BEM for everyone else

Using emoji to explain the principles of BEM

Dave House
Jan 11 · 6 min read

I'd heard the term Block Element Modifier (BEM) used by many of the frontend developers I've worked with over the years. It wasn't until I started on my current project that I actually needed to understand what it was or what the benefits of using it might be. After hearing the same questions about CSS class names from users of the product I work on, I decided to create a short intro to BEM primarily aimed at designers, although I hope it helps anyone else that's new to it.

What is BEM?

The BEM methodology is a way of naming CSS classes.

Following BEM means naming classes in a way that they relate to each other, this is achieved using a combination of double hyphens `--` and double underscores `__` to help convey the structure of the HTML markup in the class names themselves.

BEM looks like this:

```
.block {}
.block__element {}
.block--modifier {}
```

This can be hard to understand if you've never seen it before, especially if the first time you see it is in context of an existing codebase.

For example if you see `radio-button--inline` and `radio-button__input` within a codebase or a coded example in a design system, you may not understand the difference between double hyphens or double underscores. If you are completely new to BEM, you'd be forgiven for thinking that perhaps this could be just an inconsistency from multiple developers working on a project or an unscrutinised code review.

To help new users understand BEM, I've found it helpful to step out of web development and apply the principles to other things.

BEM-ojis

Love them or hate them, emojis have enough commonality and variants to help explain BEM.

For this I'm going to start with a basic 'expressionless face' emoji as it can be considered a good base for modification into other face emojis.

Block

If we make the base for the 'expressionless face' emoji our 'block' might look something like this in HTML:

A background of an emoji

```
<div class="face">
</div>
```

Element

We can consider the eyes and mouth to be 'elements' of 'face', these elements are prefixed by the name of the 'block' followed by a double underscore:

An ' expressionless face'
emoji

```
<div class="face">
 <div class="face__eyes">
 </div>
 <div class="face__mouth">
 </div>
</div>
```

Naming things like this makes it easier for someone looking at the CSS to understand the relationship between the 'block' and the 'element', in this case `face` , `face__eyes` and `face__mouth` .

Modifier

Modifiers are used in addition to 'blocks' or 'elements' to change or 'modify' something about the default style. Modifiers are signified with a double hyphen.

If we wanted to modify the 'expressionless face' to become a 'neutral face' then we would add a modifier to the `face__eyes` element, for example `face__eyes--open` .

In the HTML markup that would look like this:

An 'expressionless face'
becomes a 'neutral face'
by modifying the eyes
element

```
<div class="face">
 <div class="face__eyes face__eyes--open">
 </div>
 <div class="face__mouth">
```

```
 </div>
</div>
```

Modifiers don't work on their own

Modifiers are always used in addition to a 'block' or 'element' as they require the CSS from the base class to add styling on top of.

If we made up some imaginary CSS for the emoji's mouth it might look something like this:

```
.face__mouth {
  background-color: brown;
  border: 1px solid yellow;
  box-shadow: inset 0px 0px 15px maroon;
  height: 10px;
  width: 40px;
  border-radius: 30px;
}

.face__mouth--open {
  /* Make mouth square */
  width: 30px;
  height: 30px;
  /* Make square a circle */
  border-radius: 50%;
}
```

If you simply replaced the `face__mouth` class with the modifier class `face__mouth--open` it would actually result in the emoji having no mouth at all, as the `background colour`, `border` and internal `box-shadow` are only declared inside the `face__mouth` class:

What the emoji would look like without the `face__mouth` element

```
<div class="face">
 <div class="face__eyes face__eyes--open">
 </div>
 <div class="face__mouth--open">
 </div>
</div>
```

To make this example work you need to have the class `face__mouth` included in your HTML for `face__mouth--open` to modify.

What the emoji looks like with the `face__mouth` element and the **face__mouth--open** modifier

```
<div class="face">
 <div class="face__eyes face__eyes--open">
 </div>
 <div class="face__mouth face__mouth--open">
 </div>
</div>
```

Modifying the block

So far we've only added modifiers to elements — the eyes and the mouth in the previous examples, but blocks themselves can also be modified.

For this example we can borrow from the iOS emoji skin tone modifiers. In this instance, using the 'wave' emoji as our block, we have the default skin tone applied to the class `wave`.

A "wave" emoji using
the default iOS yellow

```
<div class="wave">
</div>
```

This block could be modified with the classes `wave--skin-tone-light`, `wave--skin-tone-medium-light`, `wave--skin-tone-medium`, `wave--skin-tone-medium-dark` or `wave--skin-tone-dark`.

Applying the **wave--skin-tone-medium** modifier to the **wave** block

```
<div class="wave wave--skin-tone-medium">
</div>
```

Examples from GOV.UK Frontend

I work at the Government Digital Service as a designer on the team that looks after the GOV.UK Design System and it's frontend codebase GOV.UK Frontend.

To see what BEM looks like in more of a real life setting, here are some examples from GOV.UK Frontend.

Buttons

Here we have a button with the class `govuk-button`, this can be modified into what we call a 'Start button' by adding the class `govuk-button--start`.

