CSS3 Text & Font

CSS3 Text Effects

- CSS3 contains several new text features:
  - text-shadow
  - word-wrap

- Browser support
  - Internet Explorer does not yet support the text-shadow property
  - Firefox, Chrome, Safari, and Opera support the text-shadow property
  - All major browsers support the word-wrap property

CSS3 Text Shadow

- In CSS3, the text-shadow property applies shadow to text

```
h1 {
    text-shadow: 5px 5px 5px #FF0000;
}
```

  - Values of **text-shadow** are:
    - Horizontal shadow
    - Vertical shadow
    - Blur distance
    - Color of the the shadow

CSS3 Word Wrapping

- If a word is too long to fit within an area, it expands outside
- In CSS3, the word-wrap property allows you to force the text to wrap - even if it means splitting it in the middle of a word

```
p {word-wrap:break-word;}
```

- word-wrap can have one of two values:
  - normal : Break words only at allowed break points
  - break-word : Allows unbreakable words to be broken

CSS3 Fonts

- With CSS3, web designers are no longer forced to use only "web safe" fonts
  - Before CSS3, web designers had to use fonts that were already installed on the user's computer

- With CSS3, web designers can use whatever font they like
  - Simply include the font file in the page, and it will be downloaded automatically to the browser when needed

- The selected font is described with the @font-face rule
  - In the @font-face rule you define a name for the font, and the URL to the font file:

CSS3 Fonts…

```
@font-face {
    font-family:myFirstFont;
    src:url('Sansation_Light.ttf'),
        url('Sansation_Light.eot') format("opentype"); /* IE */
}
```

- Browser support
  - Internet Explorer only support fonts of type .eot (Embedded OpenType)
  - Firefox, Chrome, Safari, and Opera support fonts of type .ttf (True Type Fonts) and .otf (OpenType Fonts)

CSS3 Font Descriptors

- The font descriptors that can be defined inside the @font-face rule:
- font-family: *name*
  - Required. Defines a name for the font
- url: *url*
  - Required. Defines the URL to the font file
- font-style: normal | italic | oblique
  - Optional. Defines how the font should be styled. Default is "normal"
- unicode-range: *unicode-range*
  - Optional. Defines the range of UNICODE characters the font supports. Default is "U+0-10FFFF"

CSS3 Font Descriptors…

- font-stretch : *values can be:*
  - normal | condensed | ultra-condensed | extra-condensed | semi-condensed | expanded | semi-expanded | extra-expanded | ultra-expanded
    - Optional. Defines how the font should be stretched. Default is "normal"

- font-weight : *values can be:*
  - normal | bold | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900
    - Optional. Defines the boldness of the font. Default is "normal"

CSS3 Transformations

move, scale, turn, spin, and stretch

CSS3 Transforms

- With CSS3 transform, we can move, scale, turn, spin, and stretch elements
- A transform is an effect that lets an element change shape, size and position
- You can transform your elements using 2D or 3D transformation
- Browser Support
  - Internet Explorer 9 requires the prefix -ms-
  - Firefox requires the prefix -moz-
  - Chrome and Safari requires the prefix -webkit-
  - Opera requires the prefix -o-

2D Transforms

- The transform property has the following methods:
  - translate()
  - rotate()
  - scale()
  - skew()

- Usage:

```
div {
    transform: method( args );
}
```

The translate() Method

- With the translate() method, the element moves from its current position, depending on the parameters given for the left (X-axis) and the top (Y-axis) position:

```
div {
    transform: translate(50px,100px);
    -ms-transform: translate(50px,100px); /* IE 9 */
    -webkit-transform: translate(50px,100px); /*Safari /Chrome*/
    -o-transform: translate(50px,100px); /* Opera */
    -moz-transform: translate(50px,100px); /* Firefox */
}
```

  - The value translate(50px,100px) moves the element 50 pixels from the left, and 100 pixels from the top
  - Arguments can be either in measure units or %
  - You can also use translateX() and translateY() methods

The rotate() Method

- With the rotate() method, the element rotates clockwise at a given degree. Negative values are allowed and rotates the element counter-clockwise:

```
div {
transform: rotate(30deg);
-ms-transform: rotate(30deg); /* IE 9 */
-webkit-transform: rotate(30deg); /* Safari / Chrome */
-o-transform: rotate(30deg); /* Opera */
-moz-transform: rotate(30deg); /* Firefox */
}
```

