CSS SELECTORS

This slideshow explores all 54 of the **CSS1, CSS2 and CSS3 selectors**.

Levels of CSS

The W3C and Recommendations

The **World Wide Web Consortium**, or the W3C is an international community that develops open standards for the Web.

The W3C produces **specifications** on a wide range of web-related topics (including HTML, XHTML and CSS).

A **W3C Recommendation** is a specification that, after extensive consensus-building, has been endorsed by W3C Members and the Director.

The W3C follows the **five steps** when advancing a specification to Recommendation.

1. Publication of the First Public Working Draft

Document maturity level: Working Draft (WD)

2. Last Call announcement

Document maturity level: Working Draft (WD)

3. Call for Implementations

Document maturity level:
Candidate Recommendation (CR)

4. Call for Review of a Proposed Recommendation

Document maturity level: Proposed Recommendation (PR)

5. Publication as a Recommendation

Document maturity level: W3C Recommendation (REC)

CSS1 and CSS2

CSS Level 1 (CSS1) became a W3C Recommendation on 17 December 1996. It was revised on 11 January 1999.

CSS Level 2 (CSS2) became a W3C Recommendation on 12 May 1998. It was revised on 11 April 2008.

CSS Level 2 Revision 1 (CSS2.1) became a W3C Recommendation on 7 June 2011.

CSS3

Instead of writing CSS3 as a single specification, the W3C decided to **divide CSS3 into a series of individual specifications - called “modules”**.

This process allows the W3C to bring modules through the five step advancement process **at different rates** - rather than wait till all the individual modules are defined.

There are a wide range of **CSS3 modules** such as:

Media Queries
Selectors Level 3
CSS Color Module Level 3

http://www.w3.org/Style/CSS/current-work.en.html

Is there a CSS4?

Despite some of the confusing titles given to various CSS3 modules, **there is no such things as CSS4**!

“Some of our modules start out at level 3, if they extend something from CSS2.1. Others start out at level 1, if they're something new (for example, Flexbox). However, the level that a module is at has no correlation with what version of CSS it's in. They're all CSS3 (or

Simple selectors

Simple selectors allow you to target HTML elements directly, as well as targeting elements that contain class or ID attributes.

Type selector (CSS1)

The **type selector** is written using an element type (e).

```css
/* syntax */
E { }

/* example */
h1 { }
```

How does it work?

The type selector targets every instance of the element type regardless of their position in the document tree.

In the following example, the “p” selector targets **any instance of the <p> element**.

```
/* CSS selector */
p { }

<!-- HTML markup -->
<p></p>
<div></div>
<p></p>
```

Support

The type selector is well supported across all modern browsers.

Let’s look at type selectors

Class selector (CSS1)

The **class selector** is written using an optional element, followed by a “.”, followed by the class name.

```css
/* syntax */
.class-name { }
E.class-name { }

/* example */
.intro { }
```

Class values are **case-sensitive**. In the following example, browsers interpret ".intro" and ".Intro" as different classes.

```
/* case sensitive classes */
.Intro { }
.intro { }
```

Classes can begin with **letters, digits, hyphens and underscores**.

```css
/* class names */
.active { }
.480wide { }
.-members { }
._classname { }
```

However, if class values start with a number or a special character, these characters or numbers **must be escaped** when writing CSS selectors.

```
/* escaped number */
.\34 80wide { }

<!-- HTML markup -->
<div class="480wide"></div>
```

How does it work?

The class selector targets any HTML element that has the relevant class value regardless of their position in the document tree.

In the following example, the “.intro” selector targets any element that **contains a class of “intro”**.

```
/* CSS selector */
.intro { }

<!-- HTML markup -->
<p class="intro"></p>
<div class="intro"></div>
```

Support

The class selector is well supported across all modern browsers.

Elements can be added before the "." to **make a selector more specific**.

CSS

```
p.intro { }
div.intro { }
```

In the following example, the “p.intro” selector targets **only <p> elements that contain a class of “intro”**. The <div> is not targeted.

CSS
```css
p.intro { }
```

HTML
```html
<p class="intro"></p>
<div class="intro"></div>
```

Multiple classes can be joined together to make a selector more specific. The joined class selectors cannot contain whitespace.

CSS

```css
.one.two { }
```

In the following example, the ".one.two" selector targets any element that has a class of **both "one" and "two"**.

CSS

```
.one.two { }
```

HTML

```
<p class="one"></p>
<p class="one two"></p>
<p class="two"></p>
```

The class attribute values within the HTML document can be written in any order. The value “one two” is **the same as** “two one”.

CSS

```css
.one.two { }
```

HTML

```html
<p class="one two"></p>
<p class="two one"></p>
```

Multiple classes can also be written in any order within the CSS. The selector “.one.two” is **the same as** “.two.one”.

CSS

```css
.two.one { }
```

HTML

```html
<p class="one two"></p>
```

Support

Multiple class selectors are not supported by Internet Explorer 6.

Let’s look at class selectors

ID selector
(CSS1)

The **ID selector** is written using an optional element, followed by a “#”, followed by the ID name.

```css
/* syntax */
#id-name { }
E#id-name { }

/* example */
#sidebar { }
```

ID can begin with **letters, digits, hyphens and underscores**.

```css
/* ID names */
#active { }
#480wide { }
#-members { }
#_classname { }
```

However, if ID values start with a number or a special character, these characters or numbers **must be escaped** when writing CSS selectors.

```
/* escaped number */
#\34 80wide { }

<!-- HTML markup -->
<div id="480wide"></div>
```

How does it work?

The ID selector targets any HTML element that has the relevant ID value regardless of its position in the document tree.

In the following example, the “#nav” selector targets **any instance** of the an element that contains an ID of “nav”.

```
/* CSS selector */
#nav { }

<!-- HTML markup -->
<p id="nav"></p>
```

Support

The ID selector is well supported across all modern browsers.

Let’s look at ID selectors

Universal selector (CSS2)

The **universal selector** is written using a “*”.

```css
/* syntax */
* { }
```

How does it work?

The universal selector targets all elements within the document.

In the following example, the universal selector targets **every element in the document**.

```
/* CSS selector */
* { }

<!-- HTML markup -->
<body>
<div></div>
</body>
```

Support

The universal selector is well supported across all modern browsers.

Let’s look at universal selectors

Combinator selectors

Combinators allow you to combine individual selectors into new types of selectors.

Descendant selectors (CSS1)

The **descendant selector** is written using two or more selectors separated by whitespace.

```css
/* syntax */
E F { }

/* example */
ul li a { }
```

How does it work?

The descendant selector targets only elements that are descendants of other elements.

In the following example, the <a> that is a **descendant of the <p> will be selected**, but not the <a> that is a descendant of the <div>.

```
/* CSS selector */
p a { }

<!-- HTML markup -->
<p><a href="#">Link</a></p>
<div><a href="#">Link</a></div>
```

The key to descendant selectors is understanding **paths to elements**. There is a path to every element starting with the <html> element.

Paths can be **written in full or in part**, depending on your need. Paths can also skip one or more levels.

```css
/* example paths */
body .container .nav ul li a { }
.container .nav ul li a { }
.nav ul li a { }
ul li a { }
ul a { }
li a { }
a { }
```

When writing descendant selectors, try to keep your selectors as short as possible. Each selector should **only be as specific as needed**.

```
/* only as specific as needed */
.nav ul li a { }
.nav a {}
```

Support

The descendant selector is well supported across all modern browsers.

Let’s look at descendant selectors

Child selectors (CSS2)

The **child selector** is written using two selectors separated by a ">".

```
/* syntax */
E > F { }

/* example */
p > a { }
```

How does it work?

The child selector targets any element that is a direct child of another element. Only child elements, rather than descendant elements, will be selected.

In the following example, the <a> that is **a child of the <div> will be selected**, but not the <a> that is a descendant of the <div>.

```
/* CSS selector */
div > a { }

<!-- HTML markup -->
<div><a>link</a></div>
<div><p><a>link</a></p></div>
```

Support

The child selector is not supported by Internet Explorer 6.

Adjacent sibling selectors (CSS2)

The **adjacent sibling** selector is written using two selectors separated by a "+".

```
/* syntax */
E + F { }

/* example */
h2 + h3 { }
```

How does it work?

The Adjacent sibling selector targets the sibling (element that shares the same parent) immediately following a defined element.

In the following example, only the <h3> that is **adjacent to (or comes directly after) the <h2>** will be selected.

```
/* CSS selector */
h2 + h3 { }

<!-- HTML markup -->
<h2>Content</h2>
<h3>Content</h3>
<h3>Content</h3>
```

Support

The adjacent sibling selector is not supported by Internet Explorer 6.

General sibling selectors (CSS3)

The **general sibling** selector is written using two selectors separated by a "~".

```css
/* syntax */
E ~ F { }

/* example */
h2 ~ h3 { }
```

How does it work?

The General sibling selector targets any sibling (element that shares the same parent) that follows a defined element.

In the following example, any **<h3> that appears after the <h2> in source order will be selected** - as long as they share the same parent.

```
/* CSS selector */
h2 ~ h3 { }

<!-- HTML markup -->
<h2>Content</h2>
<h3>Content</h3>
<h3>Content</h3>
```

Support

The general sibling selector is not supported by Internet Explorer 6.

Link pseudo-classes

Pseudo-classes (or "fake" classes) allow you to style specific attributes or states that do not exist in the document tree.

:link pseudo-class (CSS1)

The **:link pseudo-class selector** is written using an <a> element, followed by “:”, followed by “link”.

```css
/* syntax */
E:link { }

/* example */
a:link { }
```

How does it work?

The :link pseudo-class selector targets any link that is defined as “unvisited”.

Support

The :link pseudo-class selector is well supported across all modern browsers.

:visited pseudo-class (CSS1)

The **:visited pseudo-class selector** is written using an <a> element, followed by “:”, followed by “visited”.

```css
/* syntax */
E:visited { }

/* example */
a:visited { }
```

How does it work?

The :visited pseudo-class selector targets any link that is defined as “visited”.

Due to **potential security issues**, there are only a few properties that you can use to style visited links. These include:

color, background-color, border-color, outline-color, fill, and stroke

You can read more about **potential security issues associated with visited links** here:

http://dbaron.org/mozilla/visited-privacy

http://blog.mozilla.org/security/2010/03/31/plugging-the-css-history-leak/

https://developer.mozilla.org/en-US/docs/Web/CSS/Privacy_and_the_:visited_selector

Support

The :visited pseudo-class selector is well supported across all modern browsers.

:focus pseudo-class
(CSS2)

The **:focus pseudo-class selector** is written using an element, followed by “:”, followed by “focus”.

```
/* syntax */
E:focus { }

/* example */
a:focus { }
```

How does it work?

The :focus pseudo-class selector targets elements that have focus (ones that accept keyboard events).

Support

The :focus pseudo-class selector is not supported by IE6 or IE7.

:hover pseudo-class (CSS2)

The **:hover pseudo-class selector** is written using an element (which does not have to be an <a> element), followed by “:”, followed by “hover”.

```css
/* syntax */
E:hover { }

/* example */
a:hover { }
```

How does it work?

The :hover pseudo-class selector targets any element when the user’s cursor is over the element - but the user has not activated it.

Support

IE6 only supports :hover on elements with the “href” attribute.

:active pseudo-class (CSS1)

The **:active pseudo-class selector** is written using an <a> element, followed by “:”, followed by “active”.

```css
/* syntax */
E:active { }

/* example */
a:active { }
```

How does it work?

The :active pseudo-class selector targets any element that is currently being activated by the user.

Support issue 1

IE6 and IE7 only support :active on elements with the “href” attribute.

Support issue 2

IE6 incorrectly applies this pseudo-class to links that have input focus (incorrectly applies :active for :focus).

Support issue 3

IE6 has a "sticky active state" (active state remains active when user returns to page via the back button).

Order specified

A lot of people use the phrase **"love hate"** to remember link order:

Lo**v**e **Ha**te
Link, Visited, Hover, Active

```css
/* order */
a:link { }
a:visited { }
a:hover { }
a:active { }
```

The only problem is that it **does not include the “focus” state**, which is very important.

Another option is to use **“Lord Vader, Former Handle Anakin”**.

Lord **V**ader, **F**ormer **H**andle **A**nakin
Link, Visited, Focus, Hover, Active

```css
/* order */
a:link { }
a:visited { }
a:focus { }
a:hover { }
a:active { }
```

:lang(c)
pseudo-class

Defining a language

All web pages **should be defined with a language** to assist devices, search engines, assistive technologies etc.

The primary language for each web page can be **defined using the “lang” attribute** inside the <html> element.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Title</title>
</head>
<body>
    <h1>Hello world</h1>
</body>
</html>
```

To define languages for sections of a web page, the “lang” attribute can be **added to relevant elements**.

```html
<!-- Italian -->
<div lang="it">
    Mi sei mancato molto!
</div>

<!-- Latvian -->
<span lang="lv">
    Labvakar
</span>
```

The values inside the “lang” attribute are called **Language subtags**. These subtags are defined in the IANA Language Subtag Registry.

http://www.iana.org/assignments/language-subtag-registry

The :lang selector (CSS2)

The **:lang(c) pseudo-class** selector is written using an optional element, followed by a “:”, followed by “lang”, followed by a language subtag placed inside brackets (c).

```css
/* syntax */
:lang(c) { }
E:lang(c) { }

/* French example */
p:lang(fr) { }
/* Punjabi example */
span:lang(pa) { }
/* Singapore example */
:lang(sg) { }
```

How does it work?

The :lang(c) pseudo-class selector targets any element that contains the “lang” attribute and the relevant language subtag value.

In the following example, only the <p> element with the **(fr) lang subtag** will be selected.

```
/* CSS selector */
p:lang(fr) { }


<!-- HTML markup -->
<p lang="fr">Adieu</p>
<p lang="jw">Sugeng rawuh</p>
```

Support

The :lang(c) pseudo-class selector is not supported in IE6 or IE7.

Attribute selectors

Attribute selectors are used to select elements based on their attributes or their attribute and values.

```html
<!-- attribute -->
<p class="intro"></p>

<!-- attribute value -->
<p class="intro"></p>
```

[att] attribute selector (CSS2)

The **[att] attribute** selector is written using an element followed by “[”, followed by the relevant attribute, followed by “]”.

```css
/* syntax */
E[att] { }

/* example */
input[required] { }
```

How does it work?

The [att] attribute selector targets any element with the relevant attribute.

In the following example, **any input with a required attribute** will be selected.

```css
/* CSS selector */
input[required] { }
```

```html
<!-- HTML markup -->
<input type="text" required>
<input type="text">
```

Support

The [att] attribute selector is not supported by IE6.

[att=val] attribute selector (CSS2)

The **[att=val] attribute** selector is written using an element followed by “[”, followed by the relevant attribute, followed by “=”, followed by the relevant value (within no quotes, single quotes or double quotes), followed by “]”.

```css
/* syntax */
E[att=val] { }
E[att='val'] { }
E[att="val"] { }

/* example */
p[class="blue"] { }
```

How does it work?

The [att=val] attribute selector targets any element with the relevant attribute value.

In the following example, any <p> with a **class of "blue"** will be selected. Multiple values and hyphen-separated values will not be selected.

```
/* CSS selector */
p[class="blue"] { }


<!-- HTML markup -->
<p class="blue"></p>
<p class="blue red"></p>
<p class="blue-new"></p>
<p class="new-blue"></p>
```

Support

The [att=val] attribute selector is not supported by IE6.

[att~=val] attribute selector (CSS2)

The **[att~=val] attribute** selector is written using an element followed by “[”, followed by the relevant attribute, followed by “~” (tilde), followed by “=“, followed by the relevant value (within no quotes, single quotes or double quotes), followed by “]”.

