<b> and <i> tags vs <strong> and <em> tags

<b> and <i> are explicit - they specify bold and italic respectively.

<strong> and <em> are semantic - they specify that the enclosed text should be "strong" or "emphasised" in some way

The <strong> element is for content that is of greater importance, while the <b> element is used to draw attention to text without indicating that it's more important.

**Usage**

<strong> is used for the content that is of strong importance like, notes and warnings. This element is rendered by default using font-weight: bold style. However, it should not be used simply to apply bold styling; use the CSS [font-weight](https://developer.mozilla.org/en-US/docs/Web/CSS/font-weight) property for that purpose. Use the [<b>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/b) element to draw attention to certain text without indicating a higher level of importance. Use the [<em>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/em) element to mark text that has stress emphasis.

**Practical difference**

Think about visually impaired users, If a screen-reader came across an <i> tag, it wouldn't know what to do. But if it comes across a <em> tag, it knows that whatever is within should be emphasized to the listener. And therein you get the practical difference.

**<em> vs <strong>**

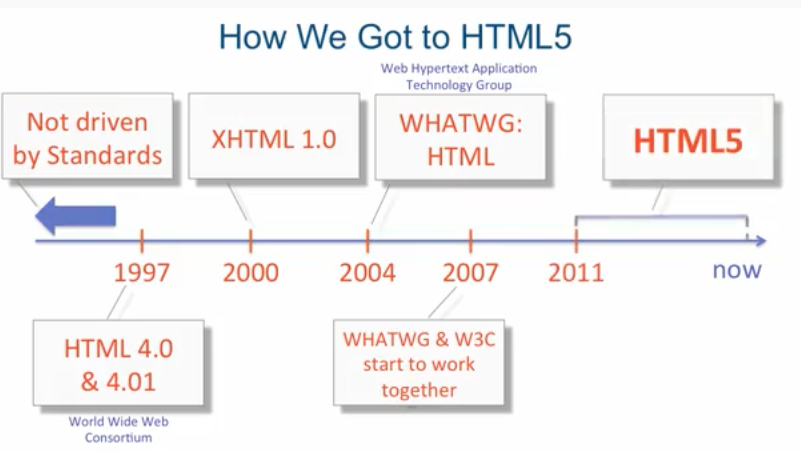
While <em> is used to change the meaning of a sentence as spoken emphasis does ("I love carrots" vs. "I love carrots"), <strong> is used to give portions of a sentence added importance (e.g., "**Warning!** This is **very dangerous.**") Both <strong> and <em> can be nested to increase the relative degree of importance or stress emphasis, respectively.

References:

<https://stackoverflow.com/questions/271743/whats-the-difference-between-b-and-strong-i-and-em>

