All html elements are rectangular.

Each html tag is an element in a DOM.

<!DOCTYPE html> This tells the browser that the type of the document is html document.

Styling can be added to HTML elements in 3 ways:

* Inline - using a **style attribute** in HTML elements
* Internal - using a **<style> element** in the HTML <head> section
* External - using one or more **external CSS files**

The most common way to add styling, is to keep the styles in separate CSS files.

CSS Specificity:

Specificity is the means by which a browser decides which CSS property values are the most relevant to an element and therefore will be applied. Specificity is only based on the matching rules which are composed of [css selectors](https://developer.mozilla.org/en/CSS/CSS_Reference#Selectors) of different sorts.

**How is it calculated?**

The specificity is a weight that is applied to a given CSS declaration based on the count of each[selector type](https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity#selector-type). In the case of specificity equality, the latest declaration found in the CSS is applied to the element. Specificity only applies when the same element is targeted. CSS rules that [directly target an element](https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity#directly-targeted-elements) will always take precedence over rules that an element inherits from an ancestor.

### **Selector Types**

The following list of selector types is by increasing specificity:

1. Universal selectors (i.e., \*).
2. Type selectors (e.g., h1) and pseudo-elements (e.g., :before).
3. Class selectors (e.g., .example), attributes selectors (e.g., [type="radio"]) and pseudo-classes (e.g., :hover).
4. ID selectors (e.g., #example).

Inline style added to an element (e.g., style="font-weight:bold") always overwrites any styles in the CSS and thus can be though as having the biggest specificity.

### **The !important exception**

When an !important rule is used on a style declaration, this declaration overrides any other declaration made in the CSS, wherever it is in the declaration list. Although, !important has nothing to do with specificity. Using !important is **bad practice** and should be avoided because it makes debugging more difficult by breaking the natural [cascading](https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade) in your stylesheets. When two conflicting declarations with the !important rule are applied to the same element, the declaration with greater specificity will be applied.

**Some rules of thumb**

* **Always** look for a way to use specificity before even considering !important
* **Only** use !important on page-specific css that overrides site-wide or foreign css (from ExtJs or YUI for example).
* **Never** use !important when you're writing a plugin/mashup.
* **Never** use !important on site-wide css.

**Instead of using !important, you can:**

1. Make better use of CSS cascading properties
2. Use more specific rules. By indicating one or more elements before the element you're selecting the rule becomes more specific and gets higher priority:

<div id="test">

<span>Text</span>

</div>

div#test span { color: green }

div span { color: blue }

span { color: red }

No matter what the order, the text will be green because that rule is most specific. (Also, the rule for blue overwrites the rule for red, notwithstanding the order of the rules)

**You should use it when:**

A) Scenario one:

1. You have a global CSS file that sets visual aspects of your site globally
2. You (or others) use inline styles on elements themselves which is a very bad practice

In this case you could set certain styles in your global CSS file as important thus overriding inline styles set directly on elements.

Real world example: Some badly written **jQuery plugins** that use inline styles.

B) Another scenario

#someElement p {

color: blue;

}

p.awesome {

color: red;

}

How do you make awesome paragraphs always turn red, even ones inside #someElement? Without !important, the first rule will have more specificity and will win over the second rule.

**How to override !important**

A) Simply add another CSS rule with !important, and either give the selector a higher specificity (adding an additional tag, id or class to the selector), or add a CSS rule with the same selector at a later point than the existing one (in a tie, the last one defined wins).

Some examples with a higher specificity:

table td {height: 50px !important;}

.myTable td {height: 50px !important;}

#myTable td {height: 50px !important;}

B) Or add the same selector after the existing one:

td {height: 50px !important;}

C) Or rewrite the original rule to avoid the use of !important altogether.

### **The :not exception**

The negation pseudo-class :not is not considered a pseudo-class in the specificity calculation. But selectors placed into the negation pseudo-class count as normal selectors when determining the count of [selector types](https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity#selector-type).

Here is a CSS chunk:

div.outer p {

color:orange;

}

div:not(.outer) p {

color: lime;

}

when used with the following HTML:

<div class="outer">

<p>This is in the outer div.</p>

<div class="inner">

<p>This text is in the inner div.</p>

</div>

</div>

Shall appear on the screen as:

This is in the outer div.

