# HTML5:
## An Introduction


an internet.com® Developer eBook

# Contents...

## HTML5: An Introduction


*HTML5 is an emerging standard web developers are embracing everywhere. The HTML5 standards define new functionality that is being embraced by most of the key web browsers, including Microsoft Internet Explorer 9. In this ebook, you will be introduced to the basic features of HTML5 that you can use within your web sites now.*

*This content was adapted from QuinStreet's HTMLGoodies website. Contributors:.David Fiedler and Scott Clark.*





# Web Developer Basics: Learning About HTML5

By David Fiedler and Scott Clark

HTML5, depending on who you listen to, may be either a disruptive new technology that has the potential to bring entire companies to their knees, or a smooth transition from current HTML 4.0 that promises to make life much easier for developers. Both are at least partially true, and in this continuing series, I hope to help you make sense out of HTML5: both business sense and nuts-and-bolts coding-level sense.


HTML5 is most definitely a work in progress. It began to take shape back in 2004, and the official specification may not be actually complete until the year 2022! But HTML5 is already here, in everything from your current desktop browser to your new smartphone, so there's no problem with getting started.

## So Let's Get Started with HTML5

Perhaps the most important thing to understand about HTML5 is not the coding details and changes themselves, but the high-level functions they give you access to. In fact, HTML5 is all about high-level functions rather than details. For instance, instead of thinking of multimedia objects and then defining them as video or audio and so on, in HTML5 you can simply write something like:

```
<video src="watchthis.mp4" width="640"
height="480">
  <a href="watchthis.mp4">Here's my video</a>
</video>
```

This functional methodology extends even to typical page coding. We're all used to writing complex pages in terms of low-level objects like , which is kind of amorphous and easy to lose track of. So we often attempt to keep track of things by coding like this:

```
<div id="header">
<H1>Web Developer Basics: Learning About
HTML5</H1>
<p class="credit">by David Fiedler</p>
</div>
```

In HTML5, we can cut right to the chase. We're writing a header, and now we can code it that way:

```
<header>
<H1>Web Developer Basics: Learning About
HTML5</H1>
<p class="credit">by David Fiedler</p>
</header>
```

So what, you might say at this point. Well, it's not just the header of a page that we can now view as a complete functional object, it's almost everything we use on a daily basis: <header>, <footer>, <article>, <section>, <nav>, <menu>, <figure>. This gives us tremendous flexibility

in terms of how we can think of the page. So it's not just easier to understand the structure of the page, it's easier to correctly code the structure of the page.

## Begin At the Beginning

The beginning of many modern HTML 4.0 pages looks something like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-strict.dtd">
```

But in our brave new world of HTML5, all we need is:

```
<!DOCTYPE html>
```

Similarly, the complex XHTML boilerplate declarations many people use can be simply replaced by:

```
<html lang="en">
```

and encoding declarations such as

```
<meta http-equiv="Content-Type" content="text/
html; charset=utf-8">
```

can be toned down to a mere

```
<meta charset="utf-8">
```

Oh, and we may as well get this next bit out of the way now, even though I hesitate to mention it for fear of being responsible for people writing near-incomprehensible HTML5 pages. You no longer need those double quotes around attributes, so that <p class=credit> is now as legal as <p class="credit">. But please use this power wisely.

## A Bit of Magic

Just to show that HTML5 isn't only about structure and saving keystrokes here and there, here's a nice example of an attribute feature that is simple on the surface, but demonstrates some real power. Paste this simple little document into a text file, and call it something like foo.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
     <meta charset="utf-8">
     <title>You Can Edit This</title>
</head>
<body>
     <h1>I Mean, You Can Really Edit This</h1>
     <p contenteditable=true>
     Now is the time for all good cats to come
to the aid of their catnip.
     </p>
</body>
</html>
```

The only new thing here that will jump out at you is the attribute of contenteditable on the paragraph tag. You can use this on any element, not just a paragraph, and it takes effect for everything within that element. Now, open this file using any modern browser and you'll see that you can indeed edit the paragraph - but not the heading! - Right in the browser.

