**CSS | Pseudo-Classes**

**Pseudo-class**

The CSS pseudo-classes allow you to style the dynamic states of an element such as hover, active and focus state, as well as elements that are existing in the document tree but can't be targeted via the use of other selectors without adding any IDs or classes to them, for example, targeting the first or last child elements.

A pseudo-class starts with a colon (:). Its syntax can be given with:

selector: pseudo-class { property: value; }

## Anchor Pseudo-classes

These pseudo-classes let you style unvisited links differently from visited ones. The most common styling technique is to remove underlines from visited links.

   a:link {

        color: blue;

      }

      a:visited {

        text-decoration: none;

      }

Some anchor pseudo-classes are dynamic, they're applied as a result of user interaction with the document like on hover, or on focus etc.

   a:hover {

        color: red;

      }

      a:active {

        color: gray;

      }

      a:focus {

        color: yellow;

      }

These pseudo-classes change how the links are rendered in response to user actions.

:link  – It adds style to the unvisited link.

:visited  – It selects the visited links and adds special styles to them. Its possible values can be any color name in a valid format.

:hover – This pseudo-class adds a special style to an element when the user moves the cursor over it. If you want to make it effective, it must be applied after the ":link" and "**:**visited"pseudo-classes.

:active  – It applies when the elements are clicked or activated. It selects the activated element.

:focus – applies when the element has keyboard focus. This is not limited to links, but can be used (and really should be used) on inputs and textareas as well.

## Input & link related pseudo class selectors

:focus – It selects the elements that are currently focused on by the user. It is generally used in input elements of the forms and triggers when the user clicks on it.

:enabled – Selects inputs that are in the default state of enabled and ready to be used.

:disabled – Selects inputs that have the disabled attribute. A lot of browsers will make the input a faded out gray, you can control that with this selector.

:checked – Selects checkboxes that are, wait for it, checked.

:required – Selects inputs with the required attribute.

## The :first-child Pseudo-class

The :first-child pseudo-class matches an element that is the first child element of some other element. The selector ol li:first-child in the example below select the first list item of an ordered list and removes the top border form it.

      ol li:first-child {

        color: red;

      }

## The :last-child Pseudo-class

The :last-child pseudo-class matches an element that is the last child element of some other element. The selector ul li:last-child in the example below select the last list item from an unordered list and removes the right border from it.

  ul li:last-child {

        color: blue;

      }

## The :nth-child Pseudo-class

The CSS3 introduces a new :nth-child pseudo-class that allows you to target one or more specific children of a given parent element. The basic syntax of this selector can be given with :nth-child(N), where N is an argument, which can be a number, a keyword (even or odd), or an expression of the form xn+y where x and y are integers (e.g. 1n, 2n, 2n+1, …).

       table tr:nth-child(2n) td {

        background: #eee;

      }

The style rules in the example above simply highlight the alternate table row, without adding any IDs or classes to the <table> elements.

**Tip:** The CSS :nth-child(N) selector is very useful in the situations where you have to select the elements that appears inside the document tree in a specific interval or pattern like at even or odd places, etc.

## 

### CSS | Animation

Transitions do a great job of building out visual interactions from one state to another, and are perfect for these kinds of single state changes. However, when more control is required, transitions need to have multiple states. In return, this is where animations pick up where transitions leave off.

CSS animation is a method of animating certain HTML elements without having to use processor and memory-hungry JavaScript or Flash. There's no limit to the number or frequency of CSS properties that can be changed. CSS animations are initiated by specifying keyframes for the animation: these keyframes contain the styles that the element will have.

CSS animations are made up of two basic building blocks.

1. **Keyframes** - define the stages and styles of the animation.
2. **Animation Properties** - assign the @keyframes to a specific CSS element and define *how* it is animated.

**#1: Keyframes**

Keyframes are the foundation of CSS animations. They define what the animation looks like at each stage of the animation timeline. Each @keyframes is composed of:

* **Name of the animation:** A name that describes the animation, for example, bounceIn.
* **Stages of the animation:** Each stage of the animation is represented as a percentage. 0% represents the beginning state of the animation. 100% represents the ending state of the animation. Multiple intermediate states can be added in between.
* **CSS Properties:** The CSS properties defined for each stage of the animation timeline.

Let’s take a look at a simple @keyframes I’ve named “bounceIn”. This @keyframes has three stages. At the first stage (0%), the element is at opacity 0 and scaled down to 10 percent of its default size, using CSS transform scale. At the second stage (60%) the element fades in to full opacity and grows to 120 percent of its default size. At the final stage (100%), it scales down slightly and returns to its default size.

The @keyframes are added to your main CSS file.

