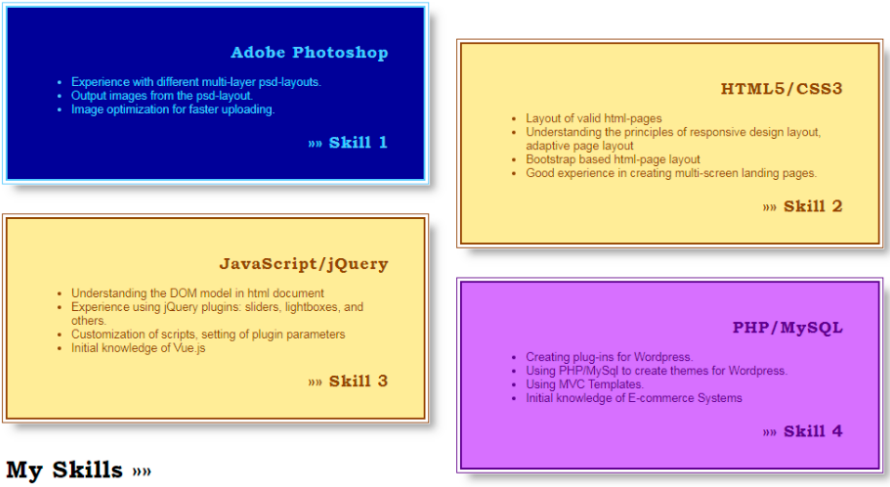


Figure 60

> **Note.** *This method can be used where you do not have access to html markup for any reason, but there is access to css rules, for example, when setting up a CMS template (content management system such as Wordpress, Joomla or Droopal).*

The last thing that remains is the wrap clearing, which we spoke about in the last lesson. If we now set a gray background color for the header of the first level, we get an interesting picture.

```
h1{background-color: #ccc;}
```

It turns out that the header spread to the whole area of our blocks. This is very clearly seen in the screenshot below (Fig. 61).

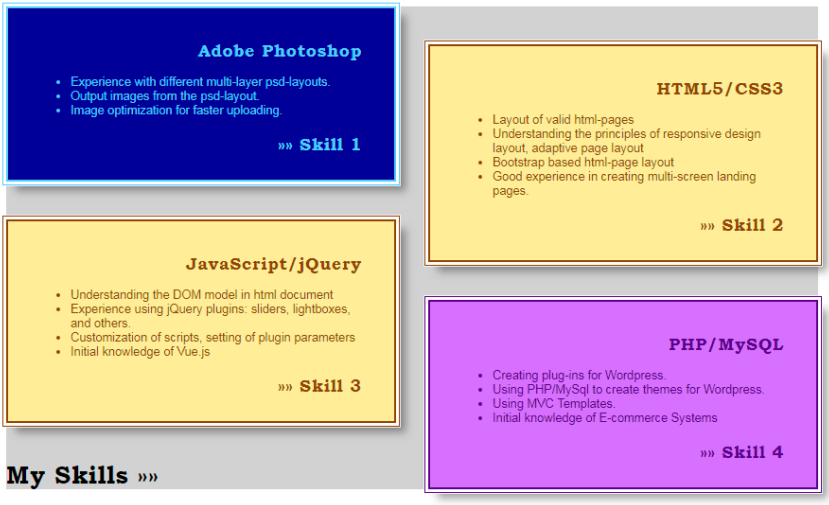


Figure 61

Since the h1 element is at the very bottom of our markup, it makes sense to undo wrapping with the help of the clear:both property for this element (Fig. 62):

```
h1 {
    background-color: #ccc;
    clear: both;
}
```

Now the header has taken its usual size according to the size of the text. Remove the selection with the background color, leaving one rule (Fig. 63):

```
h1 { clear: both; }
```

The final version of the *pseudoclasses-example.html* file.

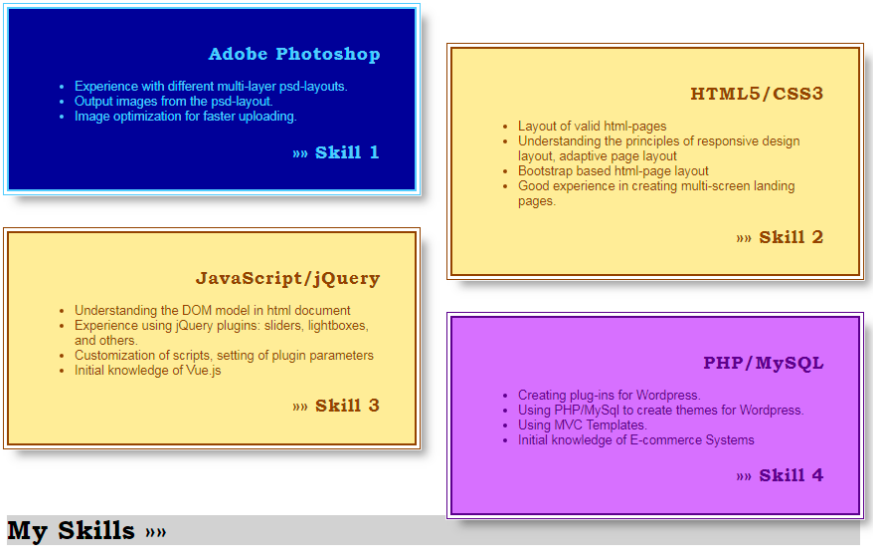


Figure 62

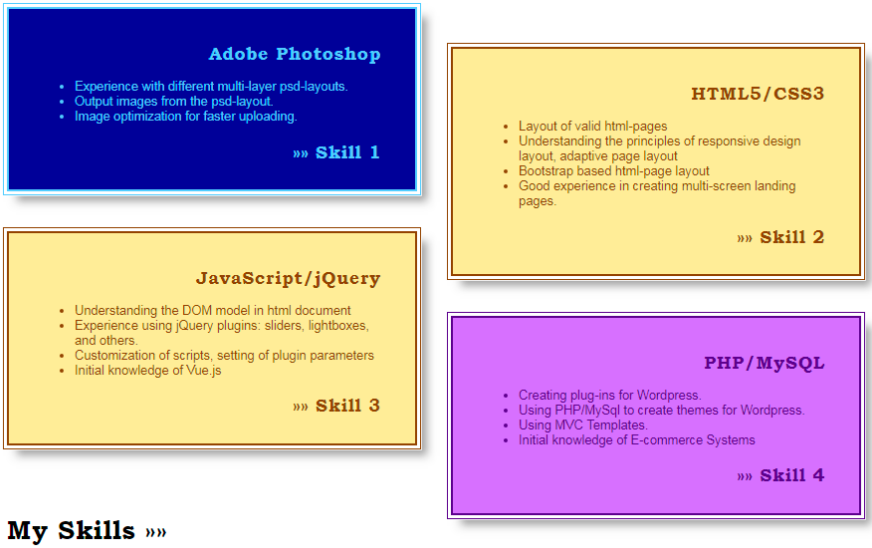


Figure 63

Related links:

1. https://www.w3schools.com/cssref/sel_nth-of-type.asp.
2. https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-of-type.
3. https://css-tricks.com/snippets/html/glyphs/.

Creating Columns Using Inline-block Elements

Let's return to inline-block elements and consider how we can use them when formatting various elements of the html-page.

Another advantage of inline-block elements in comparison with inline ones is the ability to create columns when assigning a width property to them. Due to they are inline, these elements are arrayed in columns next to each other without using the wrapping with the float: left or right property and their inherent flaws (zeroing the height of the parent element, wrapping by below elements).

Here, however, are also surprises. The feature of inline-block elements is that a browser calculates their width with one space indent after the closing tag. The size of this indent depends on the size and family of the font used and usually is 3-5 px at the standard font size of 16 px. If you want to use inline-block elements to create columns, you will need to take this indent into account when creating the columns.

