Customizing lists

This post will a look at different ways to customize HTML lists, both ordered and unordered using CSS.

As with anything on the web, you should test these techniques in your target browsers as well as with assistive technologies.

There is a possibility that using pseudo-elements created with `::before` may not work with some assistive technology and using Unicode characters representing images may have accessibility implications.

That said, let’s look at some of these ideas.

Text-based list items

You can change the display of bullet items using the list-style-type property.

There are three things that we can do with this attribute:

1. We can copy the Unicode character directly
2. We can use escaped entities for the Unicode character
3. We can choose to remove the bullet for the list item altogether

```
/* 1 */
ul {
  list-style-type: "✳";
}

"✳"/* 1 */
ul  list-style-type: "🚀";
}

/* "🚀"/* 2 */
ulst-style-type: "\2734";
}

/* 3 */"\2734"/* 3 */
```

```
ul { list-style-type: none; }
```

Styling bulleted list markers

In addition to (or instead of) adding custom bullet markers, we can style the markers using the ::marker pseudo-element.

> The following block of code will color all the bulleted list markers magenta.

```
ul li::marker {
  color: magenta;
}
```

The `::marker` pseudo element only accepts some properties. The allowed properties include:

- All font properties
- The white-space property
- color
- text-combine-upright, unicode-bidi and direction properties
- The content property
- All animation properties
- All transition properties

Image list items

In addition to textual representatins, we can use images for our bulleted list markers using the list-style-image and the url() function to reference the image from a local or remote source.

```
li {
  list-style-image: url("/images/arrow-right-svgrepo-com.svg");
}

li {
  list-style-image: url("https://example.com/path/to/arrow-right-svgrepo-c
}
```

As far as I can understand there’s no way to style the bullet without changing the technique to using background images and pseudo-elements created with `::before`.

```
.image-list {
  list-style-type: none;
  padding: 0;
  margin: 0;
}

.image-list li {
  margin: 0;
  padding: 0 30px 5px 35px;
  list-style: none;
  background-image: url("https://example.com/path/to/arrow-right-svgrepo-c

  background-repeat: no-repeat;
  background-position: left center;
  background-size: 25px;
}
```

If you want to use this technique, it is imperative that you test it on your target browsers. While working on it, I had a very hard time getting the image bullets to line up with the text of the line item.

Nested Counters

The work I’ve done with counters in the past has involved a single counter at a time. However there are times when we might want to work with more than one counter at a time. For example, we may want to associate more than one counter with a given item.

For example, we may want to use something like 1.1 or 1.2.1 in our ordered lists, figures, tables, and other sequentially numbered content.

This will require a different way of thinking about the counters. Rather than using a series of `counter` attributes, we will use a single counters function that will return a concatenated strings with all the parent counters.

The idea is as follows:

1. At every `ol` we create a new `list-counter` counter by resetting it (even if it doesn't exist)
2. Every `li` element increases the value of the `list-counter` counter at the current level
3. We use the `counters` function to collect all the values for the `list-counter` counter and display them with `.` as the separator
4. We also do some styling to separate the counter from the text of the item

```
ol {
  list-style-type: none;
  counter-reset: list-counter; /* 1 */
}

li {
  counter-increment: list-counter; /* 2 */
}

li::before {
  content: counters(list-counter, "."); "."/* 3 */
  margin-right: 10px; /* 4 */
  font-size:16px;
}
```

Setting starting point and reversing the numbers

When you reset a counter, you can specify the starting number for the counter (the default is 0 so the first increment will be 1)

The following code will generate an ordered list starting with at 11 and incrementing by one with each following element.

```
ol {
    counter-reset: list-counter 10;
    list-style-type: none;
}

li {
  counter-increment: list-counter;
}

li::before {
  content: counter(list-counter);
  margin: 0 1em 0 0;
}
```

The specification also privides for a way to create reverse counting, starting with the total number of items and decrementing by one on each iteration.

So far, this is only supported in Firefox so we have to code defensively using the @supports at-rule.