But wait, there's more! Change that paragraph as much as you like, then save the page to your computer as a new HTML file. Open it up in a text editor...and presto, the source code has changed to reflect the text changes you made in the browser. Shazam! ■

# Web Developer Basics: Differences between HTML 4 and HTML5

By David Fiedler and Scott Clark

Now it's time to take a few steps back and take a look at some of the differences between HTML 4 and HTML5.

This is intended to be a useful overview, not an exhaustive reference, but remember that things are still and always changing. The complete, up to date list of all the technical differences may always be found on the W3C's site. You also may want to refer to this document for the actual details of the HTML5 specification itself.


The first thing you should know is that, perhaps for the first time, the development of a language standard is acknowledging the real world. In order to keep file compatibility with the current standard - which is technically HTML 4.01 - the brave decision was made to separate the way the web browser renders files from the way we, as developers, must write them. So the browser, or "user agent", must still process HTML4 constructs like the center element, because there will still be millions of files on the Internet that happen to use it. But we won't be writing any more HTML with center; it's simply being dropped from the language (use CSS instead). This compatibility goes both ways: older browsers can (and will) simply ignore HTML5 code without screwing things up.

## No More Frames

This is great news to those of us who slogged through the 1990s. To be exact, the elements frame, frameset, and noframes are being removed from the language, as well as acronym, applet, basefont, big, blink, center , dir, font, isindex, strike , tt and u. All of these can be handled using CSS or other methods.

You'll also have to learn to get along without using tables for layout; while tables themselves are still part of HTML5, they're not intended for placing pixels any more. Here's what the spec says:

"Tables must not be used as layout aids. Historically, some Web authors have misused tables in HTML as a way to control their page layout. This usage is non-conforming, because tools attempting to extract tabular data from such documents would obtain very confusing results."

So all the attributes that let people create those perfectly laid-out, tinted tables are gone, like align, bgcolor, border, cellpadding, cellspacing, height, nowrap, rules, valign, and the big one: width. The mantra: use CSS instead.

I've been trying my best to break it to you slowly, but frankly, all presentational elements are coming out of HTML5. My advice: learn lots more CSS, until you can quote chapter and verse in your sleep.

## Good News

The good news is that even though this is a big change, it's a change for the better. Browsers of the future (just

another month or two!) will become more powerful because of the move towards the cloud, so that they'll be able to handle more on their own. We've already seen that with things like Ajax, and now with video/audio embedding and such, it will be far easier for us to code in a straightforward manner and let the browser figure out the details. For instance, new structure elements include article, aside, figcaption, figure, footer, header, hgroup, nav, section, and summary, all of which refer to the structure of the document itself and leave rendering to the browser.

There are still some new elements that deal with text on a detailed level, however: you'll code wbr when you think it's possible to do a line break, but the browser will decide for you. Another hint element is bdi, used to mark an area where bidirectional text formatting can be done (primarily for mixing left-right and right-left languages in a single document). Its complement, bdo, lets you explicitly override and force a particular directionality. For even more slick internationalization, the elements ruby, rp, and rt are included for ruby annotations, which are meant for pronunciation aids rather than for Ruby On Rails programmers.

The more high-level new elements include things like canvas, meant for specifying an area for drawing a bitmapped graphic on the fly, such as a data graph or game graphic; meter is a placeholder for a numeric measurement of an expected size (and is eerily similar to format in ancient FORTRAN), while progress is its graphical counterpart, to be used where you want a progress bar. Last, but not least, there are the multimedia elements (audio, video, source, embed) that we cover in another segment of this ebook. ■

# Web Developer Basics: Multimedia in HTML5

By David Fiedler and Scott Clark

Now you'll see how support for various multimedia formats in HTML5 will make things much easier for you as a developer... eventually.

## See Me, Hear Me

We'll start with the good news. HTML5 is fairly intelligent about picking the right default for presenting the most optimum audio or video. Couple that with the absolute minimum coding that's needed to handle multimedia in HTML5 and you have a pretty good situation for developers.