In this section, we will consider 2 ways to make a part of a page in the form of columns and an entire page in the form of a grid using display: inline — block elements, as well as ways to "get rid" of a space indent.

The first example is in the *column-inline-block.html* file in the examples folder of this lesson. In it we will create 4 columns.

To begin with, we use the *Emmet abbreviation* to form the html-markup of a block with columns:

```
.wrap>h1{The Best Offer}+.product*4>h2.
product-title{Product $}+.product-descr>p*2>lorem12^^.
price[data-price]{Price}+a.btn[#]>{Add to cart}
```

This rather complex abbreviation will create a div element with the *wrap* class, inside of which there will be a header of the first level and 4 div-s with the *product* class. Each of the product div-s has a second level header with the *product-title* class and 'Product' text with a sequence number from 1 to 4, which creates the Emmet plug-in from the sign of $ and *4. The header is followed by a div with a *product-descr* class with two paragraphs of a template text of 12 words. After the product description (*product-descr*), in the parent element *.product,* there is a div with the *price* class, which has the *data-price* attribute in the brackets after the class name. We will use it to create a price using the ::after pseudo-element. At the very end of the abbreviation there is a link with the *btn* class and the text 'Add to cart'.

Actually, you can quickly see all this long description if you press the Tab key at the end of the abbreviation:

```
<div class="wrap">
    <h1>The Best Offer</h1>
    <div class="product">
        <h2 class="product-title">Product 1</h2>
        <div class="product-descr">
            <p>Lorem ipsum dolor sit …</p>
            <p>Perferendis cum accusantiumducimus,
               eamagni…
            </p>
        </div>
```

```
        <div class="price" data-price="22.99">
            Price
        </div>
        <a href="#" class="btn">Add to cart</a>
    </div>

    <div class="product">
        <h2 class="product-title">Product 2</h2>
        <div class="product-descr">
            <p>Lorem ipsum dolor sit amet…</p>
            <p>Voluptatum corporis ...</p>
        </div>
        <div class="price" data-price="14.99">
            Price
        </div>
            <a href="#" class="btn">Add to cart</a>
    </div>

    …
</div>
```

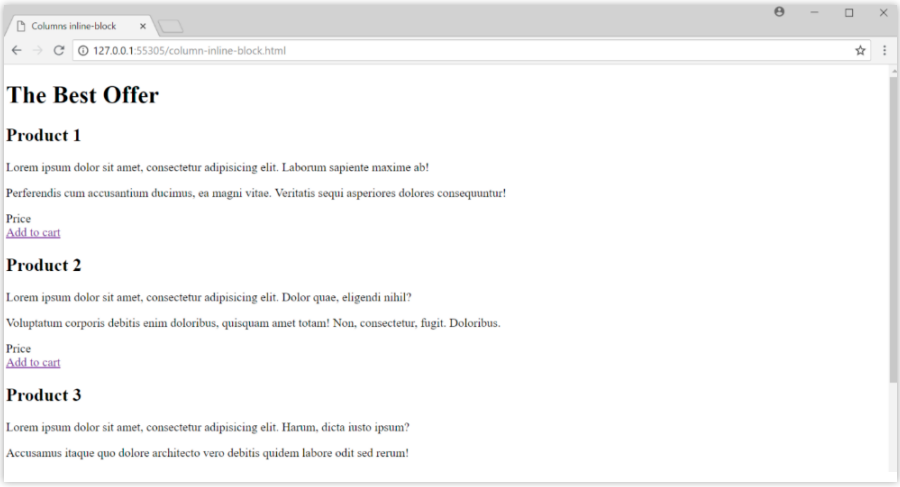


Figure 64

In this markup, the figures have already been added in the *data-price* attribute. You can arrange them yourself.

At the moment, the markup looks very simple in a browser (Fig. 64).

Let's set css-formatting:

```
body {
    font-family: Verdana, Geneva, sans-serif;
    background-color: #bcbcbc;
}
.wrap {
    max-width: 1200px;
    width: 90%;
    min-width: 768px;
    margin: 20px auto;
}
h1 {
    text-align: center;
    margin-bottom: 2em;
}
```

Since such formatting rules met often enough, I will not comment on them. Changes on the page look like this (Fig. 65):

The div's width with the *wrap* class for the elements nested in it is 100%. Thus, for each of the products that we have 4, the width should be 25%, calculated by such a simple formula: 100%:4 = 25%.

We use this value for the css rules of the *product* class:

```
.product {
    display: inline-block;
    width: 25%;
    background-color: #fff;
}
```

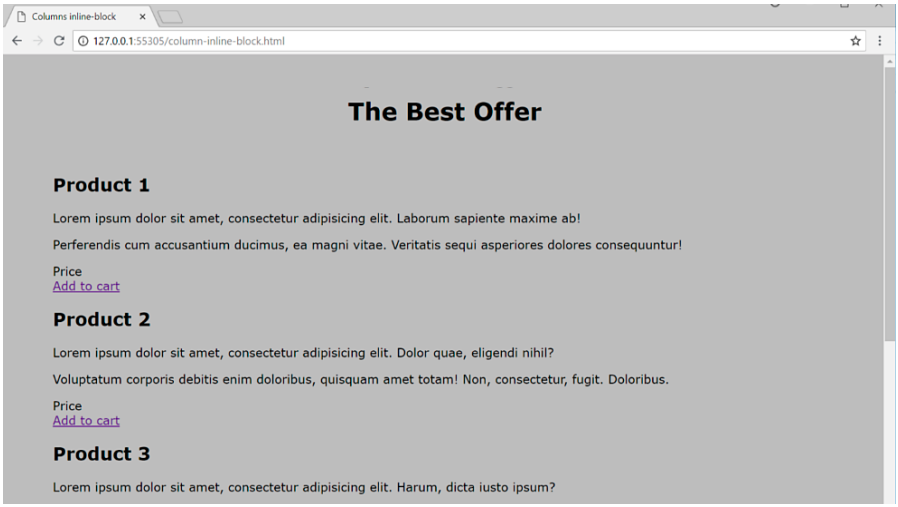


Figure 65

We will get an interesting situation, in which a small distance appeared between the blocks, although we did not set margin for them, but the last column "moved" to the bottom line (Fig. 66).

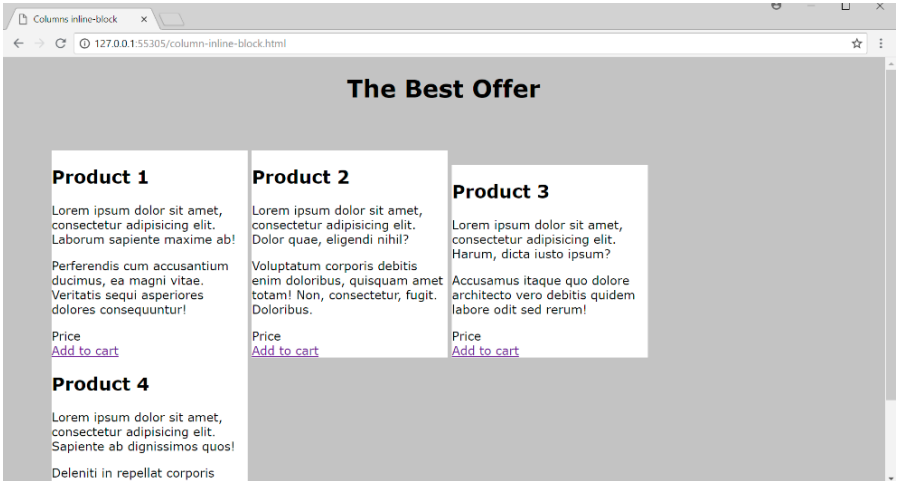


Figure 66

What's the matter? Why did it happen? The created picture is related to the display of inline-block elements, namely, that the closing tag (in our case for the *product* class) and the line break after it, executed by the Enter key, is converted to one space in HTML for any inline or inline-block elements. It is this that we observe between the columns. Its value is approximately 5 px or 0.25-0.3em for the font selected by us. However, this smallness is enough to move the last column down.

