Introduction

On smaller brochure sites, how you organize your styles isn’t usually a big concern. You get in there, write some CSS, or maybe even some SASS. You compile it all into a single stylesheet with SASS’s production settings, and then you aggregate it to get all the stylesheets from modules into a nice tidy package.

However, when it comes to larger, more complex projects, how you organize your code is the key to efficiency in at least these three ways: it affects how long it takes you to write code, how much of that code you’ll have to write and how much loading your browser will have to do. This becomes especially important when you’re working with teams of themers, and when high performance is essential.

This is also true for long-term projects with legacy code (read ["How to Scale and Maintain Legacy CSS with Sass and SMACSS"](http://webuild.envato.com/blog/how-to-scale-and-maintain-legacy-css-with-sass-and-smacss/) — some nice SMACSS and BEM mixing in there).

Methodologies

There are plenty of [methodologies](https://github.com/ikkou/awesome-css#architecture) out there aiming to reduce the CSS footprint, organize cooperation among programmers and maintain large CSS codebases. This is obvious in large projects like Twitter, Facebook and [Github](http://markdotto.com/2014/07/23/githubs-css/#two-bundles), but other projects often grow into some “Huge CSS file” state pretty quickly.

[OOCSS](http://oocss.org/)

Separating container and content with CSS “objects”

[SMACSS](http://smacss.com/)

Style-guide to write your CSS with five categories for CSS rules

[SUITCSS](http://suitcss.github.io/)

Structured class names and meaningful hyphens

[Atomic](http://github.com/nemophrost/atomic-css)

Breaking down styles into atomic, or indivisible, pieces

Why BEM over the others?

No matter what methodology you choose to use in your projects, you will benefit from the advantages of more structured CSS and UI. Some styles are less strict and more flexible, while others are easier to understand and adapt in a team.

The reason I choose BEM over other methodologies comes down to this: it is less confusing than the other methods (i.e. SMACSS) but still provides us the good architecture we want (i.e. OOCSS) and with a recognizable terminology.

Mark McDonnell, [Maintainable CSS with BEM](http://www.integralist.co.uk/posts/bem.html#4)

Blocks, Elements and Modifiers

You will not be surprised to hear that BEM is an abbreviation of the key elements of the methodology — Block, Element and Modifier. BEM’s strict naming rules can be found [here](http://getbem.com/naming/).

Block

Standalone entity that is meaningful on its own.

Examples

header, container, menu, checkbox, input

Element

A part of a block that has no standalone meaning and is semantically tied to its block.

Examples

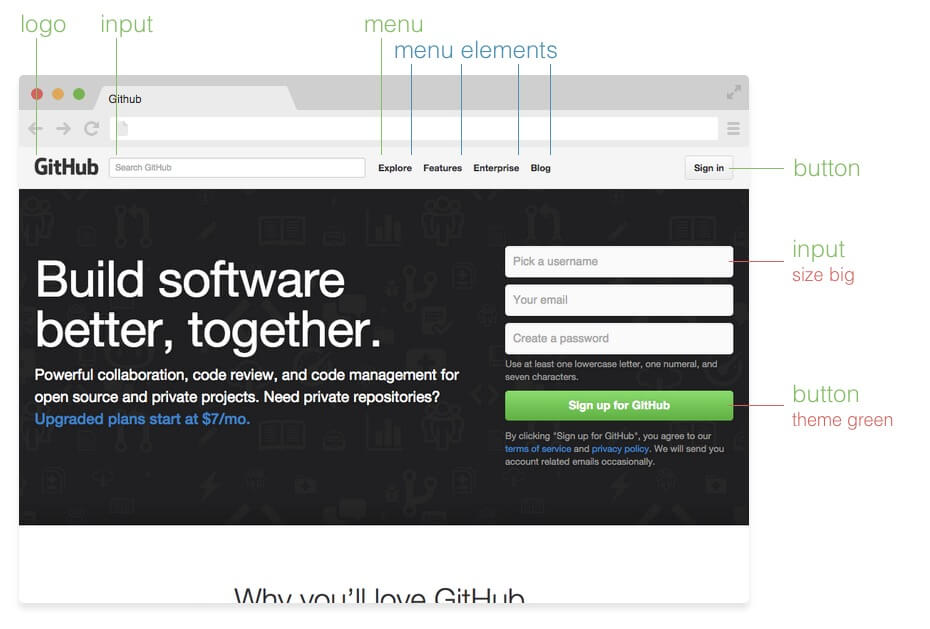
menu item, list item, checkbox caption, header title

Modifier

A flag on a block or element. Use them to change appearance or behavior.

