HTML5 & CSS3

Presentation by Uplatz

Contact Us: https://training.uplatz.com/
Email: info@uplatz.com
Phone:+44 7836 212635

Table Of Contents:

Shapes for Floats

Parameter	Details
None	A value of none means that the float area (the area that is used for wrapping content around a float element) is unaffected. This is the default/initial value.
basic-shape	Refers to one among inset(), circle(), ellipse() or polygon(). Using one of these functions and it values the shape is defined.
shape-box	Refers to one among margin-box, border-box, padding-box, content-box. When only <shape-box> is provided (without <basic-shape>) this box is the shape. When it is used along with <basic-shape>, this acts as the reference box.
image	When an image is provided as value, the shape is computed based on the alpha channel of the image specified.

Shape Outside with Basic Shape – circle()

With the shape-outside CSS property one can define shape values for the float area so that the inline content wraps around the shape instead of the float's box.

CSS

```
img:nth-of-type(1) {
 shape-outside: circle(80px at 50% 50%);
 float: left;
 width: 200px;
}
img:nth-of-type(2) {
 shape-outside: circle(80px at 50% 50%);
 float: right;
 width: 200px;
}
```

```
p {
 text-align: center;
 line-height: 30px; /* purely for demo */
}
```

HTML

```
<img src="http://images.clipartpanda.com/circle-
clip-art-circlergb.jpg">
<img src="http://images.clipartpanda.com/circle-
clip-art-circlergb.jpg">
<p>Some paragraph whose text content is required
to be wrapped such that it follows the curve of
the circle on either side. And then there is some filler
text just to make the text long enough.
Lorem Ipsum Dolor Sit Amet....</p>
```

- In the above example, both the images are actually square images and when the text is placed without the shapeoutside property, it will not flow around the circle on either side.
- It will flow around the containing box of the image only.
- With shape-outside the float area is re-defined as a circle and the content is made to flow around this imaginary circle that is created using shape-outside.
- The imaginary circle that is used to re-define the float area is a circle with radius of 80px drawn from the center-mid point of the image's reference box.

Shape margin

- The shape-margin CSS property adds a margin to shape-outside.

CSS

```
img:nth-of-type(1) {
 shape-outside: circle(80px at 50% 50%);
 shape-margin: 10px;
 float: left;
 width: 200px;
}
img:nth-of-type(2) {
 shape-outside: circle(80px at 50% 50%);
 shape-margin: 10px;
 float: right;
 width: 200px;
}
p {
 text-align: center;
 line-height: 30px; /* purely for demo */
}
```

HTML

```
<img src="http://images.clipartpanda.com/circle-clip-
art-circlergb.jpg">
<img src="http://images.clipartpanda.com/circle-clip-
art-circlergb.jpg">
<p>Some paragraph whose text content is required to
be wrapped such that it follows the curve of
the circle on either side. And then there is some filler
text just to make the text long enough.
Lorem Ipsum Dolor Sit Amet....</p>
```

- In this example, a 10px margin is added around the shape using shape-margin.
- This creates a bit more space between the imaginary circle that defines the float area and the actual content that is flowing around

Counters:

Parameter **Details**

counter-name This is the name of the counter that needs to be created or incremented or printed. It can be any custom name as the developer wishes.

Integer This integer is an optional value that when provided next to the counter name will represent the initial valu counter-reset properties) or the value by which the counter should be incremented (in counter-increment).

None This is the initial value for all 3 counter-* properties. When this value is used for counterincrement, the value of none of the counters are affected. When this is used for the other two, no counter is created.

counter-style This specifies the style in which the counter value needs to be displayed. It supports all values supported by the list-style-type property. If none is used then the counter value is not printed at all.

connector-string This represents the string that must be placed between the values of two different counter levels (like the "." in "2.1.1"). e of the counter (in counter-set,

Applying roman numerals styling to the counter output

CSS:

```
body {
 counter-reset: item-counter;
}
.item {
 counter-increment: item-counter;
}
.item:before {
 content: counter(item-counter, upper-roman) ". "; /*
by specifying the upper-roman as style the
output would be in roman numbers */
}
```

HTML

```
<div class='item'>Item No: 1</div>
<div class='item'>Item No: 2</div>
<div class='item'>Item No: 3</div>
```

- In the above example, the counter's output would be displayed as I, II, III (roman numbers) instead of the usual 1, 2, 3 as the developer has explicitly specified the counter's style

Number each item using CSS Counter

CSS

```
body {
 counter-reset: item-counter; /* create the counter */
}
.item {
 counter-increment: item-counter; /* increment the
counter every time an element with class "item"
is encountered */
}
.item-header:before {
 content: counter(item-counter) ". "; /* print the value of
the counter before the header and
append a "." to it */
}
/* just for demo */
.item {
 border: 1px solid;
 height: 100px;
 margin-bottom: 10px;
}
```

```
.item-header {
 border-bottom: 1px solid;
 height: 40px;
 line-height: 40px;
 padding: 5px;
}
.item-content {
 padding: 8px;
}
```

