Chapter 35: 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 value of the counter (in `counter-set`, `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 `counter-increment`, 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").

Section 35.1: 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.

Section 35.2: 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>
```

The above example numbers every "item" in the page and adds the item's number before its header (using `content` property of `.item-header` element's `:before` pseudo). A live demo of this code is available here.

Section 35.3: 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>
```

```
  </li>
  <li>Level  2
    <ul>
      <li>Level 2.1
        <ul>
          <li>Level 2.1.1</li>
          <li>Level 2.1.2</li>
        </ul>
      </li>
    </ul>
  </li>
  <li>Level 3</li>
</ul>
```

The above is an example of multi-level numbering using CSS counters. It makes use of the ***self-nesting*** concept of counters. Self nesting is a concept where if an element already has a counter with the given name but is having to create another then it creates it as a child of the existing counter. Here, the second level `ul` already inherits the `list-item-number` counter from its parent but then has to create its own `list-item-number` (for its children `li`) and so creates `list-item-number[1]` (counter for second level) and nests it under `list-item-number[0]` (counter for first level). Thus it achieves the multi-level numbering.

The output is printed using the `counters()` function instead of the `counter()` function because the `counters()` function is designed to prefix the value of all higher level counters (parent) when printing the output.