Examples

disabled, highlighted, checked, fixed, size big, color yellow



Under the hood

Let’s look how one particular element on a page can be implemented in BEM. We will take button from [GitHub](http://primercss.io/buttons/):

http://getbem.com/assets/github_buttons.jpg

We can have a normal button for usual cases, and two more states for different ones. Because we style blocks by class selectors with BEM, we can implement them using any tags we want (button, a or even div). The naming rules tell us to use block--modifier-value syntax.

HTML

<button class="button">

Normal button

</button>

<button class="button button--state-success">

Success button

</button>

<button class="button button--state-danger">

Danger button

</button>

CSS

.button {

display: inline-block;

border-radius: 3px;

padding: 7px 12px;

border: 1px solid #D5D5D5;

background-image: linear-gradient(#EEE, #DDD);

font: 700 13px/18px Helvetica, arial;

}

.button--state-success {

color: #FFF;

background: #569E3D linear-gradient(#79D858, #569E3D) repeat-x;

border-color: #4A993E;

}

.button--state-danger {

color: #900;

}

Benefits

Modularity

Block styles are never dependent on other elements on a page, so you will never experience [problems from cascading](http://www.phase2technology.com/blog/used-and-abused-css-inheritance-and-our-misuse-of-the-cascade/).

You also get the ability to transfer blocks from your finished projects to new ones.

Reusability

Composing independent blocks in different ways, and reusing them intelligently, reduces the amount of CSS code that you will have to maintain.

With a set of style guidelines in place, you can build a library of blocks, making your CSS super effective.

Structure

BEM methodology gives your CSS code a solid structure that remains simple and easy to understand.

Further Reading

* [‘Why BEM?’ in a nutshell](http://blog.decaf.de/2015/06/24/why-bem-in-a-nutshell/)
* [MindBEMding](http://csswizardry.com/2013/01/mindbemding-getting-your-head-round-bem-syntax/) — getting your head ’round BEM syntax
* [CSS guidelines](http://cssguidelin.es/#bem-like-naming)
* [BEM methodology for small projects](http://www.smashingmagazine.com/2014/07/17/bem-methodology-for-small-projects/)
* [BEM It! for Brandwatch](http://www.slideshare.net/MaxShirshin/bem-it-for-brandwatch)
* [Used and Abused](http://www.phase2technology.com/blog/used-and-abused-css-inheritance-and-our-misuse-of-the-cascade/) — CSS Inheritance and Our Misuse of the Cascade.
* [Objects in Space](https://medium.com/objects-in-space/objects-in-space-f6f404727) — A style-guide for modular SASS development using SMACSS and BEM
* [How to Scale and Maintain Legacy CSS with Sass and SMACSS](http://webuild.envato.com/blog/how-to-scale-and-maintain-legacy-css-with-sass-and-smacss/)
* [Building a modular My Health Skills with BEM and Sass](http://www.bluegg.co.uk/building-my-health-skills-part-3/)
* [Building My Health Skills — Part 3](http://www.bluegg.co.uk/building-my-health-skills-part-3/)

Case study

We hope to write "How to migrate an existing project to BEM" soon. In the meantime you can watch this nice presentation by Nicole Sullivan — "[CSS preprocessor performance](http://www.youtube.com/watch?v=0NDyopLKE1w)". She gives a very good overview of the problems she encounters in the majority of websites and offers ways to track them down and handle them.

Naming

There are only two hard problems in Computer Science: cache invalidation and naming things — *Phil Karlton*

It is a known fact that the right styleguide can significantly increase development speed, debugging, and the implementation of new features in legacy code. Sadly, most CSS codebases are sometimes developed without any structure or naming conventions. This leads to an unmaintainable CSS codebase in the long term.

The BEM approach ensures that everyone who participates in the development of a website works with a single codebase and speaks the same language. Using proper naming will prepare you for the changes in design of the website.

Block

Encapsulates a standalone entity that is meaningful on its own. While blocks can be nested and interact with each other, semantically they remain equal; there is no precedence or hierarchy. Holistic entities without DOM representation (such as controllers or models) can be blocks as well.

Naming

Block names may consist of Latin letters, digits, and dashes. To form a CSS class, add a short prefix for namespacing: .block

HTML

Any DOM node can be a block if it accepts a class name.

