On the importance of CSS classes in WordPress theme development

While developing a simple Wordpress theme, I was struck (literally) by the fact that the documentation on CSS and Wordpress is lacking of some of the most essential parts of CSS development. One question really needs an answer: how can CSS work together with a dynamic environment like Wordpress? CSS is a static language because it was designed when the web was made up of static HTML pages. CSS has no conditional statements, if we don't consider the newest features of `@media` rules as conditional constructs. Let's try to answer to this question.

Classes and code reusability

Reusability is the key to keep our code organized and easy to maintain. When dealing with CSS class names, two are the points to keep in mind:

1. semantics of class names
2. cascade and specificity

Semantics of class names

You should avoid presentational class names, such as `bold` or `red`. Instead, use descriptive names that specify what is the purpose of a particular class, such as `evident` or `highlight`.

Cascade and specificity

Classes can be combined together to form cumulative styles. For example, suppose that you have the following class:

```
.highlight {
  background-color: #ffc;
  font-weight: bold;
}
```

Then you can specify another class in your stylesheet:

```
.note {
  background-image: url("note.png");
  background-repeat: no-repeat;
  height: 100%;  /* IE6/7 hasLayout */
  padding: 1.2em;
}
```

Finally, you can combine them in your markup:

```
<div class="note highlight">..</div>
```

The final result is an element whose styles are the sum of the rules specified in the two classes. Of course you can use a single class or both classes on different elements, because we didn't specify an element type on our class selectors. Only remember to avoid clashing style rules, because if you do so, then the rules for specificity will be applied.

Another digression on multiple classes

Suppose that you have some floated images. You want that some of these images show a doubled border and in this case there are two classes that can be joined together:

```
.alignleft {
  float: left;
  margin-right: 5px;
}

.picture {
    padding: 5px;
    border: 5px double #aaa;
}
```

Then you add the above classes on an `img` element, like so:

```
<img src="pic.gif" alt="Picture" class="alignleft
picture" />
```

However, there is an important thing to keep in mind when using a solution like this: you should always specify style rules that don't conflict with each other. If you don't do so, then the cascading rules will decide which rule is the winner (e.g. when a class comes first or later in the source or when a rule is followed by an `!important` statement).

Anyway, if you aren't really worried about these issues, then multiple classes can actually be a useful way to apply styles in an elegant, reusable and semantic way.

Return of classitis?

Let's face it: without CSS classes, you can't write a good stylesheet for Wordpress. This happens because a Wordpress theme is not only made up

by simple and predictable page elements, but also by third-party additional gadgets, widgets, plugins and, what's more, user-defined code.

Writing down all the possible use cases is practically impossible. That's why CSS classes are so useful: they allow you to write generalized styles that can be applied to a wide variety of possible scenarios.

Without classes, you cannot handle third-party elements. For example:

```
div.widget {

  margin-top: 1em;
  margin-bottom: 1em;

}
```

Now let's say that you have a plugin that installs a list of popular posts. You may have the following final markup:

```
<div class="widget custom-widget"></div>
```

You launch your site and you notice that the list is not centered. With classes, you have just to write this:

```
div.custom-widget {
  width: 200px;
  margin-left: auto;
  margin-right: auto;
```

```
}
```

The final result will be a list of links vertically spaced and horizontally centered. The same base class, `widget`, can also be applied to many other custom plugins and widgets.

Classes allow you to add per-page styles. Now you may say: "Wait a minute, and the `body#id` technique?". Well, this technique is surely useful when you have a static site with just a few pages. But what happens when you have dozens pages and you have to choose an ID for each page? This will surely get you mad, because it's something that's quite difficult to handle in a live production environment.

Note

Just for the record, I've created a simple utility function that emulates the behavior of the default `body_class()` WordPress function. This function generates IDs instead of classes. You can find it on GitHub.

So you can write something like this:

```
  <body class="home alternate-banner xmas"></body>
```

that can be used to add some special styles just for the Christmas time:

```
body.xmas {
  background-color: #c00;
}

body.alternate-banner.xmas #header {
```

```
  background: url(images/santa-claus.jpg) no-repeat;
}
```

To sum up: classes are not bad when you need them. They're bad only when you have a simpler choice.