HyperText Markup Language

HTML&CSS

STEP IT ACADEMY

Lesson 4/1

Animation in Web Design

Contents

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

Animation in Web Design

An **animation** is one of the constituent parts of website design. Through animation, you can draw the user's attention to the most important elements of the site. UI/UX designers think over what elements on the site will be in motion.

- UI stands for User Interface;
- UX stands for User Experience.

UI/UX designers are specialists who create user interfaces while thinking about how users will interact with the interface in order to get what they want, to achieve the goal.

Examples of animation on websites:

- smoothly changing background and button shadow on hover at https://www.microsoft.com/en-us/microsoft-365/enterprise;

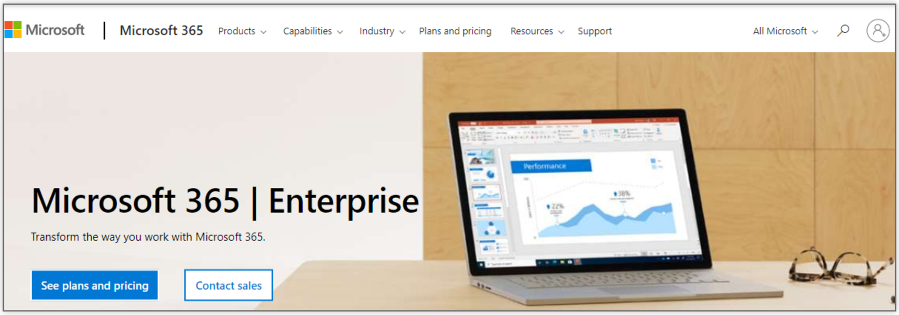


Figure 1. The button is not hovered

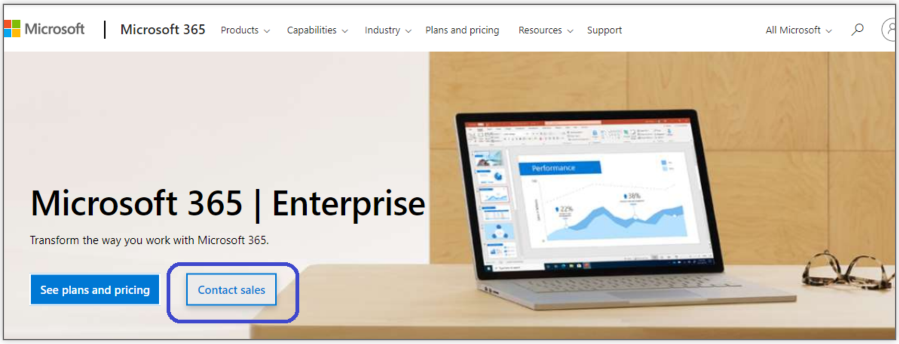


Figure 2. The Contact sales button is hovered, the button background changes, and a shadow is added

- a moving background with movie covers on the Apple website: https://www.apple.com/tv/;

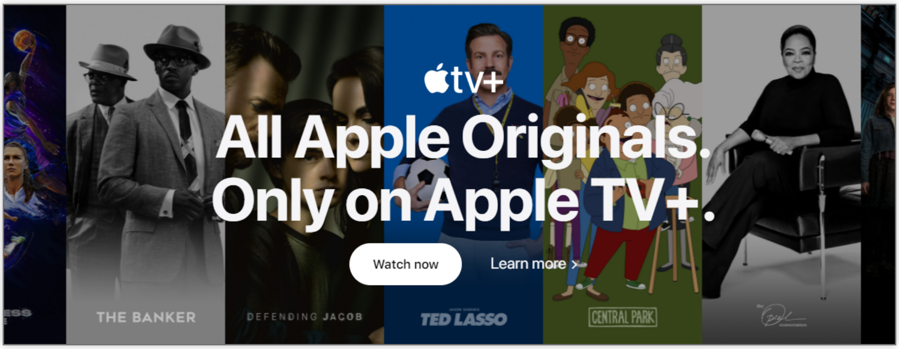


Figure 3. Example. Website: https://www.apple.com/tv/

- a smoothly changing background of buttons and text of links when hovered: https://dragone.com/en;

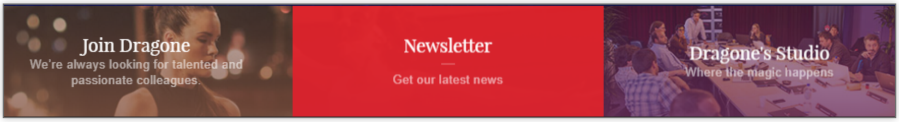


Figure 4. Example. Website: https://dragone.com/en

Quality animation evokes positive emotions in the user.

CSS3 properties allow you to create simple animations in the browser without using JavaScript. CSS animations are also effective and, in some cases, are more beneficial than JavaScript-based animations. Several years ago, the then-popular jQuery library, when paired with JavaScript, allowed you to create various types of animations on a page. In terms of performance, the jQuery library is inferior to CSS properties when creating animations. However, newer technologies such as the GSAP library based on JavaScript have sometimes even better performance. So at this point in time, the performance of CSS animations is in most cases almost identical to animations created with JavaScript. An important aspect of choosing a tool for creating an animation is the ability to control specific places in the animation. Browsers provide the ability to stop/restart a CSS animation, but there is no way to refer to any specific part of the animation, there is no way to change the direction of the element path in a certain section of the animation, etc. JavaScript gives you full control over the animation.

CSS animations are suitable for performing simple movements of an element even if compatibility with legacy browsers is not required.

JavaScript allows you to create complex animations that are interactive.

The effect of moving elements on the site can be created using CSS3 properties:

- transitions;
- animation;

- transformation. It should be noted that transformation does not create animation, but only transforms (moves, rotates, enlarges) an element. Transformations are often used in conjunction with animation and transition properties.

CSS Transitions. Transition

CSS transitions allow CSS properties to change from one value to another within a certain period of time. A simple example of this behavior is changing the background color of a link button using the :hover pseudo-class, which instantly changes the values of the background-color property in CSS.

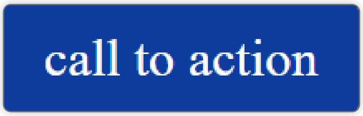


Figure 5. The button in a default state, not hovered

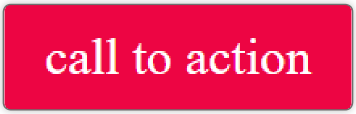


Figure 6. Background color of the button when hovered

Let's create this button. (*The example code is attached to the method.package, in a file named "ButtonCallToAction.html"*)

HTML code of the link button:

```
<a href="#">call to action</a>
```

Inactive link, default state.

CSS code:

```
a{
    text-decoration: none;
    display: inline-block;
    font-size: 2rem;
    background-color: #003fa3;
    border-radius: 5px;
    color: #ffffff;
    padding: 15px 25px;
}
```

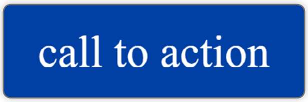


Figure 7. Inactive link, default state

The link in the :hover state, final state:

CSS code:

```
a:hover{
    background-color: #012763;
}
```

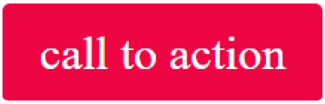


Figure 8. The link in the :hover state, final state

Let's supplement the first rule with the transition property:

CSS code:

```
a{
    text-decoration: none;
    display: inline-block;
    font-size:  2rem;
    background-color: #003fa3;
    border-radius: 5px;
    color: #ffffff;
    padding: 15px 25px;
    transition: all 0.8s ease 0s;
}
```

transition: all 0.8s ease 0s; is a shorthand property that defines a transition. The first value defines the list of properties whose transitions should be animated. The second value defines the transition time. The third value defines the temporal function. The fourth value determines the delay time before the animation starts. Please note that the all keyword affected only the background color, because only the background-color property is specified for the :hover state (on mouseover), i.e., only the background color changes.

After adding the shorthand transition property, the background color of the link button will smoothly transition from the first (default) state to the second (final, state :hover) within 0.8 seconds.