```
/* syntax */
E[att~=val] { }
E[att~='val'] { }
E[att~="val"] { }

/* example */
p[class~="blue"] { }
```

How does it work?

The [att~=val] attribute selector targets any space-separated instances of a value.

In the following example, any <p> element with a class value of “blue”, **including space separated instances**, will be selected.

```css
/* CSS selector */
p[class~="blue"] { }
```

```html
<!-- HTML markup -->
<p class="blue"></p>
<p class="blue red"></p>
<p class="red blue"></p>
<p class="blue-new"></p>
<p class="new-blue"></p>
```

Support

The [att~=val] attribute selector is not supported by IE6.

[att|=val] attribute selector (CSS2)

The **[att|=val] attribute** selector is written using an element followed by “[”, followed by the relevant attribute, followed by “|” (pipe or vertical bar), followed by “=”, followed by the relevant value (within no quotes, single quotes or double quotes), followed by “]”.

```
/* syntax */
E[att|=val] { }
E[att|='val'] { }
E[att|="val"] { }

/* example */
p[class|="blue"] { }
```

How does it work?

The [att|=value] attribute selector targets any element with an attribute value that matches, including hyphen-separated values.

In the following example, any <p> element with **a class value of “blue” or “blue-”** will be selected.

```
/* CSS selector */
p[class|="blue"] { }


<!-- HTML markup -->
<p class="blue"></p>
<p class="blue red"></p>
<p class="blue-new"></p>
<p class="new-blue"></p>
```

Support

The [att|=val] attribute selector is not supported by IE6.

[att^=val] attribute selector (CSS3)

The **[att^=val] attribute** selector is written using an element followed by “[”, followed by the relevant attribute, followed by “^” (caret), followed by “=”, followed by the relevant value (within no quotes, single quotes or double quotes), followed by “]”.

```
/* syntax */
E[att^=val] { }
E[att^='val'] { }
E[att^="val"] { }

/* example */
a[href^="http"] { }
```

How does it work?

The [att^=val] attribute selector targets any element whose attribute starts with the value.

In the following example, any <a> element with an “href” **that starts with “http”** will be selected. This could potentially allow you to style external links (that require a full URL), as opposed to local links (which may not require a full URL).

```
/* CSS selector */
a[href^="http"] { }


<!-- HTML markup -->
<a href="http://abc.com"></a>
```

Support

The [att^=val] attribute selector is not supported by IE6.

[att$=val] attribute selector (CSS3)

The **[att$=val] attribute** selector is written using an element followed by “[”, followed by the relevant attribute, followed by “$” (dollar sign), followed by “=” followed by the relevant value (within no quotes, single quotes or double quotes), followed by “]”.

```css
/* syntax */
E[att$=val] { }
E[att$='val'] { }
E[att$="val"] { }

/* example */
a[href$=".pdf"] { }
```

How does it work?

The [att$=val] attribute selector targets any element whose attribute ends with the value.

In the following example, any <a> element with an “href” that **ends with “.pdf”** will be selected. This would allow you to style all links to PDF files.

```
/* CSS selector */
a[href$=".pdf"] { }


<!-- HTML markup -->
<a href="form.pdf">Link</a>
```

Support

The [att$=val] attribute selector is not supported by IE6.

[att*=val] attribute selector (CSS3)

The **[att*=val] attribute** selector is written using an element followed by “[”, followed by the relevant attribute, followed by “*” (asterisk), followed by “=” followed by the relevant value (within no quotes, single quotes or double quotes), followed by “]”.

```css
/* syntax */
E[att*=val] { }
E[att*='val'] { }
E[att*="val"] { }

/* example */
a[href*="abc.com"] { }
```

How does it work?

The [att*=val] attribute selector target any element whose attribute value matches a string.

In the following example, any <a> element with an “href” value that **contains the string of “abc.com”** will be selected.

```
/* CSS selector */
a[href*="abc.com"] { }

<!-- HTML markup -->
<a href="http://www.abc.com">Link</a>
```

Support

The [att*=val] attribute selector is not supported by IE6.

Structural pseudo-classes

Structural pseudo-classes allow you to select elements based on their position within the overall document structure.

Structurally
unstable

Be aware **structural pseudo-classes are considered “unstable”**. If the document structure changes, the structural pseudo-class may be applied to a different element, or no element at all.

Understanding “(n)”

Four of the structural pseudo-class selectors include **“(n)” as part of the syntax**.

```css
/* CSS selectors */
:nth-child(n)
:nth-last-child(n)
:nth-of-type(n)
:nth-last-of-type(n)
```

Authors can use **six different types of values** inside these brackets.

```
/* syntax */
:nth-child(n)
:nth-child([integer]n) /* (3n) */
:nth-child([integer]n+[integer]) /* (3n+1) */
:nth-child([integer]n-[integer]) /* (3n-1) */
:nth-child(odd)
:nth-child(even)
```

We will talk about these six options in more detail as we cover each selector. For now, **let's look at how the "3n", "3n+1" and "3n-1" work**.

The “n” is an algebraic expression that represents **a set of all integers, in this case starting at zero**.

The value **“3n”** is equivalent to “**(3 x n)**”, which means it would target the following elements:

(3 x 0) = 0 = No element
(3 x 1) = 3 = 3rd element
(3 x 2) = 6 = 6th element
(3 x 3) = 9 = 9th element

The value **“3n+1”** is equivalent to “**(3 x n) + 1**”, which means it would target the following elements:

(3 x 0) + 1 = 1 = 1st element
(3 x 1) + 1 = 4 = 4rd element
(3 x 2) + 1 = 7 = 7th element
(3 x 3) + 1 = 10 = 10th element

The value **“3n-1”** is equivalent to “**(3 x n) - 1**”, which means it would target the following elements:

(3 x 0) - 1 = 0 = No element
(3 x 1) - 1 = 2 = 2nd element
(3 x 2) - 1 = 5 = 5th element
(3 x 3) - 1 = 8 = 8th element

:first-child
pseudo-class
(CSS2)

The :**first-child pseudo-class** selector is written using an element followed by “:”, followed by “first-child”.

```css
/* syntax */
E:first-child { }

/* example */
li:first-child { }
```

How does it work?

The :first-child pseudo-class selector targets elements that are the first child of some other element.

In the following example, only the **first <li> element will be selected**.

```css
/* CSS selector */
li:first-child { }
```

```html
<!-- HTML markup -->
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
</ul>
```

1st child

Support

The :first-child selector is not supported in IE6. IE7 and IE8 have buggy support.

Internet Explorer 7

If an element is added dynamically as a first-child, IE7 does not update the styles to target this new element.

Internet Explorer 8

If an element is added dynamically as a first-child, IE8 will style both the previous first-child element and the newly inserted first-child element, until focus is removed from the previous first-child.

:last-child pseudo-class (CSS3)

The **:last-child pseudo-class** selector is written using an element followed by “:”, followed by “last-child”.

```
/* syntax */
E:last-child { }

/* example */
li:last-child { }
```

How does it work?

The :last-child pseudo-class selector targets elements that are the last child of some other element.

In the following example, only the **last <li> element will be selected**.

```
/* CSS selector */
li:last-child { }


<!-- HTML markup -->
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
</ul>
```

Last child

Support

The :last-child pseudo-class selector is not supported by IE6, IE7 or IE8.

:only-child pseudo-class (CSS3)

The **:only-child pseudo-class** selector is written using an element followed by “:”, followed by “only-child”.

```css
/* syntax */
E:only-child { }

/* example */
em:only-child { }
```

How does it work?

The :only-child pseudo-class selector targets elements that are the only child of some other element.

In the following example, **any <em> that is an only child** will be selected.

```
/* CSS selector */
em:only-child { }


<!-- HTML markup -->
<p>
    <em>1</em>
</p>
<p>
    <em>1</em><em>2</em>
</p>
```

Only child

Support

The :only-child pseudo-class selector is not supported by IE6, IE7 or IE8.

:first-of-type pseudo-class (CSS3)

The **:first-of-type pseudo-class** selector is written using an element followed by “:”, followed by “first-of-type”.

```
/* syntax */
E:first-of-type { }

/* example */
p:first-of-type { }
```

How does it work?

The :first-of-type pseudo-class selector targets elements that are the first of its type within a parent.

In the following example, only the **first <p> element** will be selected.

```
/* CSS selector */
p:first-of-type { }


<!-- HTML markup -->
<p>Content</p>
<p>Content</p>
<p>Content</p>
```

Support

The :first-of-type pseudo-class selector is not supported by IE6, IE7 or IE8.

:last-of-type pseudo-class (CSS3)

The **:last-of-type pseudo-class** selector is written using an element followed by “:”, followed by “last-of-type”.

```css
/* syntax */
E:last-of-type { }

/* example */
p:last-of-type { }
```

How does it work?

The :last-of-type pseudo-class selector targets elements that are the last of its type within a parent.

In the following example, only the **last <p> element** will be selected.

```
/* CSS selector */
p:last-of-type { }

<!-- HTML markup -->
<p>Content</p>
<p>Content</p>
<p>Content</p>
```

Support

The :last-of-type pseudo-class selector is not supported by IE6, IE7 or IE8.

:only-of-type
pseudo-class
(CSS3)

The **:only-of-type pseudo-class** selector is written using an element followed by “:”, followed by “only-of-type”.

```css
/* syntax */
E:only-of-type { }

/* example */
p:only-of-type { }
```

How does it work?

The :only-of-type pseudo-class selector targets elements that are the only element of that type within the parent.

In the following example, the <p> that is the **only element of its type** within a parent will be selected.

```
/* CSS selector */
p:only-of-type { }


<!-- HTML markup -->
<div>Content</div>
<div>Content</div>
<p>Content</p>
```

Support

The :only-of-type pseudo-class selector is not supported by IE6, IE7 or IE8.

:nth-child(n) pseudo-class (CSS3)

The **:nth-child pseudo-class** selector is written using an element followed by “:”, followed by “nth-child”, followed by brackets. The brackets can contain six different types of values.

```css
/* syntax */
E:nth-child(n) { }

/* example */
li:nth-child(4) { }
```

How does it work?

The :nth-child pseudo-class selector targets specific child elements within the parent. The six different types of values used inside the brackets dictate which child elements will be targeted.

Option 1: The brackets can contain an integer value.

```css
/* CSS selector */
li:nth-child(3) { }
```

```
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>
```

3rd child

Option 2: The brackets can contain an integer value and the letter “n”.

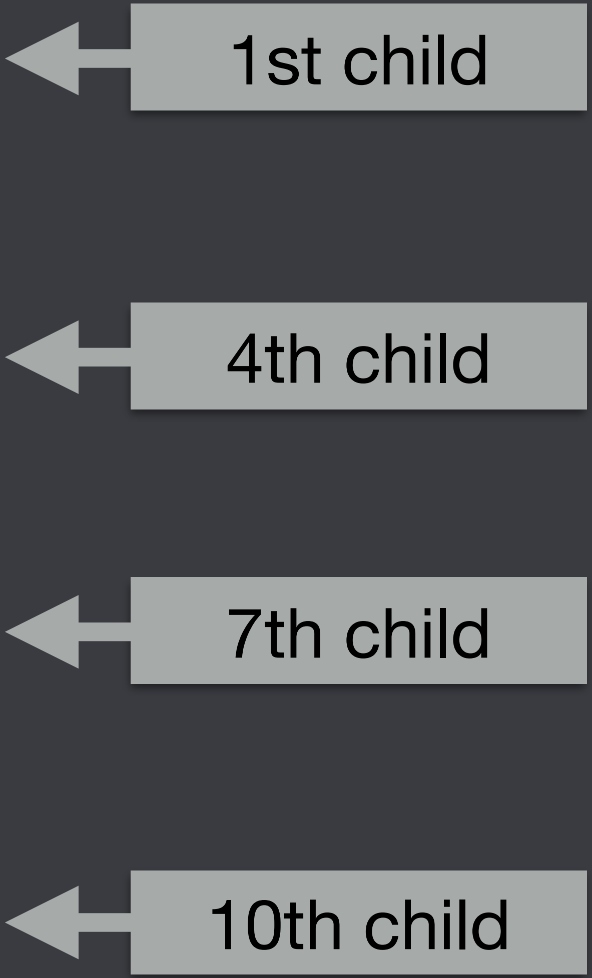
```css
/* CSS selector */
li:nth-child(3n) { }

/*
(3 x 0) = 0 = No element
(3 x 1) = 3 = 3rd element
(3 x 2) = 6 = 6th element
(3 x 3) = 9 = 9th element
(3 x 4) = 12 = 12th element
(3 x 5) = 15 = 15th element
*/
```

```
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>
    <li></li>      3rd child
    <li></li>
    <li></li>
    <li></li>      6th child
    <li></li>
    <li></li>
    <li></li>      9th child
    <li></li>
</ul>
```

Option 3: The brackets can contain an integer value, the letter “n”, a plus (+) symbol and another integer value.

```
/* CSS selector */
li:nth-child(3n+1) { }

/*
(3 x 0) + 1 = 1 = 1st element
(3 x 1) + 1 = 4 = 4th element
(3 x 2) + 1 = 7 = 6th element
(3 x 3) + 1 = 10 = 10th element
(3 x 4) + 1 = 13 = 13th element
(3 x 5) + 1 = 16 = 16th element
*/
```

```
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>
```


1st child
4th child
7th child
10th child

Option 4: The brackets can contain an integer value, the letter “n”, a minus (-) symbol and another integer value.

```
/* CSS selector */
li:nth-child(3n-1) { }

/*
(3 x 0) - 1 = -1 = No element
(3 x 1) - 1 = 2 = 2nd element
(3 x 2) - 1 = 5 = 5th element
(3 x 3) - 1 = 8 = 8th element
(3 x 4) - 1 = 11 = 11th element
(3 x 5) - 1 = 14 = 14th element
*/
```

```html
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>
```

2nd child

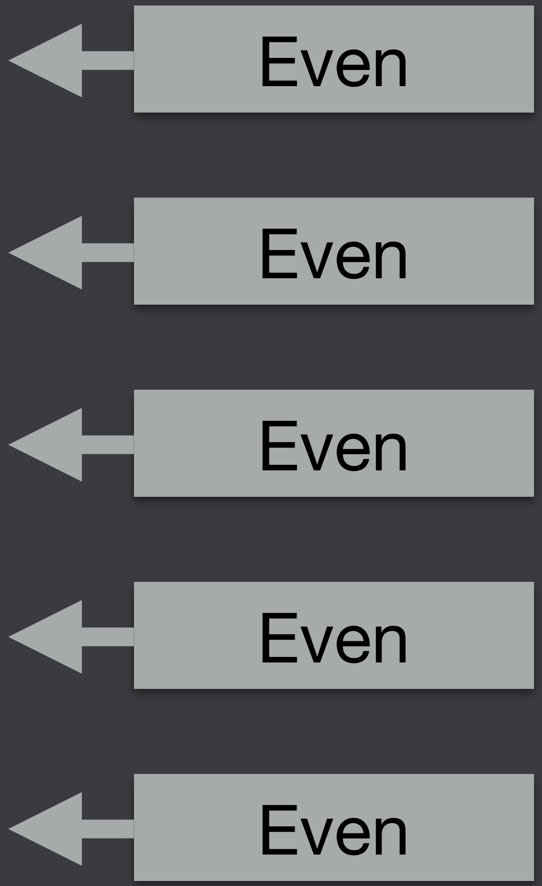
5th child

8th child

Option 5: The brackets can contain the “even” keyword.

```css
/* CSS selector */
li:nth-child(even) { }
```

```html
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>
```



Option 6: The brackets can contain the “odd” keyword.