<div class="block">...</div>

CSS

* Use class name selector only
* No tag name or ids
* No dependency on other blocks/elements on a page

.block { color: #042; }

Element

Parts of a block and have no standalone meaning. Any element is semantically tied to its block.

Naming

Element names may consist of Latin letters, digits, dashes and underscores. CSS class is formed as block name plus two underscores plus element name: .block\_\_elem

HTML

Any DOM node within a block can be an element. Within a given block, all elements are semantically equal.

<div class="block">

...

<span class="block\_\_elem"></span>

</div>

CSS

* Use class name selector only
* No tag name or ids
* No dependency on other blocks/elements on a page

**Good**

.block\_\_elem { color: #042; }

**Bad**

.block .block\_\_elem { color: #042; }

div.block\_\_elem { color: #042; }

Modifier

Flags on blocks or elements. Use them to change appearance, behavior or state.

Naming

Modifier names may consist of Latin letters, digits, dashes and underscores. CSS class is formed as block’s or element’s name plus two dashes: .block--mod or .block\_\_elem--mod and .block--color-black with .block--color-red. Spaces in complicated modifiers are replaced by dash.

HTML

Modifier is an extra class name which you add to a block/element DOM node. Add modifier classes only to blocks/elements they modify, and keep the original class:

**Good**

<div class="block block--mod">...</div>

<div class="block block--size-big

block--shadow-yes">...</div>

**Bad**

<div class="block--mod">...</div>

CSS

Use modifier class name as selector:

.block--hidden { }

To alter elements based on a block-level modifier:

.block--mod .block\_\_elem { }

Element modifier:

.block\_\_elem--mod { }

Example

Suppose you have block form with modifiers theme: "xmas" and simple: true and with elements input and submit, and element submit with its own modifier disabled: true for not submitting form while it is not filled:

HTML

<form class="form form--theme-xmas form--simple">

<input class="form\_\_input" type="text" />

<input

class="form\_\_submit form\_\_submit--disabled"

type="submit" />

</form>

CSS

.form { }

.form--theme-xmas { }

.form--simple { }

.form\_\_input { }

.form\_\_submit { }

.form\_\_submit--disabled { }

FAQ

These Frequently Asked Question are real questions of developers started with BEM, answered by the GetBEM community. Feel free to [ask your question](https://github.com/getbem/getbem.com/issues/new?title=FAQ:%20Type%20your%20question%20here) too, and we will answer it as well.

* [Why should I choose BEM and not another CSS modular solution?](http://getbem.com/faq/#why-bem)
* [Why are the modifier CSS classes not represented as a combined selector?](http://getbem.com/faq/#why-the-modifier-classes-are-prefixed)
* [Why do I need CSS classes for block instead of using semantic custom tags?](http://getbem.com/faq/#custom-tags-for-blocks)
* [Why do I need to combine block and prefixed modifier class for a modified block?](http://getbem.com/faq/#block-modifier-mix)
* [Can a block modifier affect elements?](http://getbem.com/faq/#block-modifier-affects-elements)
* [Can I create a global modifier applicable to any block?](http://getbem.com/faq/#can-i-create-global-modifier)
* [Can I combine a tag and a class in selector like button.button?](http://getbem.com/faq/#encapsulating-tag-selector)
* [Is this good to name modifiers corresponding to what they have in CSS? Like .block\_\_element--border-bottom-5px.](http://getbem.com/faq/#css-modifier-names)
* [What would be a class name for an element inside another element? .block\_\_el1\_\_el2?](http://getbem.com/faq/#css-nested-elements)
* [I've heard that BEM does not recommend global CSS resets? Why?](http://getbem.com/faq/#global-css-resets)
* [Did not find the answer?](http://getbem.com/faq/#ask-your-question)

Why should I choose BEM as a modular solution for CSS?

There are some other modular solutions for CSS (such as OOCSS, AMCSS, SMACSS, SUITCSS). What are the reasons to choose BEM?

BEM provides solutions for all the frontend technologies: CSS, JavaScript, templating; and also for building process of your web application. The methodology is applicable anywhere. However, to apply this in JavaScript and templating you would need special frameworks whereas in CSS you may just follow the methodological recommendations. The CSS part of BEM is the easiest to take into your development process. This is why many use only it. On the other hand, if lately you would found your project fully BEMed (in CSS) and yourself happy for its grown maintenance, you probably would take next step in modularizing your web application. BEM CSS will be easier to coordinate with modular JavaScript and blocks-based project file structure.

