CSS Animations

# Transitions

transition is a css property that allows to change other properties of an html element smoothly over a given duration.

like doing something on hover etc.

button:hover {

    background-color: green;

}

but the above code makes the change of color happen so fast that its like on and off switch. transition is not smooth.

For transition we need two things:

1. css property which will be changed
2. duration – amount of time the above property change will take to complete.

As you will see blow we added transition property (1) and duration (2) to the button. now the change on hover happens over 1s smoothly.

button {

    height: 90px;

    width: 290px;

    font-size: 28px;

    color: white;

    border: none;

    background-color: blue;

    transition-property: background-color;

    transition-duration: 1s;

}

button:hover {

    background-color: green;

}

Note: we can also give duration in ms like 3000ms.

## Transition options

besides the above 2 properties required we have 2 more properties

1. transition timing function
2. transition delay

### Transition timing function

it defines how the change in property will occur. example if on hover the bg color changes at equal rate or changes rapidly first then slowly. it has 5 values:

1. ease: this is default value. change starts slow, goes faster, ends slow
2. linear: change occurs at same rate
3. ease-in: change starts slow and speeds up in the end
4. ease-out: change starts fast and slows in the end
5. ease-in-out: start slow, speed up, ends slow. similar to ease.

.move {

    margin: 2em;

    width: 150px;

    height: 50px;

    padding: 10px 15px;

    background-color: red;

    color: #ffffff;

    text-align: center;

    line-height: 50px;

    border-radius: .5em;

    font-size: 20px;

    transition-property: translate();

    transition-duration: 3s;

}

here transition is applied on translate property and duration is 3s

we can define timing function as follows

#linear {

    transition-timing-function: linear;

}

same for ease-in etc.

.move:hover {

    transform: translate(900px, 0);

}

on hover we translate 900px on x axis and 0 on y axis.

### Transition delay

time to wait before start of transition.

accepts time in seconds and milliseconds

In the button bg color change example if we add

    transition-property: background-color;

    transition-duration: 3000ms;

    transition-delay: 2s;

then the color change on hover starts after 2s and the change takes 3s as per duration given.

## Applying transition to different properties of the same element

just keep adding properties space separated to transition property.

**NOTE: ON SPECIFIC CHROME VERSIONS THE ANIMATIONS HAPPEN ON LOAD OF PAGE, TO AVOID THAT ADD TRANSITION PROPERTIES TO THE ACTION LIKE HOVER ETC.**

button {

    height: 90px;

    width: 290px;

    font-size: 28px;

    color: white;

    border: none;

    background-color: blue;

    /\* transition-property: background-color color width;

    transition-duration: 3000ms;

    transition-delay: 2s; \*/

}

button:hover {

    transition-property: background-color color width;

    transition-duration: 3000ms;

    transition-delay: 2s;

    background-color: green;

    color: black;

    width: 600px;

}

instead of using individual properties we can also define ‘all’ for transitioning all property changes.

    transition-property: all;

    transition-duration: 3000ms;

    transition-delay: 2s;

**Shorthand to write transition duration and property**

    transition: background-color 3s ease-in-out 1s,

    color 2s ease 4s;

note here that color change starts after background color has changed (1s delay and 3s duration)

1st param is propert, 2nd is duration 3rd is timing function and 4th is delay. we can have multiple comma separated properties.

**NOTE: to get oval shape use border-radius 50%**

we can use shorthand for all

    transition: all 3s ease-in-out 1s;

but then we cannot control individual durations and delays.

## What properties can be transitioned

animatable properties are those which have gradual value changes and hence changes can be smooth through transition but some properties just jump from old to new value without any intermediate values and these are non-animatable.

Examples of non-animatable properties

1. background image (image changes from one to other, no intermediate state)
2. display
3. border-style
4. position
5. float
6. background-repeat
7. font-family etc.

Animatable

1. width
2. height
3. border-width
4. padding
5. margin
6. line-height
7. opacity
8. color
9. background-color

check mozilla dev site for more

# Transforms

we use them to change elements. we change size shape position etc of elements.

transform property has 4 functions to control how are elements will be displayed

1. scale – resize elements (incr decr height or width)
2. translate – move element position (horizontally, vertically)
3. rotate – rotate by degrees or number of turns in 2d
4. skew – skew element along x or y axis.

## translate

img {

    width: 300px;

    display: block;

    margin: auto;

    margin-top: 200px;

    transition: transform 2s;

}

img:hover {

    transform: translate(-500px, 200px);

}

this will take image to left 500px and bottom 200px. remember 0,0 is top left corner.