```css
/* CSS selector */
li:nth-child(odd) { }
```

```
<!-- HTML markup -->
<ul>
    <li></li>      Odd
    <li></li>
    <li></li>      Odd
    <li></li>
    <li></li>      Odd
    <li></li>
    <li></li>      Odd
    <li></li>
    <li></li>      Odd
    <li></li>
</ul>
```

Support

The :nth-child pseudo-class selector is not supported by IE6, IE7 or IE8.

:nth-last-child(n) pseudo-class (CSS3)

The **:nth-last-child pseudo-class** selector is written using an element followed by “:”, followed by “nth-last-child”, followed by brackets. The brackets can contain six different types of values.

```
/* syntax */
E:nth-last-child(n) { }

/* example */
li:nth-last-child(2) { }
```

How does it work?

The :nth-last-child pseudo-class selector targets specific child elements within the parent. The six different types of values used inside the brackets dictate which child elements will be targeted.

Option 1: The brackets can contain an integer value.

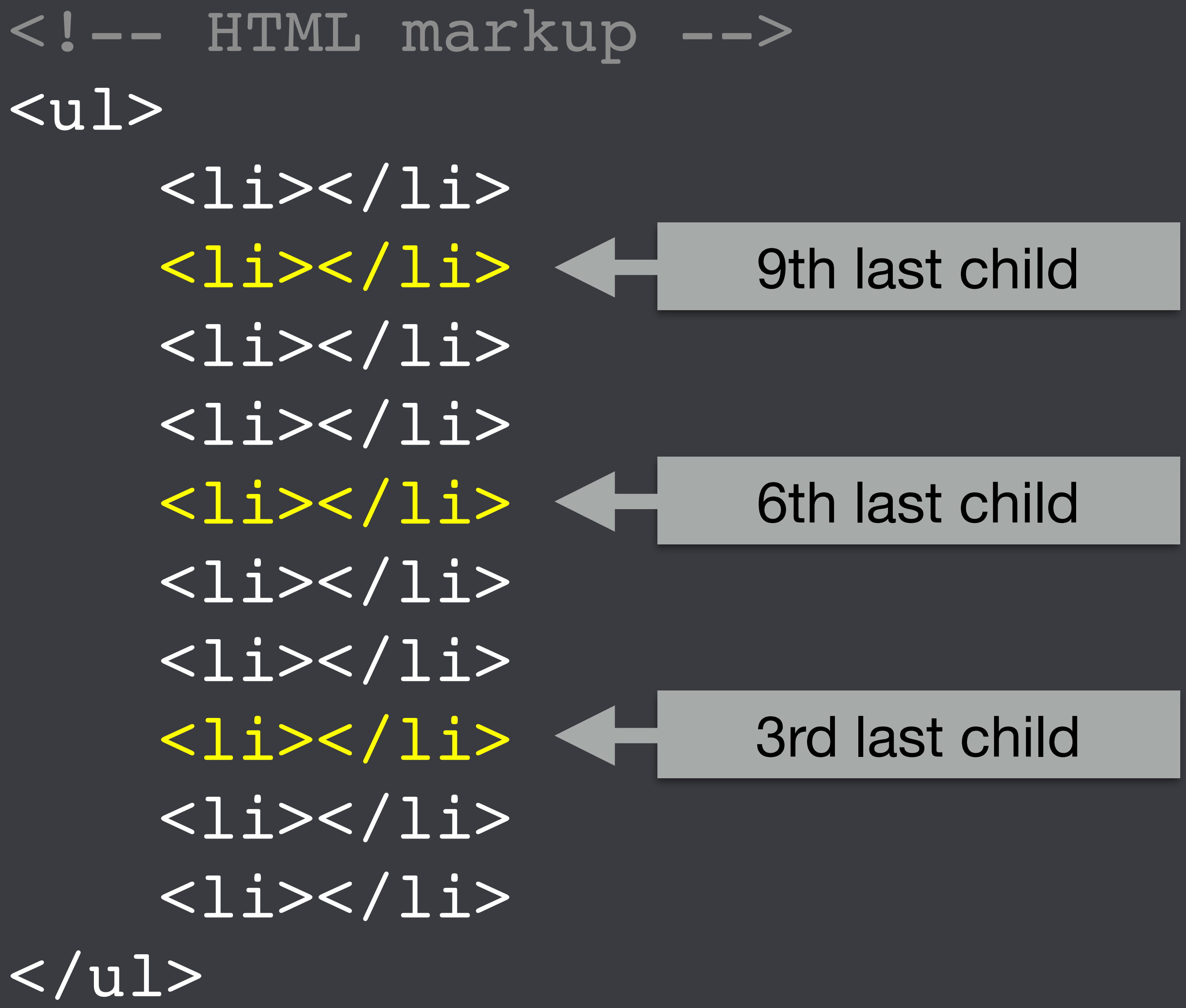
```css
/* CSS selector */
li:nth-last-child(3) { }
```

```html
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>
```

3rd last child

Option 2: The brackets can contain an integer value and the letter “n”.

```
/* CSS selector */
li:nth-last-child(3n) { }

/*
(3 x 0) = 0 = No element
(3 x 1) = 3 = 3rd last element
(3 x 2) = 6 = 6th last element
(3 x 3) = 9 = 9th last element
(3 x 4) = 12 = 12th last element
(3 x 5) = 15 = 15th last element
*/
```



```
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>
```

9th last child
6th last child
3rd last child

Option 3: The brackets can contain an integer value, the letter “n”, a plus (+) symbol and another integer value.

```css
/* CSS selector */
li:nth-last-child(3n+1) { }

/*
(3 x 0) + 1 = 1 = Last element
(3 x 1) + 1 = 4 = 4th last element
(3 x 2) + 1 = 7 = 6th last element
(3 x 3) + 1 = 10 = 10th last element
(3 x 4) + 1 = 13 = 13th last element
(3 x 5) + 1 = 16 = 16th last element
*/
```

```
<!-- HTML markup -->
<ul>
    <li></li>     10th last child
    <li></li>
    <li></li>
    <li></li>     7th last child
    <li></li>
    <li></li>
    <li></li>     4th last child
    <li></li>
    <li></li>
    <li></li>     Last child
</ul>
```

Option 4: The brackets can contain an integer value, the letter “n”, a minus (-) symbol and another integer value.

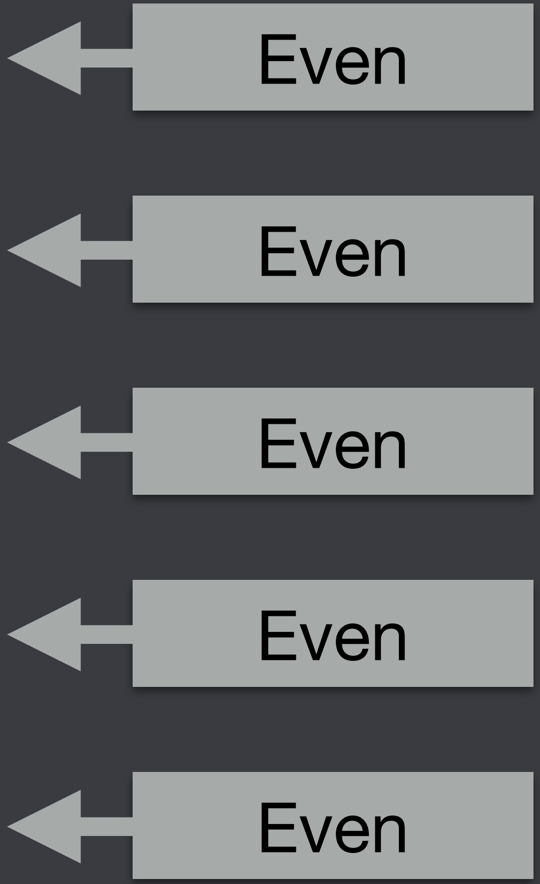
```
/* CSS selector */
li:nth-last-child(3n-1) { }

/*
(3 x 0) - 1 = -1 = No element
(3 x 1) - 1 = 2 = 2nd last element
(3 x 2) - 1 = 5 = 5th last element
(3 x 3) - 1 = 8 = 8th last element
(3 x 4) - 1 = 11 = 11th last element
(3 x 5) - 1 = 14 = 14th last element
*/
```

```
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>
    <li></li>      8th last child
    <li></li>
    <li></li>
    <li></li>      5th last child
    <li></li>
    <li></li>
    <li></li>      2nd last child
    <li></li>
</ul>
```

Option 5: The brackets can contain the “even” keyword.

```css
/* CSS selector */
li:nth-last-child(even) { }
```

```
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>
```


Even
Even
Even
Even
Even

Option 6: The brackets can contain the “odd” keyword.

```css
/* CSS selector */
li:nth-last-child(odd) { }
```

```
<!-- HTML markup -->
<ul>
    <li></li>
    <li></li>        Odd
    <li></li>
    <li></li>        Odd
    <li></li>
    <li></li>        Odd
    <li></li>
    <li></li>        Odd
    <li></li>
    <li></li>        Odd
</ul>
```

Support

The :nth-last-child pseudo-class selector is not supported by IE6, IE7 or IE8.

:nth-of-type(n) pseudo-class (CSS3)

The **:nth-of-type pseudo-class** selector is written using an element followed by “:”, followed by “nth-of-type”, followed by brackets. The brackets can contain six different types of values.

```
/* syntax */
E:nth-of-type(n) { }

/* example */
li:nth-of-type(2) { }
```

How does it work?

The :nth-of-type pseudo-class selector targets specific child elements within the parent. The six different types of values used inside the brackets dictate which child elements will be targeted.

Option 1: The brackets can contain an integer value.

```css
/* CSS selector */
p:nth-of-type(3) { }
```

```
<!-- HTML markup -->
<div>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
</div>
```

3rd <p> element

Option 2: The brackets can contain an integer value and the letter “n”.

```
/* CSS selector */
p:nth-of-type(3n) { }

/*
(3 x 0) = 0 = No element
(3 x 1) = 3 = 3rd <p> element
(3 x 2) = 6 = 6th <p> element
(3 x 3) = 9 = 9th <p> element
(3 x 4) = 12 = 12th <p> element
(3 x 5) = 15 = 15th <p> element
*/
```

```
<!-- HTML markup -->
<div>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
</div>
```

3rd <p> element

6th <p> element

Option 3: The brackets can contain an integer value, the letter “n”, a plus (+) symbol and another integer value.

```
/* CSS selector */
p:nth-of-type(3n+1) { }

/*
(3 x 0) + 1 = 1 = 1st <p> element
(3 x 1) + 1 = 4 = 4th <p> element
(3 x 2) + 1 = 7 = 6th <p> element
(3 x 3) + 1 = 10 = 10th <p> element
(3 x 4) + 1 = 13 = 13th <p> element
(3 x 5) + 1 = 16 = 16th <p> element
*/
```

```
<!-- HTML markup -->
<div>
    <p></p>    ← 1st <p> element
    <p></p>
    <div></div>
    <p></p>
    <p></p>    ← 4th <p> element
    <p></p>
    <div></div>
    <p></p>
    <p></p>    ← 7th <p> element
    <p></p>
</div>
```

Option 4: The brackets can contain an integer value, the letter “n”, a minus (-) symbol and another integer value.

```css
/* CSS selector */
p:nth-of-type(3n-1) { }

/*
(3 x 0) - 1 = -1 = No element
(3 x 1) - 1 = 2 = 2nd <p> element
(3 x 2) - 1 = 5 = 5th <p> element
(3 x 3) - 1 = 8 = 8th <p> element
(3 x 4) - 1 = 11 = 11th <p> element
(3 x 5) - 1 = 14 = 14th <p> element
*/
```

```
<!-- HTML markup -->
<div>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
</div>
```

2nd <p> element

5th <p> element

8th <p> element

Option 5: The brackets can contain the “even” keyword.

```css
/* CSS selector */
p:nth-of-type(even) { }
```

```
<!-- HTML markup -->
<div>
    <p></p>
    <p></p>          Even <p> elements
    <div></div>
    <p></p>
    <p></p>          Even <p> elements
    <p></p>
    <div></div>
    <p></p>          Even <p> elements
    <p></p>
    <p></p>          Even <p> elements
</div>
```

Option 6: The brackets can contain the “odd” keyword.

```
/* CSS selector */
p:nth-of-type(odd) { }
```

```
<!-- HTML markup -->
<div>
    <p></p>  ←  Odd <p> elements
    <p></p>
    <div></div>
    <p></p>  ←  Odd <p> elements
    <p></p>
    <p></p>  ←  Odd <p> elements
    <div></div>
    <p></p>
    <p></p>  ←  Odd <p> elements
    <p></p>
</div>
```

Support

The :nth-of-type pseudo-class selector is not supported by IE6, IE7 or IE8.

:nth-last-of-type(n) pseudo-class (CSS3)

The **:nth-last-of-type pseudo-class** selector is written using an element followed by “:”, followed by “nth-last-of-type”, followed by brackets. The brackets can contain six different types of values.

```
/* syntax */
E:nth-last-of-type(n) { }

/* example */
li:nth-last-of-type(4) { }
```

How does it work?

The :nth-last-of-type pseudo-class selector targets specific child elements within the parent. The six different types of values used inside the brackets dictate which child elements will be targeted.

Option 1: The brackets can contain an integer value.

```css
/* CSS selector */
p:nth-last-of-type(3) { }
```

```
<!-- HTML markup -->
<div>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
</div>
```

3rd last <p> element

Option 2: The brackets can contain an integer value and the letter “n”.

```css
/* CSS selector */
p:nth-last-of-type(3n) { }

/*
(3 x 0) = 0 = No element
(3 x 1) = 3 = 3rd last <p> element
(3 x 2) = 6 = 6th last <p> element
(3 x 3) = 9 = 9th last <p> element
(3 x 4) = 12 = 12th last <p> element
(3 x 5) = 15 = 15th last <p> element
*/
```

```html
<!-- HTML markup -->
<div>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
</div>
```

6th last <p> element

3rd last <p> element

Option 3: The brackets can contain an integer value, the letter “n”, a plus (+) symbol and another integer value.