  - The value rotate(30deg) rotates the element 30 degrees clockwise

The scale() Method

- With the scale() method, the element increases or decreases the size, depending on the parameters given for the width (X-axis) and the height (Y-axis):

```
div {
transform: scale(2,4);
-ms-transform: scale(2,4); /* IE 9 */
-webkit-transform: scale(2,4); /* Safari / Chrome */
-o-transform: scale(2,4); /* Opera */
-moz-transform: scale(2,4); /* Firefox */
}
```

  - The value scale(2,4) transforms the width to be twice its original size, and the height 4 times its original size
  - You can also use fractional values
  - If only one value is given to **scale()**, then the same value is applied for X & Y
  - You can also use scaleX() or scaleY() methods

The skew() Method

- With the skew() method, the element turns in a given angle, depending on the parameters given for the horizontal (X-axis) and the vertical (Y-axis) lines:

```
div {
transform: skew(30deg,20deg);
-ms-transform: skew(30deg,20deg); /* IE 9 */
-webkit-transform: skew(30deg,20deg); /* Safari / Chrome */
-o-transform: skew(30deg,20deg); /* Opera */
-moz-transform: skew(30deg,20deg); /* Firefox */
}
```

  - The value skew(30deg,20deg) turns the element 30 degrees around the X-axis, and 20 degrees around the Y-axis

Combining Transforms

- For using multiple transforms at once, list the transform values within the transform property one after the other without the use of commas

- Example:

```
div.content {
    transform: rotate(25deg) scale(.75);
    -ms-transform: rotate(25deg) scale(.75);
    -webkit-transform: rotate(25deg) scale(.75);
    -o-transform: rotate(25deg) scale(.75);
    -moz-transform: rotate(25deg) scale(.75);
}
```

What do you get with this?

- The HTML:

```
<div id="wrapper">
    <div class="shape">
      <div class="side top"> </div>
      <div class="side left"> </div>
      <div class="side right"> </div>
    </div>
</div>
```

- The CSS:

```
div#wrapper { margin:50px auto; width:85%; }
.shape { height: 230px; position: relative; }
.side { height: 100px; position: absolute; width: 100px; }
```

What do you get with this?...

- The CSS...

```
.top {
  background: #9acc53;
  -webkit-transform: rotate(-45deg) skew(15deg, 15deg);
}
.left {
  background: #8ec63f;
  -webkit-transform: rotate(15deg) skew(15deg, 15deg)
    translate(-50%, 100%);
}
.right {
  background: #80b239;
  -webkit-transform: rotate(-15deg) skew(-15deg, -15deg)
    translate(50%, 100%);
}
```

Transform Origin

- The default point of origin for transforms are the center of the element – 50% horizontal, 50% vertical
- With **transform-origin** property, the default position can be changed

- Example:

```
div.content {
    transform: rotate(30deg);
    -webkit-transform: rotate(30deg);
    -webkit-transform-origin: bottom right;
}
```

  - Values can be given as top / bottom, left / right, in %, in specific units like px
  - Note that this property collides with translate() as both try to move the element

3D Transforms

- CSS3 allows you to format your elements using 3D transforms
- 3D Transform has two methods:
  - rotateX()
  - rotateY()

The rotateX() Method

- With the rotateX() method, the element rotates around its X-axis at a given degree

```
div {
transform: rotateX(120deg);
-webkit-transform: rotateX(120deg); /* Safari and Chrome */
}
```

The rotateY() Method

- With the rotateY() method, the element rotates around its Y-axis at a given degree

```
div {
transform: rotateY(130deg);
-webkit-transform: rotateY(130deg); /* Safari and Chrome */
}
```

The rotateZ() Method

- With the rotateZ() method, the element rotates around its Z-axis at a given degree

```
div {
transform: rotateZ(130deg);
-webkit-transform: rotateZ(130deg);
}
```

Perspective

- In order for three-dimensional transforms to work the elements need a perspective from which to transform

- The perspective can be thought of as a point from which you are viewing the element

- The **perspective()** method in transform takes one argument:

```
.box {
    transform: perspective(200px) rotateX(45deg);
    -webkit-transform: perspective(200px) rotateX(45deg);
}
```

Perspective Origin

- **perspective-origin()** property, similar to *tranform-origin*, sets the point of origin for the perspective

- Example:

```
.box {
    transform: perspective(200px) rotateX(45deg);
    perspective-origin: 0 0;
}
```

The scaleZ() Method

- scaleZ() method scales the element on the z-axis

- Example:

```
div.content {
    width:250px; height:250px; background-color:#ff0;
    -webkit-transform: perspective(200px) scaleZ(.25)
    rotateX(45deg);
}
```

The translateZ() Method

- translateZ() method translates the element on the z-axis
  - A negative value pushes an element further away on the z axis, resulting in a smaller element

- Example:

```
div.box1 {
    transform: perspective(200px) translateZ(-50px);
}
div.box2 {
    transform: perspective(200px) translateZ(50px);
}
```

The skewZ() method?