To only move along x axis

    transform: translateX(-500px);

for Y

    transform: translateY(-500px);

## Scale

changes height and width of element

img {

    width: 300px;

    display: block;

    margin: auto;

    margin-top: 200px;

    transition: transform 2s;

}

img:hover {

    transform: scaleX(2);

}

number passed to scaleX is relative to element. so width increases to twice the original value

similar we have scaleY

and for both height and width we have scale(2).

to decrease use values less than 1 like 0.5 to reduce to half the size.

we also write like scale(2,2) where first 2 is for width doubling and second is for height.

## Rotate

it takes angle to which we want to rotate by like 60deg

img:hover {

    transform: rotate(20deg);

}

rotate 20deg clockwise

use -20deg for anti clockwise

img:hover {

    transform: rotate(1turn);

}

1 turn means 360 degree clockwise. use -1 for anticlockwise

for half turn we can use 0.5

we can also use grad. 400 grad is equal to 360 degree

## Skew

like above we have skewX, skewY and skew() with first param for x and second for y.

we use degrees to tell how much to skew.

img:hover {

    transform: skew(20deg, 40deg);

}

like all other functions use negative to change direction within the axis.

# Transform origin

Changes the point from which the transform is applied.

Keyword options include top, bottom, right, left

div {

    width: 150px;

    height: 150px;

    background-color: #27ae60;

    display: block;

    margin: auto;

    margin-bottom: 70px;

    color: white;

    font-size: 24px;

    font-family: 'Roboto Slab', serif;

    text-align: center;

    line-height: 150px;

    transition: transform 1s;

}

div:hover {

    transform: rotate(45deg);

}

.top {

    transform-origin: top;

}

.right {

    transform-origin: right;

}

.left {

    transform-origin: left;

}

.bottom {

    transform-origin: bottom;

}

.top-right {

    transform-origin: top right;

}

.top-left {

    transform-origin: top left;

}

.bottom-right {

    transform-origin: bottom right;

}

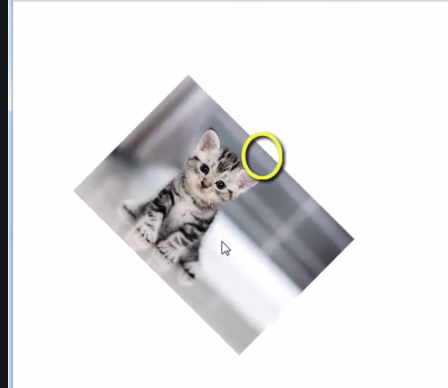
.bottom-left {

    transform-origin: bottom left;

}

By default origin is center of div

Top means top center



Right



Bottom



Left



Top right



We can also give in percentage:

Calculation done from top left corner (0,0) 100% on x axis means top right and 100 percent on y axis

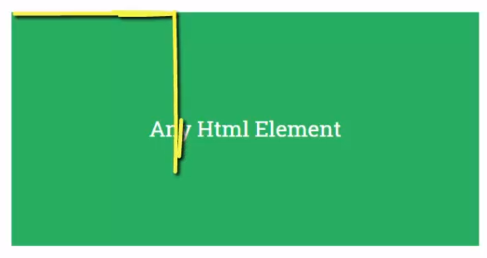
Means bottom left.

For center we use transform-origin: 50% 50%

    transition: transform 1s;

    transform-origin: 30% 80%;

30 percent to right and 80 percent bottom starting from top left corner.



We can also give pixels

    transition: transform 1s;

    transform-origin: 300px 150px;

# 3d transforms

We have z axis as well.

We need to create perspective (distance between our eye and screen) for using z axis in a 2d monitor.

perspective needs to be added to parent/enclosing element. if img tag inside body then body is where perspective should be given.

<!doctype html>

<html>

<head>

<meta charset="UTF-8">

<title>translate in 3D</title>

<style type="text/css">

body {

    perspective: 500px;

}

img {

    width: 300px;

    display: block;

    margin: auto;

    margin-top: 200px;

    transition: transform 1s;

}

img:hover {

    transform: translateZ(800px);

}

</style>

</head>

<body>

    <img src="cat.jpg" alt="">

</body>

</html>

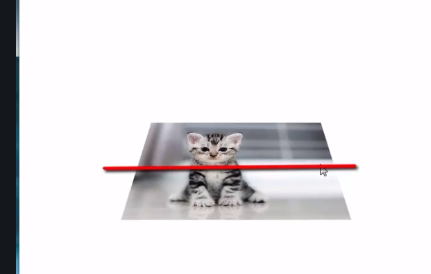
it appears to grow but actually the image is coming closer to us. if we negative it will go away hence appear smaller.

perspective is 1000 so assumption is screen is 1000px from my eyes and image comes 300px closer if we give translate as 300px. if perspective is 500px then image will appear much larger than in case of 1000px since 300 out of 500 is much closer.

if translateZ is more than perspective it means image will go past our eyes and we wont be able to see it.