```css
/* CSS selector */
p:nth-last-of-type(3n+1) { }

/*
(3 x 0) + 1 = 1 = Last <p> element
(3 x 1) + 1 = 4 = 4th last <p> element
(3 x 2) + 1 = 7 = 7th last <p> element
(3 x 3) + 1 = 10 = 10th last <p> element
(3 x 4) + 1 = 13 = 13th last <p> element
(3 x 5) + 1 = 16 = 16th last <p> element
*/
```

```
<!-- HTML markup -->
<div>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
</div>
```

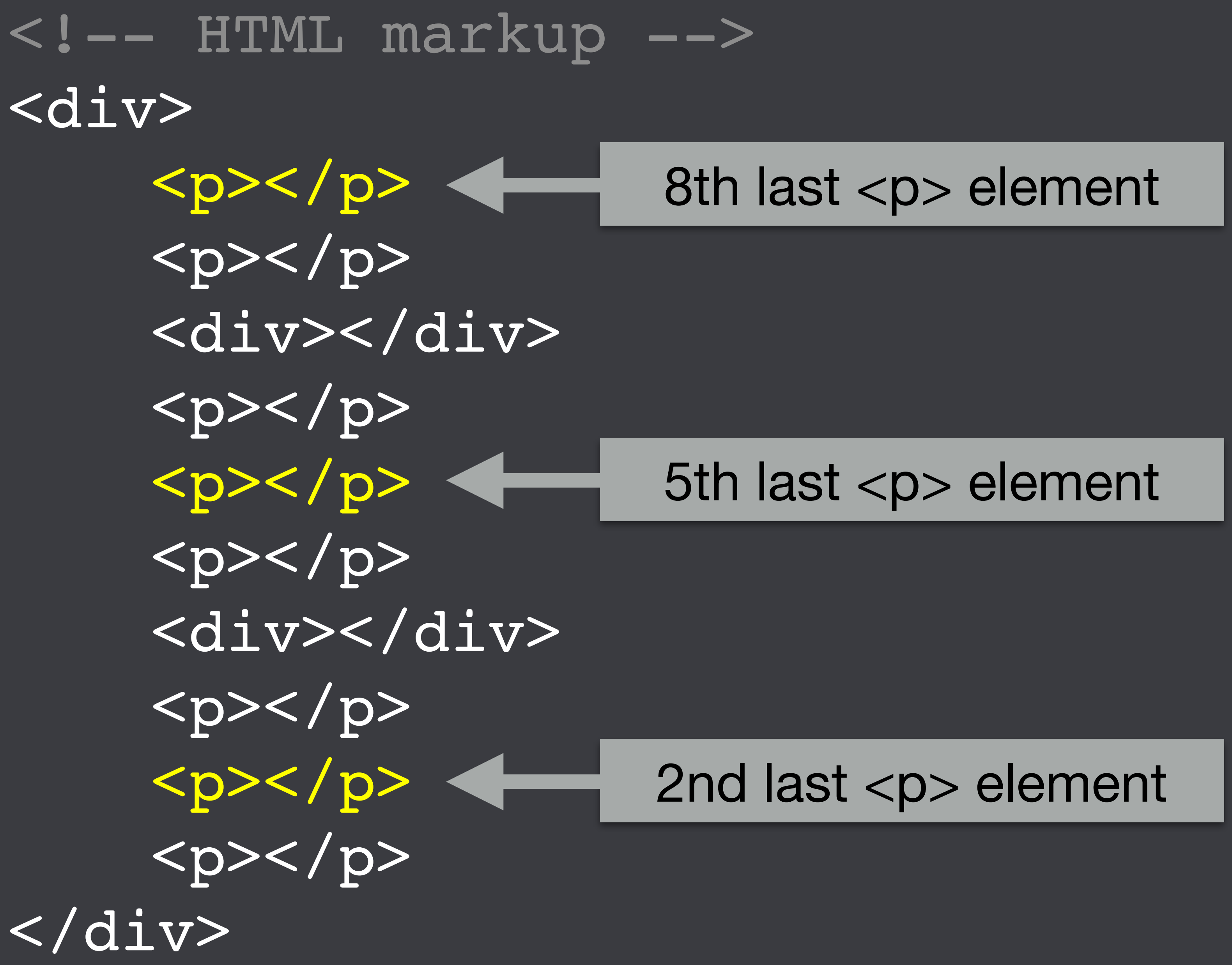
7th last <p> element

4th last <p> element

Last <p> element

Option 4: The brackets can contain an integer value, the letter “n”, a minus (-) symbol and another integer value.

```css
/* CSS selector */
p:nth-last-of-type(3n-1) { }

/*
(3 x 0) - 1 = -1 = No element
(3 x 1) - 1 = 2 = 2nd last <p> element
(3 x 2) - 1 = 5 = 5th last <p> element
(3 x 3) - 1 = 8 = 8th last <p> element
(3 x 4) - 1 = 11 = 11th last <p> element
(3 x 5) - 1 = 14 = 14th last <p> element
*/
```



```
<!-- HTML markup -->
<div>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
    <div></div>
    <p></p>
    <p></p>
    <p></p>
</div>
```

8th last <p> element
5th last <p> element
2nd last <p> element

Option 5: The brackets can contain the “even” keyword. The example below targets every “even” <p> child element starting from the end.

```css
/* CSS selector */
p:nth-last-of-type(even) { }
```

```
<!-- HTML markup -->
<div>
    <p></p>          Even <p> elements
    <p></p>
    <div></div>
    <p></p>          Even <p> elements
    <p></p>
    <p></p>          Even <p> elements
    <div></div>
    <p></p>
    <p></p>          Even <p> elements
    <p></p>
</div>
```

Option 6: The brackets can contain the “odd” keyword. The example below targets every “odd” <p> child element starting from the end.

```css
/* CSS selector */
p:nth-last-of-type(odd) { }
```

```
<!-- HTML markup -->
<div>
    <p></p>
    <p></p>   ←  Odd <p> elements
    <div></div>
    <p></p>
    <p></p>   ←  Odd <p> elements
    <p></p>
    <div></div>
    <p></p>   ←  Odd <p> elements
    <p></p>
    <p></p>   ←  Odd <p> elements
</div>
```

Support

The :nth-last-of-type pseudo-class selector is not supported by IE6, IE7 or IE8.

:root pseudo-class (CSS3)

The **:root pseudo-class** selector is written using a ":", followed by "root".

```css
/* syntax */
:root { }
```

How does it work?

The :root pseudo-class selector targets the document root element. In HTML documents, the root element is always the HTML element.

In the following example, **the root element** (the <html> element) will be selected.

```
/* css selector */
:root { }


<!-- HTML markup -->
<!DOCTYPE html>
<html>
<head>
    <title>Title - site name</title>
</head>
<body>
</body>
</html>
```

Support

The :root pseudo-class selector is not supported by IE6, IE7 or IE8.

:empty pseudo-class (CSS3)

The **:empty pseudo-class** selector is written using an element, followed by “:”, followed by “empty”.

```
/* syntax */
E:empty { }

/* example */
p:empty { }
```

How does it work?

The :empty pseudo-class selector targets elements that have no children (no element, text nodes or even character spaces).

In the following example, only the **empty <p> element** will be selected. The <p> that does not contain content, but does contain a character space, is not selected as it is not considered empty.

```
/* CSS selector */
p:empty { }


<!-- HTML markup -->
<p>Content</p>
<p></p>
<p>Content</p>
<p> </p>
```

Support

The :empty pseudo-class selector is not supported by IE6, IE7 or IE8.

User-interface pseudo-classes

User interface pseudo-classes allow you to style various aspects of form-related elements.

:disabled
pseudo-class
(CSS3)

The **:disabled pseudo-class** selector is written using an element followed by “:”, followed by “disabled”.

```
/* syntax */
E:disabled { }

/* example */
input:disabled { }
```

How does it work?

The :disabled pseudo-class selector targets form elements that use the “disabled” attribute.

In the following example, only the input element that has **a “disabled” attribute will be selected**.

```
/* CSS selector */
input:disabled { }


<!-- HTML markup -->
<input type="text" disabled>
<input type="text">
```

Support

The :disabled pseudo-class selector is not supported by IE6, IE7 or IE8.

:enabled pseudo-class (CSS3)

The **:enabled pseudo-class** selector is written using an element followed by “:”, followed by “enabled”.

```
/* syntax */
E:enabled { }

/* example */
input:enabled { }
```

How does it work?

The :enabled pseudo-class selector targets form elements that do not have a “disabled” attribute.

In the following example, the input element that **does not have a “disabled” attribute** will be selected.

```
/* CSS selector */
input:enabled { }


<!-- HTML markup -->
<input type="text" disabled>
<input type="text">
```

Support

The :enabled pseudo-class selector is not supported by IE6, IE7 or IE8.

:checked pseudo-class (CSS3)

The **:checked pseudo-class** selector is written using an element followed by “:”, followed by “checked”.

```
/* syntax */
E:checked { }

/* example */
input[type=radio]:checked { }
```

How does it work?

The :checked pseudo-class selector targets form elements (such as radio buttons or checkboxes) that use the “checked” attribute.

In the following example, the input element that **has a “checked” attribute** will be selected.

```
/* CSS selector */
input:checked { }


<!-- HTML markup -->
<input type="radio" checked>
<input type="radio">
```

Support

The :checked pseudo-class selector is not supported by IE6, IE7 or IE8.

:indeterminate pseudo-class (CSS3)

The **:indeterminate pseudo-class** selector is written using an element, followed by “:”, followed by “indeterminate”.

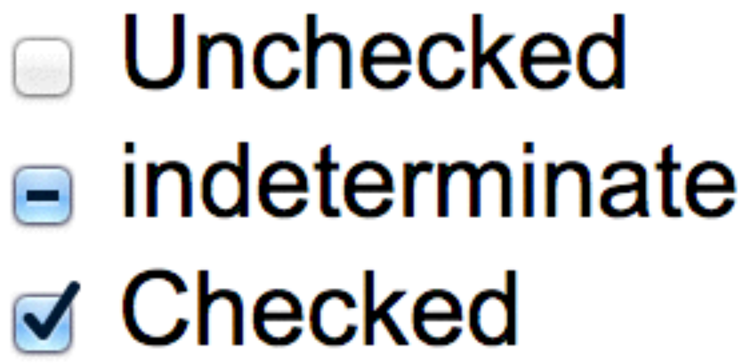
```
/* syntax */
E:indeterminate { }

/* example */
input:indeterminate { }
```

Different states

From a markup perspective, checkbox inputs **can only have two states**: checked or unchecked.

```html
<!-- checkbox inputs -->
<input type="checkbox">
<input type="checkbox" checked>
```

However, checkbox inputs can have **three visible states**: checked, unchecked, or indeterminate.


Unchecked
indeterminate
Checked

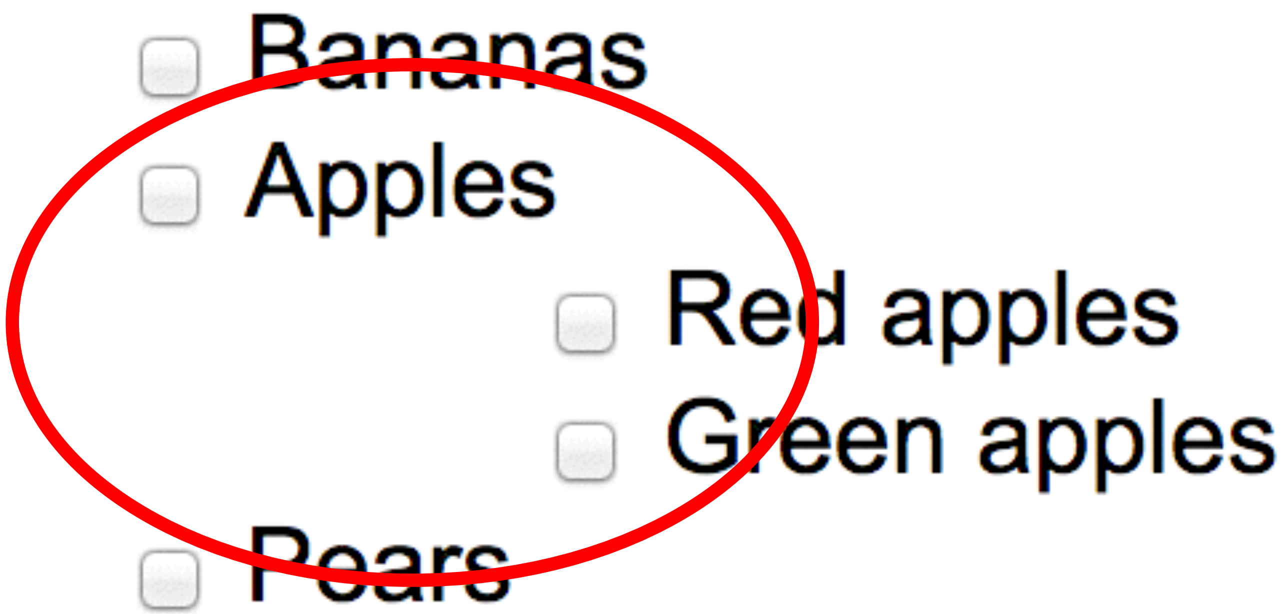
The only way to make a checkbox display in the indeterminate state is **by using JavaScript or JQuery**.

```
// JavaScript
var checkbox = document.getElementById("my-
checkbox");
checkbox.indeterminate = true;

// JQuery
$(“#my-checkbox").prop("indeterminate",
true);
```

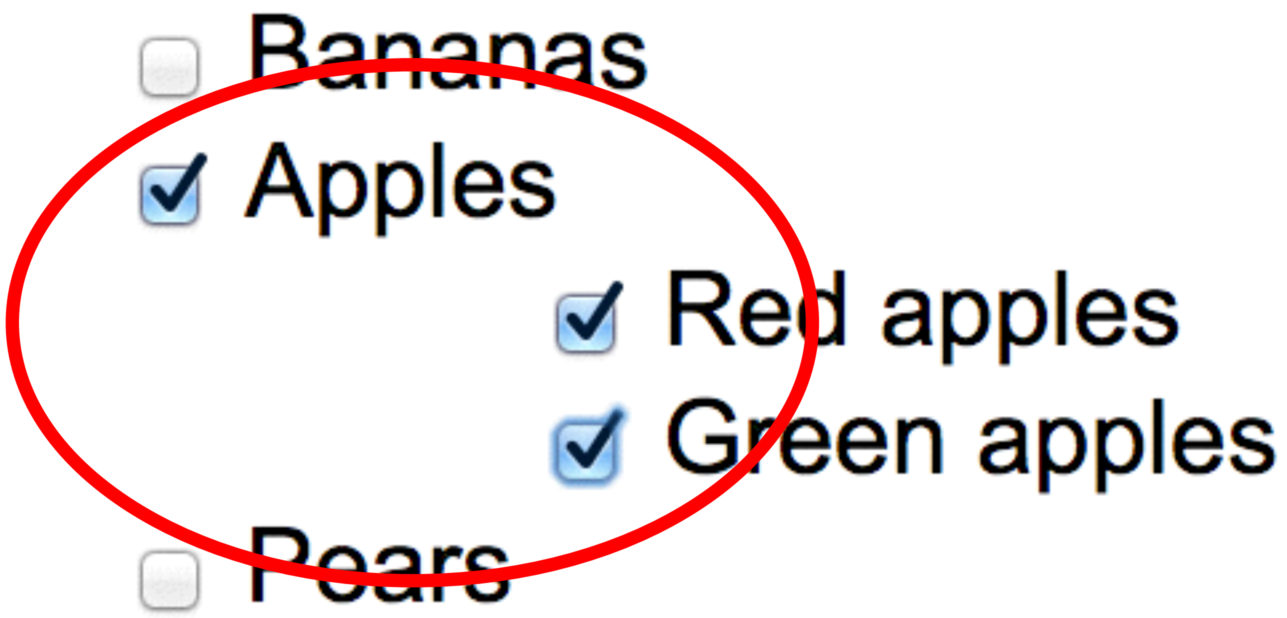
Why use indeterminate?

Let’s look at an example where you may have a series of checkbox inputs that have **child checkbox inputs**.

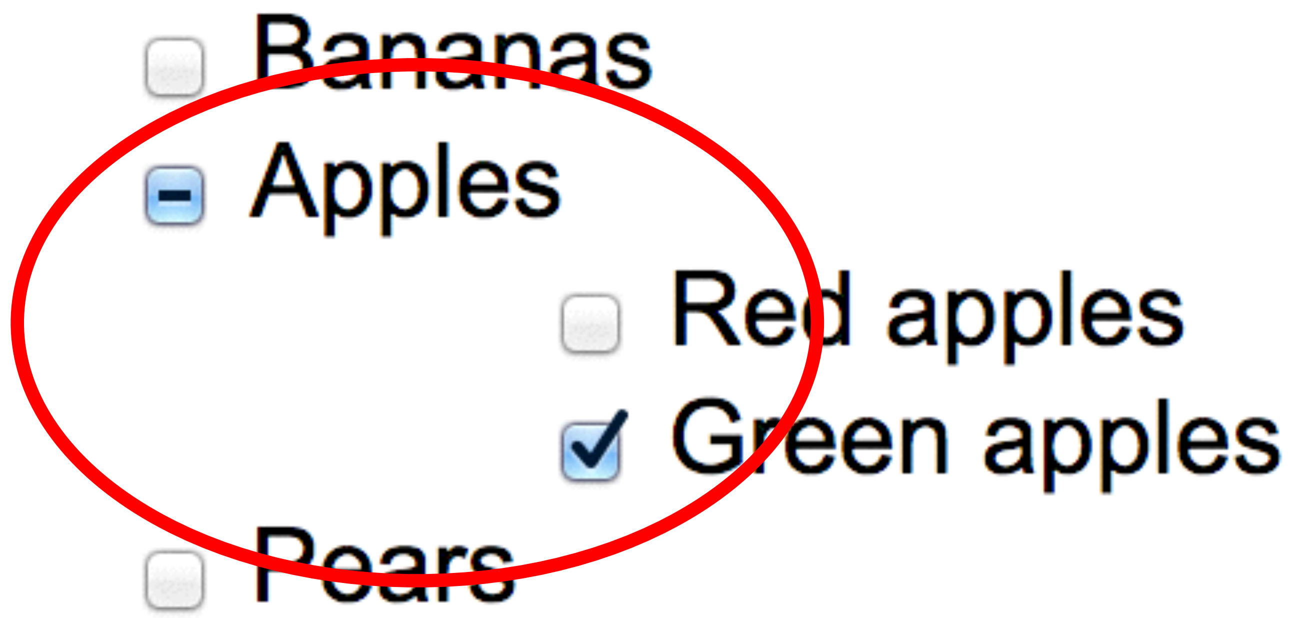
Option 1: If none of the child inputs are checked, the parent input **should also be displayed as unchecked**.


Bananas
Apples
Red apples
Green apples
Pears

Option 2: If all the child inputs are checked, the parent input **should also be displayed as checked**.


Bananas
Apples
Red apples
Green apples
Pears

Option 3: If only some of the child inputs are checked, the parent input **could be displayed as indeterminate**.


Bananas
Apples
Red apples
Green apples
Pears

Support

The :indeterminate pseudo-class selector is not supported by IE6, IE7 or IE8.

:default pseudo-class (CSS3)

The **:default pseudo-class** selector is written using an element followed by “:”, followed by “default”.