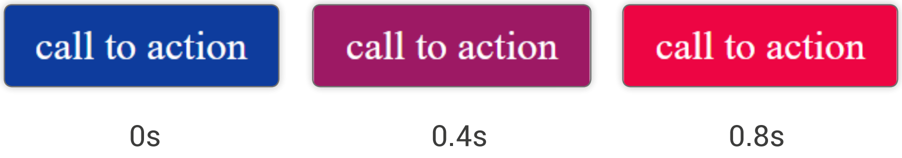


Figure 9. Changes in the background color over time

It is worth noting that the transition is applied to the original, unselected element. This is done so that we could add other styles in other states of the link and also perform a transition for them. The transition property must be applied to the element in the state from which the transition will take place.

You can declare the transition using the shorthand transition property (as mentioned in the example above) or using 4 separate properties: transition-property, transition-duration, transition-timing-function, transition-delay. In this case, the first rule will look like this:

```
a{
    text-decoration: none;
    display: inline-block;
    font-size: 2rem;
    background-color: #003fa3;
    border-radius: 5px;
    color: #ffffff;
    padding: 15px 25px;
    transition-property: background-color;
    transition-duration: 0.8s;
    transition-timing-function: ease;
    transition-delay: 0s;
}
```

where

- transition-property defines a list of modifiable properties. You can specify a single property or a list of properties to be changed, separated by commas.

Let's add the change of the text color and font size properties to the rule for the selected element:

CSS code:

```
a:hover{
    background-color: #012763;
    font-size:  2.1rem;
    color: #ffff00;
}
```

In order to smoothly change only the properties of the background color and font size, it is necessary to specify exactly these comma-separated properties in the values of transition-property:

```
transition-property: background-color, font-size;
```

The all value means that all measurable properties of the element will change. The none value means that no property will change.

The CSS properties that change the appearance of the element or its position in the HTML document are subject to change. These properties include: width, height, left, top, right, bottom, color, opacity, background-color, margin-left, border, and so on (a complete list of modifiable properties can be found at developer.mozilla.org).

It should be noted that the transition can be applied to the pair of property values for which the measured midpoint can be determined. For example, you can apply a transition from 400px to 800px to the width property, so there is a certain midpoint — 600px. But an element whose property changes from display: none to display: block cannot have a transition since it is impossible to determine a measured midpoint between these values. It is also impossible to apply the transition from width: auto to width: 1000px. An exception is the visibility property, because you can switch from visible to hidden.

- transition-duration determines the transition time indicated in s (seconds) or in ms (milliseconds). It is a required property under which the transition will be created. If several properties are specified in the transition-property property, you can specify a different duration of the transition for each property, separated by commas:

  CSS code:

```
a{
    text-decoration: none;
    display: inline-block;
    font-size:  2rem;
    background-color: #003fa3;
    border-radius: 5px;
    color: #ffffff;
    padding: 15px 25px;
    transition-property: background-color, font-size;
    transition-duration: 0.8s, 1s;
    transition-timing-function:  ease;
    transition-delay: 0s;
}
```

- transition-timing-function is a timing function that determines how the specified properties will change. There are several standard functions. It is also possible to write your own function using Bezier curves. This website https://cubic-bezier.com/ lets you compare the timing functions and see how they differ.
  - ▹ subic-bezier (x1, x2, x3, x4) is the value for its function. The values x1 and x3 are from 0 to 1.

Standard functions:

- ▹ ease or cubic-bezier (0.25, 0.1, 0.25, 1.0) — the transition starts quickly, the speed increases by the end. It is used by default;
- ▹ linear or cubic-bezier (0.0, 0.0, 1.0, 1.0) — linear speed, constant;
- ▹ ease-in or cubic-bezier (0.42, 0, 1.0, 1.0) — the transition starts slowly, the speed increases by the end;
- ▹ ease-out or cubic-bezier (0, 0, 0.58, 1.0) — the transition starts quickly, the speed decreases by the end;
- ▹ ease-in-out or cubic-bezier (0.42, 0, 0.58, 1.0) — the transition starts slowly, increases in the middle, and decreases again by the end. The function is similar to ease, but the slow down is more pronounced.

If the transition-property property has several properties specified, you can specify different timing functions for each property, separated by commas:

CSS code:

```
a{
    text-decoration: none;
    display: inline-block;
    font-size:  2rem;
    background-color: #003fa3;
    border-radius: 5px;
    color: #ffffff;
    padding: 15px 25px;
    transition-property: background-color, font-size;
    transition-duration: 0.8s, 1s;
    transition-timing-function:  ease, linear;
    transition-delay: 0s;
}
```

- transition-delay — determines the delay before the start of the transition, indicated in s or ms. If the value is not specified, then it defaults to 0, and the transition is created immediately. If the transition-property property has several properties specified, the delay for each property can be specified, separated by commas:

  CSS code:

```
a{
    text-decoration: none;
    display: inline-block;
    font-size: 2rem;
    background-color: #003fa3;
    border-radius: 5px;
    color: #ffffff;
    padding: 15px 25px;
    transition-property: background-color, font-size;
    transition-duration: 0.8s, 1s;
    transition-timing-function: ease, linear;
    transition-delay: 0s, 0.2s;
}
```

Declaring a transition for multiple CSS properties using a shorthand transition property:

CSS code:

```
a{
    text-decoration: none;
    display: inline-block;
    font-size:  2rem;
    background-color: #003fa3;
    border-radius: 5px;
    color: #ffffff;
    padding: 15px 25px;
    transition: background-color 0.8s ease 0s,
                font-size 1s linear 0.2s;
}
```

If you are not planning on changing the defaults transition-delay (0s) and transition-timing-function (ease), they can be omitted from the CSS rule. In this case, the rule will look as follows:

CSS code:

```
a{
    /* all required styles */
    transition: background-color 0.8s,  font-size 1s;
}
```

If you plan to make a transition for all changeable properties of an element in the same time interval, then it is enough to indicate only the transition time:

CSS code:

```
a{
    /* all required styles */
    transition: 1s;
}
```

Example

The curtain goes up when the block is hovered. It can be a button that takes you to a subsection of the site.

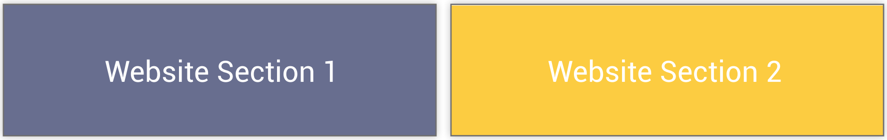


Figure 10. Elements in the default state, not hovered

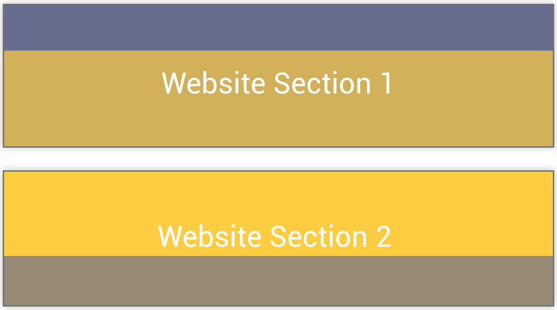


Figure 11. Elements in the on hover state.
A semi-transparent additional background ("curtain")
appears from the bottom of the element
when the latter is hovered

HTML code:

```
<div class="block bl-1">
    <a href="#">Website Section №1</a>
</div>

<div class="block bl-2">
    <a href="#">Website Section №2</a>
</div>
```

CSS code:

```
/* Add styles for the block containing the link */
.block{
    display: inline-block;
    height: 80px;
    width: 20vw;
    position: relative;
}

/* Add styles for links in blocks */
.block a{
    display: block;
    width: 100%;
    height: 100%;
    color: #fff;
    text-align: center;
    text-decoration: none;
    padding: 2em 0;
    position: absolute;
    z-index: 2;
}

/* Background color for each block */
.bl-1{
    background: #686E8F;
}

.bl-2{
    background: #FCCC41;
}

/*
   Using the :: before pseudo-element, add a curtain
   of a different color going up for each block
*/
.bl-1::before {
    content: '';
    background-color: rgba(252, 204, 65, 0.7);
    position: absolute;
    bottom: 0;
    left: 0;
   right: 0;
```

```
    height: 0%;
    transition: 1s;
}

.bl-2::before {
    content: '';
    background-color: rgba(104, 110, 143, 0.7);
    position: absolute;
    bottom: 0;
    left: 0;
    right: 0;
    height: 0%;
    transition: 1s;
}

/* On hover, the curtain's height will be 100% */
.bl-1:hover::before, .bl-2:hover::before{
    height: 100%;
}
```

The transition property is added for the default, unselected element (*the code is attached to this lesson, you can find it in a file named ButtonSecondBackground.html*).