<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/strong>

History of HTML

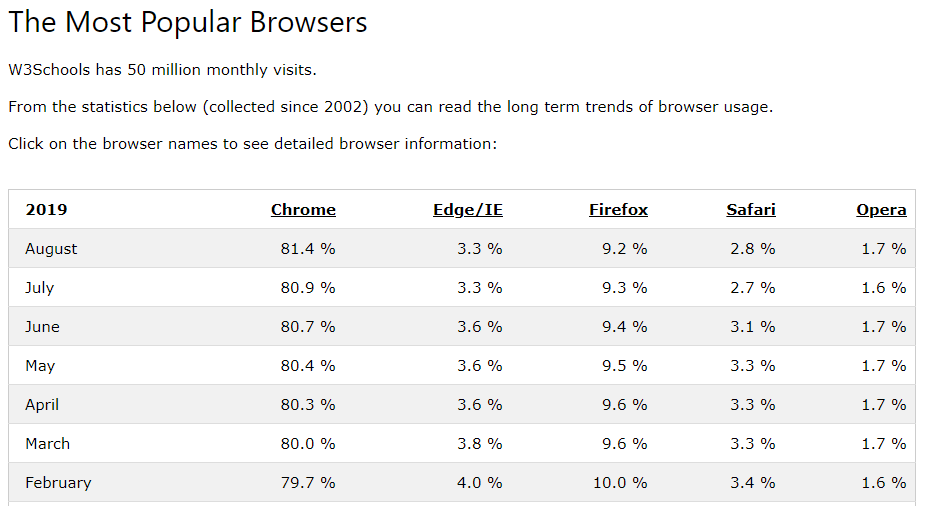


1. 1991: **HTML 1** (1991) "HTML1" is a name for Tim Berners-Lee's original HTML.
2. Before 1997: There were no community standards, so browsers community did what they wanted, invented new tags, implemented same tags differently. You could go to a website and being told that your browser is not compatible with this website and you have to get a different browser in order to event view the website.
3. Around 1997: The W3C came up set of standards browsers actually started to pay somewhat attention to as HTML 4 and quickly updated to HTML 4.01.
4. Around 2000: W3C came up with another standard called XHTML 1.0 and that specification is based on XML and W3C wanted to keep going with it an produce XHTML 2.0. Now the browser vendors who already had a history not 100% listening to the HTML standards, they decides the while W3C thing is moving way too slowly.
5. 2004: Browser vendors bended together and creates another group that produce specifications that was called WHATWG. Development of HTML is closed by W3C when it decides to instead focus on XHTML. WHATWG is formed to develop HTML further, with the aim of reflecting the modern dynamic web, while keeping backwards compatibility with existing HTML code. This group is much less democratic than W3C, they has one central editor to take final decision.
6. 2007: W3C and WHATWG started working together and what they produced is what we have right now HTML5.
7. The way they divided it up is W3C is going to be in charge of HTML5 standards as this is what their bread and butter is. And WHATWG is not going to give a version to their HTML and they say it’s HTML and it is constantly evolving it. WHATWG is that which really implements new features and W3C what it does is they cherry pick some of more successful things and bring then to the official standard.



Now major modern browsers are evergreen browsers and they silently update themselves.

To check which is used most, refer to <https://www.w3schools.com/browsers/default.asp>



Chrome is pretty important browser to make sure your web page works in.

Note: Microsoft Edge is a web browser developed by Microsoft.

Microsoft Edge was released in July, 2015. Microsoft Edge replaced Internet Explorer.

HTML5 Semantic tags

So, before the introduction of these semantics, sections were done this way:

<div id="main-section">

your content

</div>

<div id="sidebar">

your sidebar content

</div>

HTML5 semantics were supposed to save us from situations like the above while also being descriptive.

Styles were added to sections like the above then, and I don't see why we can't do the same now. There isn't really any rule against adding styles to HTML5 semantics but ensure that HTML5 semantics are used for unique elements in the first place.

Avoid adding styles to semantic elements. This is because these elements add nothing new when we are speaking about how things 'look'.

Well, HTML5 semantic (nav, header, footer, etc...) were created to help us give meaningful and self-descriptive names to sections of our web pages. They are expected to be unique.

Program: Slider to observe Border width, margin and padding

Features:

1. CSS: overflow
2. HTML: output tag
3. JavaScript: oninput event, getElementsByClassName, getElementById, style.backgroundColor, style.paddingBottom, style.border

<!DOCTYPE html>

<html>

<body>

<div style="width:600px; height: 300px; overflow:hidden; border: 2px dotted red; margin: 0px auto; align-item: center" id="container">

<div class="myH1" style="background:yellow">Hello!</div>

<div class="myH1" style="background:yellow">Hello!</div>

</div>

<p>Notice that, keep some border around 39px and keeping padding = 0, if you set margin=30px, then the <b>bottom margin of first box adds up to top margin of second box.</b>So, if bottom margin of first=20px and top margin of second=20px then the margin between both the boxes will be 20px only However, same is not observed for padding</p>

<label> Border<input type="range" id="bSlider" max="100" min="0" step="5" value="50"></label>

<output id="borderValue"></output>

<label> Margin<input type="range" id="mSlider" max="100" min="0" step="5" value="50"></label>

