

CSS Style Guide

CSS Style Guide



validity · names · name style
· selectors · shorthand ·
delimiters · order · stops ·
quotations

Google HTML/CSS Style Guide

Table of Contents

[1 Background](#)

[2 General](#)

[2.1 General Style Rules](#)

[2.2 General Formatting Rules](#)

[2.3 General Meta Rules](#)

[3 HTML](#)

[3.1 HTML Style Rules](#)

[3.2 HTML Formatting Rules](#)

[4 CSS](#)

[4.1 CSS Style Rules](#)

[4.2 CSS Formatting Rules](#)

[4.3 CSS Meta Rules](#)

[Parting Words](#)



CSS Validation Service

Check Cascading Style Sheets (CSS) and (X)HTML documents with style sheets

By URI

By file upload

By direct input

Validate by URI

Enter the URI of a document (HTML with CSS or CSS only) you would like validated:

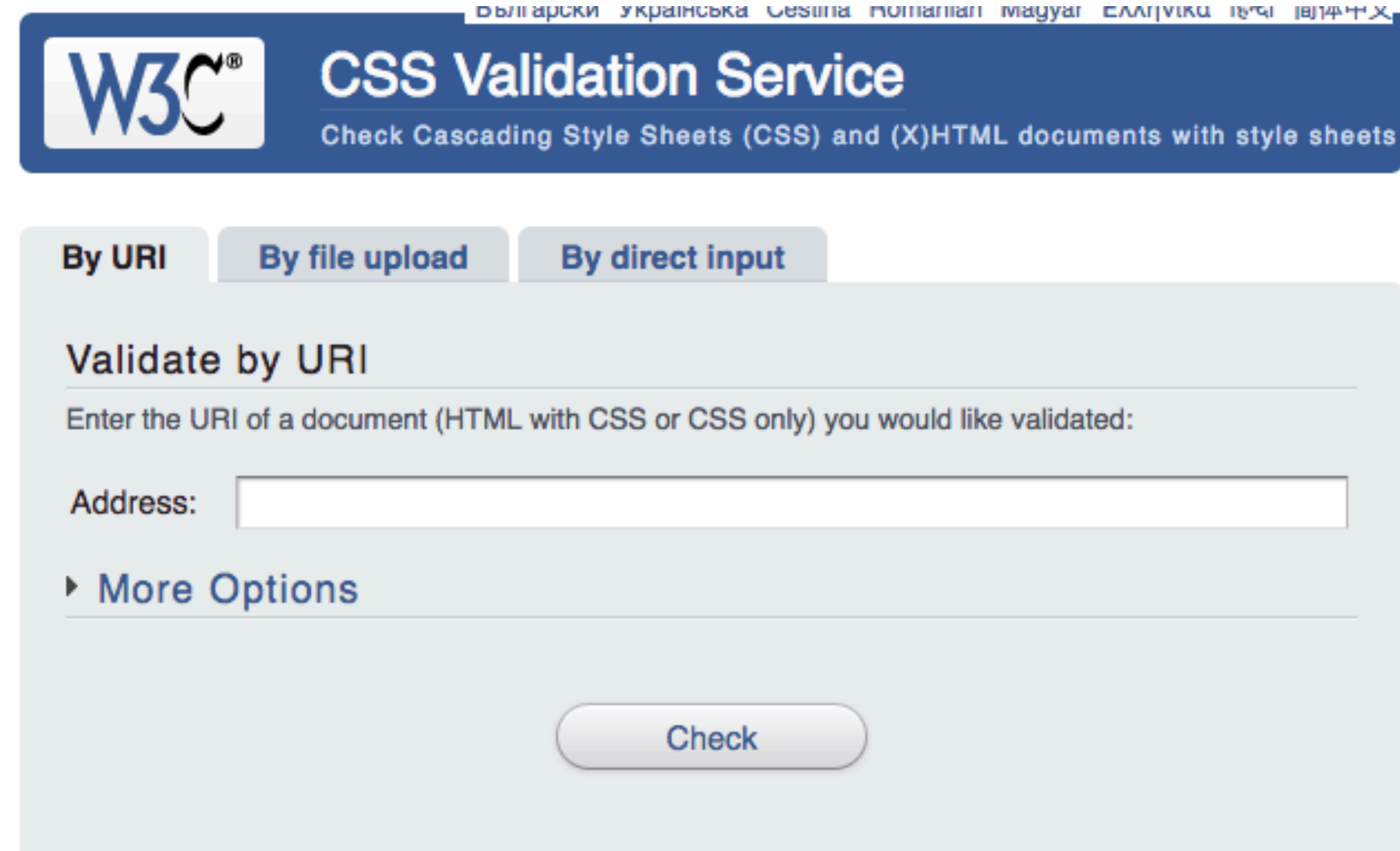
Address:

▶ [More Options](#)

Check

CSS Validity

Use valid CSS where possible.



The image shows the W3C CSS Validation Service interface. At the top, there is a blue header with the W3C logo and the text "CSS Validation Service" and "Check Cascading Style Sheets (CSS) and (X)HTML documents with style sheets". Below the header, there are three tabs: "By URI", "By file upload", and "By direct input". The "By URI" tab is selected. The main content area is titled "Validate by URI" and contains the instruction "Enter the URI of a document (HTML with CSS or CSS only) you would like validated:". Below this is a text input field labeled "Address:". There is also a link for "More Options" and a "Check" button at the bottom.

Unless dealing with CSS validator bugs or requiring proprietary syntax, use valid CSS code.

Use tools such as the W3C CSS validator to test.

Using valid CSS is a measurable baseline quality attribute that allows to spot CSS code that may not have any effect and can be removed, and that ensures proper CSS usage.

```
/* Not recommended: meaningless */  
#yee-1901 {}
```

```
/* Not recommended: presentational */  
.button-green {}  
.clear {}
```

```
/* Recommended: specific */  
#gallery {}  
#login {}  
.video {}
```

```
/* Recommended: generic */  
.aux {}  
.alt {}
```

ID and Class Naming

Use meaningful or generic ID and class names.

Instead of presentational or cryptic names, always use ID and class names that reflect the purpose of the element in question, or that are otherwise generic.

```
/* Not recommended: meaningless */  
#yee-1901 {}  
  
/* Not recommended: presentational */  
.button-green {}  
.clear {}
```

```
/* Recommended: specific */  
#gallery {}  
#login {}  
.video {}  
  
/* Recommended: generic */  
.aux {}  
.alt {}
```

Using functional or generic names reduces the probability of unnecessary document or template changes.

```
/* Not recommended */  
#navigation {}  
.atr {}
```

```
/* Recommended */  
#nav {}  
.author {}
```

ID and Class Name Style

Use ID and class names that are as short as possible but as long as necessary.

```
/* Not recommended */  
#navigation {}  
.atr {}
```

```
/* Recommended */  
#nav {}  
.author {}
```

Try to convey what an ID or class is about while being as brief as possible.

Using ID and class names this way contributes to acceptable levels of understandability and code efficiency.


```
/* Not recommended */  
ul#example {}  
div.error {}
```

```
/* Recommended */  
#example {}  
.error {}
```

Type Selectors

Avoid qualifying ID and class names with type selectors.

```
/* Not recommended */  
ul#example {}  
div.error {}
```

```
/* Recommended */  
#example {}  
.error {}
```

Unless necessary (for example with helper classes), do not use element names in conjunction with IDs or classes.

Avoiding unnecessary ancestor selectors is useful for performance reasons.

```
/* Not recommended */  
border-top-style: none;  
font-family: palatino, georgia, serif;  
font-size: 100%;  
line-height: 1.6;  
padding-bottom: 2em;  
padding-left: 1em;  
padding-right: 1em;  
padding-top: 0;
```

```
/* Recommended */  
border-top: 0;  
font: 100%/1.6 palatino, georgia, serif;  
padding: 0 1em 2em;
```

Shorthand Properties

Use shorthand properties where possible.

```
/* Not recommended */  
border-top-style: none;  
font-family: palatino, georgia, serif;  
font-size: 100%;  
line-height: 1.6;  
padding-bottom: 2em;  
padding-left: 1em;  
padding-right: 1em;  
padding-top: 0;
```

```
/* Recommended */  
border-top: 0;  
font: 100%/1.6 palatino, georgia, serif;  
padding: 0 1em 2em;
```

CSS offers a variety of shorthand properties (like font) that should be used whenever possible, even in cases where only one value is explicitly set.