In order to remove this indent, we use the function calc(), which appeared CSS 3:

```
.product {
    display: inline-block;
    width: calc(25% - .27em);
    background-color: #fff;
}
```

The situation with the columns has changed — now they are placed in one row, and at any screen resolution (Fig. 67, 68).

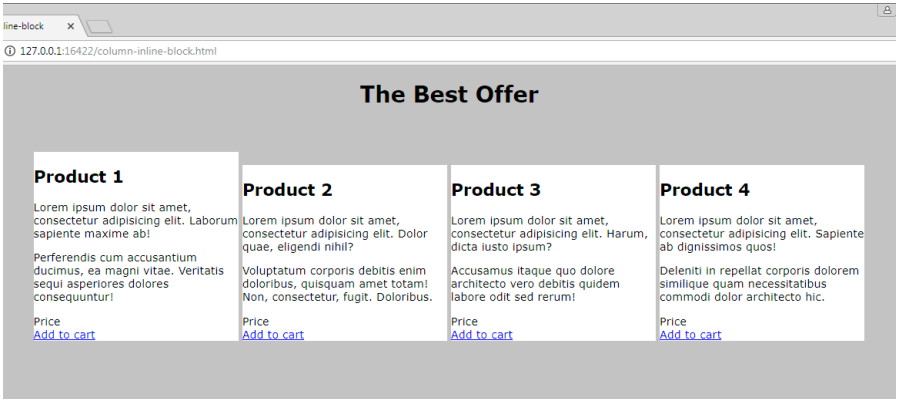


Figure 67

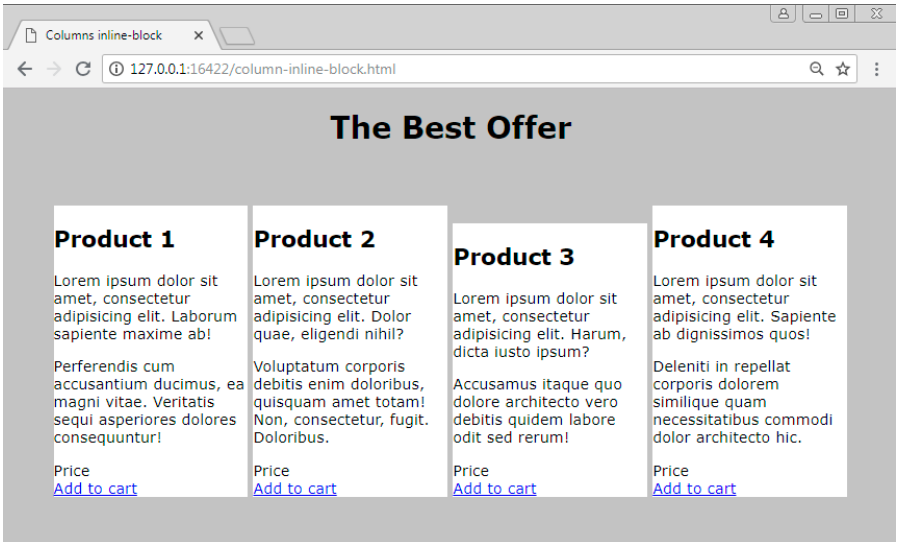


Figure 68

Now it is desirable to align blocks with products along the top line. The vertical-align property with the top value is designed for this (Fig. 69):

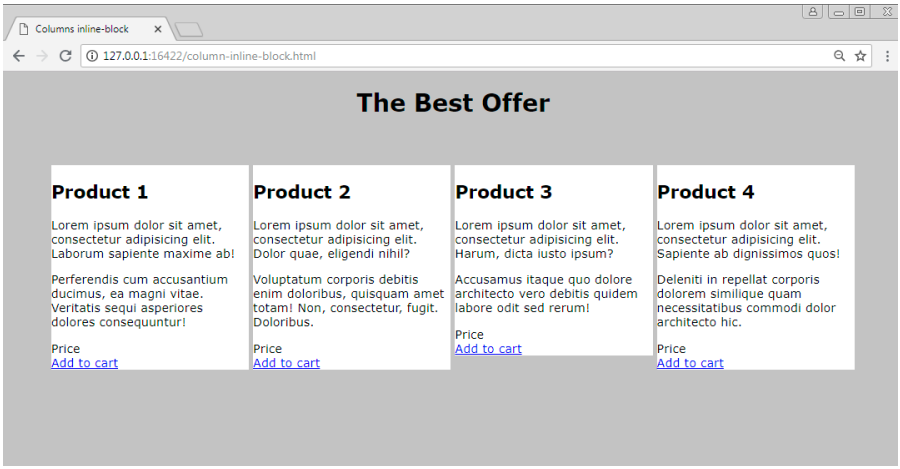


Figure 69

```
.product {
    display: inline-block;
    width: calc(25% - .27em);
    background-color: #fff;
    vertical-align: top;
}
```

To improve the appearance, add padding to the *product* class.

```
.product {
    display: inline-block;
    width: calc(25% - .27em);
    background-color: #fff;
    vertical-align: top;
    padding: 0 15px;
}
```

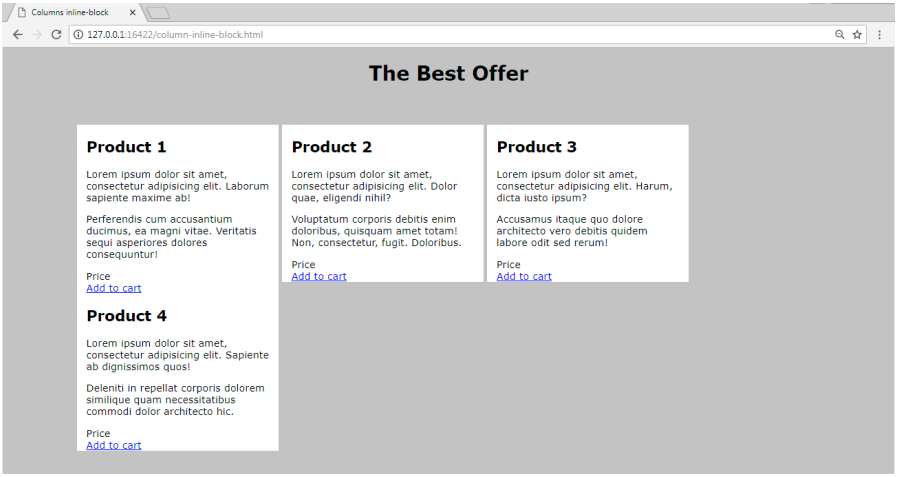


Figure 70

After this, we again get the “moving” of the last column down. Now this offset is due to the increase in the calculated

width of the column in a browser due to the fact that the real width is made up of the width property and the padding property, i.e. the value of 25% – .27em added 30px of padding, and the columns again ceased to fit 100% of the width of the parent element with the *wrap* class (Fig. 70).

This problem is solved by decreasing the value of the width property or by using box-sizing: border-box:

```
.product {
    display: inline-block;
    width: calc(25% - 30px - .27em);
    background-color: #fff;
    vertical-align: top;
    padding: 0 15px;
}
```

or:

```
.product {
    box-sizing: border-box;
    display: inline-block;
    width: calc(25% - .27em);
    background-color: #fff;
    vertical-align: top;
    padding: 0 15px;
}
```

Figure 71

The second way is more universal, so use it in the code (Fig. 71).

Indents of one space is perhaps too small to separate our columns. Therefore add margin-right of 2% and reduce width of the column by this value. Also, use margin-bottom to add a margin at the bottom:

```
.product {
    box-sizing: border-box;
    display: inline-block;
    width: calc(23% - .27em);
    background-color: #fff;
    vertical-align: top;
    padding: 0 15px;
    margin-right: 2%;
    margin-bottom: 20px;
}
```

Now everything seems to be in order. However, if you open the Property inspector, you can see that there is extra space to the right of the last column. This is margin-right of the last column (fig 72).