- There is no skewZ() method

Transform Style

- On occasions, 3D transforms will be applied on elements which is nested within a parent element which is also being transformed

- In this event, the nested, transformed elements will not appear in their own three-dimensional space

- To allow nested elements to transform in their own three-dimensional plane use the **transform-style** property with the **preserve-3d** value

The transform-style Property

- The transform-style property needs to be placed on the parent element, above any nested transforms
  - It can have values preserve-3d or flat
  - The preserve-3d value allows the transformed children elements to appear in their own three-dimensional plane
  - The flat value forces the transformed children elements to lie flat on the two-dimensional plane

- Example:

transform-style Example

- The HTML:

```
<div class="rotate three-d">
  <div class="box">Box 1</div>
</div>
<div class="rotate">
  <div class="box">Box 2</div>
</div>
```

- The CSS:

```
.rotate {
    border:1px dotted #333;
    -webkit-transform: rotateY(45deg);
    -webkit-perspective: 200px;
}
```

transform-style Example

- The CSS…

```
.three-d {
    -webkit-transform-style: preserve-3d;
}
.box {
    width:150px; height:150px; background-color:#f90;
    border-radius: 9px; margin-bottom:50px; text-
    align:center;
    -webkit-transform: rotateX(15deg) translateZ(20px);
    -webkit-transform-origin: 0 0;
}
```

Backface Visibility

- Elements can be transformed in a way that makes them face away from the screen
  - For example, with rotateY(180deg)
- **backface-visibility** property makes the elements show from the back
  - A value of **hidden** will hide all the elements
  - A value of **visible** will displays the elements
- Example:

```
.box1 {
    transform: rotateY(180deg);
    backface-visibility: hidden;
}
.box2 {
    transform: rotateY(180deg);
    backface-visibility: visible;
}
```

CSS3 Transitions & Animations

CSS3 Transitions

- With CSS3 transitions, you can alter the appearance and behavior of an element
- The appearance / behavior is altered whenever a state change occurs, like hover, focus or active

- There are four transition related properties –
  - transition-property
  - transition-duration
  - transition-timing-function
  - transition-delay

- The states at which transitions take place are
  - :hover
  - :focus
  - :active
  - :target

Transition Example

- The HTML:

```
<div class="box">hover to start</div>
```

- The CSS:

```
div.box {
    width:250px; height:250px; border-radius:12px;
    background-color:#ff0;
    transition-property: background;
    transition-duration: 2s;
    transition-timing-function: linear;
}

div.box:hover {
    background-color:#FF6600;
}
```

transition-property

- **transition-property** determines what properties will be altered
  - Not all properties may be transitioned, only properties that have an identifiable halfway point can be transitioned
  - Colors, sizes, font sizes, and the like may be transitioned
  - Properties like *display* cannot be transitioned
  - Multiple properties can be transitioned by specifying them as a comma separated list:

```
div.box {
    transition-property: background, border-radius;
}
```

transition-duration

- **transition-duration** sets the time for the transition to take place
  - The value can be given in seconds (s) or milliseconds (ms)
  - The values can be given in fractions too
  - When transitioning multiple properties, set multiple durations, as comma separated list

```
div.box {
    transition-property: background, border-radius;
    transition-duration: .2s, 1s;
}
```

transition-timing-function

- **transition-timing-function** property is used to set the speed in which a transition will move
- It can take one of these values –
  - linear = uniform speed throughout the animation
  - ease-in = start slow and gradually speed up
  - ease-out = start quick and end slow
  - ease-in-out = start slow, speed up, end slow
  - With multiple transform properties, provide comma separated list

```
div.box {
    transition-property: background, border-radius;
    transition-duration: .2s, 1s;
    transition-timing-function: linear, ease-in;
}
```

transition-delay

- **transition-delay** sets a time value, that determines how long a transition should be stalled before executing

```
div.box {
    width:250px; height:250px;
    background-color:#FFFF00; border-radius:6px;
    transition-property: background, border-radius;
    transition-duration: .2s, 1s;
    transition-timing-function: linear, ease-in;
    transition-delay: 0, 1s;
}
div.box:hover {
    background-color:#FF6600;
    border-radius: 50%;
}
```