If speaking about CSS modular solutions only, the key feature of BEM is block's independence. Following the CSS recommendations enables to put a block into any place on a page and be sure that is won't be affected by its surroundings. Also, if you would lately need to nest another block into the current one, their full compatibility is guaranteed. In other words, when maintaining your web application you would be able to move blocks across the page, add others and combine them.

BEM CSS unambiguously defines which CSS belongs to a piece of interface and so using it gives answers to questions "Can I remove this piece of code?" and "What happens and which interface parts will be affected if I change this piece of code?".

Why the modifier CSS classes are not represented as a combined selector?

BEM recommends to modify blocks like this <div class="block block--mod">. Why not to use the simple version like <div class="block mod">? Since we now have combined selectors .block.mod, it's easy to define all the CSS properties to it.

The recommendation to prefix modifier CSS class with its block name has multiple reasons.

Firstly, since it is possible to mix several blocks and elements at the same DOM node, we need to ensure that a modifier would affect only the block it belongs to. Let's say that we have a menu item element and a button mixed together. In HTML this construction is represented by the following markup:

<div class="menu\_\_item button"></div>

In this case adding .active modifier to them would affect both.

<div class="menu\_\_item button active"></div>

All the 3 sit at the same DOM node, so it is impossible to differentiate if we mean menu\_\_item.active or button.active. Whereas in the prefixed case the naming button--active unambiguously says as that this is only the button that has to be affected.

Another point is CSS specificity. The combined selectors are more specific (means more important) than single class selectors. This means that you might have trouble when redefining them with parent block code.

<div class="header">

<button class="button active">

</div>

If you already have .button.active selector in your code, the specificity of your redefining .header .button would be exactly the same as the specificity of modifier combined selector which makes you dependent on the order of the CSS rules declared. Whereas if you use a prefixed modifier, you can always be sure that the cascade selector .header .button will overwrite the .button--active modifier.

This makes life easier especially for maintainable projects.

The third point is that looking at the <div class="block block--mod"> markup you can clearly see the block structure. It is easy to recognize that we have a block and its modifier and there is no different interpretations here. Unfortunately a grasp onto <div class="block mod"> code does not give such information. Depending on what are the exact CSS classes sometimes it is impossible to recognize if we have a block and a modifier or a mix of 2 blocks here. This might be even more confusing if the names of the entities are complex or contracted/abbreviated (which sometimes happens in big projects).  
Clear understanding of a block structure is especially helpful when looking for corresponding code on a file system.

You will also appreciate .block--mod practice when refactoring and use global search over all your project files. Imagine the same looking for not-prefixed .mod and all the HTML pieces it might be in.

And lastly, from a development process standpoint the difference between .block.mod and .block--mod is only one symbol. Using - instead of . costs nothing but it brings all the benefits listed above. Moreover, since pre-processor began to support BEM notation, it is pretty natural to write &--mod there and finally get a modifier declared as it was recommended.

Why do I need CSS classes for block instead of using semantic custom tags?

Blocks can be represented as custom tags which we may define CSS rules for. Looks like we do not need CSS classes for blocks at all. They can be used for modifiers only, like <button class="mod"/>.

Using custom tags as block selectors is indeed one of the BEMish solutions and can be used. However this variant is less flexible than the recommended "class" approach.

This is more likely that you would need to prefix modifier classes with their block name to provide them namespace. The details are uncovered in ["Why the modifier CSS classes are prefixed with their parent block name?"](http://getbem.com/faq/#why-the-modifier-classes-are-prefixed) question. So, finally the custom-tag version of a block is like <block class="block--mod"/>. This does not look very different from <div class="block block--mod"> especially assuming that being tag-independent you can use any custom node and stay with <block class="block block--mod">.

Second drawback is that "tag" version makes using the mixes of blocks impossible whereas the "class" version represent that naturally by <div class="block1 block2">.

And the last clench against such an approach is that in many cases you are not able to represent your blocks with custom tags at all. For a link block you definitely need <a> tag, and the same for <input>.

Why do I need to combine block and prefixed modifier class for a modified block?

Why does both block's and modifier's class sit together in the modified block like <div class=”block block--mod”>?

Everything about a modified block can be described in .block--mod. If there is something common between 2 modifiers, it's possible to use preprocessor's mixins to avoid copy-paste.