The bad news is that because the people diligently working on the HTML5 specification tried to compromise between open formats and de facto standard formats and so on, support for native codecs in HTML5 is slightly lacking: there isn't any. It's up to the browser to support formats, and up to the developer to supply them. What's emerged from that are a few relatively new standards. So unless you're a video or audio enthusiast, you may have to learn a few new things (and worse, convert your legacy media!). But when all is said and done, programs do all the hard work anyway, so it's certainly not a deal-breaker.

```
<video controls width="640" height="480"
src="sample.mp4" poster="sample.jpg">
</video>
```

That's all the code you need to display MPEG-4/H.264 video in an HTML5 browser that supports the MPEG-4/H.264 video format, with a few extra goodies even thrown in such as a predefined video size, default video controls, and a still thumbnail. Unfortunately, at this writing, only Apple's Safari and Google's Chrome browsers will work natively with this particular format.

## Free Me

MPEG-4 format has one other problem which is not technical; it's not a "free" format. It's covered by patents and there are licensing fees involved, at least for broadcasters and browser manufacturers. That's why Mozilla Firefox, Opera, and a few other browsers support Ogg Theora. Google recently announced it was freeing the VP8 video format it had acquired with On2 Technologies, and simultaneously announced broad industry support for the WebM container format using VP8 video and Vorbis audio.

As many developers will remember from the 1990s, the reality of competing standards is that browsers end up implementing them differently, so you have to keep the differences in mind. Luckily, "all it takes" at this point in time for universal support (except for some mobile platforms) is to have all three different formats available for each of your videos to satisfy all the HTML5 browsers and some extra code...plus a fallback scheme for non-HTML5 browsers (which basically means the all-purpose

Flash plugin). Love it or hate it, you can't live without it yet.

```
<video controls width="640" height="480"
poster="sample.jpg">
  <source src="sample.mp4" type='video/mp4;
codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="sample.webm" type='video/webm;
codecs="vp8, vorbis"'>
  <source src="sample.ogv"  type='video/ogg;
codecs="theora, vorbis"'>
  <object width="640" height="480"
type="application/x-shockwave-flash"
data="flowplayer-3.2.1.swf">
    <param name="movie" value="flowplayer-
3.2.1.swf">
    <param name="allowfullscreen" value="true">
    <param name="flashvars"
value='config={"clip": {"url": "http://yourdomain.
com/videos/sample.mp4", "autoPlay":false,
"autoBuffering":true}}'>
  <p>If you can read this, you're using a pre-
HTML5 browser without Flash.</p>
  </object>
</video>
```

This extra MIME type specification certainly isn't there to pretty up the code, and it's not strictly necessary, but if you don't write it in accurately, then the browser is going to have to download each video file format in order until it finds one that it can play. And by that, we mean downloading the entire video file...so in the worst case, that would be a full three times! So it takes a few extra seconds to paste in those types and codecs, but now you know why you should do it.

Now that you've seen some of the video markup, the audio will seem simple by comparison. Once more, it's quite possible to let the HTML5 browser do a lot of the work:

```
<audio controls src="sample.ogg">
</audio>
```

And again, in the real world, it's recommended to provide a number of format options for the browser:

```
<audio controls preload="metadata">
  <source src="sample.mp3">
  <source src="sample.ogg">
  <!-- Flash fallback code or text would go
here -->
</audio>
```

Both the <video> and <audio> can take another important parameter, shown above, called preload (which was formerly implemented as autobuffer with slightly different syntax). The values for preload can be auto (download the media file to the browser in advance), none (do not preload the media), or metadata (just download enough metadata to discover the duration and other information of the media file). So with a little extra care in coding, you can make the user's experience much better. ■