**Rotate z axis**

it is similar to normal rotate since z axis is line coming towards our eyes.

rotateX() make img rotate along horizontal axis

rotateY works similarly across y axis. if we rotate by 90deg image disappears since its parallel to our eyesight. we see a line with 0 thickness, so disappears.

# Animations

.icon::before{

content: "";

position: absolute;

width: 120%;

height: 120%;

background: #10ac84;

left: -110%;

top: 90%;

transform: rotate(45deg);

}

.icon:hover::before{

animation: animation 0.7s 1 forwards;

}

@keyframes animation {

0%{

left: -110%;

top: 90%;

}50%{

left: 15%;

top: -30%;

}100%{

top: -10%;

left: -10%;

}

}

To create animation use @keyframes give it a name “animation” in this case

Define at what % of time what attribute values we want to give.

Use animation property on hover or other event to apply animation.

We write forwards so that after animation finishes the html remains in that state.

1. Define a css animation rule and give it a name using @keyframes
2. Add to html element you want to animate

body {

    margin: 0;

}

div {

    width: 150px;

    height: 150px;

    background-color: red;

    animation-name: moving;

    animation-duration: 3s;

}

@keyframes moving {

    from { transform: translateX(0); }

    to { transform: translateX(1000px); }

}

In this rule we define starting state to final state using “from” and “to”

## Animation name

Name of animation we defined the rule in. Eg “moving”

## Animation duration

The duration in which the animation completes

## Animation delay

Wait for some time before starting animation

    animation-duration: 5s;

    animation-fill-mode: both;

    animation-delay: 3s;

}

## More Stages

We can add percentage of time passed instead of “from” and “to” keywords

@keyframes moving {

    0% { transform: translateX(0); }

    10% { transform: translateX(1000px); }

    100% { transform: translate(1000px, 500px); }

}

## Fill mode property

Tells what to do with element after animation ends

Without this by default after animation completes the element goes back to initial state which was before the animation started.

Options

1. None – does nothing after animation window. Element returns to original position. This is default, whatever we define in keyframes will have no affect after the animation window.
2. Forwards – element takes last stage of animation (mentioned in keyframes) after animation is finished.
3. Backwards – element starts from the first stage of animation (mentioned in keyframes) and not the position or state of the actual element. But after animation ends we again go to original state of element.
4. Both – applies both forwards and backwards.

div {

    width: 150px;

    height: 150px;

    background-color: red;

    animation-name: moving;

    animation-duration: 5s;

    animation-fill-mode: both;

    animation-delay: 3s;

}

@keyframes moving {

    0% { transform: translateX(500px); }

    50% { transform: translateX(1000px); }

    100% { transform: translate(1000px, 500px); }

}

## Animation iteration count

How many times iteration should be repeated. Accepts values in numbers. 2 means animation repeated 2 times.

div {

    width: 150px;

    height: 150px;

    background-color: red;

    animation-name: moving;

    animation-duration: 5s;

    animation-fill-mode: both;

    animation-iteration-count: 2;

    animation-timing-function: linear;

}

We can give count as “infinite” to repeat indefinitely

Animation timing function takes same values as transition timing functions: ease in, linear, ease-in-out.

# Miscellaneous points

Center element horizontally and vertically using flexbox

body {

height: 100vh; // this is required else flex align doesn’t work

justify-content: center;

align-items: center;

}

## pseudo elements

allows us to add css before or after an html element

body {

    height: 100vh;

    display: flex;

    justify-content: center;

    align-items: center;

}

a {

    text-decoration: none;

    color: #262626;

    font-family: sans-serif;

    font-size: 45px;

    border: 5px solid #262626;

    padding: 40px 80px;

    position: relative;

    overflow: hidden;

}

a:before {

    content: '';

    position: absolute;

    left: 0;

    top: 0;

    background-color: #fff200;

    height: 100%;

    width: 100%;

    z-index: -1;

    transform-origin: bottom left;

    transform: rotate(-90deg);

