Drawing text

← Previous Next →

After having seen how to apply styles and colors in the previous chapter, we will now have a look at how to draw text onto the canvas.

Drawing text

The canvas rendering context provides two methods to render text:

`fillText(text, x, y [, maxWidth])`

Fills a given text at the given (x,y) position. Optionally with a maximum width to draw.

`strokeText(text, x, y [, maxWidth])`

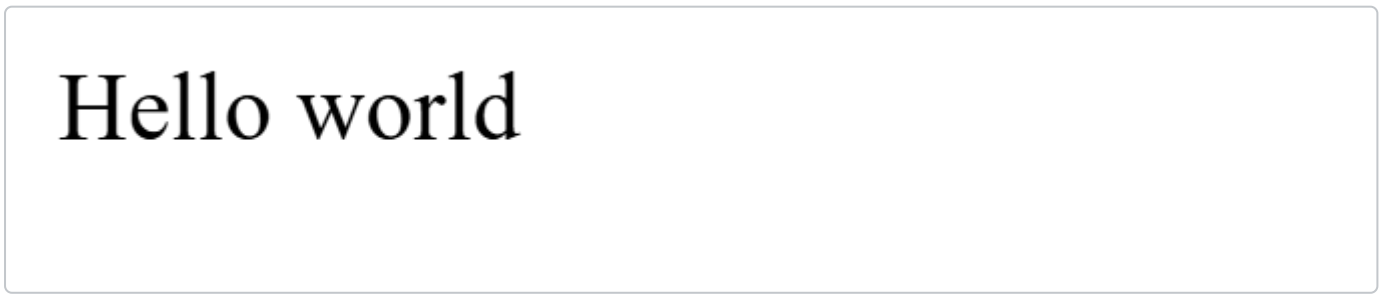
Strokes a given text at the given (x,y) position. Optionally with a maximum width to draw.

A `fillText` example

The text is filled using the current `fillStyle`.

JS

```
function draw() {
  const ctx = document.getElementById("canvas").getContext("2d");
  ctx.font = "48px serif";
  ctx.fillText("Hello world", 10, 50);
}
```



A `strokeText` example

The text is filled using the current `strokeStyle`.

JS

```
function draw() {
  const ctx = document.getElementById("canvas").getContext("2d");
  ctx.font = "48px serif";
  ctx.strokeText("Hello world", 10, 50);
}
```

Styling text

In the examples above we are already making use of the `font` property to make the text a bit larger than the default size. There are some more properties which let you adjust the way the text gets displayed on the canvas:

`font = value`

The current text style being used when drawing text. This string uses the same syntax as the CSS `font` property. The default font is 10px sans-serif.

`textAlign = value`

Text alignment setting. Possible values: `start`, `end`, `left`, `right` or `center`. The default value is `start`.

`textBaseline = value`

Baseline alignment setting. Possible values: `top`, `hanging`, `middle`, `alphabetic`, `ideographic`, `bottom`. The default value is `alphabetic`.

`direction = value`

Directionality. Possible values: `ltr`, `rtl`, `inherit`. The default value is `inherit`.

These properties might be familiar to you, if you have worked with CSS before.

The following diagram from the HTML spec demonstrates the various baselines supported by the `textBaseline` property.

A `textBaseline` example

This example demonstrates the various `textBaseline` property values. See the `CanvasRenderingContext2D.textBaseline` page for more information and detailed examples.

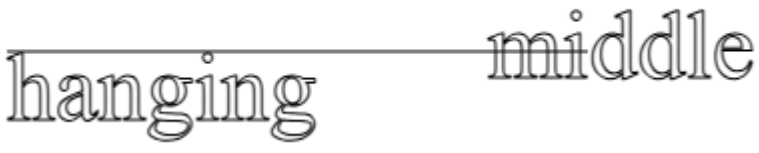
JS

```
function draw() {
  const ctx = document.getElementById("canvas").getContext("2d");
  ctx.font = "48px serif";

  ctx.textBaseline = "hanging";
  ctx.strokeText("hanging", 10, 50);

  ctx.textBaseline = "middle";
  ctx.strokeText("middle", 250, 50);

  ctx.beginPath();
  ctx.moveTo(10, 50);
  ctx.lineTo(300, 50);
  ctx.stroke();
}
```



Advanced text measurements

In the case you need to obtain more details about the text, the following method allows you to measure it.

`measureText()`

Returns a `TextMetrics` object containing the width, in pixels, that the specified text will be when drawn in the current text style.

The following code snippet shows how you can measure a text and get its width.

JS

```
function draw() {
  const ctx = document.getElementById("canvas").getContext("2d");
  const text = ctx.measureText("foo"); // TextMetrics object
  text.width; // 16;
}
```

Accessibility concerns

The `<canvas>` element is just a bitmap and does not provide information about any drawn objects. Text written on canvas can cause legibility issues with users relying on screen magnification. The pixels within a canvas element do not scale and can become blurry with magnification. This is because they are not a vector but letter-shaped collection of pixels. When zooming in on it, the pixels become bigger.

Canvas content is not exposed to accessibility tools like semantic HTML is. In general, you should avoid using canvas in an accessible website or app. An alternative is to use HTML elements or SVG instead of canvas.

← Previous Next →