A standard button

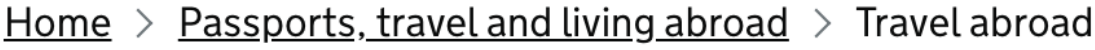
```
<button class="govuk-button">Save and continue</button>
```

A standard button becomes a start button by adding the class **govuk-button--start**

```
<button class="govuk-button govuk-button--start">Start now</button>
```

Breadcrumbs

In this example of the `govuk-breadcrumbs` component, the 'block' `govuk-breadcrumbs` contains the 'elements' `govuk-breadcrumbs__list`, `govuk-breadcrumbs__list-item` and `govuk-breadcrumbs__link`.



An example of the breadcrumbs component

```
<div class="govuk-breadcrumbs">
  <ol class="govuk-breadcrumbs__list">
    <li class="govuk-breadcrumbs__list-item">
      <a class="govuk-breadcrumbs__link" href="#">Home</a>
    </li>
    <li class="govuk-breadcrumbs__list-item">
      <a class="govuk-breadcrumbs__link" href="#">Passports, travel
and living abroad</a>
    </li>
    <li class="govuk-breadcrumbs__list-item" aria-
current="page">Travel abroad
    </li>
  </ol>
</div>
```

Why use BEM at all?

The choice to use BEM or any other CSS naming methodology should be based on your individual project. For large distributed codebases BEM could be an ideal choice, for a small marketing site it's probably overkill.

For us, an example of why we moved to BEM can be seen in our previous frontend codebase GOV.UK Elements, which often reused classes on multiple components.

For example, the 'inset text' example in GOV.UK Elements uses a class called `panel`, which applied padding around an element with a border to the left:

> It can take up to 8 weeks to register a lasting power of attorney if there are no mistakes in the application.

The 'Inset text' component from GOV.UK Elements

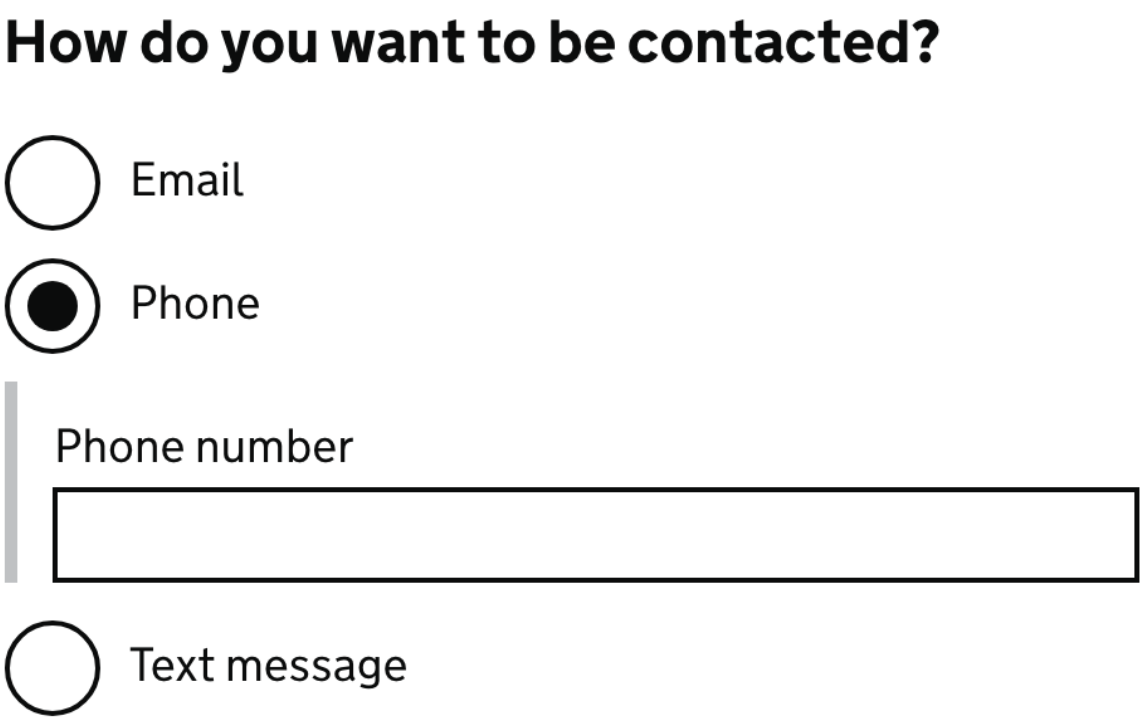
'Hidden text' also used the `panel` class:

▼ Help with nationality

> If you're not sure about your nationality, try to find out from an official document like a passport or national ID card.
>
> We need to know your nationality so we can work out which elections you're entitled to vote in. If you can't provide your nationality, you'll have to send copies of identity documents through the post.

The 'Hidden text' component from GOV.UK Elements

Conditionally revealing content also used the `panel` class:



Conditionally revealing content from GOV.UK Elements

While in some situations this would actually be the ideal way of doing things, in our scenario a developer making a change to the styling of the `panel` class, thinking they were styling 'inset text', could affect all of the other components that rely on that class without realising.

In GOV.UK Frontend, we use BEM to style Inset Text, Details and conditionally revealing content independently of one another. This means there will be a much better understanding of what effect changing these classes will have, as it makes components easier to test and see when something breaks.

Beyond the basics

I hope this simple overview has helped you to get your head around BEM, if you want to continue to learn about it you should read the official docs.

CSS Design Systems Frontend