CSS Transformation. The Transform Property

Using transformations (the transform property), the element can be scaled, rotated, skewed, shifted without affecting other elements on the page.

Transforms are often used in conjunction with transitions. Transitions allow you to smooth the transformation process of an element.

There are 2D and 3D transformations. Let's begin with 2D transformations. The transformation occurs on two axes: X (*horizontally*) and Y (*vertically*).

Elements are transformed with the transform property, which defines the transformation type of the element, and the transform-origin property, which defines the starting point of the transformation.

The transform-origin Property

Default values: 50% 50% 0. The first value indicates the x-axis, the element's horizontal movement. The second value indicates the y-axis, the element's vertical movement. These values are calculated from the top left-hand corner of the element. The third value specifies the z-axis and is used in 3D transformations. The values can be specified in the element's length units, percentages, or keywords center, left, right, top, bottom. The z-axis is optional.

If the transformation of an element is assumed to be relative to its center (transform-origin: 50% 50% — default value), you do not need to declare this property.

Depending on the origin set with the transform-origin property, rotation, for example, (transform: rotate ();) will have a different effect:

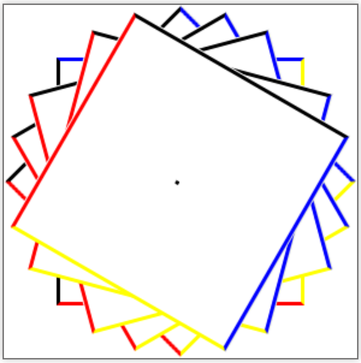


Figure 12. Default value: transform-origin: 50% 50%

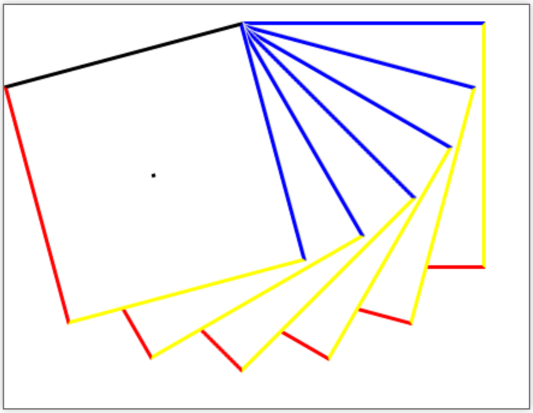


Figure 13. Transform-origin: 0 0

Functions of the transform property

- translate (x,y) — changes the location of the element to the x value from left to right and to the y value from top to bottom. If negative values are specified, the shift will be directed from right to left and from bottom to top:

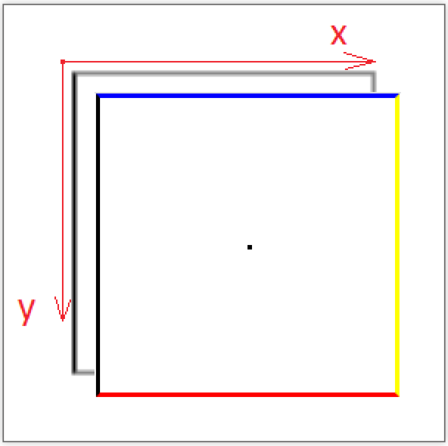


Figure 14. Repositioning of an element along the X and Y-axes

HTML code:

```
<div class="example">
    <div class="point"></div> <!-- Central point -->
</div>
```

CSS code:

```
/* Styles for a square with borders different in color */
.example{
    width: 200px;
    height: 200px;
    border: 3px solid black;
    border-top-color: blue;
    border-right-color: yellow;
    border-bottom-color: red;
    transform: translate(10px, 10px);
}

/* Styles for point in the center */
.point{
    width: 3px;
    height: 3px;
    background-color: black;
    position: relative;
    top: 50%;
    left: 50%;

}
```

The transform: translate (10px, 10px); property moves the element along the x- and y-axes by 10px in relation to its original position. Legal values are element's size units or percentages. (*Sample code is attached to the method.package, a file named "translate.html"*)

- translateX(x) — is similar to translate (x,y), only it specifies one value to shift the element along the x-axis, left or right.
- translateY(y) — is similar to translate (x,y), only it specifies one value to shift the element along the y-axis, up or down.
- scale (x,y) — scales the element. The values from 0 to 1 reduce the size of the element. The values that are greater than 1 increase the size of the element. Negative values flip the element. It should be noted that as the element is enlarged, the quality of the displayed element may deteriorate. It is valid for block-level elements (display: block; display: inline-block;). Why only for these elements? Because scaling is the simultaneous increase of the element in width and height. You can change the width/height of a block or inline element. The width and height of inline elements are calculated according to the content of the inline element.

The transform: scale (1.2,1.2); property will enlarge the button on hover by 1.2 times in height and width

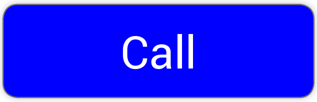


Figure 15. Default state

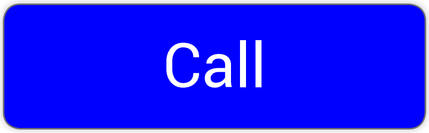


Figure 16. On hover

HTML code:

```
<a href="#">Call</a>
```

CSS code:

```
/* Styles for the link button */
a{
    display: inline-block;
    border-radius: 5px;
    padding: 15px 30px;
    background-color: blue;
    color: #fff;
    font-weight: bold;
    text-transform: uppercase;
    text-decoration: none;
}

a:hover{
    transform: scale(1.2,1.2);
}
```

- scaleX(x) — scales the element horizontally. The values from 0 to 1 reduce the size of the element. The values that are greater than 1 increase the size of the element. Negative values flip the element. It should be noted that as the element is enlarged, the quality may deteriorate.
- scaleY(y) — scales the element vertically. The values from 0 to 1 reduce the size of the element. The values that are greater than 1 increase the size of the element. Negative values flip the element. It should be noted that as the element is enlarged, the quality may deteriorate.
- rotate() — rotates the element around the origin (the transform-origin property is responsible for the location of

the origin). Units: degrees (deg). If the value is positive, the element is rotated clockwise. If negative, the element is rotated counterclockwise. When specifying a rotation angle greater than 360 deg, the element will rotate according to the specified value, for example, if transform: rotate (3600deg), the element will make 10 revolutions around the origin.

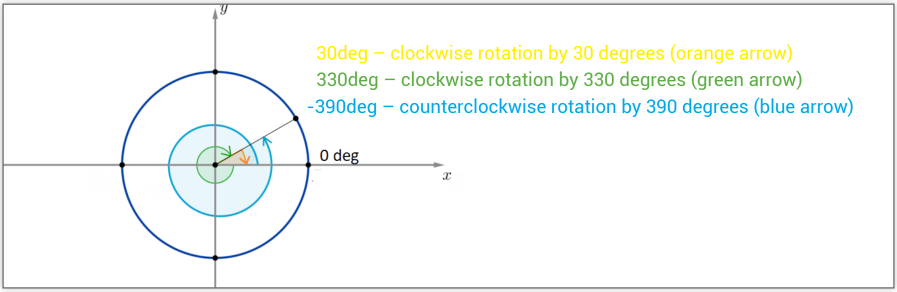

y
30deg – clockwise rotation by 30 degrees (orange arrow)
330deg – clockwise rotation by 330 degrees (green arrow)
-390deg – counterclockwise rotation by 390 degrees (blue arrow)
0 deg
x