# Web Developer Basics: The HTML5 Video Element

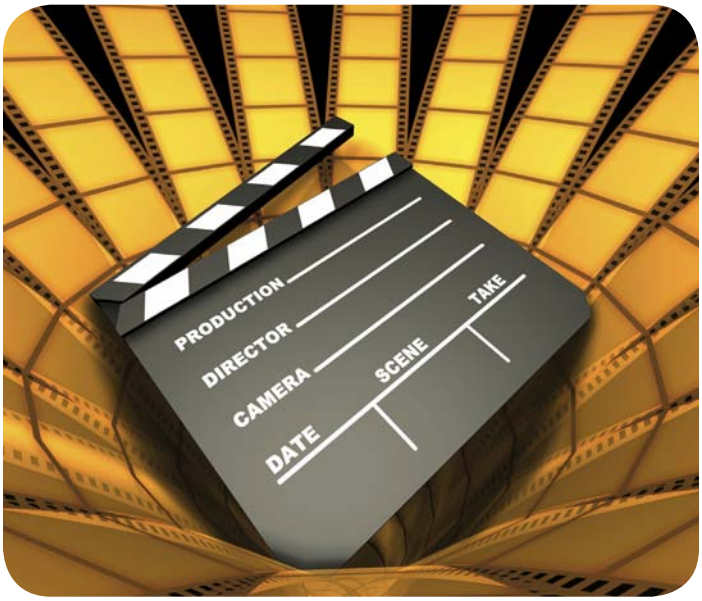
By David Fiedler and Scott Clark

HTML5's video element is conceptually easy to work with, since at bottom it's been designed like the <img> tag...just code and go:

```
<video src="sample.mp4" width="640"
height="480"></video>
```

Unfortunately, in the real world, things aren't quite so easy. Due to licensing and other restrictions, you can't simply assume that the user's software will properly display your video in MPEG-4/H.264 format, so at this point you'll have to supply the same video in open formats as well as providing a Flash-based fallback for pre-HTML5 browsers--something like this:

```
<video controls width="640" height="480"
poster="sample.jpg">
<source src="sample.mp4"  type="video/mp4">
<source src="sample.ogv"  type="video/ogg">
<source src="sample.webm" type="video/webm">
    <object width="640" height="480"
type="application/x-shockwave-flash"
data="player.swf">
        <param name="movie" value="player.swf">
        <param name="flashvars" value="controlba
r=over&amp;image=sample.jpg&amp;file=sample.mp4">
        <img src="sample.jpg" width="640"
height="360" alt="Sample"
   title="No direct video playback
capabilities, so please download the video
below">
    </object>
</video>
<p>      <strong>Download Video:</strong>
        Closed Format: <a href="sample.
mp4">"MP4"</a>
        Open Format:   <a href="sample.
ogv">"Ogg"</a>
        WebM Format:   <a href="sample.
webm">"WebM"</a>
</p>
```




Since this isn't the kind of thing most developers like to memorize, it's good that someone has written up a handy free code generator that lets you specify all your parameters and video sizes and so on. But there's more nitty-gritty to deal with, specifically transcoding those extra formats.

## Preaching To the Converted

Unless you have lots of video files and an actual software budget, your best bet is free conversion software. Currently, the best selection for that is Handbrake, which is fully open source, runs on Windows, Mac OS X, and Linux, and will convert most multimedia file formats to MP4 or Ogg Theora (when

we get to audio, you'll find that it works just as well for those formats, too). If you'd rather be modern, OS-independent, and stay in the cloud, sites such as MediaConverter and Zamzar can accommodate you with free online transcoding services (the latter site even supports the relatively new WebM format).

The full list of attributes for the HTML5 video element at this time includes the following, with usage notes:

- **src -** The URL of the video. This overrides the source element, if present.
- **poster -** The URL of a still picture to show while the video is not playing.
- **preload -** This can have the value none, metadata, or auto. Auto will download the entire file if possible; metadata will download just the parameters so that the length, size, and type of the video can be identified, and none will do nothing, which saves bandwidth.
- **autoplay -** This boolean, if present, triggers the video to play as soon as it is fully buffered or ready to stream.
- **loop -** Also a boolean; if loop is present, the video will repeat endlessly in the absence of user intervention.
- **audio -** This attribute, which controls the audio portion of the video, is still in development. Currently, it can take only a single value: muted, which means that the audio volume will initially be set to zero. The intent is to allow an autoplaying video to start and get the user's attention, but without blaring audio that would cause the user to close the entire tab in disgust :-)
- **controls -** A boolean attribute that specifies the browser should provide a set of default video controls. If it doesn't appear, you'll have to design and code your own controls.
- **width, height -** these size attributes control the size of the area reserved for the video on the page, but not necessarily its exact dimensions.

While it takes a bit more work, generally in the Javascript department, to create your own controls and error handling, all it takes is some CSS to modify the look of the video element:

```
<!DOCTYPE html>
<html lang="en">
<head>
     <meta charset="utf-8">
     <title>File in same directory</title>
<style type="text/css">
video {
        width: 800px;
        height: 600px;
        position: relative;
    }
</style>
</head>
<body>
<P>
<video source src="airplane.webm" controls
autoplay>
     Your browser does not support the video
tag.
</video>
<P>
</body>
</html>
```

# HTML5 Primer: How To Use the Audio Tag

By David Fiedler and Scott Clark

Now we're going to expand upon our discussion on multimedia and delve further into HTML5's <audio> tag.