```
/* syntax */
E:default { }

/* example */
option:default { }
```

How does it work?

The :default pseudo-class selector targets any form element falling into one of the following four categories:

1. Button elements that are their form’s “default” button (the first button in tree order, within a form).

```
/* syntax */
button:default { }

<!-- HTML markup -->
<form method="get">
    <button>Submit</button>
</form>
```

2. Input elements whose type attribute is either “submit” or “image”, and that are their form’s default button (the first submit or image input element in tree order, within a form).

```
/* syntax */
input[type="submit"]:default { }

<!-- HTML markup -->
<form method="get">
    <input type="submit" id="s1">
</form>
```

3. Input elements (radio or checkboxes) that have a checked attribute.

```
/* syntax */
input[type="radio"]:default { }

<!-- HTML markup -->
<form method="get">
    <input id="r1" type="radio" checked>
    <input id="r2" type="radio">
</form>
```

4. Option elements that have a selected attribute.

```
/* syntax */
option:default { }

<!-- HTML markup -->
<form method="get">
    <select id="s1">
        <option value="1" selected></option>
        <option value="2"></option>
    </select>
</form>
```

Support 1

The :default pseudo-class selector is not supported by IE6 - IE11.

Support 2

Webkit-based browsers (Chrome, Safari and Opera) do not support the :default pseudo-class on radio inputs, checkbox inputs or <option> elements.

Support 3

Firefox does not support the :default pseudo-class on <option> elements.

:required pseudo-class (CSS3)

The **:required pseudo-class** selector is written using an element followed by “:”, followed by “required”.

```
/* syntax */
E:required { }

/* example */
input:required { }
```

How does it work?

The :required pseudo-class selector targets form elements that use the “required” attribute.

In the following example, only the input element **that has a “required” attribute** will be selected.

```
/* CSS selector */
input:required { }


<!-- HTML markup -->
<input type="text" required>
<input type="text">
```

Support

The :required pseudo-class selector is not supported by IE6 - IE9.

:optional pseudo-class (CSS3)

The **:optional pseudo-class** selector is written using an element followed by “:”, followed by “optional”.

```
/* syntax */
E:optional { }

/* example */
input:optional { }
```

How does it work?

The :optional pseudo-class selector targets form elements that do not use the “required” attribute.

In the following example, only the input element that **does not have a “required” attribute** will be selected.

```
/* CSS selector */
input:option { }


<!-- HTML markup -->
<input type="text" required>
<input type="text">
```

Support

The :optional pseudo-class selector is not supported by IE6 - IE9.

:valid pseudo-class (CSS3)

The **:valid pseudo-class** selector is written using an element, followed by “:”, followed by “valid”.

```
/* syntax */
E:valid { }

/* example */
input:valid { }
```

How does it work?

The :valid pseudo-class selector targets elements that are determined to be valid (ie. user inputs match the relevant pattern attributes).

In the following example, only the input element **that has valid input by the user** will be selected.

```
/* CSS selector */
input:valid { }


<!-- HTML markup -->
<input pattern="[0-9]{2}">
<input type="text">
```

Support

The :valid pseudo-class selector is not supported by IE6 - IE9.

:invalid pseudo-class (CSS3)

The **:invalid pseudo-class** selector is written using an element, followed by ":", followed by "invalid".

```css
/* syntax */
E:invalid { }

/* example */
input:invalid { }
```

How does it work?

The :invalid pseudo-class selector targets form elements that are required but not filled in, or are determined to be invalid (ie. user input does not match the relevant pattern attributes).

In the following example, only the input element **that has not been filled in or has invalid input by the user** will be selected.

```
/* CSS selector */
input:invalid { }


<!-- HTML markup -->
<input pattern="[0-9]{2}">
<input type="text">
```

Support

The :invalid pseudo-class selector is not supported by IE6 - IE9.

:in-range
pseudo-class
(CSS3)

The **:in-range pseudo-class** selector is written using an element followed by “:”, followed by “in-range”.

```
/* syntax */
E:in-range { }

/* example */
input:in-range { }
```

How does it work?

The :in-range pseudo-class selector targets form elements with range limitations, where the value is within the defined limitations.

In the following example, the :in-range pseudo-class selector would be applied if the user **inserts a number that is “in-range”** (between 1 and 10).

```
/* CSS selector */
input:in-range { }


<!-- HTML markup -->
<input min="1" max="10">
```

Support

The :in-range pseudo-class selector is not supported by IE6 - IE11.

Issues with default behaviour

Let's take an example of an **input element that has has min and max attributes defined**. It has also been given “in-range” and “out-of-range” styles.

```
/* CSS selectors */
.one:in-range { border: 1px solid green; }
.one:out-of-range { border: 1px solid red; }

<!-- Input with min and max attributes -->
<input class="one" type="number" min="1"
max="10">
```

Before a user interacts with this input element, it is **neither in a state of “in-range” or “out-of-range”**.

The question is, **should this “yet-to-be-interacted-with” input be styled as “in-range” or “out-of-range”**? Or, in this case, should it have a red border or green border?

Theoretically, all browsers should display this “yet-to-be-interacted-with” input **based on the “in-range” styles**. So, the input should have a green border.

Unfortunately, if both “in-range” and “out-of-range” states are defined, **Chrome and Opera browsers** will style the “yet-to-be-interacted-with” input based on the last defined style. (Firefox does not have this issue).

So, in Chrome and Opera, if the **“out-of-range” state is defined last, the input's border will be red**, even through the user has not interacted with it yet.

One way around this issue is the write **“out-of-range” rules before “in-range” rules**.

```css
/* CSS selectors */
.one:out-of-range { border: 1px solid red; }
.one:in-range { border: 1px solid green; }
```

:out-of-range pseudo-class (CSS3)

The **:out-of-range pseudo-class** selector is written using an element followed by “:”, followed by “out-of-range”.

```
/* syntax */
E:out-of-range { }

/* example */
input:out-of-range { }
```

How does it work?

The :out-of-range pseudo-class selector targets elements with range limitations, where the value is not within the defined limitations.

In the following example, the :out-of-range selector would be applied if the user **inserts a number that is “out-of-range”** (less that 1 or greater than 10).

```
/* CSS selector */
input:out-of-range { }


<!-- HTML markup -->
<input min="1" max="10">
```

Support

The :out-of-range pseudo-class selector is not supported by IE6 - IE11.

“in-range” vs “out-of-range”

See the in-range selector information for information about how Chrome and Opera apply “in-range” and “out-of-range” rules.

:read-only
pseudo-class
(CSS3)

The **:read-only pseudo-class** selector is written using an element followed by “:”, followed by “read-only”.

```
/* syntax */
E:read-only { }

/* example */
input:read-only { }
```

How does it work?

The :read-only pseudo-class selector targets form elements that use the “readonly” attribute.

In the following example, **only the input element that has a “readonly” attribute** will be selected.

```
/* CSS selector */
input:read-only { }


<!-- HTML markup -->
<input type="text" readonly>
<input type="text">
```

Support

The :read-only pseudo-class selector is not supported by IE6 - IE11.

:read-write pseudo-class (CSS3)

The **:read-write pseudo-class** selector is written using an element followed by “:”, followed by “read-write”.

```
/* syntax */
E:read-write { }

/* example */
input:read-write { }
```

How does it work?

The :read-write pseudo-class selector targets form elements that can be altered by users and elements that do not use the “readonly” attribute.

In the following example, the input element that **has a “readonly” attribute will not be selected**.

```css
/* CSS selector */
input:read-write { }
```

```html
<!-- HTML markup -->
<input type="text" readonly>
<input type="text">
```

Support

The :read-write pseudo-class selector is not supported by IE6 - IE11.

:target pseudo-class (CSS3)

The **:target pseudo-class** selector is written using an element, followed by “:”, followed by “target”.

```css
/* syntax */
E:target { }

/* example */
h2:target { }
```

How does it work?

Some links point to a location within a web document. These links include the “#” (number or pound sign) followed by an anchor identifier.

These links are called “fragment identifiers”. Fragment identifiers link to a specific element within a web document, known as the **target element**.

```html
<!-- HTML markup -->
<a href=“#one">Link down page</a>
<p></p>
<p></p>
<p></p>
<p></p>
<h2 id="one">Target element</h2>
```

The :target pseudo-class selector allows us to select these **“target elements”**.

In the following example, only the <h2> element that has been linked to, **as a target element**, will be selected.

```
/* CSS selector */
h2:target { }


<!-- HTML markup -->
<h2></h2>
<h2 id="one"></h2>
<h2></h2>
```

Support

The :target pseudo-class selector is not supported by IE6, IE7 or IE8.

Pseudo-elements

Pseudo-elements (or "fake" elements) allow you to style elements that are not in the document tree.

Single or double?

First-line and first-letter pseudo-elements are covered in the earlier course “CSS selectors: Getting started”

In CSS1 and CSS2, pseudo-elements were defined using a single “:”. In CSS3, pseudo-elements **are defined using double “::”**.

```css
/* CSS2.1 syntax */
p:first-line { }

/* CSS3 syntax */
p::first-line { }
```

The double “::” notation was introduced to **make it easier to tell the difference** between pseudo-classes and pseudo-elements.

Most browsers support both variations. However, **if you need to support IE6-8**, it may be best to continue using the single “:” notation.

::first-line
pseudo-element
(CSS1)

The **::first-line pseudo-element** selector is written using an element, followed by “:” or “::”, followed by “first-line”.

```css
/* syntax */
E::first-line { }

/* example */
p::first-line { }
```

How does it work?
The ::first-line pseudo-element selector targets the first line of content within the relevant element.

In the following example, the **first line of the paragraph** will be selected.

```
/* CSS selector */
p::first-line { }

<!-- HTML markup —>
<p>This is the first line
of a paragraph of text</p>
```

Support

The ::first-line pseudo-element selector is buggy in IE6 and IE7.

::first-letter pseudo-element (CSS1)

The **::first-letter pseudo-element** selector is written using an element, followed by ":" or "::", followed by "first-letter".

```css
/* syntax */
E::first-letter { }

/* example */
p::first-letter { }
```

How does it work?
The ::first-letter pseudo-element selector targets the first letter in the content of the relevant element.

In the following example, the **first letter of the paragraph** will be selected.

```
/* syntax */
p::first-letter { }

<!-- HTML markup —>
<p>
   This is the first line
</p>
```

Support

The ::first-letter pseudo-element selector is buggy in IE6 and IE7.

::before
pseudo-element
(CSS2)

The **::before pseudo-element** selector is written using an element, followed by “:” or “::”, followed by “before”.

```css
/* syntax */
E::before { }

/* example */
.intro::before { }
```

The ::before pseudo-element selector is **used in conjunction with the “content” property**.

```
/* the content property */
p::before
{
    content: "hello";
}
```

How does it work?

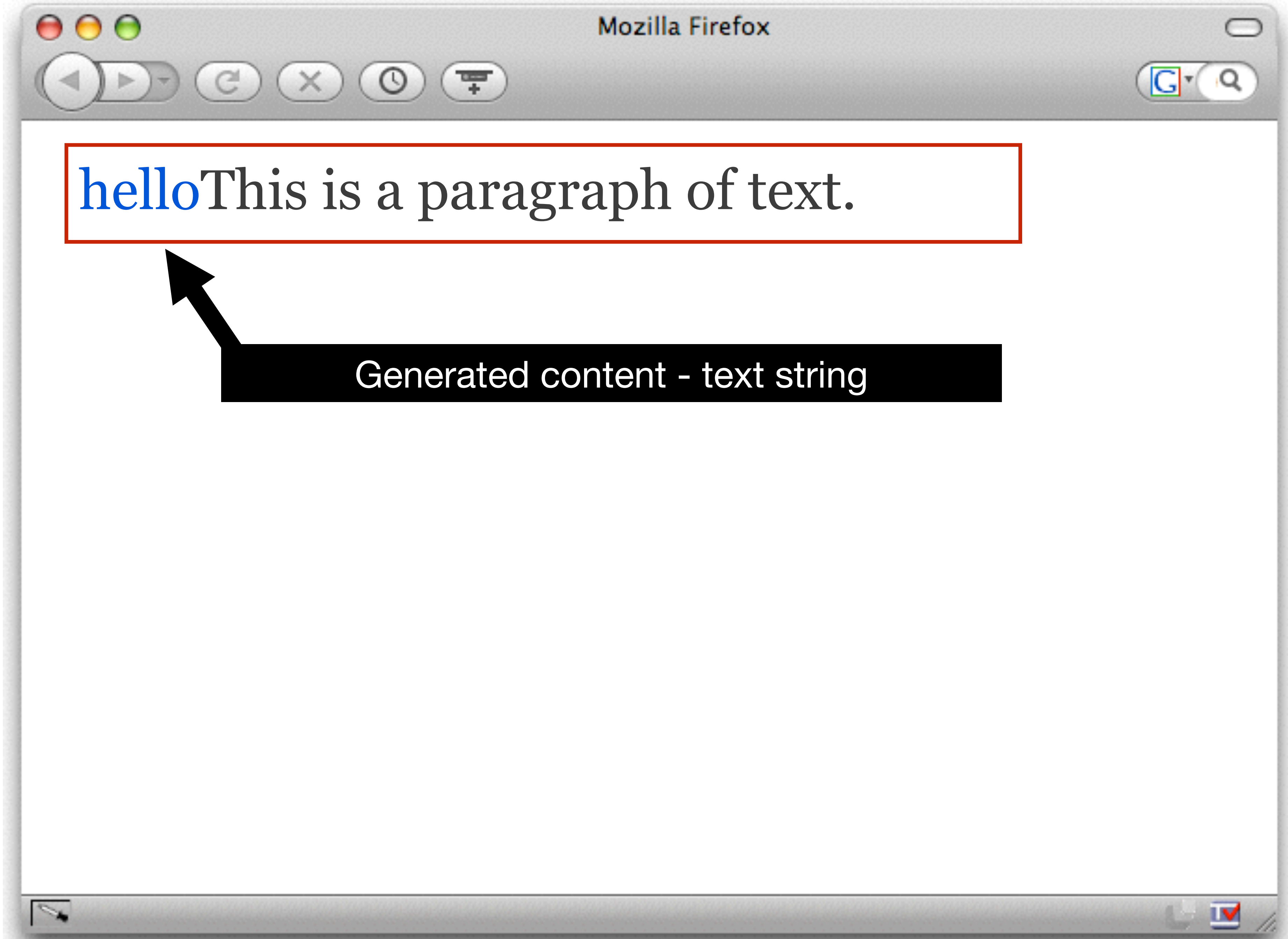
The ::before pseudo-element selector is used to generate content.

This generated content is rendered within the specified element - **before the elements real content**.