Figure 17. Counterclockwise/clockwise rotations

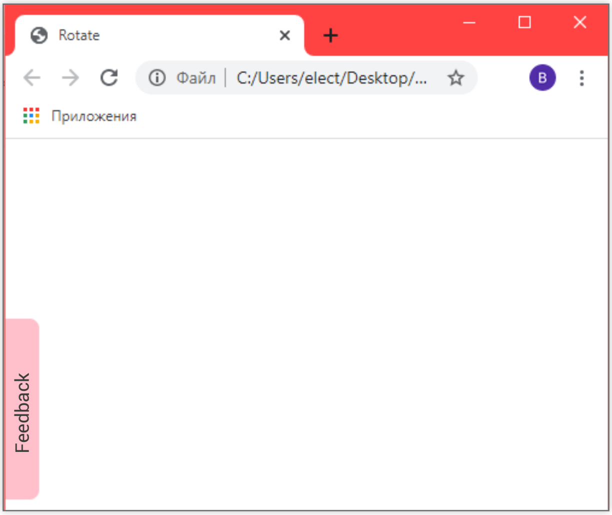
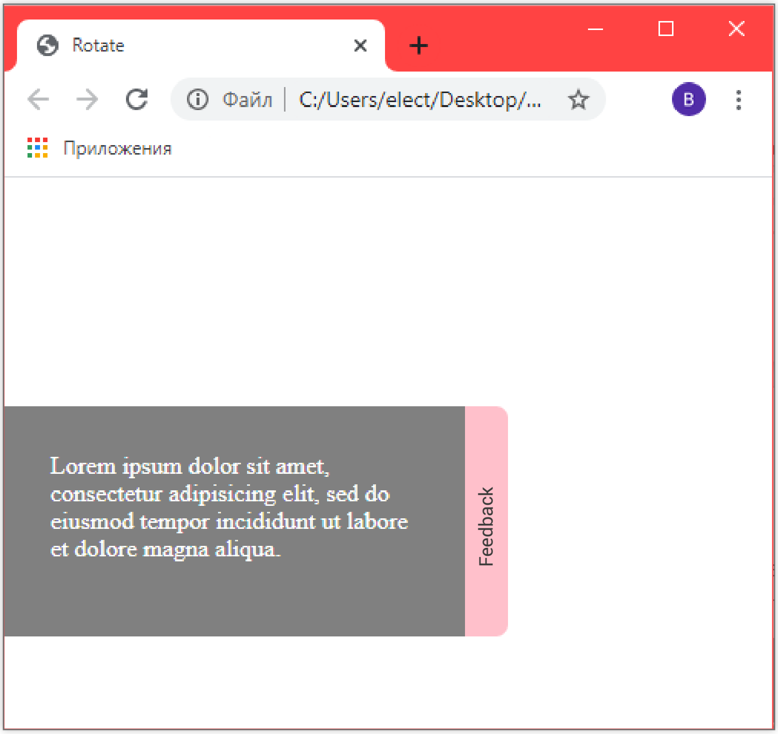


Figure 18. When the Feedback element is hovered, a window appears

Let’s create a “Feedback” element on the page. Its default state is at the left border of the browser window, and only the pink part with the “Feedback” written on it is visible (Fig. 18).

When the element is hovered, the window with instructions smoothly slides out (the example uses Lorem ipsum) (Fig. 19).



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 19. The Feedback element that appeared after hovering the Feedback caption

At this stage, we will create a Feedback element. We can make the element smoothly slide out within the Transformations and Transitions section (Fig. 20).

The Feedback element that appeared after hovering the Feedback caption.

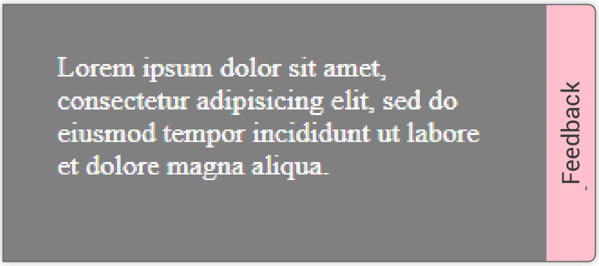

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 20. The Feedback element

The Feedback element (the ::after pseudo-element) will be rotated with the property transform: rotate(-90deg);. (*The complete example code is attached to the method.package, in a file named rotate.html*).

Let's add HTML for the Feedback element:

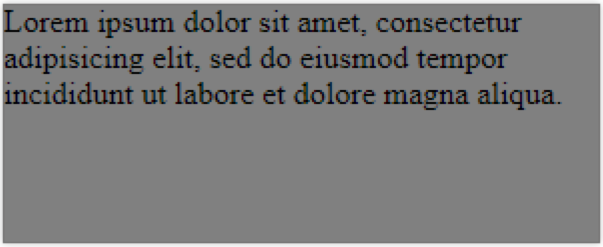
HTML code:

```
<div id="callback">
    <p>Lorem ipsum dolor sit amet, consectetur
       adipisicing elit, sed do eiusmod tempor
       incididunt ut labore et dolore magna aliqua.
    </p>
</div>
```

Styles for the Feedback element:

CSS code:

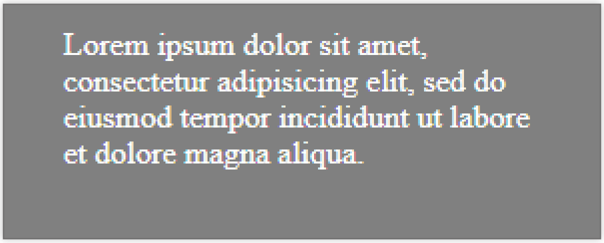
```
/* Styles for the text block */
#callback{
    width: 300px;
    height: 150px;
    background-color: grey;
}
```



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 21

```
/* styles for the paragraph */
#callback p{
    color: #fff;
    padding: 30px;
}
```

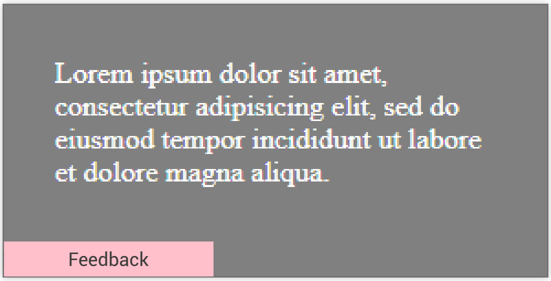


Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 22

Let's use the ::after pseudo-element to make a window slide out when the flipped Feedback caption is hovered. The content property is required for pseudo-elements. The ::after element is inline, so let's add the 'Feedback' caption to the property value:

```
/* Styles for the pseudo-element */
#callback:after{
    content: 'Feedback';
    background-color: pink;
}
```



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 23

Le'ts put the ::after pseudo-element at the end of the text block by completing styles:

```
/* Styles for the pseudo-element */
#callback:after{
    content: 'Feedback';
    background-color: pink;
    text-align: center;
    border-radius: 0 0 8px 8px;
    padding: 5px 0;
    width: 150px; /* the width of the pseudo-element
                     is equal to the block's height,
                     so there will be a rotation */
    position: absolute; /* absolute positioning */
      left: 300px; /* position to the left is equal
                      to the block's width #callback */
    top: 150px; /* position at the top is equal to
                   the block's height */
}
```

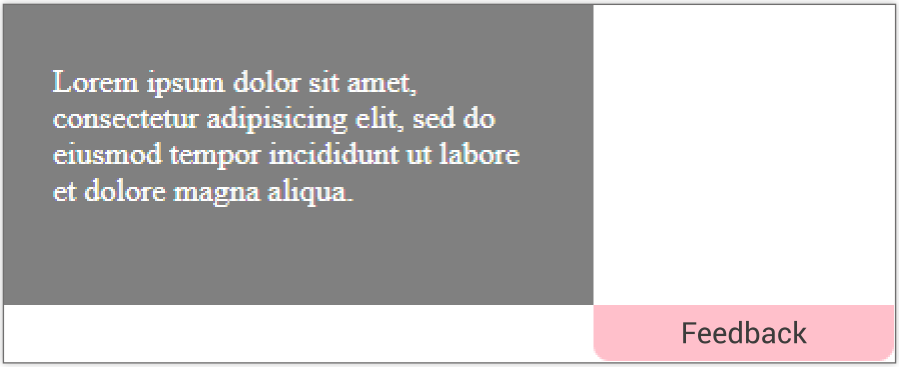
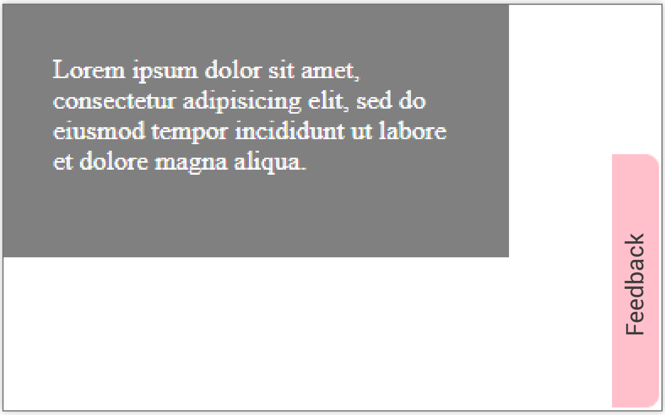