The <audio> tag is new to HTML, like the <video> tag, and allows developers to embed music on their websites (and unlike earlier attempts to add audio to a website, it isn't limited to old-fashioned midi music). That said, it does have limitations on the types of files that can be used. Currently any recent browser that is based on Webkit, such as Chrome and Safari, supports the use of regular .mp3 files. Others, such as Firefox, only support the .ogg format.

The good news is that you can either convert your files from .mp3 to .ogg (one audio conversion tool, media.io, can be used online) or just supply two versions of your audio file, one in each format. When Safari, for instance, comes across the <audio> tag, it will ignore the .mp3 file and move directly to the .ogg file.

## So How Is the <audio> Tag Used on the Page?

When the audio tag is used, it will look something like Figure 1 (obviously you will only see it if your browser supports it):


**Figure 1: The Audio Tag in Use**

You use the <audio> tag just like you use any other element:

```
<audio autoplay="autoplay" controls="controls">
   <source src="music.ogg" />
   <source src="music.mp3" />
</audio>
```

You can also include the source file's location in the beginning <audio> tag, rather than between the two, like this:

```
<audio src="music.ogg"
controls="controls">
```

Also note that you can point the src to a file located on the server with your web page (a relative URL, like /audio/music.ogg), or a file located elsewhere on the web (an absolute URL, such as http://www.yoursite.com/music.ogg).

You will likely want to include some text inside the tag so that users whose browsers do not support the <audio> tag will have a clue as to what is going on (and why they aren't seeing the audio control on the page). You do that like this:

```
<audio src="horse.ogg" controls="controls">
Your browser does not support the audio
element.
</audio>
```

You can use any HTML elements that are supported

within the <audio> tag, such as italics, bold, links, objects such as Flash, etc.

## The <audio> Tag's Attributes

The <audio> tag supports the full range of standard attributes in HTML5. These attributes are supported by all HTML5 tags, with very few exceptions. They include:

- **accesskey -** this specifies a keyboard shortcut for a given element
- **class -** this specifies a classname for a given element, to be used in conjunction with a style sheet
- **contenteditable -** specifies whether a user is allowed to edit the content
- **contextmenu -** specifies the context menu for a given element
- **dir -** specifies the direction of the text for content in a given element
- **draggable -** specifies if a user is allowed to drag a given element
- **dropzone -** specifies the event that occurs when an item or data is dragged and dropped into a given element
- **hidden -** specifies if a given element is hidden or not
- **id -** specifies a unique identification for a given element
- **lang -** specifies a language code for the content in a given element
- **spellcheck -** specifies if a specific element will need to be subjected to a spelling and grammar check
- **style -** defines an inline style for a specific element tabindex - specifies the tab order of a specific element
- **title -** specifies a title for a specific element

New attributes for the <audio> tag include the following:

- **autoplay -** if this attribute is included, the audio will begin to play once it is ready
- **controls -** if this one is included, controls for the audio file will be included on the page (which is a great idea--it is very annoying to not have a way to stop the audio from playing)
- **loop -** if this one is included, the audio will loop and play again once it has finished
- **preload -** this one has three parameters: auto, which plays once it has loaded, metadata, which only displays the data associated with the audio file, and none, which means it will not preload
- **src -** this one's value is simply the URL of the audio file you wish to play

You can see some of the new attributes in action here:

```
<audio loop="loop" autoplay="autoplay"
controls="controls">
   <source src="music.ogg" />
   <source src="music.mp3" />
</audio>
```

The <audio> tag has a lot of attributes which can be used for additional controls, including the event attributes in HTML5. Events include window events, which are triggered for the window object, form events, which are triggered by actions that occur within an HTML form, keyboard and mouse events, and media events. Many of the events are the same as those included with previous versions of HTML. ■

# Web Developer Basics: Using The HTML5 Canvas Element

By David Fiedler and Scott Clark

Canvas is a unique concept. Unlike the rest of the HTML world that consists of well-defined pieces that designers and developers love to place in exactly the right spot, the Canvas element provides a virtual Etch-a-Sketch-like area where almost anything can be made to happen. It's relatively easy to describe but, like most open-ended concepts, more difficult to characterize. Canvas is a bit-mapped area whose width and height dimensions are specified as attributes (defaulting to 300 and 150 respectively); the purpose can be virtually anything, and the intention is that a canvas element will be rendered by scripting.