This text is in the inner div.

### **Form-based specificity**

Specificity is based on the form of a selector. In the following case, the selector \*[id="foo"] counts as an attribute selector for the purpose of determining the selector's specificity, even though it selects an ID.

The following style declarations:

\*#foo {

color: green;

}

\*[id="foo"] {

color: purple;

}

when used with this markup:

<p id="foo">I am a sample text.</p>

Will end up looking like:

I am a sample text.

Because it matches the same element but the ID selector has a higher specificity.

### **Tree proximity ignorance**

The proximity of an element to other elements that are referenced in a given selector has no impact on specificity. The following style declaration:

body h1 {

color: green;

}

html h1 {

color: purple;

}

With the following HTML:

<html>

<body>

<h1>Here is a title!</h1>

</body>

</html>

Will render as:

Here is a title!

Because the two declarations have equal [selector type](https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity#selector-type) counts, but the html h1 selector is declared last.

### **Directly targeted elements versus inherited styles**

Styles for a directly targeted element will always take precedence over inherited styles, regardless of the specificity of the inherited rule.

#parent {

color: green;

}

h1 {

color: purple;

}

With the following HTML:

<html>

<body id="parent">

<h1>Here is a title!</h1>

</body>

</html>

Will also render as:

Here is a title!

Because the h1 selector targets the element specifically, but the green selector is only inherited from the parent.

element.style{

}

It is an element's inline style attribute.

Here's the specificity value

Selector Specificity Specificity in large base

inline-style 1 0 0 0 1000

id selector 0 1 0 0 100

class,pseudo,attribute selector 0 0 1 0 10

type selector and pseudo elements 0 0 0 1 1

When do we prefer using this style attribute?

When you want to set/update the styles of an element dynamically using Javascript, you'll use this styles.

How selectors work?

<https://css-tricks.com/how-css-selectors-work/>

# Q) [What's the difference between using NAV and DIV](http://stackoverflow.com/questions/18628097/whats-the-difference-between-using-nav-and-div-around-bootstrap-3-navbars)

A) <nav> is the semantic HTML5 container element for you main navigation elements.

The nav is a block level element used to denote a section of major navigational links on a page. Not all links should be wrapped within a nav element. The nav should be reserved for primary navigation sections including universal navigation, a table of contents, breadcrumbs, previous/next links, or other noteworthy groups of links.

from <http://learn.shayhowe.com/html-css/elements-semantics>

If you are using HTML5 then you should use nav.

Q) What is the difference between <section> and <div> in HTML?

A) <section> means that the content inside is grouped (i.e. relates to a single theme), and should appear as an entry in an outline of the page.

<div>, on the other hand, does not convey any meaning, aside from any found in its class, langand title attributes.

From the spec:

### **<section>**

The section element represents a generic section of a document or application. A section, in this context, is a thematic grouping of content, typically with a heading.

Examples of sections would be chapters, the various tabbed pages in a tabbed dialog box, or the numbered sections of a thesis. A Web site's home page could be split into sections for an introduction, news items, and contact information.

...

**Note: The section element is not a generic container element. When an element is needed for styling purposes or as a convenience for scripting, authors are encouraged to use the div element instead. A general rule is that the section element is appropriate only if the element's contents would be listed explicitly in the document's outline.**

(<http://dev.w3.org/html5/spec-author-view/the-section-element.html#the-section-element>)

### **<div>**

The div element has no special meaning at all. It represents its children. It can be used with the class, lang, and title attributes to mark up semantics common to a group of consecutive elements.

**Note: Authors are strongly encouraged to view the div element as an element of last resort, for when no other element is suitable. Use of the div element instead of more appropriate elements leads to poor accessibility for readers and poor maintainability for authors.**

(<http://dev.w3.org/html5/spec-author-view/the-div-element.html#the-div-element>)

Q) What is footer tag?

A) The <footer> tag defines a footer for a document or section.

A <footer> element should contain information about its containing element.

A <footer> element typically contains:

* authorship information
* copyright information
* contact information
* sitemap
* back to top links
* related documents

You can have several <footer> elements in one document.