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 24

The element must rotate counterclockwise by 90 degrees, which means that you must specify a negative value for the function rotate (-90deg);

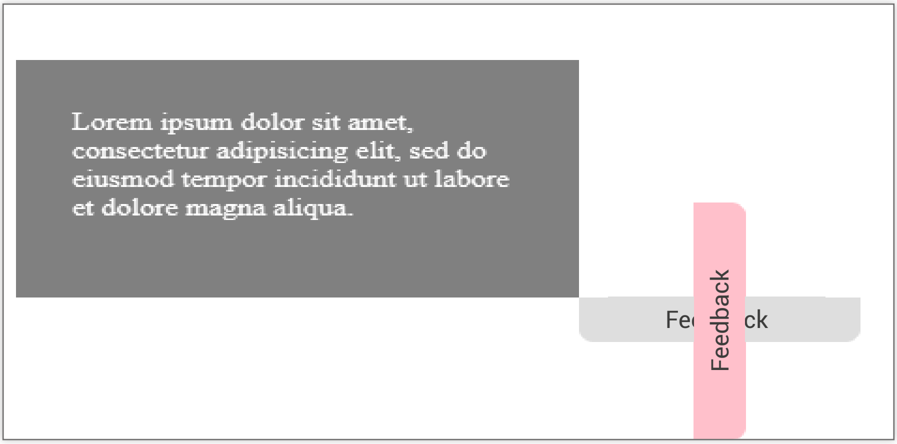
```
/* Styles for the pseudo-element */

#callback:after{
    content: 'Feedback';
    background-color: pink;
    text-align: center;
    border-radius: 0 0 8px 8px;
    padding: 5px 0;
    width: 150px; /* the width of the pseudo-element
                     is equal to the block's height,
                     so there will be a rotation */
    position: absolute; /* absolute positioning */
    left: 300px; /* position to the left is equal
                    to the block's width #callback */
    top: 150px;  /* position at the top is equal to
                    the block's height */
    transform: rotate(-90deg);

}
```



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 25

However, the element is in the wrong place, because now the origin is defined at the center of the element being rotated, and its coordinates are equal by default transform-origin: 50% 50%;. The rotation occurs relative to this point:



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 26

You need to change the position of the origin using the transform-origin property and set the values that define the location of the origin in the upper left corner of the Feedback element:

```
/* Styles for the pseudo-element */
#callback:after{
    content: 'Feedback';
    background-color: pink;
    text-align: center;
    border-radius: 0 0 8px 8px;
    padding: 5px 0;
    width: 150px; /* the width of the pseudo-element
                     is equal to the block's height,
                     so there will be a rotation */
    position: absolute; /* absolute positioning */
    left: 300px; /* position to the left is equal
                    to the block's width #callback */
    top: 150px;  /* position at the top is equal
                    to the block's height */
    transform-origin: 0 0;
    transform: rotate(-90deg);
}
```

This time the Feedback pseudo-element has rotated relative to the upper left corner, 90 degrees counterclockwise:

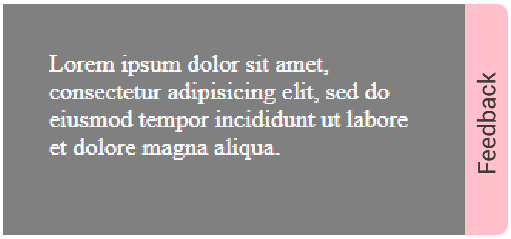

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Feedback

Figure 27

- skew (x,y) — defines the skew of the element along the X- and Y-axes. If you specify only one value, then the element will be skewed along the X-axis. Values are specified in angles, degrees, radians, and revolutions. Positive values tilt the element clockwise, negative — counterclockwise.

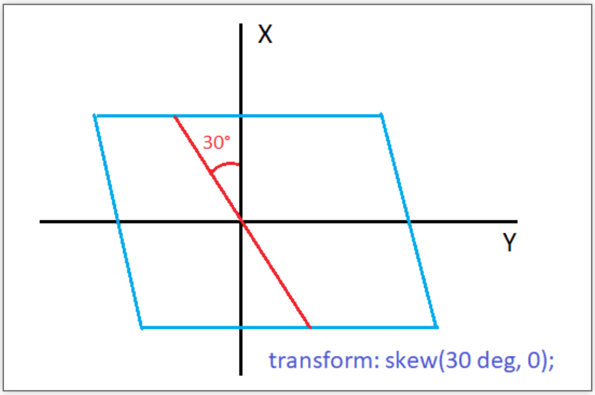


Figure 28. The skew property, horizontal skew by 30 degrees

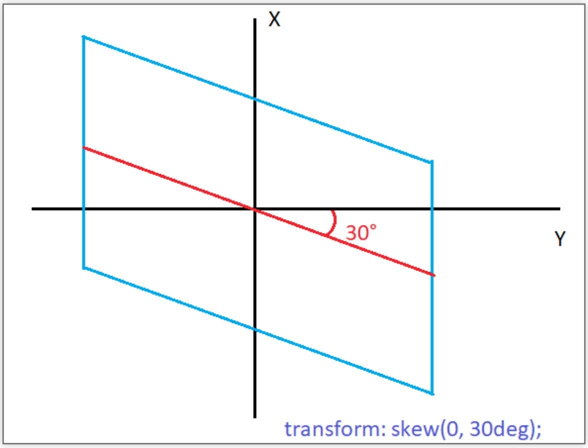


Figure 29. The skew property, vertical skew by 30 degrees

- skewX(x) — defines the skew of the element along the X- and Y-axes. If you specify only one value, then the element will be skewed along the X-axis. Values are specified in angles, degrees, radians, and revolutions. Positive values make the element tilt to the left, negative — to the right.

- skewY(y) — defines the skew of the element along the axes X and Y. If you specify only one value, then the element will be skewed along the X-axis. Values are specified in angles, degrees, radians, and revolutions. Positive values make the element skew up, negative — down.
- matrix — combines a number of transformations (scale, rotate, skew, translate) into one declaration. The syntax is matrix (a, c, b, d, tx, ty), where
  - ▹ a — horizontal scaling scaleX;
  - ▹ c — vertical skew skewY;
  - ▹ b — horizontal skew skewX;
  - ▹ d — vertical scaling scaleY;
  - ▹ tx — horizontal shift translateX;
  - ▹ ty — vertical shift translateY;

Transformation + Transitions

Transformation properties are often used in conjunction with CSS transitions. Let's consider creating a button that smoothly increases in size. (*The example code is attached to this lesson. It is in a file named scale.html*).

Figure 30. The button in a default state

Figure 31. The button is hovered and increased in size

We will increase the size of the button using the trans-form: scale(1.2,1.2); property (scaling). The smoothness of the transition will be created with the transition property.

HTML code:

```
<a href="#">Call</a>
```

CSS code:

```
/* Styles for the link button */
a{
    display: inline-block; /* inline-block allows us
             to apply scaling to the element */
    border-radius: 5px;
    padding: 15px 30px;
    background-color: blue;
    color: #fff;
    font-weight: bold;
    text-transform: uppercase;
    text-decoration: none;
}

a:hover{
    transform: scale(1.2,1.2); /* scaling the element
               in width and height */
    transition: 0,5s; /* transition from the default
               state to rescaled when the element
               is hovered within 0.5 seconds */
}
```

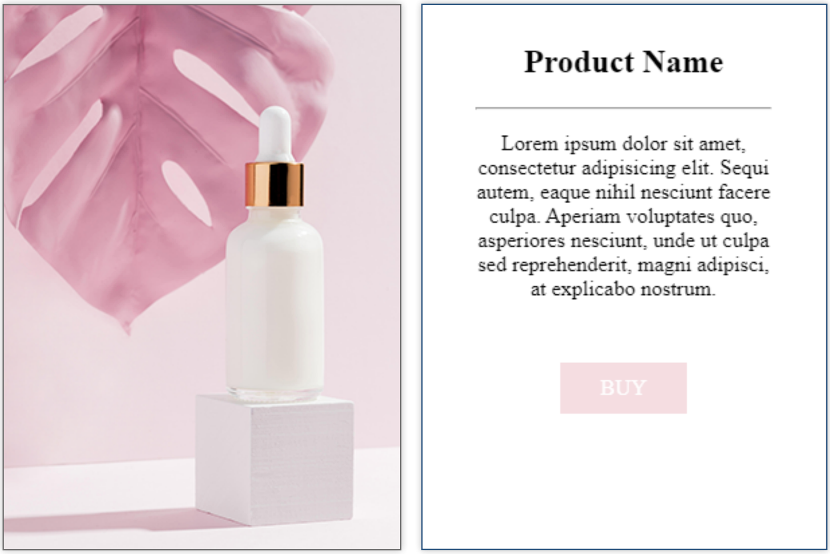
If you are going to make a transition with the standard values such as transition-property, transition-timing-function, and transition-delay, then all you have to do is specify the transition time — 0.5s.