```
<!-- HTML markup -->
<p>
   [generated content]content
</p>
```

Option 1: Generated content can include a text string. This will place the generated text string before the element's content.

```
/* Generated content - text string */
p::before { content: "hello"; }
```



helloThis is a paragraph of text.

Generated content - text string

Text strings can be written inside **double quotes or inside single quotes**.

```css
/* single quotes */
p::before { content:'hello'; }
/* double quotes */
p::before { content:"hello"; }
```

Single quotes **can be used inside double quotes** - and visa versa.

```
/* single quotes with double quotes inside */
p::before { content:'a "string"'; }

/* a "string" */


/* double quotes with single quotes inside */
p::before { content:"a 'string'"; }

/* a 'string' */
```

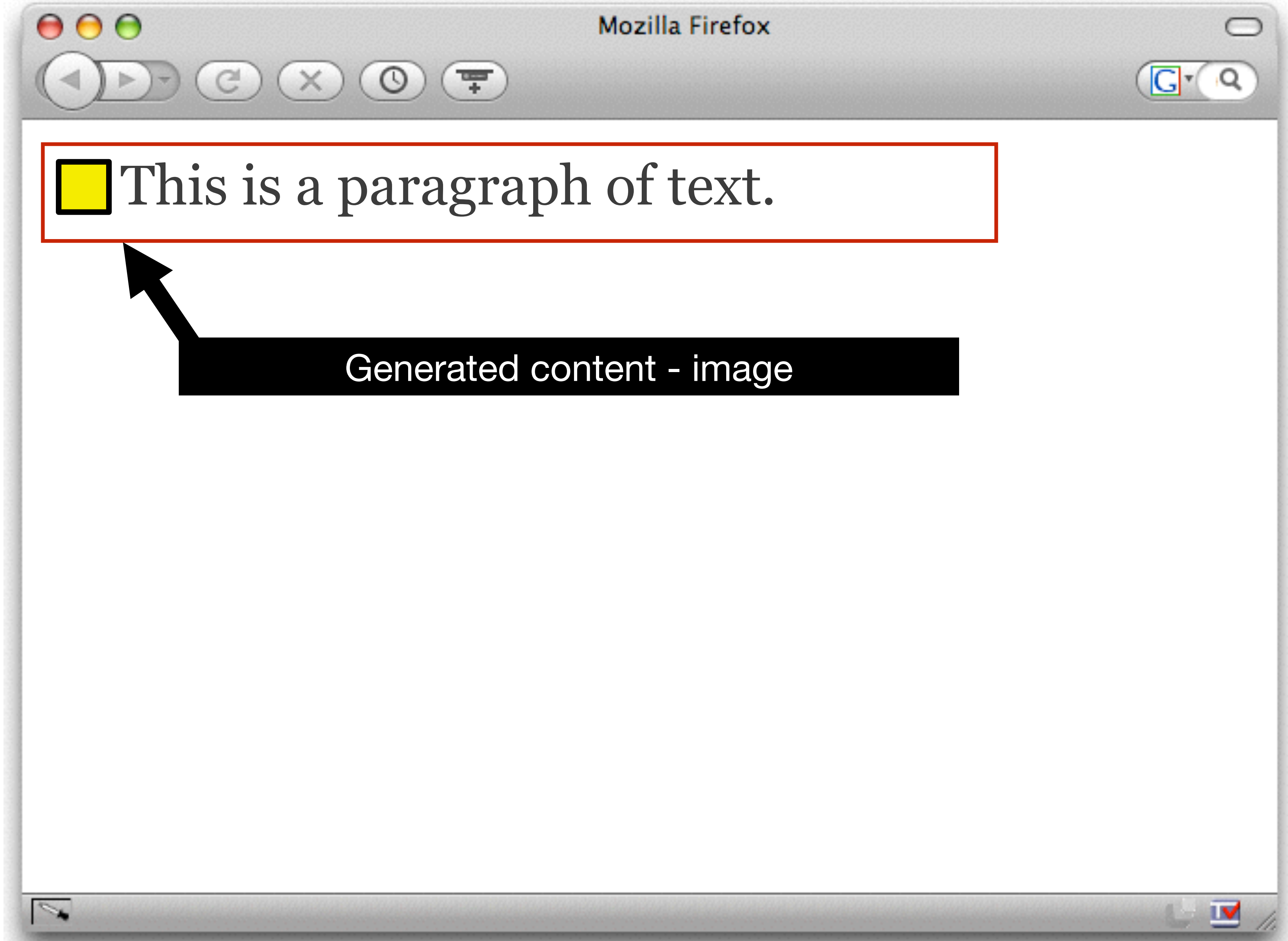
Double quotes cannot occur inside double quotes, unless they are **escaped with a backslash “\” symbol** before the start and end quotation marks - and visa versa.

```
/* escaped singe quotes inside */
p::before { content:'a \'string\''; }

/* a 'string' */


/* escaped double quotes inside */
p::before { content:"a \"string\""; }

/* a "string" */
```

Generated text **should not be used for any meaningful content** as it is totally inaccessible to assistive technologies such as Screen Readers and Refreshable Braille devices.

Option 2: Generated content can include an image. This will place the generated image before the element's content.

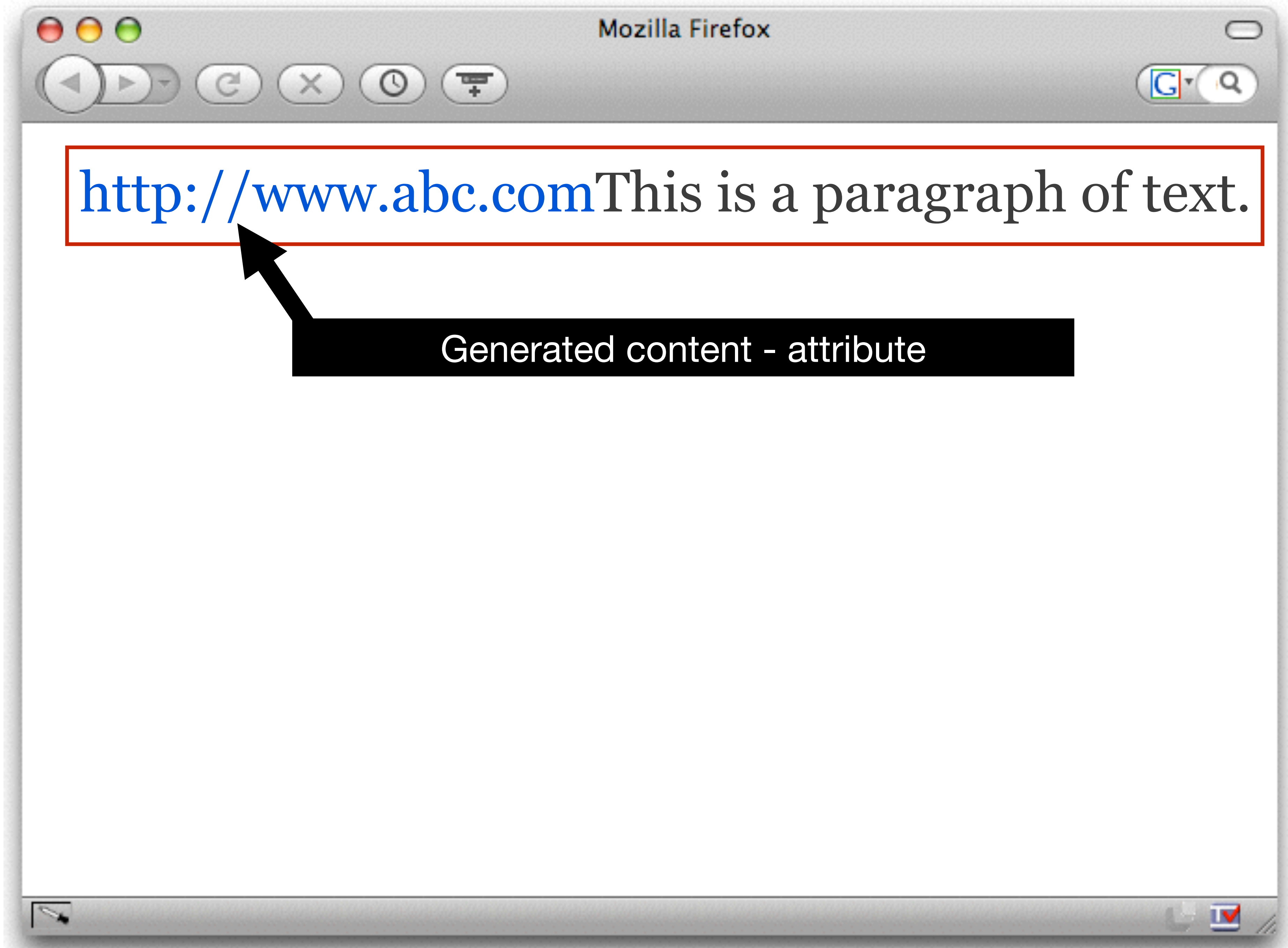
```css
/* Generated content - image */
p::before { content: url(a.gif); }
```


Mozilla Firefox

This is a paragraph of text.

Option 3: Generated content can include an attribute. This will place the value of the attribute as text before the element's content.

```
/* Generated content - attribute value */
p::before { content: attr(cite); }


<!-- HTML markup -->
<blockquote cite="http://www.abc.com">
    This is a paragraph of text.
</blockquote>
```



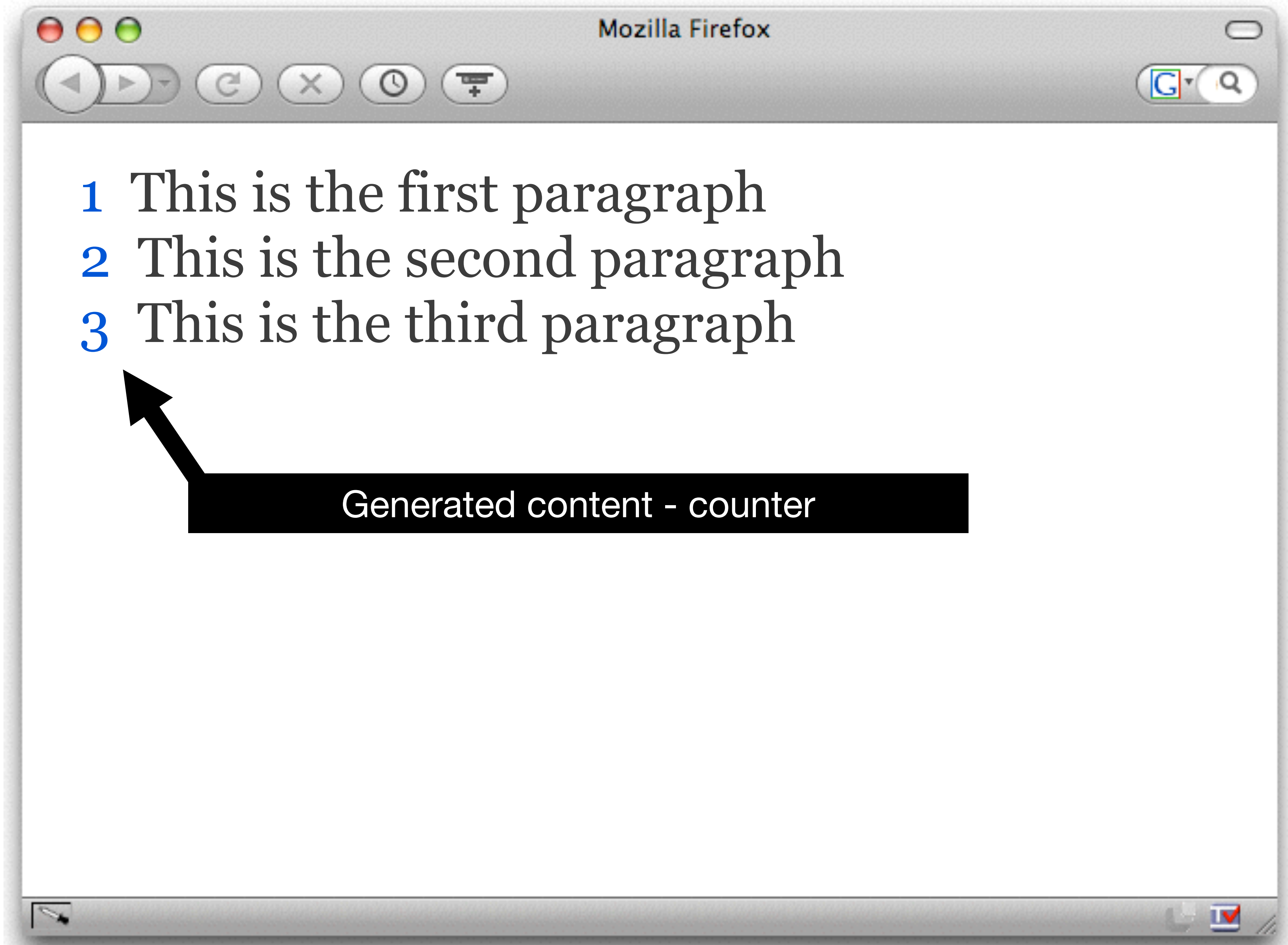
http://www.abc.comThis is a paragraph of text.

Generated content - attribute

Option 4: Generated content can include a counter. This allows authors to change the number order and numbering method of ordered lists, as well as add counter-increments to any element.