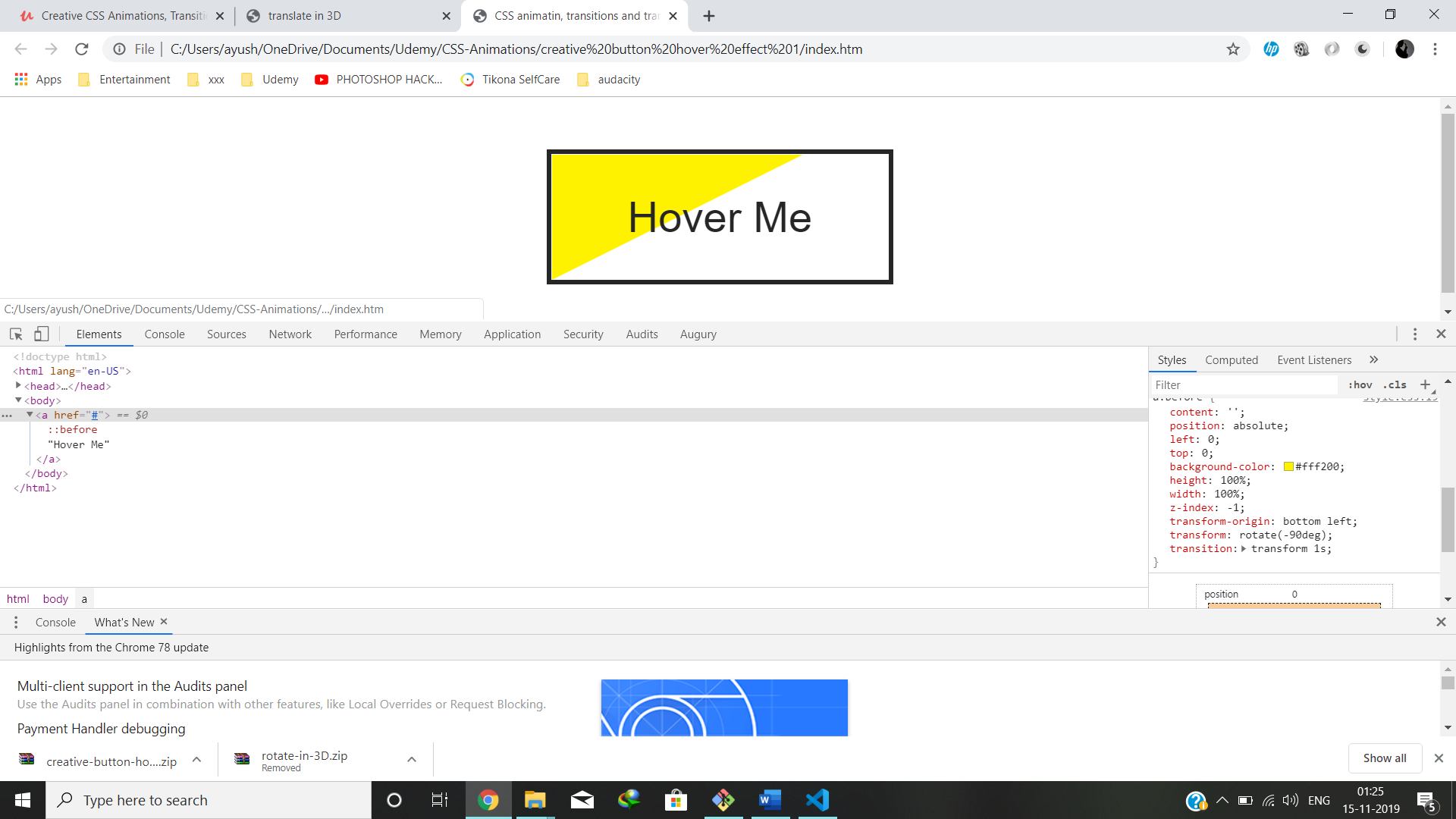
    transition: transform 1s;

}

a:hover:before {

    transform: rotate(0deg);

}



the above image is between the transition

<https://dev.to/finallynero/understanding-css-before-and-after-pseudo-elements-ml0>

use overflow hidden to hide before after pseudo elements outside the actual element.

NOTE: do line-height: 40px when height is 40px to center text vertically inside display block.

a:hover span {} is used to change span when anchor tag is hovered.

a:hover:before works same way.

When position is relative: all positions like left right etc will be with respect to current position

Position: absolute means position is calculated from parents perspective. So top 0 and left 0 would take element to to left top corner of parent.

Use css filter to attain blur saturation etc effects.