## Show Me the HTML5 Code!

Clearly the simplest possibility is <canvas></canvas>, and that gets you your very own 300x150 area, but you won't see it because the default is transparent (speaking of defaults, anything you actually draw on a canvas is black unless you specify otherwise). A more realistic beginning is something like this:

```
<canvas id="drawme" width="400" height="200">
    If you can read this, your browser does not support Canvas.
<canvas>
```

Note that, similar to the video element, you can (and should) provide fallback content in case the browser doesn't yet support the canvas element. Actually, it's more than "should"...it's "must" in the words of the W3C definition:

"When authors use the canvas element, they must also provide content that, when presented to the user, conveys essentially the same function or purpose as the bitmap canvas. This content may be placed as content of the canvas element."

So in a real-world web page, you'd want to include something that displays the closest equivalent to the contents of your fancy bitmapped graphic canvas. If your canvas showed a stock graph, you could use the current stock price. The intent is to allow the widest possible audience to see the most valuable and useful content, after all. The good news is that the current versions of all leading modern browsers already support canvas...except, of course, Internet Explorer (though it's coming in IE9).

Also, note that you must name each specific canvas element with its own id so you can address it later in your JavaScript. And while most basic tutorials show all kinds of boxes and graphic elements laboriously rendered point by point, in reality most canvas applications will involve user interactivity, image transformations, or algorithm-based graphic rendering, such as Apple's clock widget. Or all three, such as this mind-boggling Canvas Aquarium (http://widgets.opera.com/widget/5040/).

## The Mozilla Developer Network Template

Mozilla's Developer Network (http://developer.mozilla.org/) has created a nice template for use in demonstrating basic canvas functions, so we'll use that as the basis of our own example code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Canvas Tutorial Template</title>
    <script type="text/javascript">
      function draw(){
        var canvas = document.
getElementById('tutorial');
        if (canvas.getContext){
            var ctx = canvas.getContext('2d');

//
//    drawing code goes below here
//
            ctx.fillStyle = "rgb(255,0,0)";

            ctx.fillRect(0, 0, 150, 150);

//
//    drawing code goes above here
//

        }
      }
    </script>
    <style type="text/css">
      canvas { border: 1px solid black; }
    </style>
  </head>
  <body onload="draw();">

    <canvas id="tutorial" width="150"
height="150">
        Fallback content goes here.
    </canvas>
  </body>
</html>
```

The most important piece of all this is actually where you call getElementById using your named canvas id (which locates your canvas in the DOM), then call getContext to initialize the actual drawing context. Once you have a context (which we point to using ctx), you can manipulate it using anything you like in the 2D drawing APIs.

## And the Function Is....

In this case, we've used fillStyle to select solid red and fillRect to describe the boundaries of the rectangle to be drawn. You can now use this template to quickly experiment with any or all of the other functions in the APIs, by simply writing more complex code in place of those two lines. The x,y origin is defined to be at the top left, so knowing that, you can start making use of functions such as:

- fillRect (x,y, width, height) - paints the described rectangle (filled using the current fillStyle)
- strokeRect (x,y, width, height) - paints the described rectangle outline (using the current strokeStyle)
- clearRect (x,y,width,height) - clears the described rectangle and makes it transparent (transparent black, to be technical)

For true drawing, you need the ability to draw lines and curves, and place your virtual pen anywhere on the canvas, so these functions are key:

- beginPath() - starts a new drawing
- moveTo(x,y) - places the pen at the x,y location
- lineTo(x,y) - draws a line from the pen location to the x,y location
- arc(x, y, radius, startAngle, endAngle, anticlockwise) - draws an arc at x,y using radius from startAngle to endAngle (in radians). The arc is drawn clockwise unless the optional boolean anticlockwise parameter appears.
- stroke() - renders the drawing in outline form
- fill() - renders the drawing in filled form