```
/* Generated content - counter-increment */
body { counter-reset: section; }

p:before
{
    content: counter(section);
    counter-increment: section;
    color: red;
}
```

```html
<!-- HTML markup -->
<p>
    This is the first paragraph
</p>
<p>
    This is the second paragraph
</p>
<p>
    This is the third paragraph
</p>
```



1 This is the first paragraph

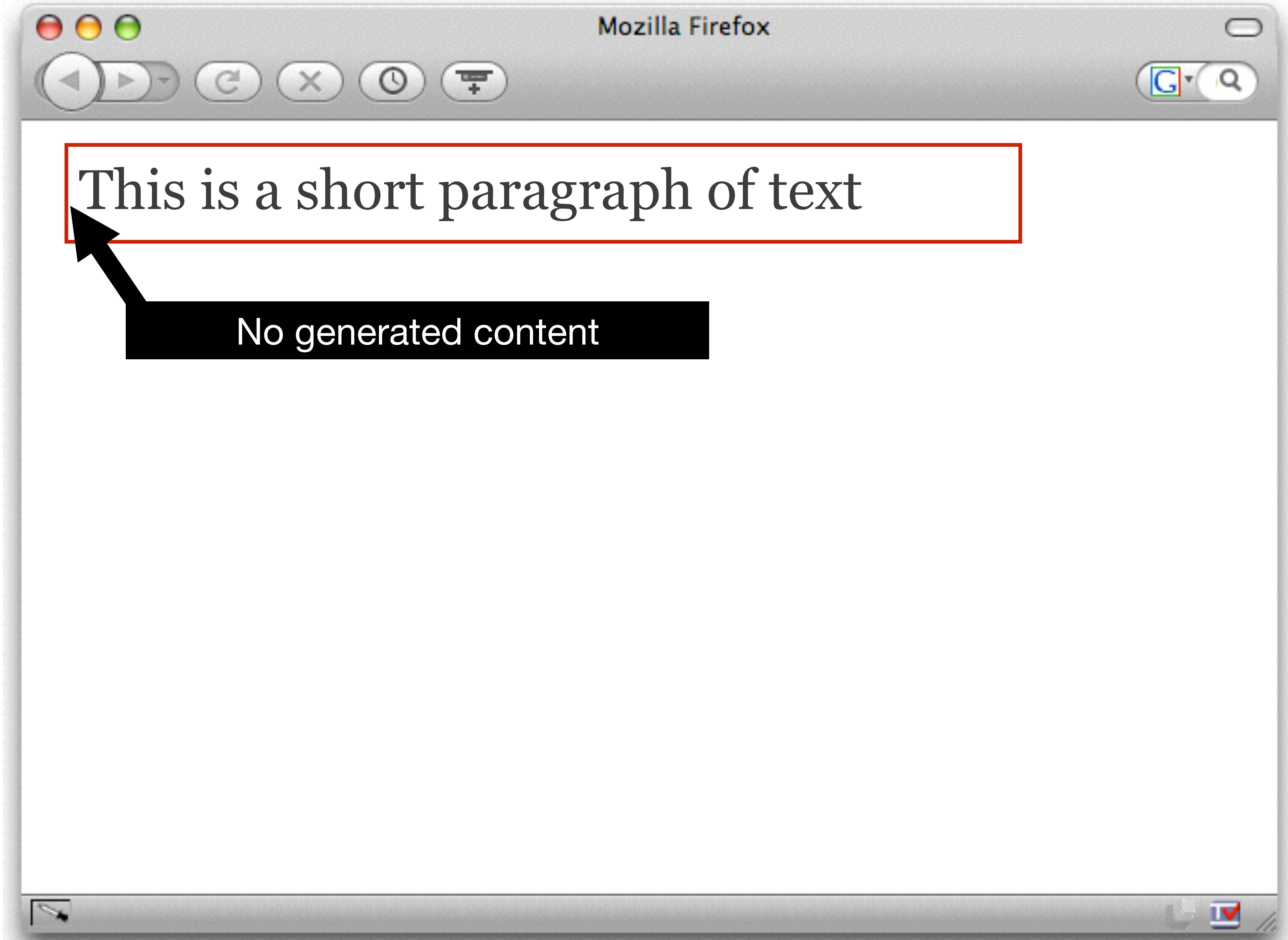
2 This is the second paragraph

3 This is the third paragraph

Generated content - counter

Option 5: Generated content can include nothing. This allows authors to position and style a new generated element.

```
/* Generated content - empty */
p::before { content: ""; }
```


Mozilla Firefox

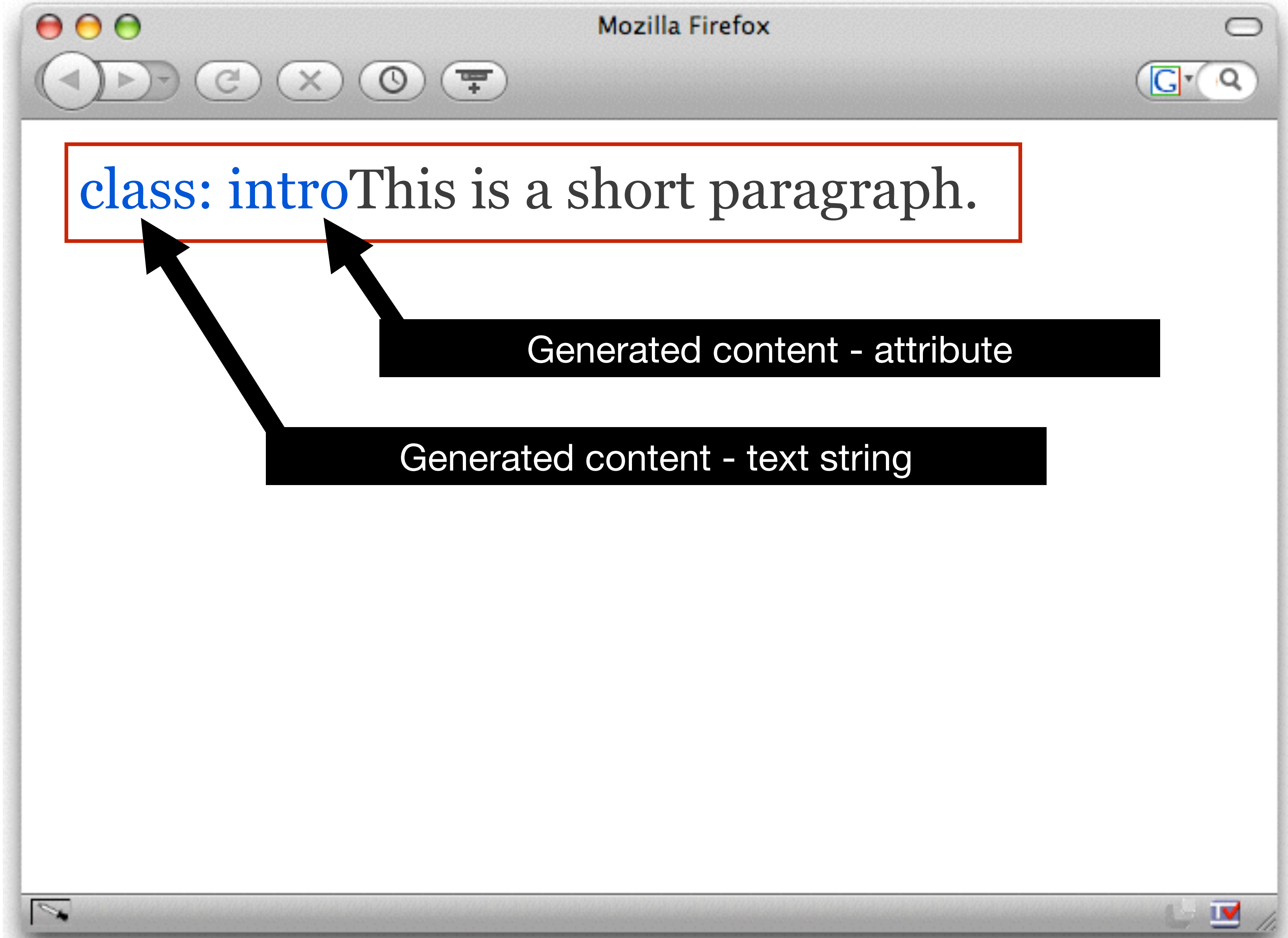
This is a short paragraph of text

No generated content

Generated content can also include **combinations of any of these five different options**.

The following example will render the **generated text string** “class: ” and then render any class names applied to the element.

```css
/* Generated content - combinations */
p::before
{ content: "class: " attr(class); }
```



class: introThis is a short paragraph.

Mozilla Firefox
Generated content - attribute
Generated content - text string

Generated content **cannot include HTML markup**.

```
/* Generated content - markup not allowed */
p::before { content: <p>test<p>; }
```

Support

The ::before pseudo-element selector is not supported by IE6 or IE7.

::after

pseudo-element

(CSS2)

The **::after pseudo-element** selector is written using an element, followed by “:” or “::”, followed by “after”.

```css
/* syntax */
E::after { }

/* example */
.intro::after { }
```

The ::after pseudo-element selector is **used in conjunction with the “content” property**.

```
/* the content property */
p::after
{
    content: "hello";
}
```

How does it work?

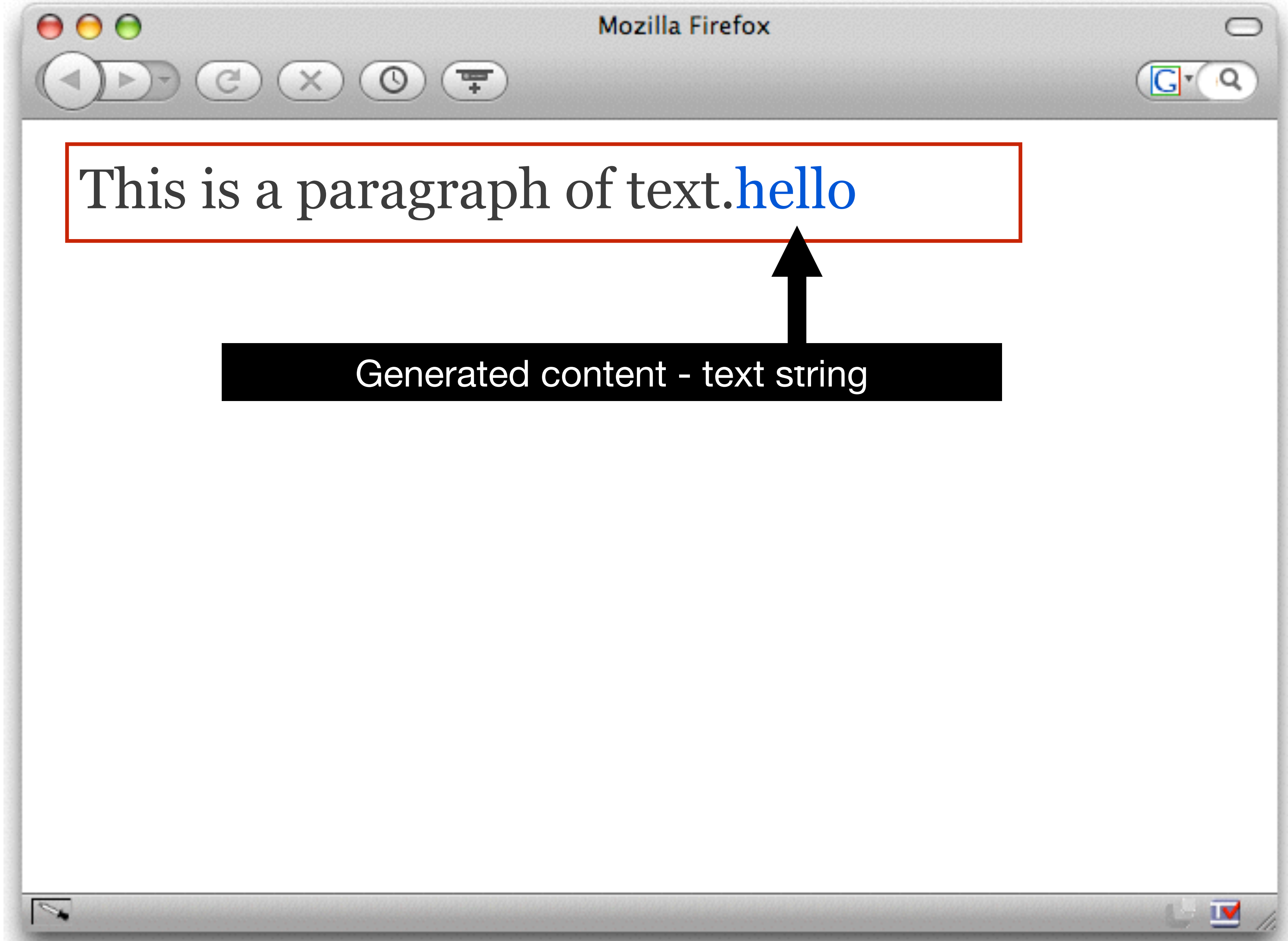
The ::after pseudo-element selector is used to generate content.

This generated content is rendered within the specified element - **after the elements real content**.

```
<!-- HTML markup -->
<p>
   content[generated content]
</p>
```

Option 1: Generated content can include a text string. This will place the generated text string after the element's content.

```css
/* Generated content - text string */
p::after { content: "hello"; }
```



This is a paragraph of text.hello

Generated content - text string

Text strings can be written inside **double quotes or inside single quotes**.

```css
/* single quotes */
p::after { content:'hello'; }
/* double quotes */
p::after { content:"hello"; }
```

Single quotes **can be used inside double quotes** - and visa versa.

```
/* single quotes with double quotes inside */
p::after { content:'a "string"'; }

/* a "string" */


/* double quotes with single quotes inside */
p::after { content:"a 'string'"; }

/* a 'string' */
```

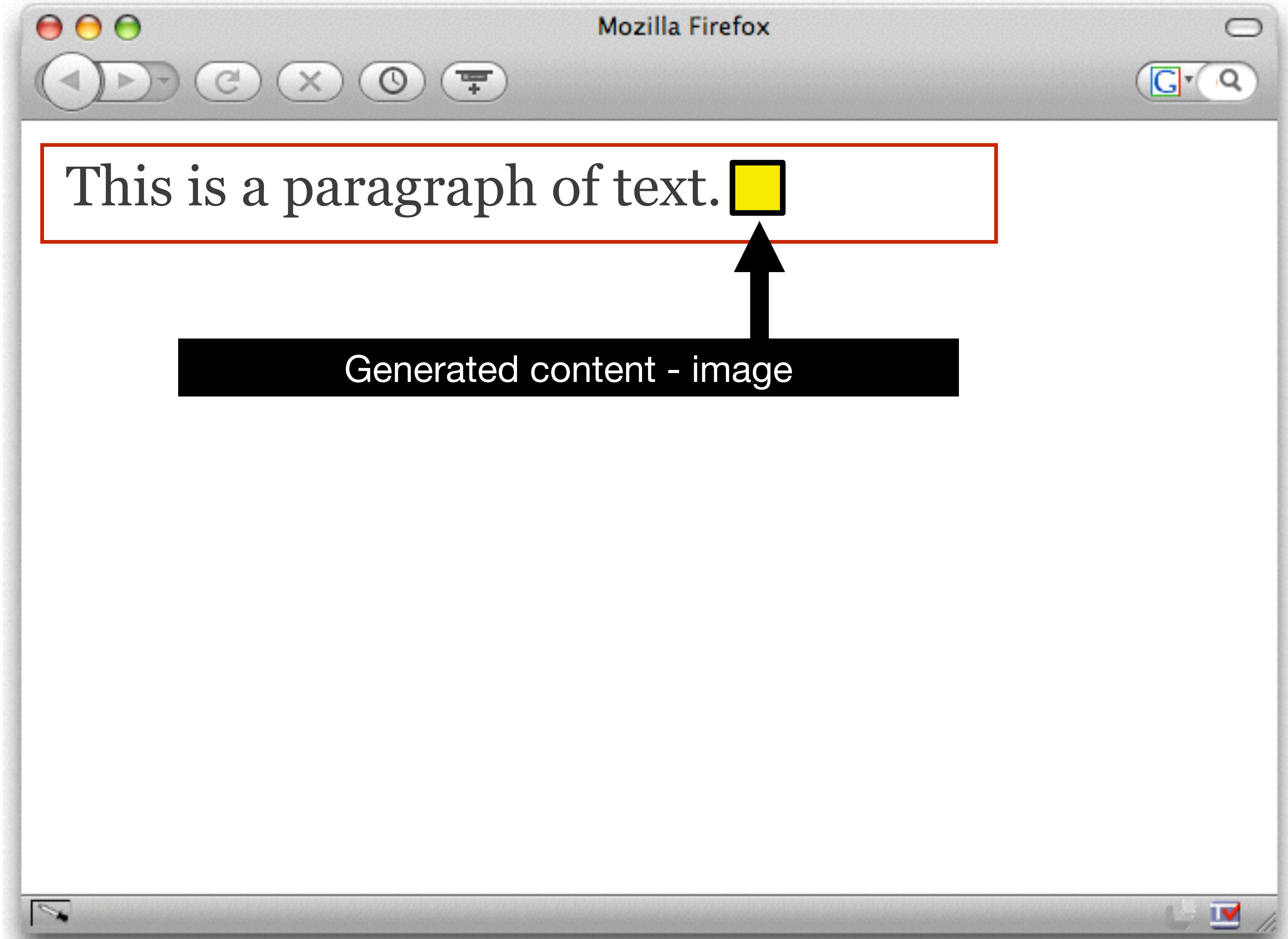
Double quotes cannot occur inside double quotes, unless they are **escaped with a backslash “\” symbol** before the start and end quotation marks - and visa versa.