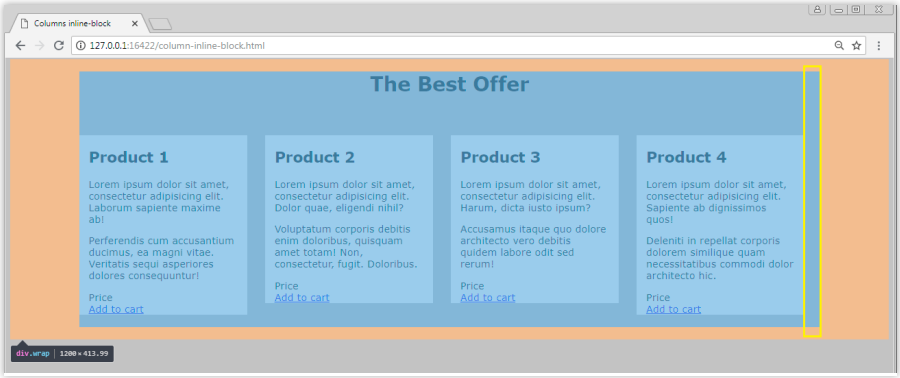


Figure 72

For our example, this is not very important, but if the page has several sections of the same size, this margin will catch your eye. Therefore, we need to redistribute the space of margins between the columns in such a way that after the last column there is no margin.

To do this, we determine that we need 3 margins of 2% each. Accordingly, we have 100% – 3*2% = 94% for columns. The width of each column will be 94%/4 = 23.5%.

Do not forget to use the calc() function for the width property:

```
.product {
    box-sizing: border-box;
    display: inline-block;
    width: calc(23.5% - .27em);
    background-color: #fff;
    vertical-align: top;
    padding: 0 15px;
    margin-right: 2%;
    margin-bottom: 20px;

}
```

In addition, in order not to get the last column moved once again, it is necessary to disable margin-right for it setting this property to 0. Let's make this using the :last-child pseudo-class:

```
.product:last-child {
    margin-right: 0;
}
```

Now all 4 columns fit perfectly within the limits of the parent element (Fig. 73):

Figure 73

Next, we design the header and description of a product:

```
.product-title {
    text-align: center;
    border-bottom: 2px dashed;
    padding-bottom: 10px;
}
.product-descr {
    font-family: Cambria, serif;
}
```

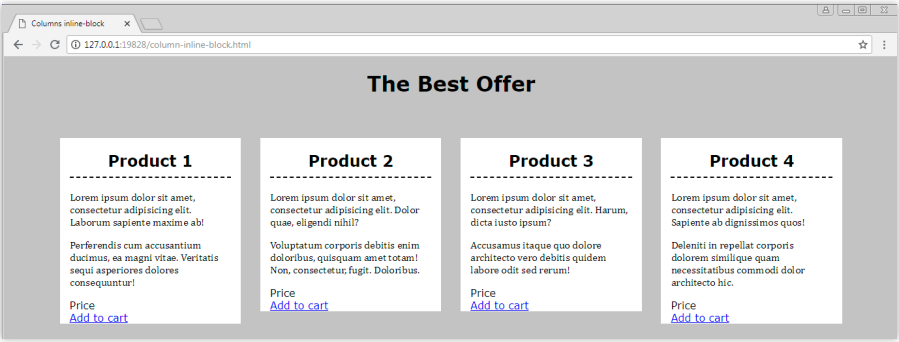


Figure 74

For the *price* class, we use the design with the::after pseudo-class. Let me remind you that in html-markup we used

data-attribute data-price with different values for different blocks with the *product* class:

```
82    <div class="wrap">
83        <h1>The Best Offer</h1>
84        <div class="product">
85            <h2 class="product-title">Product 1</h2>
86            <div class="product-descr">···</div>
90            <div class="price" data-price="22.99">Price</div>
91            <a href="#" class="btn">Add to cart</a>
92        </div>
93        <div class="product">
94            <h2 class="product-title">Product 2</h2>
95            <div class="product-descr">···</div>
100           <div class="price" data-price="14.99">Price</div>
101           <a href="#" class="btn">Add to cart</a>
102       </div>
103       <div class="product">
104           <h2 class="product-title">Product 3</h2>
105           <div class="product-descr">···</div>
110           <div class="price" data-price="16.59">Price</div>
111           <a href="#" class="btn">Add to cart</a>
112       </div>
```

Figure 75

For the content property, which must be specified in the ::after pseudo-element, the attr() function is available; its parentheses should contain the attribute whose value is to be placed into the function:

```
.price:after {
    content:
}
```

attr()

Figure 76

Add a $ sign in front of the function in quotation marks and get the following rules:

```
.price:after {
    content: "$"attr(data-price);
    color: red;
}
```

Figure 77

By using different values in the data-price attribute for different columns, we get a variety of price values for a product.

> **Note.** *in fact, we could create any span in the div with the price class and write down the value for the product price in it, and then set the red color of the text for the span. Nevertheless, data-attributes with a price may well appear in real code, but, most likely, they will record the price of a product in different currencies, for example:*
> * Price $14.99</> *
> *Using JavaScript, this data can be substituted into the span with the price-value class and shown to the user when choosing another currency on a website without reloading the page.*

Set the bold font for the div with the *price* class, and add an indent from the bottom, and move the text of the ::after pseudo-element to the right using the float: right property:

```
.price {
    font-weight: bold;
    margin-bottom: 10px;
}

.price::after {
    content: "$"attr(data-price);
    color: red;
    float: right;
}
```

Figure 78

The last element that needs formatting is the link with the text 'Add to cart' and the *btn* class. For it, we will write the following css-rules:

```
.btn {
    text-decoration: none;
    background-color: #f00;
    color: #fff;
    display: block;
    width: 100px;
    padding: 7px 20px;
    border-bottom: 2px solid black;
    margin: 15px auto 20px;
    text-align: center;
}
```

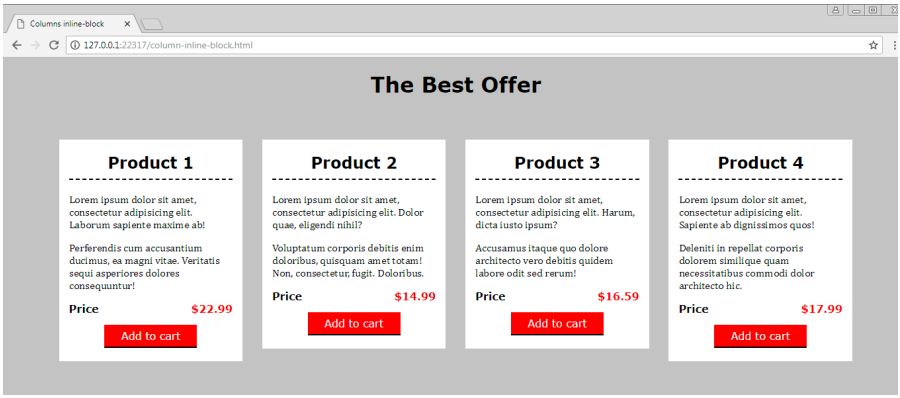


Figure 79

The final touch in the formatting of these blocks is the height adjustment of the columns with the *product* class. To do this, we use the Property inspector (F12 or CTRL+SHIFT+I). The screenshot shows that for large screens, the minimum block height should be approximately 335px.