<output id="marginValue"></output>

<label> Padding<input type="range" id="pSlider" max="100" min="0" step="5" value="50"></label>

<output id="paddingValue"></output>

</div>

<script>

document.getElementById("container").style.backgroundColor = 'yellow';

var boxList = document.getElementsByClassName("myH1");

for (var i = 0; i < boxList.length; i++) {

boxList[i].style.border = "thick solid #0000FF";

}

document.getElementById("bSlider").oninput = function() {

myBorderFunction()

};

function myBorderFunction(){

userRange = document.getElementById("bSlider").value;

document.getElementById("borderValue").innerHTML = userRange+'px ';

var elementList = document.getElementsByClassName("myH1");

for (var i = 0; i < elementList.length; i++) {

elementList[i].style.borderWidth = userRange+'px';

}

}

document.getElementById("mSlider").oninput = function() {

myMarginFunction()

};

function myMarginFunction(){

var userRange = document.getElementById("mSlider").value;

document.getElementById("marginValue").innerHTML = userRange+'px ';

var elementList = document.getElementsByClassName("myH1");

for (var i = 0; i < elementList.length; i++) {

elementList[i].style.marginBottom = userRange+'px';

elementList[i].style.marginTop = userRange+'px';

elementList[i].style.marginRight = userRange+'px';

elementList[i].style.marginLeft = userRange+'px';

}

}

document.getElementById("pSlider").oninput = function() {

myPaddingFunction()

};

function myPaddingFunction(){

userRange = document.getElementById("pSlider").value;

document.getElementById("paddingValue").innerHTML = userRange+'px ';

var elementList = document.getElementsByClassName("myH1");

for (var i = 0; i < elementList.length; i++) {

elementList[i].style.paddingBottom = userRange+'px';

elementList[i].style.paddingTop = userRange+'px';

elementList[i].style.paddingRight= userRange+'px';

elementList[i].style.paddingLeft = userRange+'px';

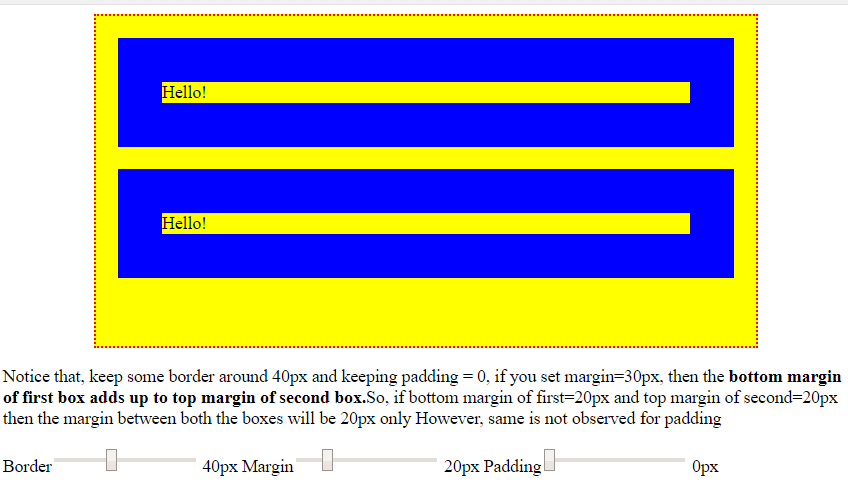
}

}

</script>

</body>

</html>



# **HTML DOM setAttribute() Method**

The setAttribute() method adds the specified attribute to an element, and gives it the specified value.

Examples:

document.getElementsByTagName("H1")[0].setAttribute("class", "democlass");

document.getElementById("myAnchor").setAttribute("href", "https://www.w3schools.com");

**Note:** Although it is possible to add the style attribute with a value to an element with this method, it is recommended that you use [properties of the Style object](https://www.w3schools.com/jsref/dom_obj_style.asp) instead for inline styling, because this will not overwrite other CSS properties that may be specified in the style attribute:

Bad:

*element*.setAttribute("style", "background-color: red;");

Good:

*element*.style.backgroundColor = "red";