Using shorthand properties is useful for code efficiency and understandability.

```
/* Not recommended: does not separate the words "demo" and "image" */  
.demoimage {}
```

```
/* Not recommended: uses underscore instead of hyphen */  
.error_status {}
```

```
/* Recommended */  
#video-id {}  
.ads-sample {}
```

ID and Class Name Delimiters

```
/* Not recommended: does not separate the words "demo" and "image" */  
.demoimage {}
```

```
/* Not recommended: uses underscore instead of hyphen */  
.error_status {}
```

```
/* Recommended */  
#video-id {}  
.ads-sample {}
```

Separate words in ID and class names by a hyphen.

Do not concatenate words and abbreviations in selectors by any characters (including none at all) other than hyphens, in order to improve understanding and scannability.

```
p {  
  background: fuchsia;  
  border: 1px solid;  
  border-radius: 4px;  
  color: black;  
  text-align: center;  
  text-indent: 2em;  
}
```

Declaration Order

```
p {  
  background: fuchsia;  
  border: 1px solid;  
  border-radius: 4px;  
  color: black;  
  text-align: center;  
  text-indent: 2em;  
}
```

Alphabetize declarations.

Put declarations in alphabetical order in order to achieve consistent code in a way that is easy to remember and maintain.


```
/* Not recommended */  
.test {  
  display: block;  
  height: 100px  
}
```

```
/* Recommended */  
.test {  
  display: block;  
  height: 100px;  
}
```

Declaration Stops

```
/* Not recommended */  
.test {  
  display: block;  
  height: 100px  
}
```

```
/* Recommended */  
.test {  
  display: block;  
  height: 100px;  
}
```

Use a semicolon after every declaration.

End every declaration with a semicolon for consistency and extensibility reasons.

```
/* Not recommended */
@import url("https://www.google.com/css/maia.css");

html {
  font-family: "open sans", arial, sans-serif;
}
```

```
/* Recommended */
@import url(https://www.google.com/css/maia.css);

html {
  font-family: 'open sans', arial, sans-serif;
}
```

CSS Quotation Marks

```
/* Not recommended */  
@import url("https://www.google.com/css/maia.css");  
  
html {  
    font-family: "open sans", arial, sans-serif;  
}
```

```
/* Recommended */  
@import url(https://www.google.com/css/maia.css);  
  
html {  
    font-family: 'open sans', arial, sans-serif;  
}
```

Use single (') rather than double (") quotation marks for attribute selectors and property values.

Do not use quotation marks in URI values (url()).

Parting Words:

“If you’re editing code, take a few minutes to look at the code around you and determine its style. If they use spaces around all their arithmetic operators, you should too. If their comments have little boxes of hash marks around them, make your comments have little boxes of hash marks around them too.”

Be consistent:

“The point of having style guidelines is to have a common vocabulary of coding so people can concentrate on what you’re saying rather than on how you’re saying it. We present global style rules here so people know the vocabulary, but local style is also important. If code you add to a file looks drastically different from the existing code around it, it throws readers out of their rhythm when they go to read it. Avoid this.”

<http://getbem.com/>

BEM — Block Element Modifier is a methodology that helps you to create reusable components and code sharing in front-end development

Easy

To use BEM, you only need to employ BEM's naming convention.

Modular

Independent blocks and CSS selectors make your code reusable and modular.

Flexible

Using BEM, methodologies and tools can be recomposed and configured the way you like.

Introduction

BEM is a highly useful, powerful, and simple naming convention that makes your front-end code easier to read and understand, easier to work with, easier to scale, more robust and explicit, and a lot more strict.



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 { }
```