```
/* escaped singe quotes inside */
p::after { content:'a \'string\''; }

/* a 'string' */


/* escaped double quotes inside */
p::after { content:"a \"string\""; }

/* a "string" */
```

Generated text **should not be used for any meaningful content** as it is totally inaccessible to assistive technologies such as Screen Readers and Refreshable Braille devices.

Option 2: Generated content can include an image. This will place the generated image after the element's content.

```css
/* Generated content - image */
p::after { content: url(a.gif); }
```

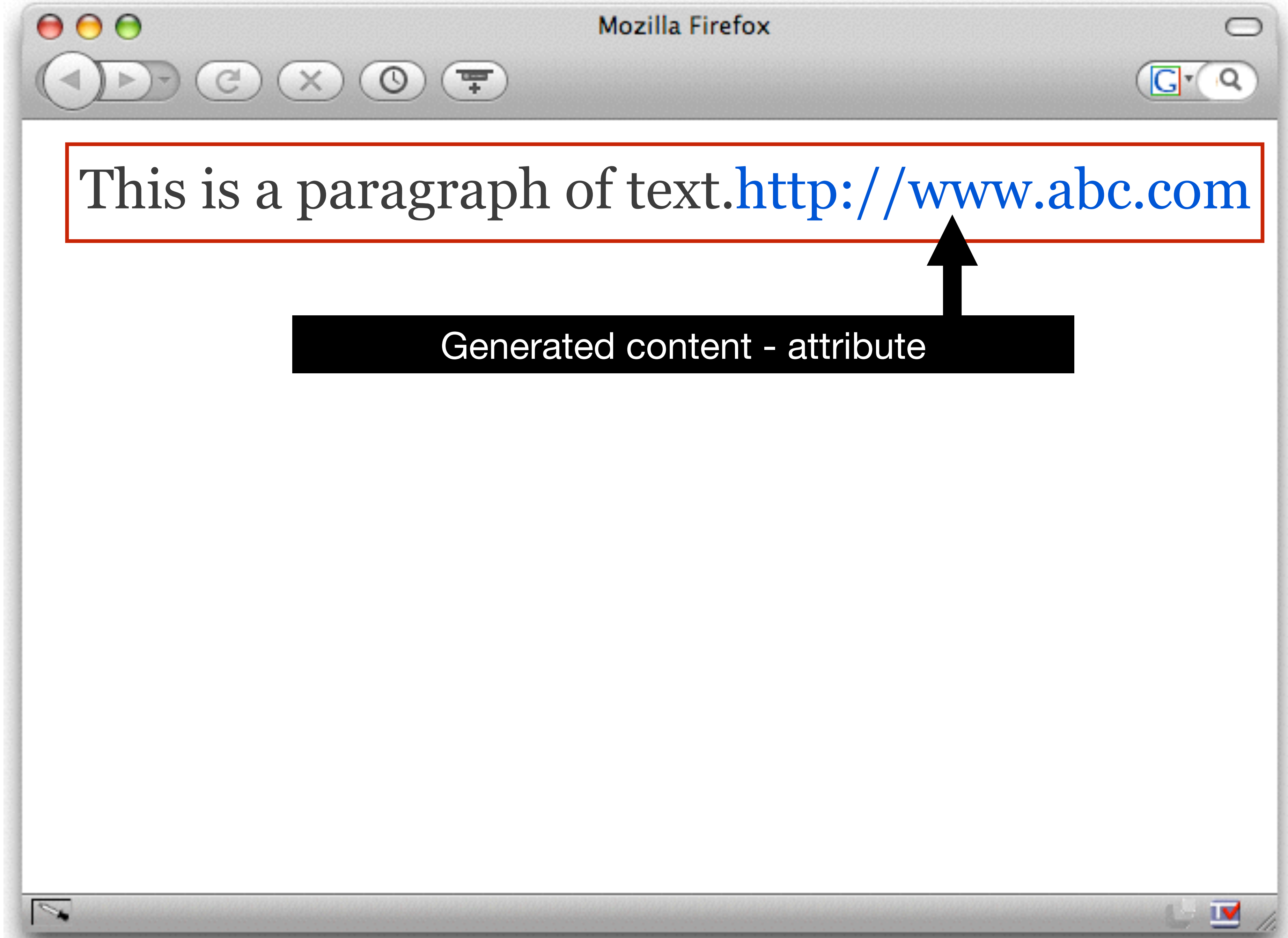


This is a paragraph of text.

Generated content - image

Option 3: Generated content can include an attribute. This will place the value of the attribute as text after the element's content.

```
/* Generated content - attribute value */
p::after { content: attr(cite); }


<!-- HTML markup -->
<blockquote cite="http://www.abc.com">
    This is a paragraph of text.
</blockquote>
```

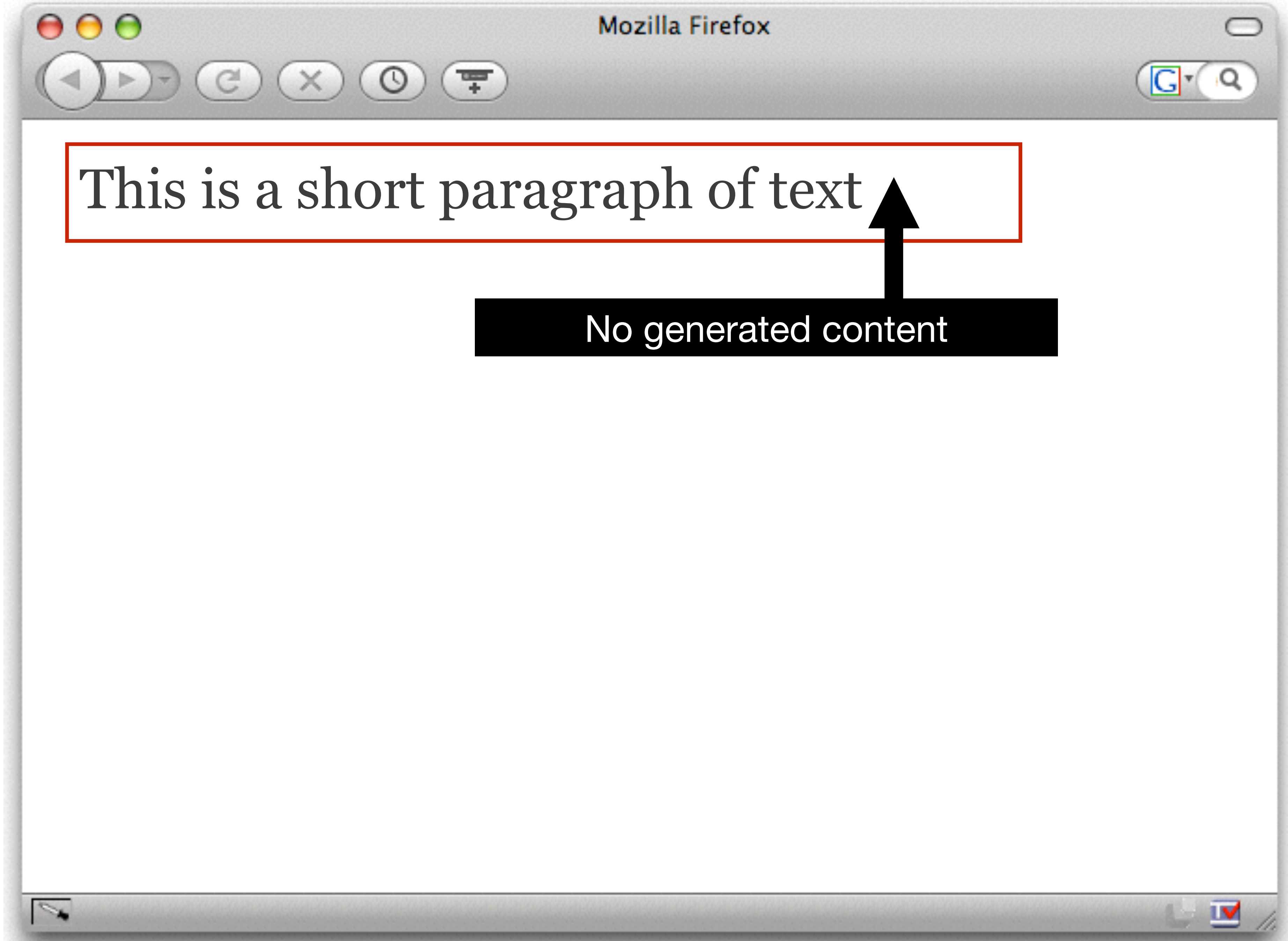


This is a paragraph of text.http://www.abc.com

Generated content - attribute

Option 4: Generated content can include nothing. This allows authors to position and style a new generated element.

```css
/* Generated content - empty */
p::after { content: ""; }
```



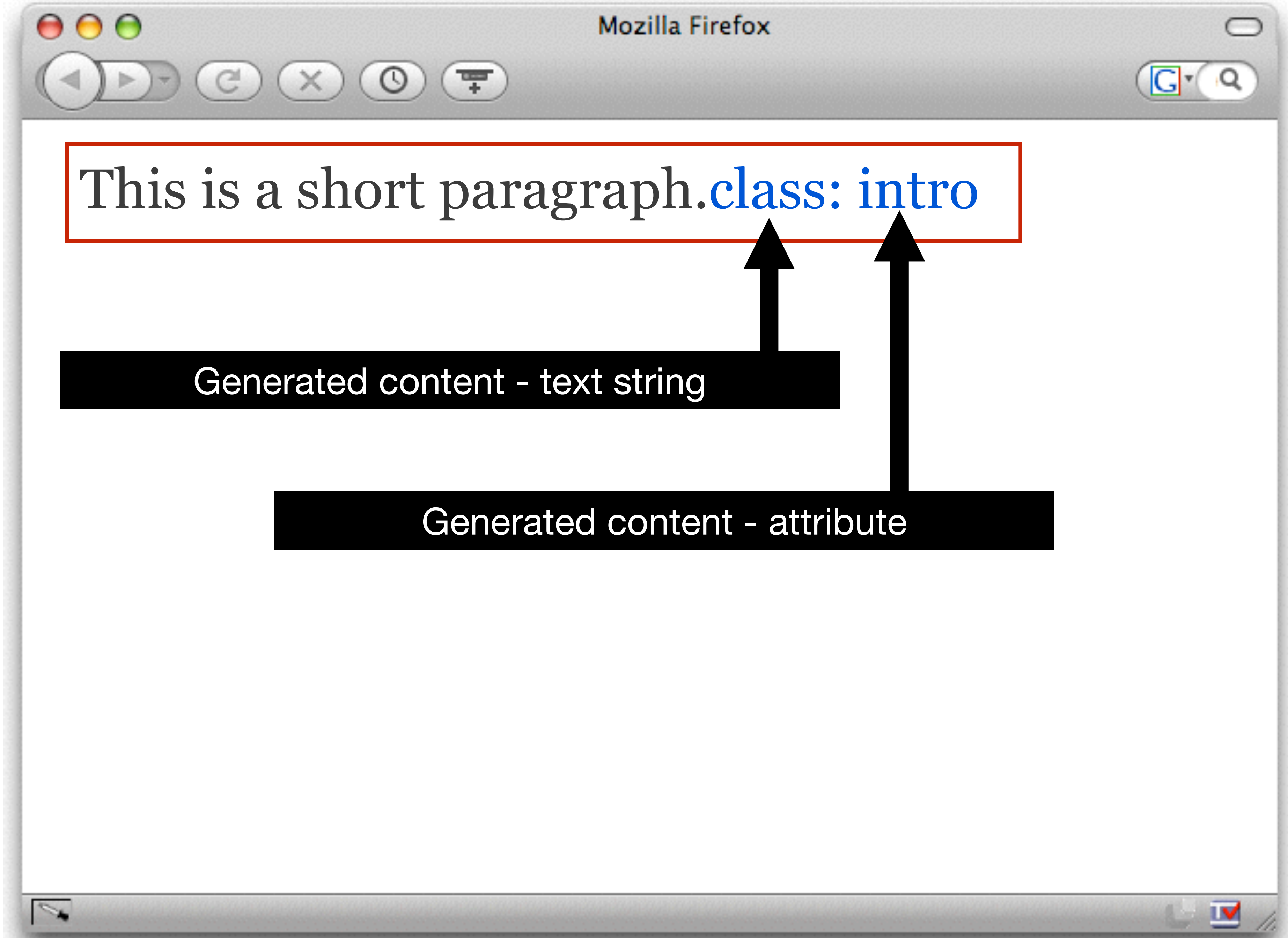
This is a short paragraph of text

No generated content

Generated content can also include **combinations of any of these four options**.

The following example will render the **generated text string** “class: ” and then render any class names applied to the element.

```
/* Generated content - combinations */
p::after
{ content: "class: " attr(class); }
```



This is a short paragraph.class: intro

Generated content - text string
Generated content - attribute

Generated content **cannot include HTML markup**.

```
/* Generated content - markup not allowed */
p::after { content: <p>test<p>; }
```

Support

The ::after pseudo-element selector is not supported by IE6 or IE7.

::selection
pseudo-element
(CSS3)

The **::selection pseudo-element** selector has been dropped from the latest W3C Basic User Interface specification

http://www.w3.org/TR/css3-ui/

“At risk” pseudo-elements (CSS3)

The following CSS3 pseudo-element selectors **are considered at risk**: ::value, ::choice, ::repeat-item, ::repeat-index

http://www.w3.org/TR/css3-ui/

Negation pseudo-class

The **:not() pseudo-class selector** is written using an element, followed by ":not", followed by a “(“, followed by a simple selector, followed by a “)”.

```css
/* syntax */
E:not(s) { }
[selector] :not(s) { }

/* examples */
div:not(p) { }
:not(.intro) { }
div :not(#news) { }
```

Only simple selectors are allowed inside the brackets.

```css
/* type selector */
p:not(p) { }
/* class selector */
p:not(.intro) { }
/* id selector */
p:not(#nav) { }
/* pseudo-class selector */
p:not(:lang(fr)) { }
/* attribute selector */
p:not([class="intro"]) { }
```

The negation selector, multiple simple selectors and pseudo-elements **are not allowed inside the brackets**.

```css
/* :not selector - not allowed */
:not(:not) { }
/* multiple simple selectors - not allowed */
:not(p, .intro) { }
/* pseudo-element selector - not allowed */
:not(::first-line) { }
```

How does it work?

The :not pseudo-class selector targets all elements except those defined by the selector.

In the following example, every element that is **not a <p> element** will be selected.

```css
/* CSS selector */
:not(p) { }
```

```html
<!-- HTML markup -->
<p></p>
<div></div>
<form>
    <label></lable>
<input>
</form>
```

In the following example, every element that **does not have a class of “intro”** will be selected.

```
/* CSS selector */
:not(.intro) { }


<!-- HTML markup -->
<p class="intro"></p>
<div></div>
<p></p>
```

In the following example, every element that **does not have an ID of “news”** will be selected.

```
/* CSS selector */
:not(#news) { }


<!-- HTML markup -->
<p id="news"></p>
<div></div>
<p></p>
```

In the following example, every element that **does not have a pseudo-class of “lang(fr)”** will be selected.

```css
/* CSS selector */
:not(:lang(fr)) { }
```

```html
<!-- HTML markup -->
<p lang="fr">Content</p>
<p lang="br">Content</p>
```

In the following example, every element that **does not have an attribute of “disabled”** will be selected.

```css
/* CSS selector */
:not([disabled]) { }
```

```html
<!-- HTML markup -->
<input type="text" disabled>
<input type="text">
```

Support

The :not pseudo-class selector is not supported in IE6, IE7 or IE8.

Russ Weakley

Max Design

Site: maxdesign.com.au

Twitter: twitter.com/russmaxdesign

Slideshare: slideshare.net/maxdesign

Linkedin: linkedin.com/in/russweakley