The transition property allows the button to slowly increase in size when being hovered (:hover state). If the cursor is removed, the button will instantly return to its default state, without a smooth transition. Therefore it is necessary to add the transition property for the default state of the button, too.

```
a{
    display: inline-block; /* inline-block allows us
                              to apply scaling to
                              the element */
    border-radius: 5px;
    padding: 15px 30px;
    background-color: blue;
    color: #fff;
    font-weight: bold;
    text-transform: uppercase;
    text-decoration: none;
    transition: 0.5s;
}
```

Let's consider another interesting example of combining transformations and transitions — the "rotation" of a product card when hovered (Fig. 32). (*The complete example code is in the attachment, in a file named ProductCard.html*).

Let's create a div with the card class, inside of which there will be blocks with the classes "product" (product image block) and "description" (product description block with the "buy" button).



Product Name

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Sequi autem, eaque nihil nesciunt facere culpa. Aperiam voluptates quo, asperiores nesciunt, unde ut culpa sed reprehenderit, magni adipisci, at explicabo nostrum.

Figure 32. The product image in its default state and the reverse side of the card, when you hover the cursor over the image

HTML code:

```
<div class="card">
    <div class="product">
        <img src="product.png">
    </div>
    <div class="description">
        <h2>Product Name</h2>
        <hr>
        <p> Lorem ipsum dolor sit amet, consectetur
            adipisicing elit. Sequi autem, eaque
            nihil nesciunt facere culpa. Aperiam
            voluptates quo, asperiores nesciunt,
            unde ut culpa sed reprehenderit, magni
            adipisci, at explicabo nostrum. </p>
        <a href="#">buy</a>
    </div>
</div>
```

In the default state, the product card will be turned face up, where the face is the image. When hovering the product image, the card will rotate along the X-axis by 180 degrees, and a description of the product with the “buy” button will appear in front of the user.

CSS code:

```
/* Styles for the parent block */
.card {
    width: 300px;
    height: 450px;
    margin: 0 auto;
    position: relative; /* positioning of the parent
              element is relative so that we could
              put the child elements .product and
             .description inside the card */
    border: 1px solid #f5dde1;
}
```

Place both sides of the card so that they overlap and fill the parent element by 100%. Center-align the content and add a background.

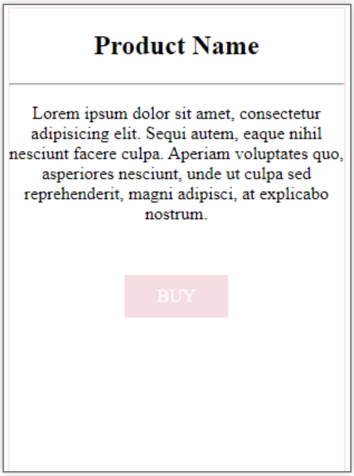
```
.product, .description {
    width: 100%; /* 100% of the parent width */
    height: 100%; /* 100% of the parent height */
    position: absolute; /* absolute positioning
                  of the elements inside the parent
                  with the card class */
              left: 0;  /* left positioning 0px */
              top: 0;   /* top positioning 0px */
    text-align:center;
    background-color: #fff;
}
```

```
/* Styles for the button of the description block
   of the product card */
.description a{
    display:inline-block;
    padding: 10px 30px;
    margin-top:30px;
    text-decoration:none;
    text-transform:uppercase;
    color:#fff;
    background-color: #f5dde1;
}
```

Let's hide the back of the card using the property backface-visibility: hidden;

```
.product, .description {backface-visibility: hidden;}
```

Since the description block is in the HTML markup after the image block, it appears on top of the image block:



Product Name

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Sequi autem, eaque nihil nesciunt facere culpa. Aperiam voluptates quo, asperiores nesciunt, unde ut culpa sed reprehenderit, magni adipisci, at explicabo nostrum.

Figure 33

Let's rotate the side with the description 180 degrees horizontally, and it will be behind the front of the image card. While in the default state, the card displays its front — image. Let's also add paddings and the box-sizing:border-box; property so that the size of the description block is calculated with paddings and border sizes:

```
.description {
   box-sizing:border-box;
   padding:40px;
   transform: rotateY(180deg); /* the description
       block rotates by 180 deg, hide it in the default
       state when the card is not hovered */
}
```

Let's set the transition property for smooth transitions when the element is hovered.

```
.product, .description {
    transition: 1s;
}
```

In order to see the description of the product, you need to add transformation properties for the blocks with the image and description:

```
.card:hover .product {
    transform: rotateY(180deg); /* rotate the image
                                   block by 180 deg */
}
.card:hover .description {
    transform: rotateY(360deg); /* return the
                                   description block */
}
```

CSS Animation. The Animation Property

In order to create an animation, you need to storyboard it first. Storyboarding is used for cartoons; you can storyboard a man running or a bird flying.

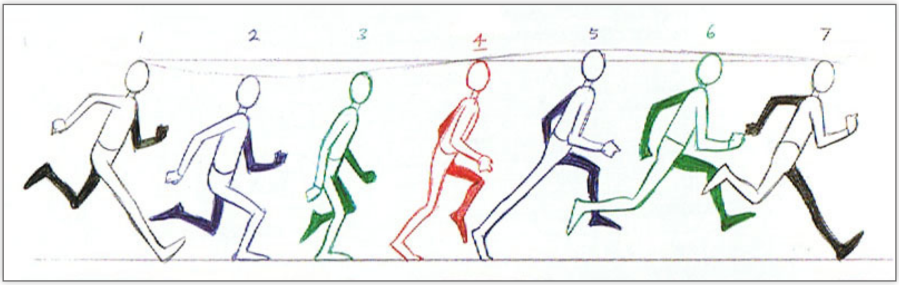


Figure 34. A man running, frame by frame

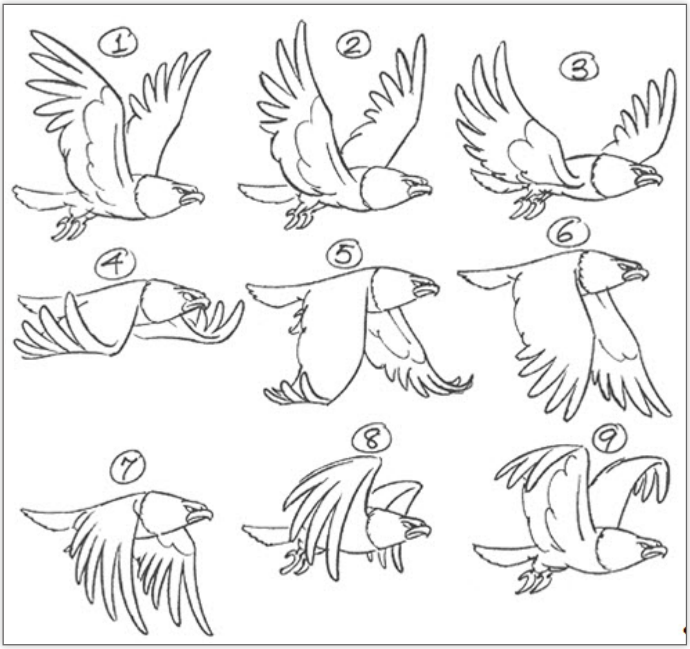


Figure 35. A drawn eagle flying, frame by frame

The same happens when animating HTML elements on a page. The simplest kind of animation is two frames, a start and an end. The browser will understand that it is necessary to animate the element between these two frames, depending on the specified conditions.

Frames are created using the special @keyframes rule. The frames are declared with the keywords from — the starting, first frame and to — the end frame, the second and the last frame. If the number of frames is more than two, then the frames are referred to as segments expressed as a percentage of the distance covered. Let's say we have three frames: 0% is the first frame, 50% is the second frame, and 100% is the third and the last frame.

