Do not do this:

.redBox { border: 1px solid red;}

Instead, do this:

.red-box { border: 1px solid red;}

This is a pretty standard CSS naming convention. It is arguably more readable.

Also, it is consistent with the CSS property names.

// Correct

.some-class { font-weight: 10em}

// Wrong

.some-class { fontWeight: 10em}

**The BEM Naming Convention**

Teams have different approaches to writing CSS selectors. Some teams use hyphen delimiters, while others prefer to use the more structured naming convention called BEM.

Generally, there are 3 problems that CSS naming conventions try to solve:

1. To know what a selector does, just by looking at its name
2. To have an idea of where a selector can be used, just by looking at it
3. To know the relationships between class names, just by looking at them

Have you ever seen class names written like this:

.nav--secondary { ...}

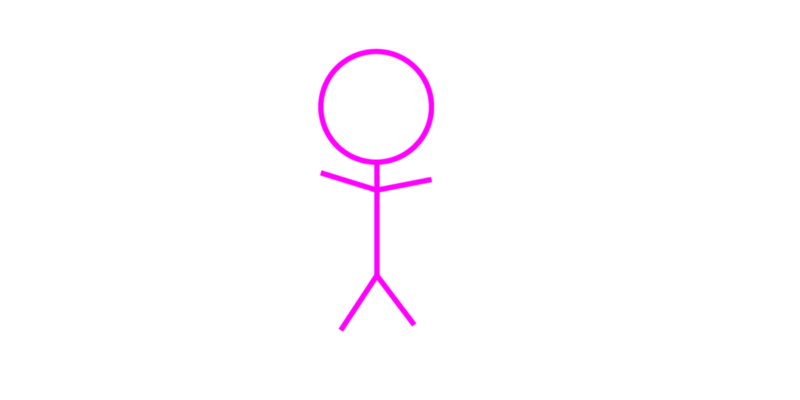
.nav\_\_header { ...}

That is the BEM naming convention.

**Explaining BEM to a 5 year Old**

BEM attempts to divide the overall user interface into small reusable components.

Consider the image below:

It is an award winning image of a stick-man :)

No, it’s not award winning :(

The stick-man represents a component, such as a block of design.

You may have already guessed that the B in BEM stands for ‘Block’.

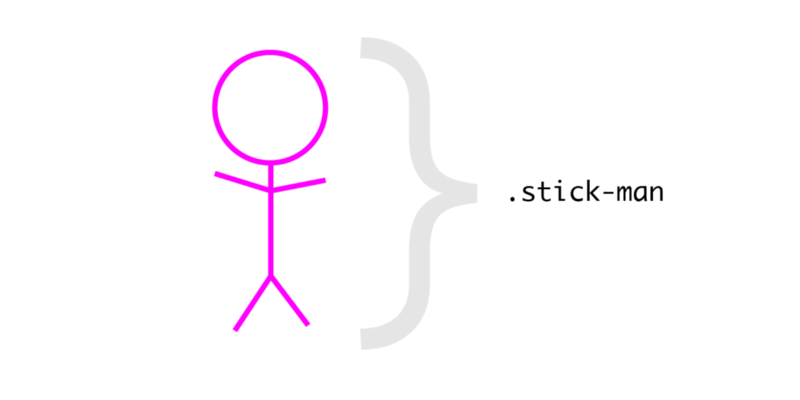
In the real world, this ‘block’ could represent a site navigation, header, footer, or any other block of design.

Following the practice explained above, an ideal class name for this component would be stick-man.

The component should be styled like so:

.stick-man { }

We have used delimited strings here. Good!

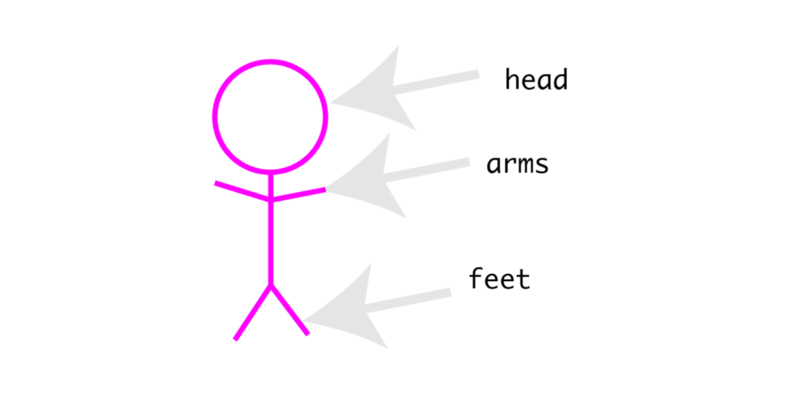


**E for Elements**

The E in ‘BEM’ stands for Elements.

Overall blocks of design rarely live in isolation.

For instance, the stick-man has a head, two gorgeous arms, and feet.



The head , feet, and arms are all elements within the component. They may be seen as child components, i.e. children of the overall parent component.

Using the BEM naming convention, element class names are derived by adding **two underscores,**followed by the element name.