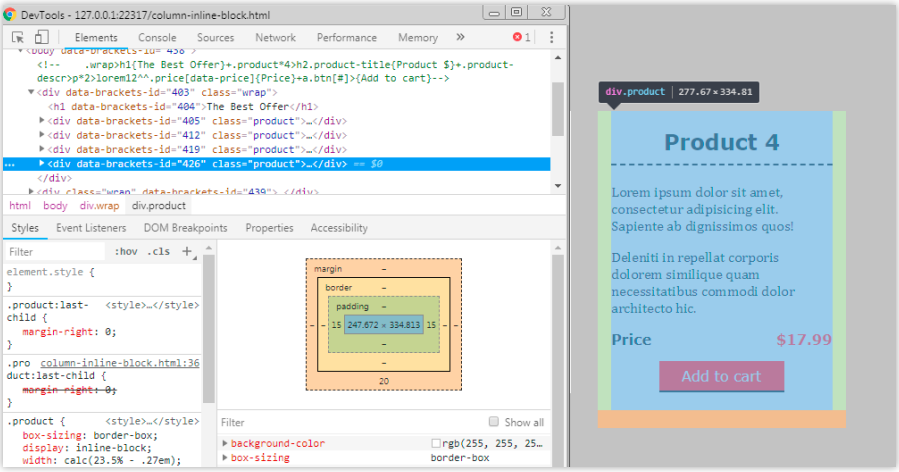


Figure 80

We use this value for the min-height property:

```
.product {
    box-sizing: border-box;
    display: inline-block;
    width: calc(23.5% - .27em);
    min-height: 335px;
    background-color: #fff;
    vertical-align: top;
    padding: 0 15px;
    margin-right: 2%;
    margin-bottom: 20px;
}
```

The result of the formatting is shown in the screenshot (Fig. 81):

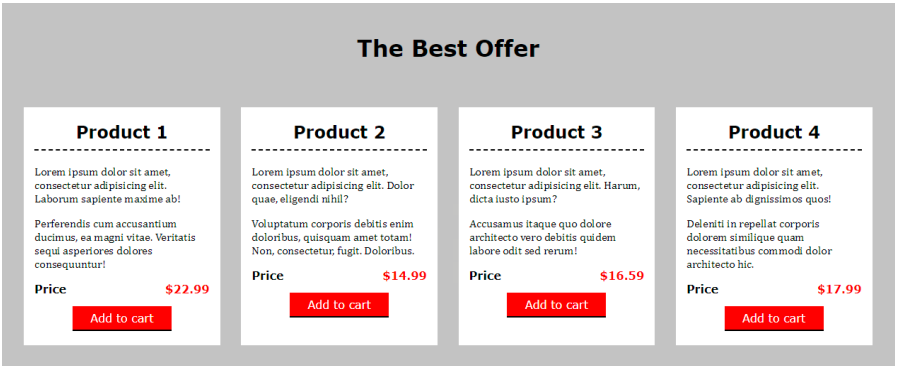


Figure 81

Related links:

1. https://developer.mozilla.org/en-US/docs/Web/CSS/calc.
2. https://css-tricks.com/a-couple-of-use-cases-for-calc/.
3. https://developer.mozilla.org/en-US/docs/Web/CSS/vertical-align.
4. https://www.w3schools.com/cssref/pr_pos_vertical-align.asp.

Use of Inline-block Elements for Page Layout

Consider how you can create a simple adaptive page based on inline-block elements (Fig. 82). In this example, adaptability will be determined by using the units of measurement for the width and height properties in % and vmax(vh), which are translated by the browser in px, depending on the width and height of the window. The code and appearance of the page can be seen in the *display-inline-block-grid.html* file in the *examples* folder of the lesson.

Figure 82

For markup we use a number of article elements with *grid-item* and *item with a digit* classes, because It is assumed that the page will contain introductory data about several articles with links to the main content. We will put them in a main element, which will need to assign some css-properties.

```
<main>
    <article class="grid-item item1">
        <h4 class="place">California</h4>
        <h2 class="post-title">Jumping Around</h2>
        <p>California's last empty jump spots</p>
        <a href="#">Discover more</a>
    </article>
    <article class="grid-item item2">
        <h4 class="place">Italy</h4>
        <h2 class="post-title">Calm Sirenity</h2>
        <p>Italy's secrets meadows and fields</p>
        <a href="#">Check them out</a>
    </article>
    …
</main>
```

The Emmet abbreviation for creating the basis of the code is found in the comments in the body of the *display-inline-block-grid.html* document or you can copy it below:

```
main>article.grid-item.item$*6>h4.place+h2.
post-title+p+a[#]{Read more}
```

As for using 2 classes for one element, this is not only permissible for any element, but it is often used in markup. In our case, the first class, *grid-item, is* necessary for assigning general rules for formatting articles, and the *item with a digit* class — for assigning different background colors.

```
.grid-item {
    display: inline-block;
    width: 50%;
    min-width: 370px;
    height: 50vmax;
}
.item1 {background-color: #ff0;}
.item2 {background-color: #f00;}
.item3 {background-color: #0f0;}
```

The CSS rules for the *.grid-item* class specify a display of the initially block article element as inline-block with a width of 50% (that is, half of the available space of the main parent element). In order for the content to look nice on different screens, the minimum width (*min-width*) should be set at 370px, and for vertical division of the page into 2 blocks we set the height of the article (*height*) to 50vmax.

Figure 83

However, the appearance of the page is far from what we planned (Fig. 83).

For more details on vmax units, see an article, although they were considered in the material of the first lesson: https://web-design-weekly.com/2014/11/18/viewport-units-vw-vh-vmin-vmax/.

Why did it happen? First, the body element has margins of 8px by default. Second, the article elements with the *display: inline-block* property have 1 space indent, which does not allow placing two elements side by side, as is done for block elements that have a float property with the value left or right. We highlight this indent in the previous example.

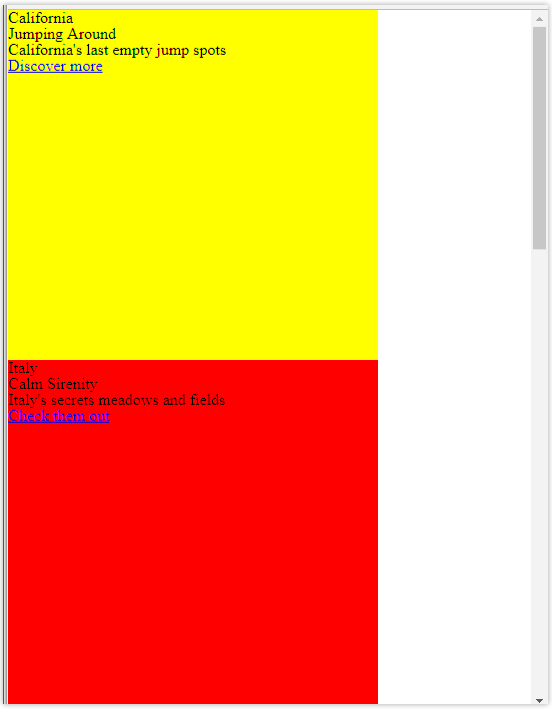


Figure 84

In order to remove the margins in the body and the headers, we'll create a *css* folder in the *examples* folder, place the reset.css file created by Eric Meyer in it, and then add the code from it to the page. To do this, add the <link> tag in the <head> block after the <title> tag:

```
<link rel="stylesheet" href="css/reset.css">
```

Emmet abbreviation for creating this tag — link.

The href attribute of this tag records the path to the css file that you want to load to format the html page.

After adding a link to this file in the link tag, all the css-rules described in it apply to the elements of our page (Fig. 84).

If you carefully compare the last two screenshots (Fig. 83, 84), you can see that after applying the rules from the reset.css file, not only the margins of the body and headers disappeared, but also the headers "lost" the bold style of the font + all the elements began to have the same font size.

However, the colored blocks are still positioned one below the other. To avoid this, let's add a negative margin to the right for the *grid-item* class:

```
.grid-item {
    …;/* all previous css-properties */
    margin-right: -.25em;

}
```

Now you can see how the *article* elements with the *.grid-item* class are located next to each other (Fig. 85).