This approach is possible thanks to preprocessors. However it brings some drawbacks which you should be aware of.

In the case of combining 2 or more modifiers at the same block <div class="block--theme--christmas block--size--big">, you would get the core block's styles twice. However this depends on the preprocessor algorithms.

When adding/removing modifiers dynamically with JavaScript, the additional modifier is more handy. Switching it off would mean only removing one CSS class from the DOM node with no need to add the core block CSS class back as it sits there forever.

Can a block modifier affect elements?

If I have a block modifier, for example xmas, and I want the elements within that block to also be xmas themed, how would it be best to do it.

Does a --xmas suffix for every element seem necessary? Or would this be the one use-case for nesting (e.g. block--xmas block\_\_elem { ... }?)

While in general BEM recommends avoiding nested selectors, in this case they are reasonable.

When creating the nested selector, you declare that one entity depends on another. Because BEM introduces independent components, such an approach is not suggested when we are speaking about 2 different blocks.

But when it comes to a block and its element, they are not of equivalent meaning. By definition, an element does not make any sense outside its parent block. So, an element is a block-dependent entity. Assuming this, it is quite normal and logical that an element is affected by the block's current state.

So, this is a common pattern in BEM to code

.my-block--xmas .my-block\_\_button {

/\* Jingle bells, jingle bells, jingle all the way.\*/

}

Can I create a global modifier applicable to any block?

I've heard that global modifiers like visible, invisible, red, opacity50 are not welcomed in BEM. Why?

I think it is useful to incorporate common properties like this in such a global class and then apply it to different blocks.

Indeed you can have 2 main CSS classes at the same DOM node. In BEM we call it mix:

<div class="block1 block2"></div>

But the important thing about it is that both block1 and block2 should be standalone blocks. This is slightly different from what people usually mean by "global modifiers", as modifiers do not have any sense on their own and are just a set of properties to change.

<div class="block globalmod"></div>

If you think that in your case you would have a global modifier, these are the problems you may face:

First of all the specificity problem appears. In a local modifier case CSS code goes like this:

.block {

display: block;

}

.block--hidden {

display: none;

}

Both block and modifier selectors have the same specificity. As modifier declaration goes after the block, it redefines the CSS properties. These styles belong to block and are stored in the block file. Thus, independently on how the resultant CSS is built from source, you will always have them in this order and be sure that redefining happens.

In the case of global modifier, its properties can be redefined by the blocks if they follow modifiers in code:

.hidden { display: none }

/\* ... \*/

.block { display: block }

<div class="block hidden">you still see me</div>

One of the possible solutions to this problem is to raise the selector specificity of global modifiers by adding !important to them. But in this case any side-effects of such a global modifier might be overwritten only by declarations with the same !important instruction.

Another way is to load global modifier CSS after all the other styles. But in this case you are not able any more to use *lazy loading* strategy for your components. The additional lazy CSS will still be loaded after the modifiers and you get the same problem.

The next problem is combination of several global modifiers at the same block.

<div class="block mod1 mod2"></div>

In this case you absolutely have no control over the block. The order of modifiers in code can be different. If it conflicts with other declarations, changing the order can fix this conflict but lead to another one. The only way would be to redefine the mess in block. And don't forget about the !important to your hack.

Also, depending on a block the same modifier can be implemented differently. Even the simple .hidden sometimes needs to be not display: none but visibility: hidden or even position: absolute; left: -9999px etc. And if you need to bring some changes into your block, it is much nicer not to waste time searching for all the places where this block can be combined with a global modifier. Especially assuming that such dependencies usually are not described anywhere.

All this hell can be avoided by encapsulating a modifier in a block like .block--mod.

Indeed using global modifiers makes the resultant code less. However if you measure the real difference in bytes it usually does not seem that big. Especially if you are using CSS optimizer which can combine selectors.

Can I combine a tag and a class in selector like button.button?

I want to use selectors like button.button to encapsulate my blocks functionality within a particular tag. If lately someone else would use in their code <h2 class="button">, such an encapsulation would prevent a conflict.

The CSS specificity of such a selector grows. .button--mod selector will not overwrite CSS properties of the block, only button.button--mod would work. You will need its modifiers to be combined with the tag as well and so do the developers who lately would redefine your block.

Lately, when a project goes larger, it's very likely that you may have input.button, span.button and a.button as well. And all the prefixed selectors for modifiers and nested elements will need 4 declarations.