For example:

.stick-man\_\_head {

}

.stick-man\_\_arms {

}

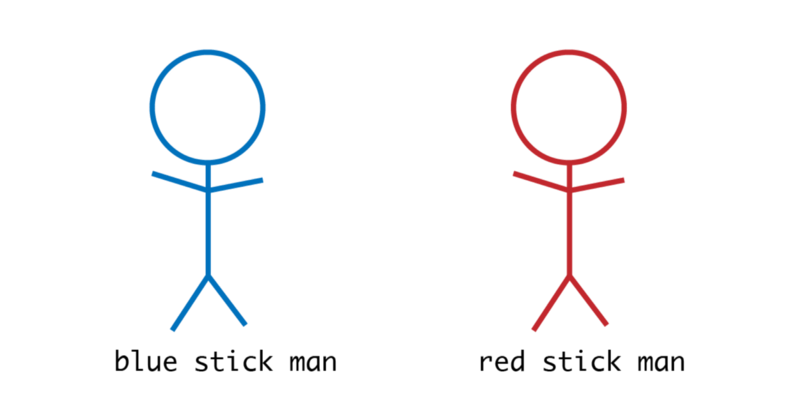
.stick-man\_\_feet {

}

**M for Modifiers**

The M in ‘BEM’ stands for Modifiers.

What if the stick-man was modified and we could have a blue or a red stick- man?



In the real world, this could be a red button or blue button. These are modifications of the component in question.

Using BEM, modifier class names are derived by adding two **hyphens**followed by the element name.

For example:

.stick-man--blue {

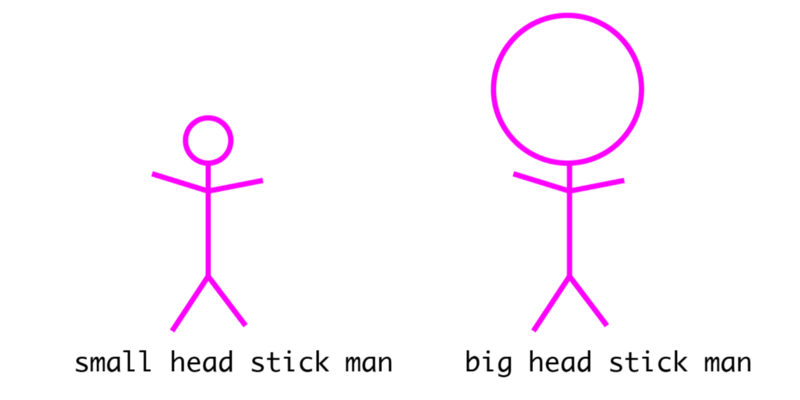
}

.stick-man--red {

}

The last example showed the parent component being modified. This is not always the case.

What if we had stick-men of different head sizes?



This time the element has been modified. Remember, the element is a child component within the overall containing block.

The .stick-man represents the Block , .stick-man\_\_head the element.

As seen in the example above, double hyphens may also be used like so:

.stick-man\_\_head--small {

}

.stick-man\_\_head--big {

}

Again, note the use of the double **hyphens** in the example above. This is used to denote a modifier.

Now you’ve got it.

That’s basically how the BEM naming convention works.

Personally, I tend to use only hyphen delimeter class names for simple projects, and BEM for more involved user interfaces.

You can [read more](http://getbem.com/naming/) about BEM.

[**BEM - Block Element Modifier**](http://getbem.com/naming/)  
[*BEM - Block Element Modifier is a methodology, that helps you to achieve reusable components and code sharing in the…*getbem.com](http://getbem.com/naming/)

**Why Use Naming Conventions?**

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

Naming things is hard. We’re trying to make things easier, and save ourselves time in the future with more maintainable code.

Naming things correctly in CSS will make your code easier to read and maintain.

If you choose to use the BEM naming convention, it will become easier to see the relationship between your design components/blocks just by looking at the markup.

Feeling confident?

**CSS Names with JavaScript Hooks**

Today is John’s first day at work.

He is handed over an HTML code that looks like this:

<div class="siteNavigation">

</div>

John has read this article and realizes this may not be the best way to name things in CSS. So he goes ahead and refactors the codebase like so:

<div class="site-navigation">

</div>

Looks good, huh?

Unknown to John, he had broken the codebase ???

How?

Somewhere in the JavaScript code, there was a relationship with the previous class name, siteNavigation:

//the Javasript code

const nav = document.querySelector('.siteNavigation')

So, with the change in the class name, the nav variable became null.

How sad.

To prevent cases like this, developers have come up with different strategies.

**1. Use js- class names**

One way to mitigate such bugs is to use a **js-\***class name to denote a relationship with the DOM element in question.

For example:

<div class="site-navigation js-site-navigation">

</div>

And in the JavaScript code:

//the Javasript code

const nav = document.querySelector('.js-site-navigation')

As a convention, anyone who sees the **js-**site-navigation class name would understand that there is a relationship with that DOM element in the JavaScript code.

**2. Use the Rel attribute**

I don’t use this technique myself, but I have seen people do.

Do you recognize this?

<link rel="stylesheet" type="text/css" href="main.css">

Basically, the **rel attribute** defines the relationship that the linked resource has to the document from which it’s referenced.

In the previous example with John, proponents of this technique would do this:

<div class="site-navigation" rel="js-site-navigation">

</div>

And in the JavaScript:

const nav = document.querySelector("[rel='js-site-navigation']")

I have my doubts about this technique, but you’re likely to come accross it in some codebases. The claim here is, *“well, there’s a relationship with Javascript, so I use the rel attribute to denote that”*.

The web is a big place with lots of different “methods” for solving the same problem.

**3. Don’t use data attributes**

Some developers use data attributes as JavaScript hooks. This isn’t right. By definition, data attributes are used **to store custom data**.