HTML

```
<div class='item'>
 <div class='item-header'>Item 1 Header</div>
 <div class='item-content'>Lorem Ipsum Dolor Sit
Amet....</div>
</div>
```

```
<div class='item'>
 <div class='item-header'>Item 2 Header</div>
 <div class='item-content'>Lorem Ipsum Dolor Sit
Amet....</div>
</div>
<div class='item'>
 <div class='item-header'>Item 3 Header</div>
 <div class='item-content'>Lorem Ipsum Dolor Sit
Amet....</div>
</div>
```

Implementing multi-level numbering using CSS Counters:

CSS

```
ul {
 list-style: none;
counter-reset: list-item-number; /* self nesting counter as
name is same for all levels */
}
li {
 counter-increment: list-item-number;
}
li:before {
 content: counters(list-item-number, ".") " "; /* usage of
counters() function means value of
counters at all higher levels are combined before
printing */
}
```

HTML

```
<ul>
 <li>Level 1
```

```
<ul>
<li>Level 1.1
<ul>
<li>Level 1.1.1</li>
</ul>
</li>
</ul>
```

Functions

calc() function

- Accepts a mathematical expression and returns a numerical value.
- It is especially useful when working with different types of units (e.g. subtracting a px value from a percentage) tocalculate the value of an attribut

- +, -, /, and * operators can all be used, and parentheses can be added to specify the order of operations if necessary.
- Use calc() to calculate the width of a div element:

```
#div1 {
 position: absolute;
 left: 50px;
 width: calc(100% - 100px);
 border: 1px solid black;
 background-color: yellow;
 padding: 5px;
 text-align: center;
}
```

Use calc() to determine the position of a background-image:

background-position: calc(50% + 17px) calc(50% + 10px), 50% 50%;

Use calc() to determine the height of an element:

height: calc(100% - 20px);

attr() function

- Returns the value of an attribute of the selected element.
- Below is a blockquote element which contains a character inside a data-* attribute which CSS can use (e.g. inside the ::before and ::after pseudo-element) using this function.

<blockquote data-mark=""></blockquote>

In the following CSS block, the character is appended before and after the text inside the element:

```
 blockquote[data-mark]::before,
 blockquote[data-mark]::after {
  content: attr(data-mark);
 }
```

var() function

The var() function allows CSS variables to be accessed.

```
/* set a variable */
:root {
 --primary-color: blue;
}
/* access variable */
selector {
 color: var(--primary-color);
}
```

radial-gradient() function:

- Creates an image representing a gradient of colors radiating from the center of the gradient
- radial-gradient(red, orange, yellow) /*A gradient coming out from the middle of the gradient, red at the center, then orange, until it is finally yellow at the edges*/

linear-gradient() function:

- Creates a image representing a linear gradient of colors.

linear-gradient(0deg, red, yellow 50%, blue);

- This creates a gradient going from bottom to top, with colors starting at red, then yellow at 50%, and finishing in blue.

Custom Properties (Variables)

- CSS Variables allow authors to create reusable values which can be used throughout a CSS document.
- For example, it's common in CSS to reuse a single color throughout a document.
- Prior to CSS Variables this would mean reusing the same color value many times throughout a document.
- With CSS Variables the color value can be
- assigned to a variable and referenced in multiple places.
- This makes changing values easier and is more semantic than using traditional CSS values.

Variable Color

```
:root {
 --red: #b00;
 --blue: #4679bd;
 --grey: #ddd;
}
.Bx1 {
 color: var(--red);
 background: var(--grey);
 border: 1px solid var(--red);
}
```

Variable Dimensions

```
:root {
 --W200: 200px;
 --W10: 10px;
}
.Bx2 {
 width: var(--W200);
 height: var(--W200);
 margin: var(--W10);
}
```

Variable Color

```
:root {
 --red: #b00;
 --blue: #4679bd;
 --grey: #ddd;
}
.Bx1 {
 color: var(--red);
background: var(--grey);
 border: 1px solid var(--red);
}
```

Variable Dimensions

```
:root {
 --W200: 200px;
 --W10: 10px;
}
.Bx2 {
 width: var(--W200);
 height: var(--W200);
 margin: var(--W10);
}
.button:hover {
 --color: blue;
```

}
.button_red {
 --color: red;
}

Valid/Invalids

- Naming When naming CSS variables, it contains only letters and dashes just like other CSS properties (eg: lineheight, -moz-box-sizing) but it should start with double dashes (--)

//These are Invalids variable names
--123color: blue;
--#color: red;
--bg_color: yellow
--$width: 100px;

//Valid variable names

--color: red;

--bg-color: yellow

--width: 100px;

CSS Variables are case sensitive.