<footer>  
  <p>Posted by: Hege Refsnes</p>  
  <p>Contact information: <a href="mailto:someone@example.com">  
  someone@example.com</a>.</p>  
</footer>

# Q) [**What is href=“#” and why is it used?**](http://stackoverflow.com/questions/4855168/what-is-href-and-why-is-it-used)

## A) About hyperlinks:

The main use of anchor tags - <a></a> - is as [**hyperlinks**](http://www.w3.org/MarkUp/html-spec/html-spec_7.html). That basically means that they take you somewhere. Hyperlinks require the href property, because it specifies a location.

### **Hashtag:**

A hashtag - # within a hyperlink specifies an html element id to which the window should be scrolled.

href="#some-id" would scroll to an element on the current page such as <div id="some-id">.

href="//site.com/#some-id" would go to site.com and scroll to the id on that page.

## Scroll to Top:

href="#" doesn't specify an id name, but does have a corresponding location - the top of the page. Clicking an anchor with href="#" will move the scroll position to the top.

[**See this demo.**](http://jsfiddle.net/yrmnmd1q/)

This is the expected behavior according to the [**w3 documentation.**](http://www.w3.org/TR/html5/browsers.html#scroll-to-fragid)

## Hyperlink placeholders:

An example where a hyperlink placeholder makes sense is within template previews. On single page demos for templates, I have often seen <a href="#"> so that the anchor tag is a hyperlink, but doesn't go anywhere. Why not leave the href property blank? A blank href property is actually a hyperlink to the current page. In other words, it will cause a page refresh. As I discussed, href="#"is also a hyperlink, and causes scrolling. Therefore, the best solution for hyperlink placeholders is actually href="#!" The idea here is that there hopefully isn't an element on the page with id="!"(who does that!?) and the hyperlink therefore refers to nothing - so nothing happens.

## About anchor tags:

Another question that you may be wondering is, "Why not just leave the href property off??". A common response I've heard is that the href property is required, so it "should" be present on anchors. This is FALSE! The href property is required only for an anchor to actually be a hyperlink! Read [**this from w3**](http://www.w3.org/html/wg/drafts/html/master/links.html#attr-hyperlink-href). So, why not just leave it off for placeholders? Browsers render default styles for elements and will change the default style of an anchor tag that doesn't have the href property. Instead, it will be considered like regular text. It even changes the browsers behavior in regards to the element. The status bar (bottom of the screen) will not be displayed when hovering an anchor without the href property. It is most optimal, then, to use a placeholder href value on an anchor to ensure it is treated as a hyperlink.

## Q) What are Semantic Elements?

A) There were always semantic elements in every version of the HTML specification. HTML5 just added some new ones. A semantic element clearly describes its meaning to both the browser and the developer.

Examples of **non-semantic** elements: <div> and <span> - Tells nothing about its content.

Examples of **semantic** elements: <form>, <table>, and <img> - Clearly defines its content.

New Semantic Elements in HTML5

Many web sites contain HTML code like: <div id="nav"> <div class="header"> <div id="footer">  
to indicate navigation, header, and footer.

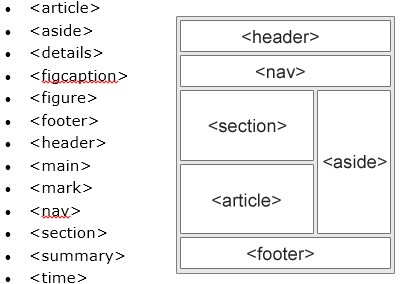
HTML5 offers new semantic elements to define different parts of a web page:

HTML5 <section> Element

The <section> element defines a section in a document.

According to W3C's HTML5 documentation: "A section is a thematic grouping of content, typically with a heading."

A Web site's home page could be split into sections for introduction, content, and contact information.



HTML5 <section> Element

The <section> element defines a section in a document.

According to W3C's HTML5 documentation: "A section is a thematic grouping of content, typically with a heading."

A Web site's home page could be split into sections for introduction, content, and contact information.

<http://www.w3schools.com/html/html5_semantic_elements.asp>

# **30 CSS Selectors**

<http://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048>