So, it is better not to tie your own hands with such prefixing. However if you still can softly-softly ensure that your blocks are used with proper tags if your provide your users with templates for every block. This is the most flexible and automatic solution.

If the templating looks overhead, there is a "documentation" approach to inform your users which tag the block CSS class would be applied to, this can be done with documenting the block code. The shortest version could be just a comment with a tag name prefixing the block declaration /\*button\*/.button. Or that can be a larger comment with full HTML piece needed to the block to function.

Is this good to name modifiers corresponding to what they have in CSS?

Thanks to mixes, we can create a lot of modifiers which represent CSS properties and assign them to blocks. But I've heard that "it is bad". For example, this selector .block\_\_element--border-bottom-5px was stamped as "awful". I am wondering why and how should the modifiers be named then?

Naming the modifiers corresponding to their CSS representation is not recommended. Indeed it looks not very nice but there are also practical reasons against it. Lately then the view of your components is changed, you will need to fix not only CSS but also the selectors. So, when you border is 6px, it would require changes in all the templates and sometimes in JavaScript.

Also, it never happens that a modifier has only one CSS property to define and will have it forever. Even if now it is only border that differentiates one state from another, this is very likely that lately you would need other CSS properties for the same state of your block. This would be messy if you define a background or padding in a modifier called "border". So, it is recommended to choose semantic names for modifiers even if they only have one property by now.

What would be a class name for an element inside another element? .block\_\_el1\_\_el2?

What should I do if my block has a complex structure and its elements are nested? CSS classes like block\_\_elem1\_\_elem2\_\_elem3 look scary.

According to BEM method, block structure should be flattened; you do not need to reflect nested DOM structure of the block. So, the class names for this case would be:

.block {}

.block\_\_elem1 {}

.block\_\_elem2 {}

.block\_\_elem3 {}

Whereas the DOM representation of the block may be nested:

<div class='block'>

<div class='block\_\_elem1'>

<div class='block\_\_elem2'>

<div class='block\_\_elem3'></div>

</div>

</div>

</div>

Besides the fact that the classes look much nicer, it makes the elements be dependent on the block only. So, you can easily move them across the block when providing changes to the interface. The changes of the block DOM structure would not need corresponding changes to the CSS code.

<div class='block'>

<div class='block\_\_elem1'>

<div class='block\_\_elem2'></div>

</div>

<div class='block\_\_elem3'></div>

</div>

I've heard that BEM does not recommend global CSS resets. Why?

CSS resets is a practise making a good showing. Many frameworks first align anything and then apply their special styles. BEM does not recommend common resets. Why? And what we are supposed to do instead?

Nothing bad would happen to your blocks if you use common reset (well, except of some special cases below). So, BEM does not prohibit to use them. But using them BEM-way would be more effective.

Common CSS reset is a set of CSS to be applied to document nodes and ensure that their default view is the same in different browsers. In most cases the CSS rules are written for tag selectors and this is not recommended in BEM (you can find a lot of explanation above).

Another point is that in BEM a block encapsulates everything which is needed for it to be displayed and function. And this is why we call the BEM blocks independent. If the block does not look properly without a third-party CSS being added onto the page, it cannot be called "independent" that much.

Assuming this all, BEM recommends every block to reset itself. If you have menu block and list block both as <ul> in your HTML, each of them should provide the reset CSS usually given to <ul>. You may worry that having several blocks with the same reset rules will case repeats in the resultant code. *But this is what CSS optimizers should do for you.* As a developer you develop every block independently, as if there are no other blocks on the same page.

In the case you don't have a CSS optimizer to combine selectors with the same set of rules, you may use preprocessors to prevent copy-paste. With every new block you can make it reset itself mixing the proper code. For example, with SASS this would look like:

.menu {

@include reset-list;

}

.menu\_\_item {

@include reset-list-item;

}

/\* ... \*/

.list {

@include reset-list;

}

.list\_\_item {

@include reset-list-item;

}

However using this mixin-way you should realize that the only reason for it is not having a proper optimizer.

Having resets for every block (besides being nice and BEMish) will also prevent problems with injecting a third-party piece of HTML/CSS markup which relies on browser defaults and so can be affected by global resets. For example, this is a known problem for webmails.

Did not find the answer? Please ask your questions!

If you didn't find the proper answer, please [ask your question](https://github.com/getbem/getbem.com/issues/new?title=Type+your+question&body=Explain+in+detail+your+question&labels=question)!