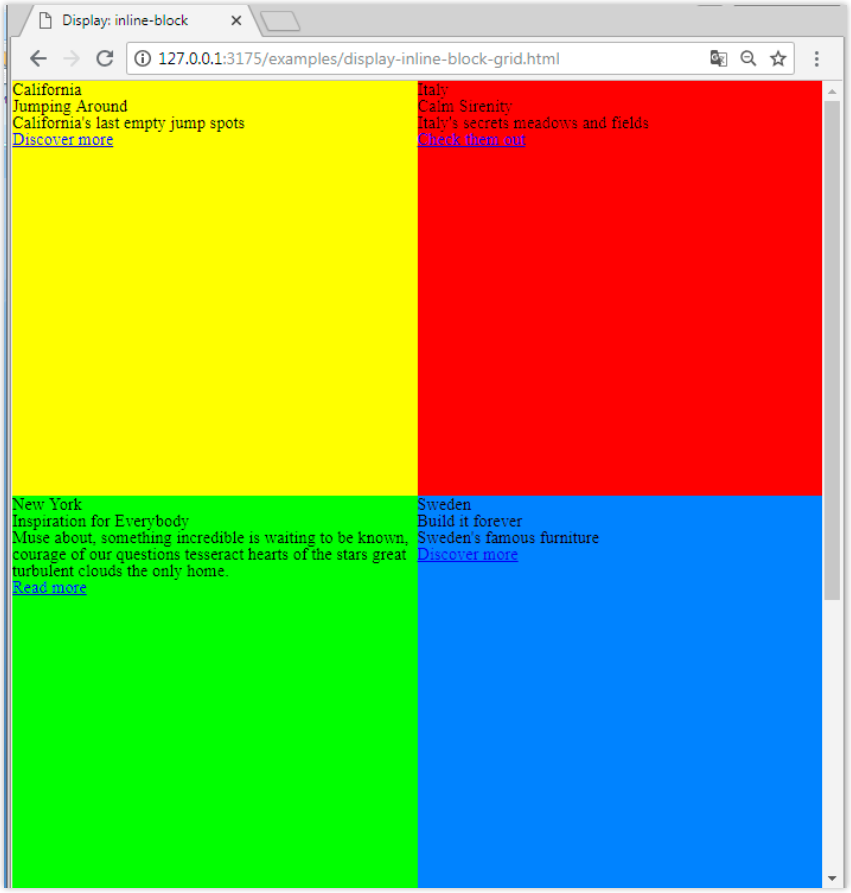


Figure 85

The second way to remove a space indent in inline-block elements is to set the parent element (in our case main) to font-size: 0, and the child (*.grid-item*) — font-size: 1rem. And the *.grid-item* font size (font-size) necessarily have to be specified in units of rem, which are calculated relative to the root html element and often are taken from the browser settings (usually 16px), and not from the parent element's font size with font-size set to 0.

```
main {font-size: 0; }
.grid-item {
… ;/* all previous css-properties */
font-size: 1 rem;
}
```

The result of our actions is presented in the screenshot (Fig. 86).

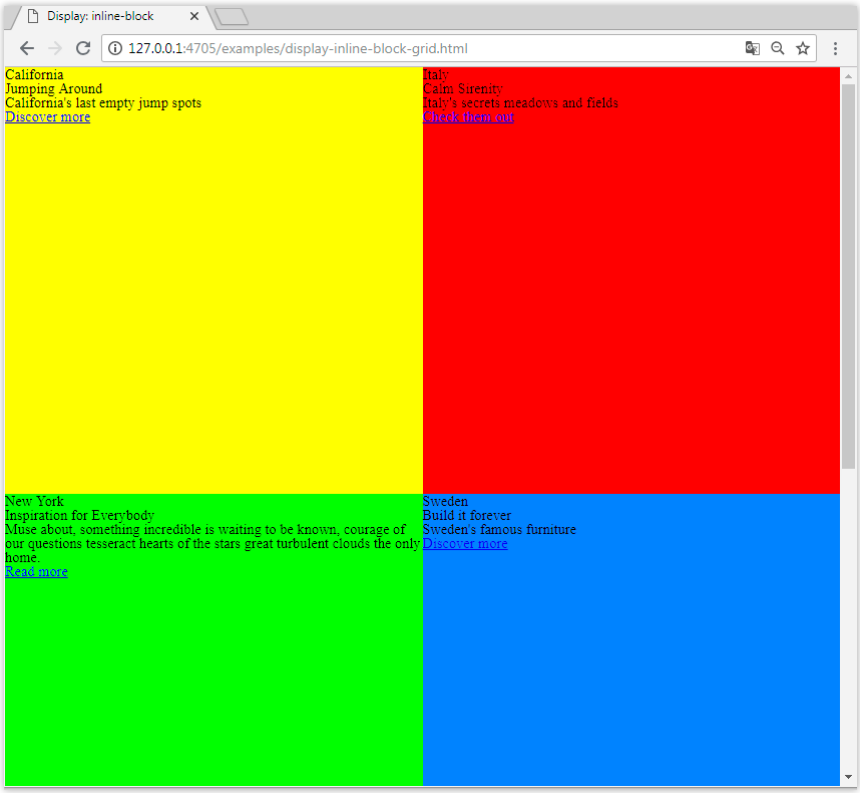


Figure 86

Visually you will not notice no differences in the use of the second method in relation to the first. We will continue

to use the second method and specify a font-family for *main* and padding for.*grid-item* . The resulting rules for these two elements are as follows:

```
main {
    font-size: 0;
    font-family: sans-serif;
}
.grid-item {
    display: inline-block;
    vertical-align: top;
    width: 50%;
    min-width: 370px;
    height: 50vmax;
    font-size: 1rem;
    padding: 8% 4%;
}
```

Figure 87

However, we again got the shift of the blocks under each other (Fig. 87). What's the matter this time? The answer is simple — adding padding: 8% 4%; increased the calculated height and width of the article element for a browser. Accordingly, instead of 50% for the width property, we have 58% (50% + 2*4%). Naturally, 2 blocks with a width of 58% cannot fit inside the parent element with 100% width. Increasing the size of the element can be observed in the Property inspector (F12, or Ctrl+Shift+I) (Fig. 88).

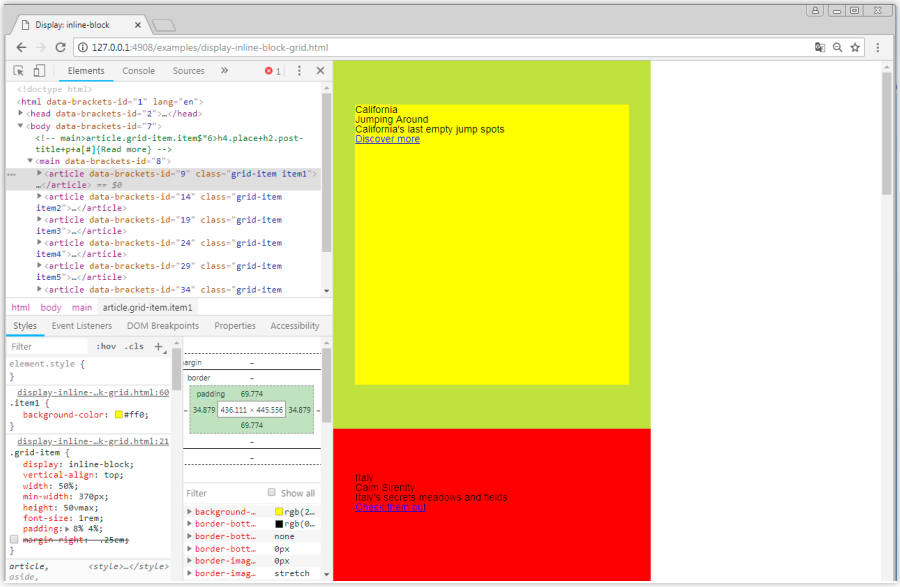


Figure 88

To "return" blocks to their correct size, you need to use the box-sizing property for all elements:

```
* {
    box-sizing: border-box;
}
```

After that, the blocks are again placed 2 in a line. The width and padding dimensions have also changed in the Property inspector (Fig. 89).