/* The variable names below are all different variables */

--pcolor: ;

--Pcolor: ;

--pColor: ;

Empty Vs Space

/* Invalid */

 --color:;

/* Valid */

```
--color: ; /* space is assigned */
```

Concatenations

```
/* Invalid - CSS doesn't support concatenation*/
.logo{
--logo-url: 'logo';
background: url('assets/img/' var(--logo-url) '.png');
}
/* Invalid - CSS bug */
.logo{
--logo-url: 'assets/img/logo.png';
background: url(var(--logo-url));
}
```

/* Valid */

```
.logo{
```

--logo-url: url('assets/img/logo.png');
background: var(--logo-url);
}

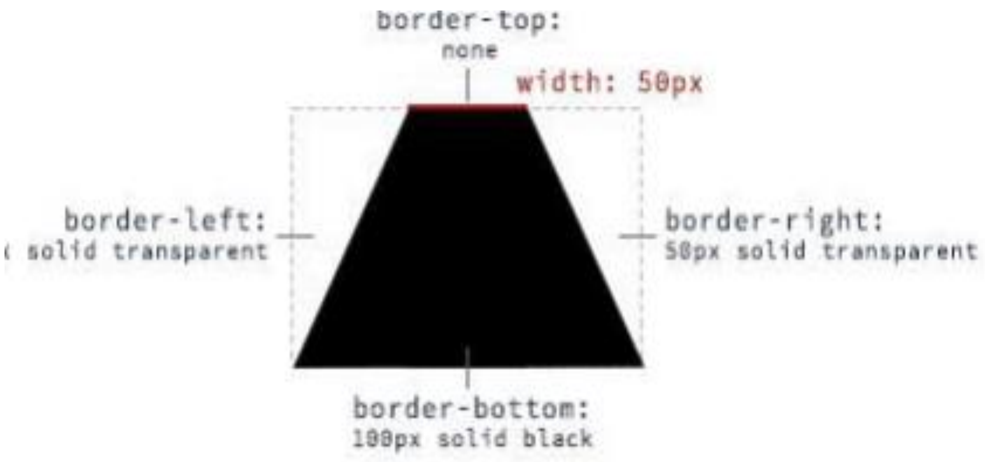
Careful when using Units

/* Invalid */
--width: 10;
width: var(--width)px;
/* Valid */
--width: 10px;
width: var(--width);
/* Valid */
--width: 10;
width: calc(1px * var(--width)); /* multiply by 1 unit to convert */
width: calc(1em * var(--width));

Single Element Shapes:

Trapezoid:

- A trapezoid can be made by a block element with zero height (height of 0px), a width greater than zero and a border, that is transparent except for one side:



L:

HTML:

```
<div class="trapezoid"></div>
```

CSS:

```
.trapezoid {
 width: 50px;
 height: 0;
 border-left: 50px solid transparent;
 border-right: 50px solid transparent;
 border-bottom: 100px solid black;
}
```

- With changing the border sides, the orientation of the trapezoid can be adjusted.

Triangle - Pointing Up

```
<div class="triangle-up"></div>
```

CSS

```
.triangle-up {
 width: 0;
 height: 0;
 border-left: 25px solid transparent;
 border-right: 25px solid transparent;
 border-bottom: 50px solid rgb(246, 156, 85);
}
```

Circles and Ellipses

Circle

- To create a circle, define an element with an equal width and height (a square) and then set the border-radius property of this element to 50%.

HTML

```
<div class="circle"></div>
```

CSS

```
.circle {
 width: 50px;
 height: 50px;
 background: rgb(246, 156, 85);
 border-radius: 50%;
}
```

Bursts

- A burst is similar to a star but with the points extending less distance from the body.
- Think of a burst shape as a

square with additional, slightly rotated, squares layered on top.

- The additional squares are created using the ::before and ::after psuedo-elements.

8 Point Burst

- An 8 point burst are 2 layered squares.
- The bottom square is the element itself, the additional square is created using the :before pseudo-element. The bottom is rotated 20°, the top square is rotated 135°.

Cube

- This example shows how to create a cube using 2D transformation methods skewX() and skewY() on pseudo elements.