Syntax:

```
@keyframes name{
    from{
       animation condition
    }
    to{
       animation condition
    }
}
```

where name is the animation name, which is custom and based on the meaning of the animation.

As a first and simple example, let's create an animation for a title that will move from left to right an infinite number of times. (*The complete example code is in the attachment, in the file named h1Animation.html*)

HTML code:

```
<h1>Some News</h1>
/* Styles for the title */
h1{
    background-color:#d4c4ff;
    width: 30vw;
    color: #fff;
}
```

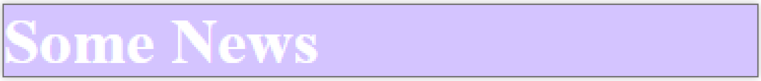


Figure 36. The title with the background

The easiest way to move the title is to set margin to left. This means that in the start frame, the margin-left will be 0 pixels, and in the end frame 500 pixels.

Let's create animation frames:

```
@keyframes myAnim{
    from{
        margin-left: 0px;
    }
    to{
        margin-left: 500px;
    }
}
```

Next, you need to call the created animation for the element using the animation property:

```
h1{
    animation: myAnim 5s linear alternate infinite;
}
```

where myAnim is an animation call created with the @keyframes rule, 5s is the animation speed from the first to the last frame, linear is a linear function, alternate determines the animation playback “forward-backward,” infinite is the number of times the animation is played, the infinite keyword means an infinite number of times.

The animation property in CSS is a shorthand notation of animation-name, animation-duration, animation-timing-function, animation-delay, animation-iteration-count, animation-direction, animation-fill-mode, animation-play-state.

- animation-name — you can set a custom name;
- animation-duration — animation duration per cycle;
- animation-timing-function — determines how the animation occurs during one cycle. This property is similar to transition in its standard functions. It is also possible to write your own function using Bezier curves. This website https://cubic-bezier.com/ lets you to compare the timing functions and see how they differ.
  - ▹ subic-bezier (x1, x2, x3, x4) is the value for its function. The values x1 and x3 are from 0 to 1.

  Standard functions:
  - ▹ ease or cubic-bezier (0.25, 0.1, 0.25, 1.0) — the transition starts quickly, the speed increases by the end. It is used by default;
  - ▹ linear or cubic-bezier (0.0, 0.0, 1.0, 1.0) — linear speed, constant;
  - ▹ ease-in or cubic-bezier (0.42, 0, 1.0, 1.0) — the transition starts slowly, the speed increases by the end;

  - ▹ ease-out or cubic-bezier (0, 0, 0.58, 1.0) — the transition starts quickly, the speed decreases by the end;
  - ▹ ease-in-out or cubic-bezier (0.42, 0, 0.58, 1.0) — the transition starts slowly, increases in the middle, and decreases again by the end. The function is similar to ease, but the slow down is more pronounced;
- animation-delay — determines the delay time before the animation starts;
- animation-iteration-count — determines how many times the animation cycle will be played before it stops. The infinite value sets an infinite number of cycles.
- animation-direction — determines the direction of the animation. There are several standard values:
  - ▹ normal — the default value, the animation starts at the first frame, loops through to the final frame, and immediately returns to the first frame;
  - ▹ reverse — the animation is played backward. It starts at the final frame, plays to the first one, and resets at the first frame, instantly returning to the end frame;
  - ▹ alternate — the animation changes direction in every cycle. It goes forward from the start frame to the end in one cycle; in the next loop, it returns from the end frame to the start, the number of times depends on the number of iterations of animation-iteration-count;
- animation-fill-mode — determines whether to apply animation styles to an element before and after the animation ends. It has the following standard values:
  - ▹ none — animation styles will not be applied before and after the animation ends;

  - ▹ forwards — after the end of the animation, the element will preserve the end keyframe's animation styles that depend on animation-direction and animation-iteration-count;
  - ▹ backwards — the element will preserve the animation styles of the start frame throughout the entire delay (animation-delay) before the start of the animation, which is determined by the animation-direction; value;
  - ▹ both — specifies the simultaneous application of the forwards and backwards values.
- animation-play-state — defines the states of pause and play for animation. It is used in scripts, the animation can be stopped and resumed using scripts, and it will start playing from the moment it was stopped, not from the start. It has two values:
  - ▹ running — the animation is played;
  - ▹ paused — the animation is paused.

In a shorthand notation of all seven properties, you can omit values that you do not plan to use, and they will be set to their default values. Our example uses the following properties: animation-name to call the created animation for the element, animation-duration — playback time of the animation cycle, animation-timing-function set to linear (linear function, linear speed, no jumps), animation-direction — playback direction set to alternate to make the title move forwards and backwards, and animation-iteration-count set to infinite to make the animation playback infinite.

A good example of applying the animation-iteration-count property set to infinite is an infinitely moving background with the movie covers at https://www.apple.com/tv/.

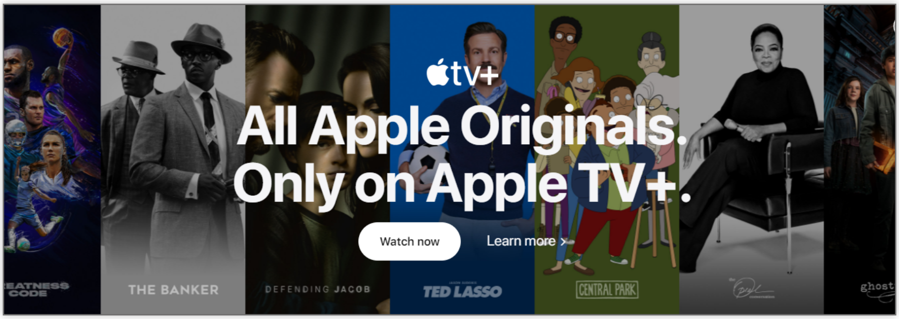


Figure 37

To create an effect of a moving background, you should describe the background-color and background-image properties in the @keyframes of the animation.

Let's create an effect of moving clouds on a blue background. (*The complete example code is in the attachment, in the file named RuningBackground.html*)

There will be nothing in the HTML code since the background image is added with CSS properties.

```
/* Styles for the page background */
body{
    background-color: #b6cbe8; /* blue background */
    background-image:url('cloud.png'); /* a png image
                            of a cloud, a transparent
                            background */
}
```

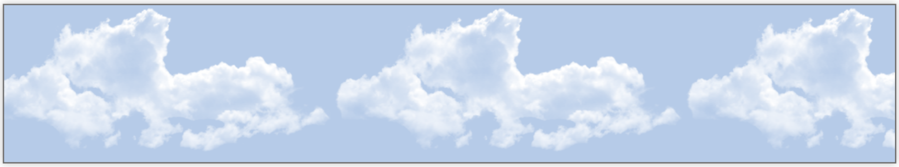


Figure 38

You can "move" the background image horizontally using the background-position-x property by the value equal to the image width. Let's set background-position-x: 0px for the first frame and background-position-x:500px for the last frame.

```
@keyframes cloud {
    from{
        background-position-x: 0px;
    }
    to{
        background-position-x: 500px;
    }
}
```

Animation call:

```
body{
    background-color: #b6cbe8;
    background-image: url('cloud.png');
    animation: cloud 10s linear infinite;
}
```

The clouds will move from left to right all the time (the infinite value).

Example of a Preloader for a Website (loader)

Preloaders are designed to inform the site visitor that the page is loading. Let's create a preloader in the form of four squares equidistant from each other. They will become white over time. This will create the animation — the effect of a square moving clockwise:

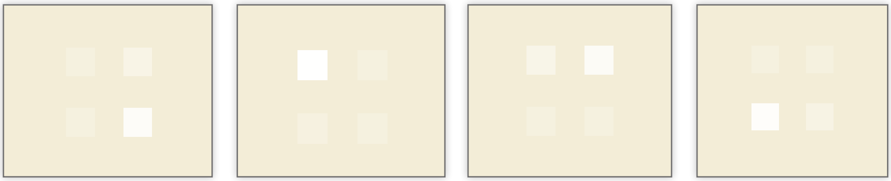


Figure 39. Preloader, 4 frames.

The example code is in the attachment, in a file named loader.html.

Let's make the moving square effect by adding a shadow. There will be four shadows. They will be added for an invisible square with a size of 10*10 pixels. Let's add shadows using the box-shadow property. This property allows you to add multiple shadows and position them relative to the desired element along the x- and y-axes. We will describe the box-shadow property when creating animation for five keyframes: 0%, 25%, 50%, 75%, 100%. On the first and last frames (0% and 100%), the box-shadow property will be the same to complete the animation cycle.