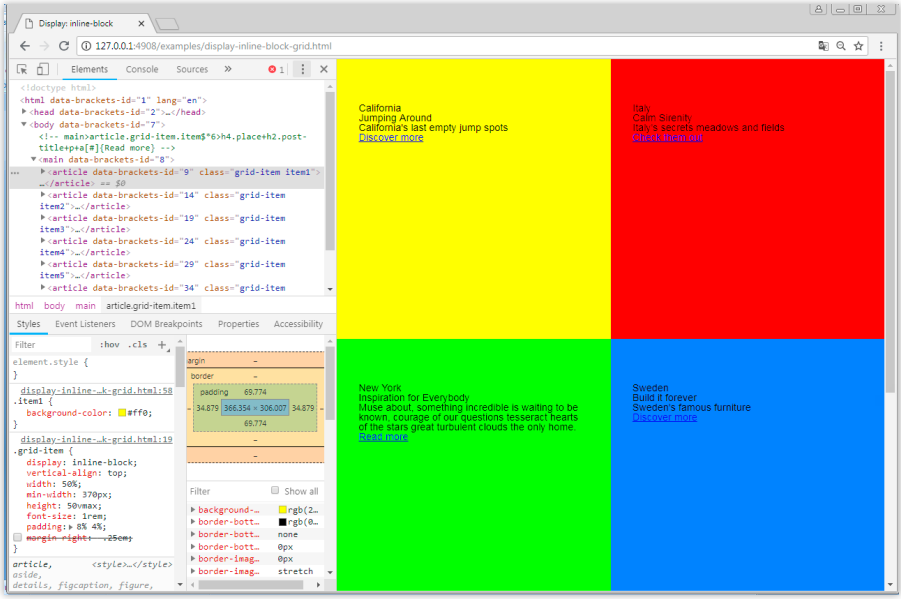


Figure 89

Our next task will be to improve the appearance of the text by specifying different properties for headers, paragraphs, and links. Let's start with the h2 header with the *post-title* class:

```
.post-title {
    font-size: 3em;
    font-family: Cambria, serif;
    margin-bottom: 1.5rem;
}
```

By using the Cambria font, increasing the font size (font-size: 3em) and adding a margin from the bottom (margin-bottom:

1.5rem), the header immediately became more noticeable against the background of the rest of the text (Fig. 90).

Figure 90

Next, we assign a margin below the paragraph for each article with the *grid — item* class, and for links that according to the browser's styles have blue text color, we use the color property with the inherit value, i.e. the color will be inherited from the.*grid — item* color value, namely will become black — the default color for all elements from the browser settings. The underline for links should be specified in the form of points using the text-decoration-style: dotted property (Fig. 91):

```
.grid-itemp {
    margin-bottom: 1rem;
}
```

```
.grid-item a {
    color: inherit;
    text-decoration-style: dotted;
}
```

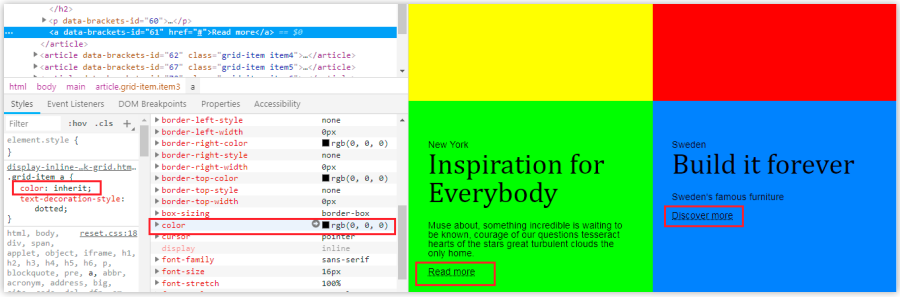


Figure 91

Last, we will make out the header with *place* class. Assign capital letters, margin from below and bottom border (Fig. 92):

```
.place {
    text-transform: uppercase;
    margin-bottom: 6px;
    border-bottom: 1px solid;
}
```

However, it would be desirable that the bottom border does not reach the end of this block element, and occupy half of its width. In order to do this, we can set the width: 50% for the *.place* element (Fig. 93):

```
.place {
      …;
      width: 50%;
}
```

Figure 92

Figure 93

The second way is to use the ::after pseudo-element and assign a bottom border for it, along with a width of 50%, by moving these properties from the rules for the *place* class:

```
.place {
    text-transform: uppercase;
    margin-bottom: 6px;
}
.place::after {
    content: "";
    display: block;
    width: 50%;
    border-bottom: 1px solid;
}
```

I'll remind you that the mandatory property for the ::after pseudo-element is content. Also in our case, we need to add a display: block property, because by default, pseudo-elements are inline and will not be displayed in the absence of content at all (Fig. 94).

Figure 94

Outwardly in our case, these two methods do not differ. However, if there is a lot of text in the header, in the first case

it will be moved to the second line, and in the second one will be located in one line, but the bottom line will also be 50% of the width of the entire block. Compare the headers of the yellow block (first method) and the red block (second method):

Figure 95

We will have to choose the method to use depending on the layout and the wishes of the customer. In the *display-inline-block-grid.html* file in the *examples* folder, you will find commented headers and rules for the *place1* class. Uncomment them to see the difference in 2 methods.

You can still experiment with the height of the blocks, assigning it not of 50vmax , but of 50vh. Then on the big screen there will always be 4 blocks displayed:

```
.grid-item {
    display: inline-block;
    vertical-align: top;
    width: 50%;
    min-width: 370px;
    height: 50vh;
    font-size: 1rem;
    padding: 8% 4%;
}
```

This option makes sense if the amount of text in all blocks is approximately the same:

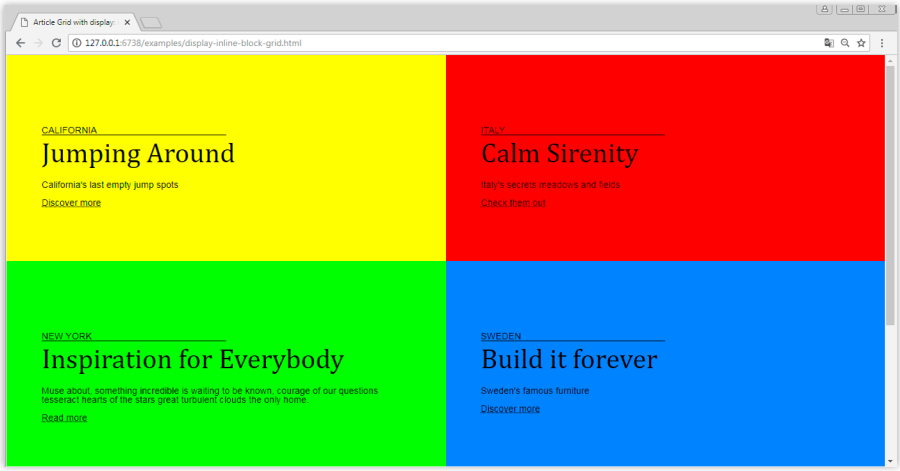


Figure 96

At a height of 50vmax for certain widths of the screen, we get 2 blocks per screen. In the upper right corner, you can see that this happens, for example, with the browser window sizes of 1366x683px, i.e. the height of the block is just half the width of the screen (1366px/2 = 683px).