HTML:

<div class="cube"></div>

CSS:

```
.cube {
 background: #dc2e2e;
 width: 100px;
 height: 100px;
 position: relative;
 margin: 50px;
}
.cube::before {
 content: '';
  display: inline-block;
  background: #f15757;
  width: 100px;
  height: 20px;
  transform: skewX(-40deg);
position: absolute;
 top: -20px;
 left: 8px;
}
.cube::after {
 content: '';
 display: inline-block;
 background: #9e1515;
 width: 16px;
 height: 100px;
 transform: skewY(-50deg);
 position: absolute;
 top: -10px;
 left: 100%;
}
```

```
 background: #C52329;
 box-shadow: inset 0 0 0 5px #333;
 transform-origin: 50% 50%;
 transform: rotateX(90deg);
 backface-visibility: hidden;
}
.face2 {
 transform-origin: 50% 50%;
 transform: rotatez(90deg) translateX(100%)
rotateY(90deg);
}
.cubeFace:before, .cubeFace:after {
 content: '';
 position: absolute;
```

```
width: 100%;
 height: 100%;
 transform-origin: 0 0;
 background: inherit;
 box-shadow: inherit;
 backface-visibility: inherit;
}
.cubeFace:before {
 top: 100%;
 left: 0;
 transform: rotateX(-90deg);
}.cubeFace:after {
 top: 0;
 left: 100%;
 transform: rotateY(90deg);
}
```

Pyramid

- This example shows how to create a pyramid using borders and 2D transformation methods skewY() and rotate() on pseudo elements

HTML:

```
<div class="pyramid"></div>
```

CSS:

```
.pyramid {
 width: 100px;
 height: 200px;
 position: relative;
 margin: 50px;
}
.pyramid::before,.pyramid::after {
```

```
content: '';
 display: inline-block;
 width: 0;
 height: 0;
 border: 50px solid;
 position: absolute;
}.pyramid::before {
 border-color: transparent transparent #ff5656
transparent;
 transform: scaleY(2) skewY(-40deg) rotate(45deg);
}
.pyramid::after {
 border-color: transparent transparent #d64444
transparent;
 transform: scaleY(2) skewY(40deg) rotate(-45de
}
```

Column:

Simple Example (column-count)

- The CSS multi-column layout makes it easy to create multiple columns of text.

Code

```
<div id="multi-columns">Lorem ipsum dolor sit amet,
consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu
fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est
laborum</div>
```

```
.multi-columns {
 -moz-column-count: 2;
 -webkit-column-count: 2;
 column-count: 2;
}
```

Inline-Block Layout

Justified navigation bar

- The horizontally justified navigation (menu) bar has some number of items that should be justified.
- The first (left) item has no left margin within the container, the last (right) item has no right margin within the container.
- The distance between items is equal, independent on the individual item width.

HTML

```
<nav>
 <ul>
 <li>abc</li>
 <li>abcdefghijkl</li>
 <li>abcdef</li>
 </ul>
</nav>
```

CSS

```
nav {
 width: 100%;
 line-height: 1.4em;
}
```

```
ul {
 list-style: none;
 display: block;
 width: 100%;
 margin: 0;
 padding: 0;
 text-align: justify;
 margin-bottom: -1.4em;
}ul:after {
 content: "";
 display: inline-block;
 width: 100%;
}
li {
 display: inline-block; }
```

Notes

- The nav, ul and li tags were chosen for their semantic meaning of 'a list of navigation (menu) items'. Other tags may also be used of course.
- The :after pseudo-element causes an extra 'line' in the ul and thus an extra, empty height of this block,

pushing other content down.

- This is solved by the negative margin-bottom, which has to have the same magnitude as the line-height (but negative).
- If the page becomes too narrow for all the items to fit, the items will break to a new line (starting from the right) and be justified on this line.
- The total height of the menu will grow as needed.

CSS Image Sprites:

A Basic Implementation:

What's an image sprite?

- An image sprite is a single asset located within an image sprite sheet.
- An image sprite sheet is an image file that

contains more than one asset that can be extracted from it.

For example:

- The image above is an image sprite sheet, and each one of those stars is a sprite within the sprite sheet.
- These sprite sheets are useful because they improve performance by reducing the number of HTTP requests a browser might have to make.

HTML

```
<div class="icon icon1"></div>
<div class="icon icon2"></div>
<div class="icon icon3"></div>
```

CSS

```
.icon {
 background: url("icons-sprite.png");
 display: inline-block;
height: 20px;
 width: 20px;
}
.icon1 {
 background-position: 0px 0px;
}
.icon2 {
 background-position: -20px 0px;
}
.icon3 {
 background-position: -40px 0px;
}
```

- By using setting the sprite's width and height and by using the background-position property in CSS (with an x and y value) you can easily extract sprites from a sprite sheet using CSS.

Summary:

- Shapes for Floats
- List Styles
- Counters
- Functions
- Single Element Shapes
- Columns
- Inline-Block Layout
- CSS Image Sprites

Thank You………

If you have any quries please write to   info@uplatz.com".