HTML code:

```
<div class="loader-container">
    <div class="loader"></div>
</div>
```

CSS code:

```
body{
    background-color: #f3edd7;
    margin: 0;
}
/* styles for the parent container of the preloader */
```

```
    .loader-container{
    width: 100vw;
    height: 100vh;
}

/* Styles for the preloader, it will not be visible
   on the page */
.loader{
    width: 10px;
    height: 10px;
    position: relative;
    top: 50%;
    margin: auto;
}
```

When declaring animation, add four shadows for each frame in keyframes. A completely white shadow will be at the box-shadow position that matches the keyframe order. Set the X and Y values for the box-shadow so that four shadows form a square, each shadow in the corner.

```
@keyframes loader {
  0%, 100% {
    box-shadow:-10px 15px 0 #ffffff,
                10px 15px 0 rgba(255, 255, 255, 0.2),
                10px 36px 0 rgba(255, 255, 255, 0.2),
               -10px 36px 0 rgba(255, 255, 255, 0.2);
  }

  25% {
    box-shadow:-10px 15px 0 rgba(255, 255, 255, 0.2),
                10px 15px 0 #ffffff,
                10px 36px 0 rgba(255, 255, 255, 0.2),
               -10px 36px 0 rgba(255, 255, 255, 0.2);
  }
```

```
   50% {
     box-shadow:-10px 15px 0 rgba(255, 255, 255, 0.2),
                 10px 15px 0 rgba(255, 255, 255, 0.2),
                 10px 36px 0 #ffffff,
                -10px 36px 0 rgba(255, 255, 255, 0.2);
   }

   75% {
     box-shadow:-10px 15px 0 rgba(255, 255, 255, 0.2),
                 10px 15px 0 rgba(255, 255, 255, 0.2),
                 10px 36px 0 rgba(255, 255, 255, 0.2),
                -10px 36px 0 #ffffff;
   }
}
```

Call the created animation for the loader element:

```
.loader{
    animation: loader 1s ease infinite;
}
```

Homework

Task 1. Implement an HTML page with a clock

Create an analog clock with moving hands — hour, minute, and second. The animation starts from the position of the hands at 12 o'clock. The second hand should complete a full circle in 60 seconds. The minute hand must move one minute after the second hand has completed a full circle, the hour hand must complete the circle in one hour, accordingly.

An example of the result:

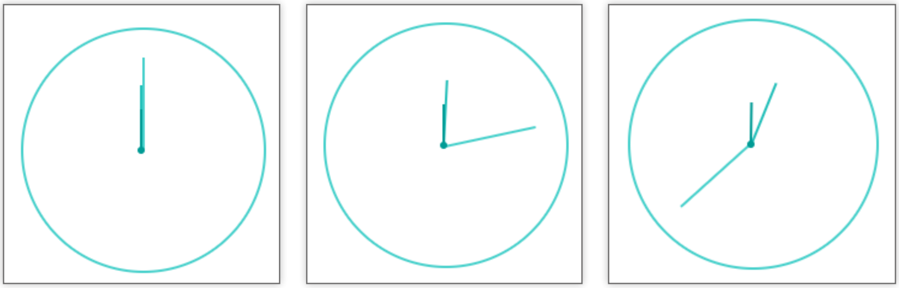


Figure 40

Task 2. Implement an HTML page with side flaps

The page initially displays two identical blocks that hide a piece of text. When any of the blocks is hovered, they slowly (in 2 seconds) open, revealing the text. If you remove the cursor, the side flaps will return to their default state.

Example of the default state (Fig. 41).

Example of the state when the block is hovered, but the animation has ended (Fig. 41).

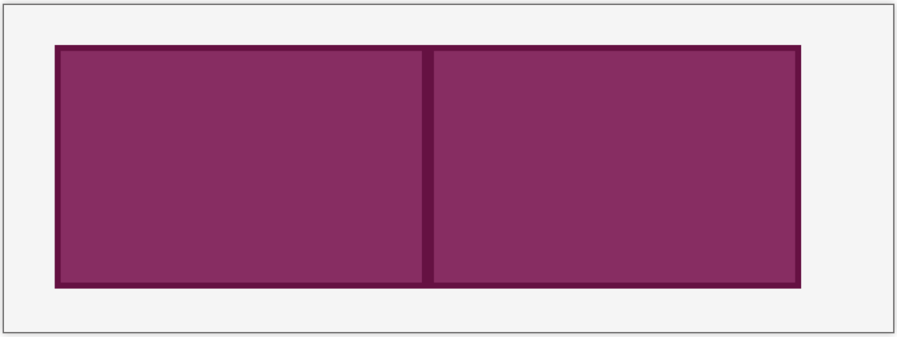


Figure 41

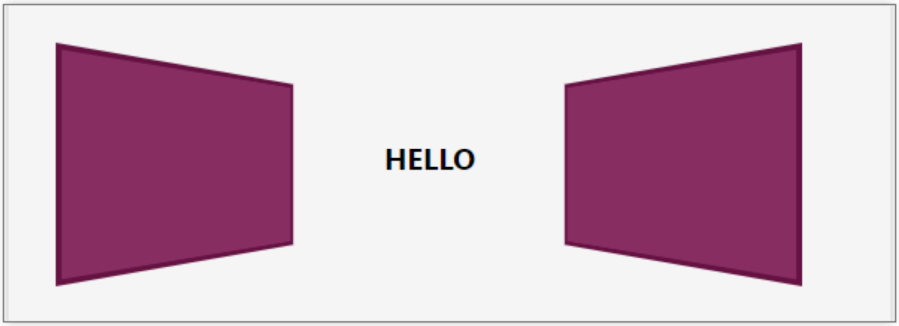


Figure 42

Task 3. Implement an HTML page with several blocks consisting of text + image

Requirements:

- the block consists of text and an image, which equally divide the available width;
- the block must be of a fixed height;
- the text should be in the middle of the allocated space;
- the image should occupy its entire width and height;
- the image should not go beyond the allocated space;

- when hovered, the block (text + image):
- must be completely covered with a translucent black plate in 0.3 seconds;
- the image must smoothly (in 1 second) enlarge, but at the same time, not go beyond the allocated space.

Images are up to you.
The example of the result (the third block is hovered):

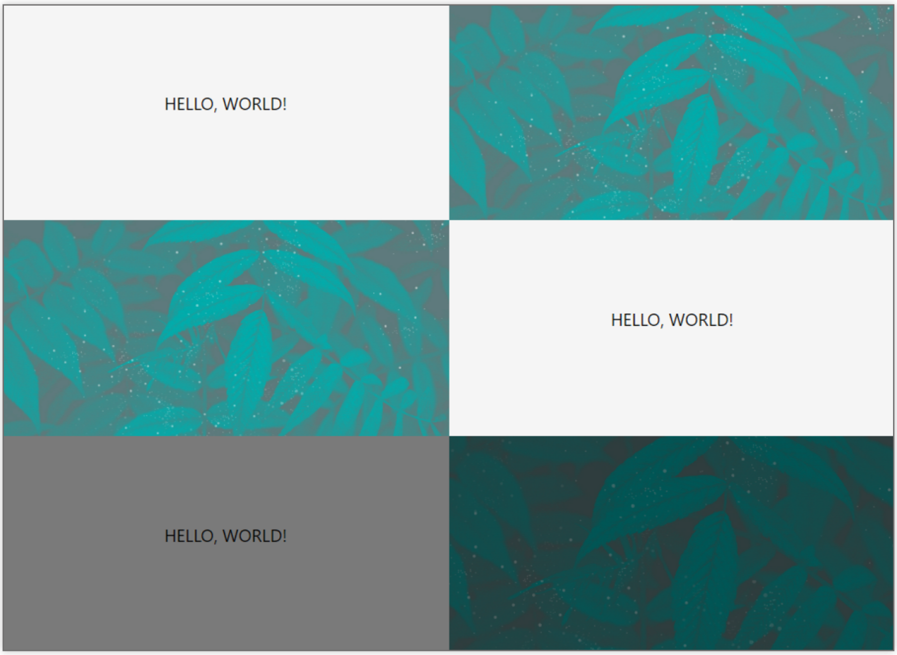


Figure 43

Lesson 4/1. Animation in Web Design