@keyframes bounceIn {

        0% {

          transform: scale(0.1);

          opacity: 0;

        }

        60% {

          transform: scale(1.2);

          opacity: 1;

        }

        100% {

          transform: scale(1);

        }

 }

(If you’re unfamiliar with CSS Transforms, you’ll want to brush up on your knowledge. Combining CSS transforms in the animations is really where the magic happens.)

**#2: Animation Properties**

Once the @keyframes are defined, the animation properties must be added in order for your animation to function.

Animation properties do two things:

1. They assign the @keyframes to the elements that you want to animate.
2. They define *how* it is animated.

The animation properties are added to the CSS selectors (or elements) that you want to animate. You must add the following two animation properties for the animation to take effect:

* animation-name: The name of the animation, defined in the @keyframes.
* animation-duration: The duration of the animation, in seconds (e.g., 5s) or milliseconds (e.g., 200ms).

Continuing with the above bounceIn example, we’ll add animation-name and animation-duration to the div that we want to animate.

div {

        animation-duration: 2s;

        animation-name: bounceIn;

    }

Shorthand syntax:

div {

        animation: bounceIn 2s;

    }

By adding both the @keyframes and the animation properties, we have a simple animation!

**Animation Property Shorthand**

Each animation property can be defined individually, but for cleaner and faster code, it’s recommended that you use the animation shorthand. All the animation properties are added to the same animation: property in the following order:

animation: [animation-name] [animation-duration]

           [animation-timing-function] [animation-delay]

          [animation-iteration-count] [animation-direction]

         [animation-fill-mode] [animation-play-state];

Just remember for the animation to function correctly, you need to follow the proper shorthand order AND specify at least the first two values.

**Sub-properties**

* animation-name: declares the name of the @keyframes at-rule to manipulate.
* animation-duration: the length of time it takes for an animation to complete one cycle.
* animation-timing-function: establishes preset acceleration curves such as ease or linear.
* animation-delay: the time between the element being loaded and the start of the animation sequence.
* animation-direction: sets the direction of the animation after the cycle. Its default resets on each cycle.
* animation-iteration-count: the number of times the animation should be performed.
* animation-fill-mode: sets which values are applied before/after the animation.  
  For example, you can set the last state of the animation to remain on screen, or you can set it to switch back to before when the animation began.
* animation-play-state: pause/play the animation.

**CSS3 Animation: keyframes timing function**

To control the rate of the animation use CSS3 transition-timing-function property.

**Syntax**

The syntax of the property is given with:

|  |  |
| --- | --- |
| animation-timing-function: | linear | ease | ease-in | ease-out | ease-in-out | cubic-bezier(n,n,n,n) |

**Property Values**

linear: The rate of transition remains constant from start to end

ease : The transition begins quickly and then gradually slows down

ease-in : Begins slowly and then ends then speeds up towards the end.

ease-out : The transition begins quickly and stays quick longer than ease, and then slows down and ends abruptly

ease-in-out: Accelerates in the beginning and then decelerates in the end.

cubic-bezier() : Transition is applied using a custom cubic bezier curve.

step-start : Tansition is applied at the start of the step.

step-end: Transiton is applied at the end of the step.

steps() : Change is introduced in steps specified within the brackets.

**CSS3 animation-direction Property**  
  
The animation-direction CSS property specifies whether the animation should play in reverse on alternate cycles or not.  
  
**Syntax**  
  
The syntax of the property is given with:

|  |  |
| --- | --- |
| animation-direction: | normal | reverse | alternate | alternate-reverse |

**Property Values**

normal: The animation should play forward in each cycle. This is default.

reverse: The animation should play backward in each cycle.

alternate: The animation plays forward in the first cycle, then play backward, then continues to alternate.

alternate-reverse: The animation plays backward in the first cycle, then play forward, then continues to alternate.

**CSS3 animation-fill-mode Property**

The animation-fill-mode CSS property specifies how a CSS animation should apply styles to its target before and after it is executing.

**Syntax**

The syntax of the property is given with:

|  |  |
| --- | --- |
| animation-fill-mode: | none | forwards | backwards | both | [initial](https://www.tutorialrepublic.com/definitions.php#initial) | [inherit](https://www.tutorialrepublic.com/definitions.php#inherit) |

**Property Values**

none: The animation will not apply any styles to the target before or after it is executing.

forwards: After the animation ends (as determined by its animation-iteration-count), the animation will apply the property values for the time the animation ended.

backwards: The animation will apply the property values defined in the keyframe that will start the first iteration of the animation, during the period defined by animation-delay property. These are either the values of the from keyframe (when animation-direction is normal or alternate) or those of the to keyframe (when animation-direction is reverse or alternate-reverse).

both: The animation will follow the rules for both forwards and backwards, thus extending the animation properties in both directions.