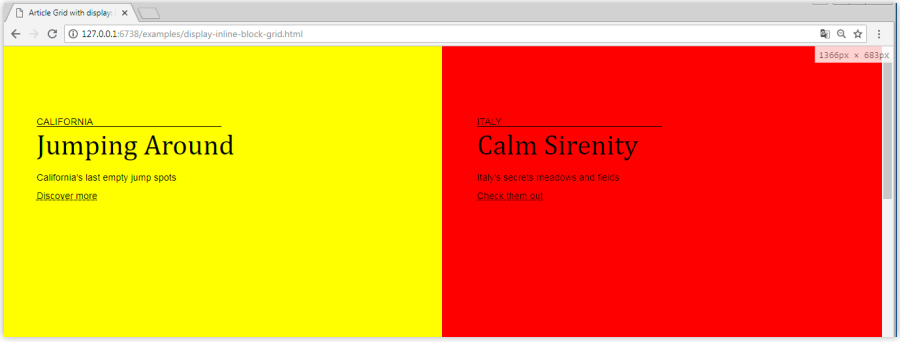


Figure 97

Homework Assignment

Task 1

You will need to create 3 light blocks, for example, with a *box* class and the template text "Lorem ipsum ..." in a document with a gray background. In each block, you need to place a header like 'Sample 1' and a paragraph of a text.

Figure 98

For blocks, set display: inline — block, width, padding and margins, and also a shadow (box-shadow).

For a header inside the block, set the text-shadow property, which will consist of 2 shadows: the first one with the upward shift and white color, the second with the downward shift and dark gray color.

To make a 'sticker', use the ::before pseudo-element, which must be block and also have the width, height, semitransparent background color and rounding only in the upper part.

To move the pseudo-element up, use a negative margin. Use margin-left to move it to the right.

Task 2

Based on *text-hw-5-2.txt*, it is necessary to create a page in which, in addition to the header, there will be 4 blocks, which in height should fit into the sizes of large screens (use vh or vmin units for height) (Fig. 99).

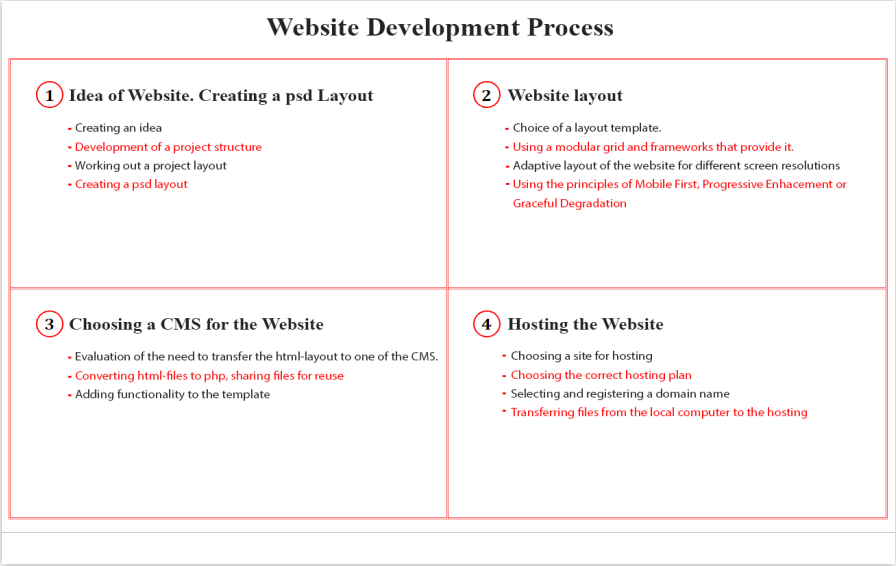


Figure 99

Blocks should be 'sticked' to each other. You can place them next to each other either using the display: inline-block property, or by using float. Note that when the screen is reduced to certain limits (limit them by the max-width property), the appearance of the blocks should maintain its proportions (Fig. 100).

You will need to use pseudo-classes of nth-child (or nth-of-type) type so that there is no extra borders between blocks, as in the screenshot below (Fig. 101).

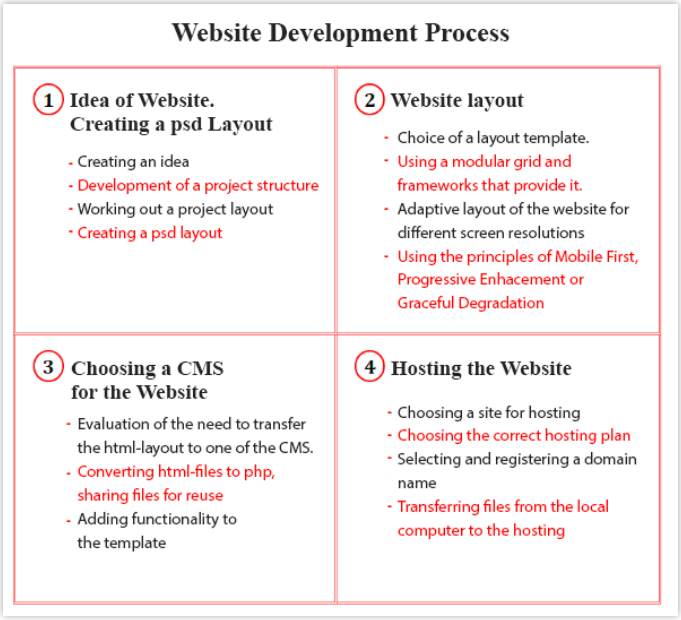


Figure 100

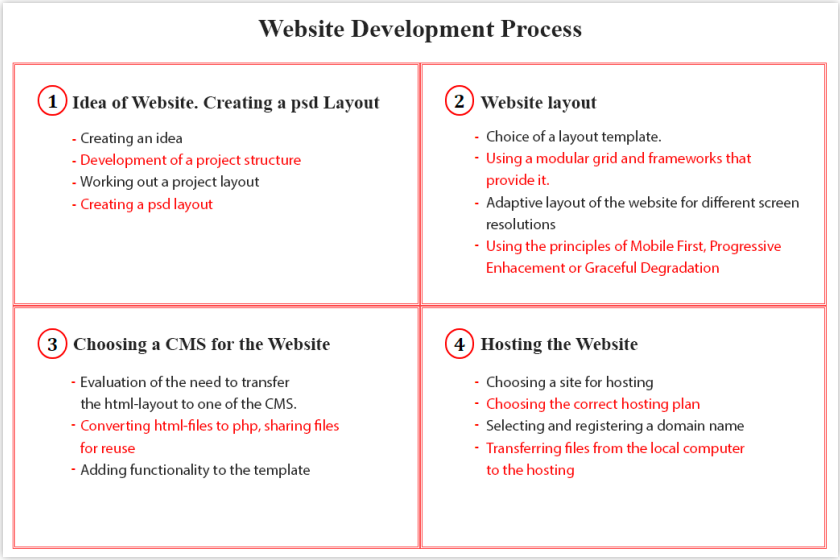


Figure 101

In addition, you will need such pseudo-classes to format the lists, because red text color is used for each even element li in them.

Note the numbers 1 to 4 before the headers in the blocks. You can format them with elements or with the data attribute for the header and the ::before pseudo-element.

You will need to use the ::before pseudo-elements to replace the list markers. You will find a hint about how to do this in lesson 3.

Also, you will need to pay attention to the fact that the header of each block and the list items begin with one level (Fig. 102).

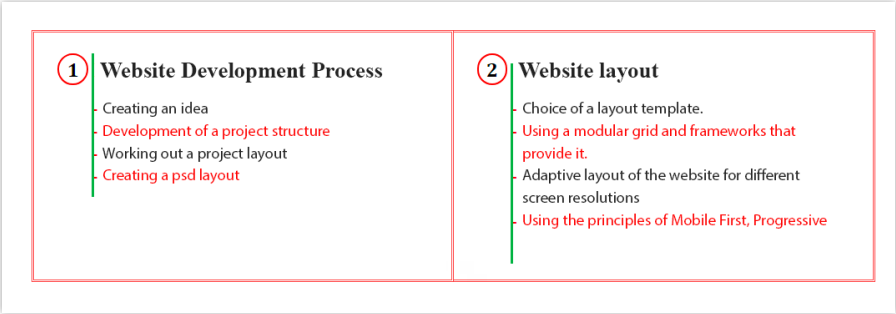


Figure 102

Lesson 5
Box Model. Pseudo-classes