The first block checks if the browser **doesn’t** support reverse lists. If it doesn’t then we insert text telling them so in the example. This is where we would also provide equivalent functionality if possible.

```
@supports not (counter-reset: reversed(list-counter)) {
  #warning::after {
    content: "Browser doesn't support reversed counters";
  }
}
"Browser doesn't support reversed counters"
```

Then we check if the browser **does** support reversed counters and run the code if it does.

The code does mostly the same as previous examples. The main differences are:

- The `counter-reset` uses the `reversed` keyword to indicate that the

counter should start from the total number of items in the list
- The `counter-increment` is actually a reduction. It increments by `-1`

```css
@supports (counter-reset: reversed(list-counter)) {
  ol {
    counter-reset: reversed(list-counter);
    list-style-type: none;
  }

  li {
    counter-increment: list-counter -1;
  }

  li::before {
    content: counter(list-counter);
    margin-right: 1em;
  }
}
```

In Firefox, the only browser that currently supports reverse counters, the counter will decrease the value for each item in the list.

In browsers that don't support reverse counters, the list will be displayed as negative numbers, increasing with each item on the list.

Predefined counter styles

CSS Counter Styles Level 3 provides a set of predefined counters that include things like upper and lower case Roman numbers, arabic numbers zero-padded and language-specific letters.

We can use these prefefined counters set as the second parameter to the `counter` or `counters` functions.

In the following example we use a `list-counter` list defined elsewhere and the `decimal-leading-zero` style to produce a list that will be zero-padded (01, 02, 03, etc).

The code below is formatter for clarity, the parameters for `counter` should all be in a single line.

```
ol li::before {
  content: counter(
    list-counter,
    decimal-leading-zero
  );
}
```

The full list of ready-made counter styles is located in the Ready-made Counter Styles from the CSS Working Group

Custom counter styles

CSS Counter Styles Level 3 defines the @counter-style at-rule to allow developers to create custom counter styles.

The first example in this section creates a `thumbs` custom style that uses the Thumbs Up Sign Emoji (👍), Thumbs Down Sign Emoji (👎), and Fisted Hand Sign Emoji (👊) as alternating bullet symbols in an unordered list

```
@counter-style thumbs {
  system: cyclic;
  symbols: "\1F44D""\1F44E""\1F4"\1F44D"uffix: "   ";
}

"  "ul {
  list-style: thumbs;
  font-size: 2em;
}
```

You can see the result in the following Codepen.

A slightly less silly hexadecimal example that will run the numbers from zero to nine and then the letters a through f to get the full 16 digits of the hexadecimal numbering system.

The code below has been formatted for clarity. All the values in `symbols` should be in a single line.

The code below is formatter for clarity, the parameters for `counter` should all be in a single line.

```
@counter-style lower-hexadecimal {
system: numeric;
symbols:
'\30'
  '\31'
  '\32'
  '\33'
  '\34'
  '\35'
  '\36'
  '\37'
  '\38'
  '\39'
  '\61'
  '\62'
  '\63'
  '\64'
  '\65'
  '\66';
}
```

With this little bit of information, browsers will be able to show you any number of characters. They are smart enough to know that the hexadecimal value for 26 is 1a and that the hexadecimal value for 32 is 20.

The Codepen below shows a list of twenty items using our lower-case hexadecimal custom counter style.

Example: Smashing Magazine

I've always loved the way that Smashing Magazine does their numbered lists.

The main difference is that the counter we use in `ol li::before` has two parameters, the counter we want to use and the style for the counter.

```
ol {
  counter-reset: list-counter;
}

ol li {
  padding-left: 2em;
  counter-increment: list-counter;
  margin-bottom: 1.1em;
  list-style-type: none;
}

ol li::before {
  content: counter(list-counter, decimal-leading-zero);
  font-weight: 700;
  color: rebeccapurple;
  margin: 0 2em 0 0 ;
}
```

The final result is show in the Codepen below