Shorthand Transition

- **transition** property is shorthand for setting all the transition properties (transition-property, transition-duration, transition-timing-function, transition-delay) is a single declaration:

```
div {
transition: width 2s linear, height 2s ease-in,
    transform 2s ease-out;
}
```

CSS3 Animations

Animations

- Transitions alter the appearance / behavior from one state to another
- However, transitions cannot handle more than one state change
- Animations are used when transitions need to have multiple states
- Animations pick up where transitions leave off

Animation Keyframes

- The **@keyframes** rule allow setting of multiple points at which an element should undergo a transition

- The @keyframes rule includes
  - the animation name
  - any animation breakpoints
  - the properties intended to be animated

- Vendor prefixed keyframe rule
  - @-webkit-keyframes
  - @-moz-keyframes
  - @-o-keyframes

@keyframes rule

```
@keyframes bounce {
  0% {
    left: 0;
    top: 0;
  }
  50% {
    left: 305px;
    top: 100px;
  }
  100% {
    left: 610px;
    top: 0;
  }
}
```

- The animation is named **bounce**
- The different keyframe breakpoints are set using percentages, starting at 0% and ending to 100%
- An intermediate breakpoint is at 50%
- The element properties to be animated are listed inside each of the breakpoints
- The keywords ***from*** and ***to*** could be used in place of 0% and 100%

animation-name

- Once the keyframes for an animation have been declared they need to be assigned to an element

- **animation-name** property is used with the animation name, identified from the @keyframes rule, as the property value

```
.stage:hover .ball {
    animation-name: bounce;
}
```

animation-duration

- Animations need a duration declared using the **animation-duration** property
  - Can be specified in seconds (s) or milliseconds (ms)
- **animation-timing-function** and **animation-delay** properties can also be given to fine tune the animation
  - Both the properties are similar to as in transition

```
.stage:hover .ball {
    animation-name: bounce;
    animation-duration: 2s;
    animation-timing-function: ease-in-out;
    animation-delay: .5s;
}
```

Animation Example

- The HTML:

```
<div class="stage">
    <div class="ball"></div>
</div>
```

- The CSS:

```
@-webkit-keyframes bounce { ... }
.stage {
    height: 150px; position: relative;
}
.ball {
    width: 50px; height: 50px;
    position: absolute;
    border-radius:50%;
    background-image: -webkit-radial-gradient(center, circle cover,
    #FFEDA3, #FFC800);
}
.stage:hover .ball { ... }
```

Customizing Animations

- By default, animations run only once
- **animation-iteration-count** may be used to repeat the animation numerous times
  - The value can be either an integer or the *infinite* keyword

```
.stage:hover .ball {
    animation-name: bounce;
    animation-duration: 2s;
    animation-timing-function: ease-in-out;
    animation-delay: .5s;
    animation-iteration-count: infinite;
}
```

animation-direction

- **animation-direction** specifies the direction in which the animation plays
- It can have one of these values:
  - normal = plays from begin to end
  - reverse = plays in the opposite direction of the breakpoints in @keyframes, that is from 100% to 0%
  - alternate = plays animation forwards and then backwards, that is 0% to 100% and then 100% to 0%
  - alternate-reverse = starts at 100%, running to 0% and then back to 100%

animation-play-state

- **animation-play-state** property allows an animation to be *running* or *paused*
  - A *paused* state is usually set on :active

```
.stage:hover .ball {
    …
    animation-direction: alternate;
}
.stage:active .ball {
    animation-play-state: paused;
}
```

animation-fill-mode

- **animation-fill-mode** property identifies how an element should be styled either before, after, or before and after an animation is run
  - accepts four keyword values:
  - none = will not apply any styles
  - forwards = will keep the styles declared within the last specified keyframe
  - backwards = will apply the styles within the first specified keyframe, before the animation has been run
  - both = will apply the behaviors from both the forwards and backwards values

Shorthand Animations

- The animation properties can be written in a shorthand format with the **animation** property
- The order of values within the animation properties should be
  - animation-name
  - animation-duration
  - animation-timing-function
  - animation-delay
  - animation-iteration-count
  - animation-direction

  - The *animation-fill-mode* should be written separately

```
.stage:hover .ball {
animation: slide 2s ease-in-out .5s infinite alternate;
}
```