Dashing D3.js

Learn D3 Faster Sign In

Adding an SVG Element

Basic Example

In the last example you added a **p** HTML element to the DOM. In this example, you will use D3.js to add an SVG element to a basic webpage. Per our previous SVG examples , you will add an SVG circle to the webpage.

Start with a basic HTML webpage:

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="d3.v2.min.js"></script>
  </head>
  <body>
    <p>Hello!</p>
  </body>
</html>
```

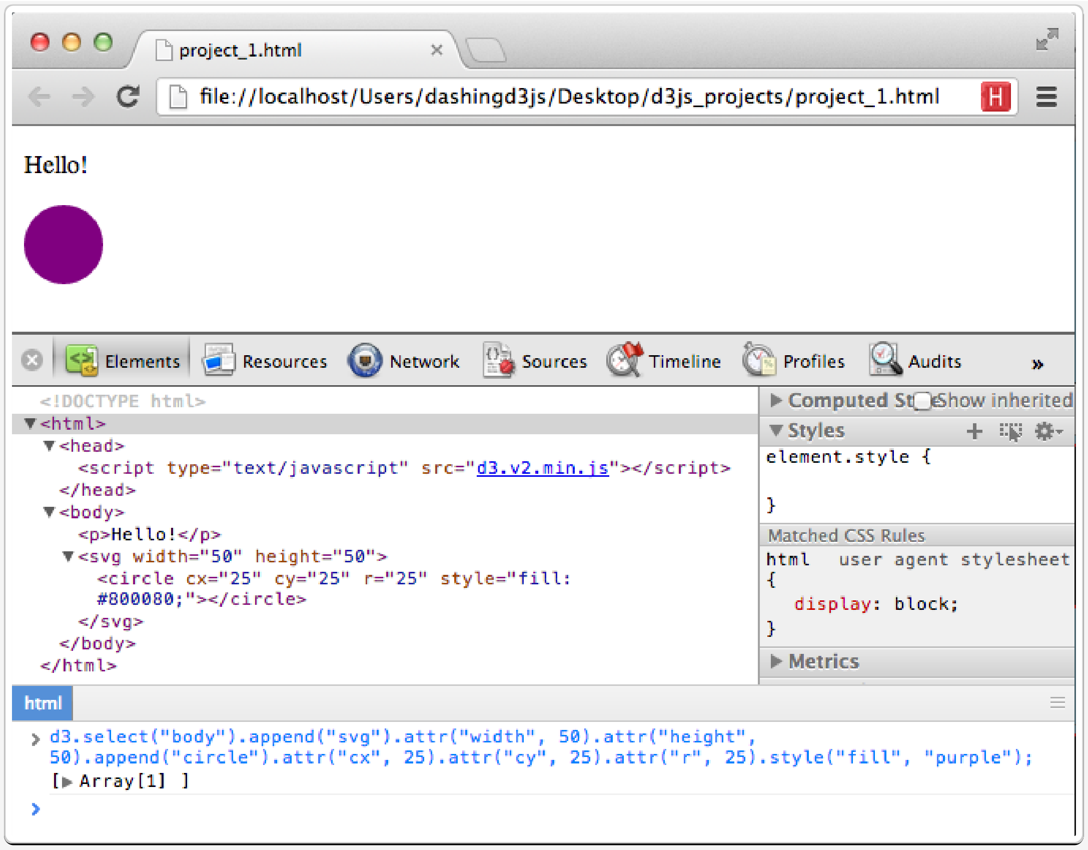
Recall that an SVG circle can be created as such:

```
<svg width="50" height="50">
  <circle cx="25" cy="25" r="25" fill="purple" />
</svg>
```

Next open the Developer Tools (Webkit Inspector). Then type the following long line into the JavaScript Console:

```
d3.select("body").append("svg").attr("width", 50).attr("height", 50).append("circle").attr("cx", 25).attr("cy", 25).attr("r", 25).style("fill", "purple");
```

This will give you the following:



Hello!

```
<!DOCTYPE html>
▼<html>
  ▼<head>
     <script type="text/javascript" src="d3.v2.min.js"></script>
   </head>
  ▼<body>
     <p>Hello!</p>
    ▼<svg width="50" height="50">
       <circle cx="25" cy="25" r="25" style="fill:
       #800080;"></circle>
     </svg>
   </body>
 </html>
```

```
element.style {

}

html    user agent stylesheet
{
   display: block;
}
```

```
> d3.select("body").append("svg").attr("width", 50).attr("height",
  50).append("circle").attr("cx", 25).attr("cy", 25).attr("r", 25).style("fill", "purple");
  [▶ Array[1] ]
>
```

Congratulations - you've added an SVG element to the DOM using D3.js!

D3.js Legibility

As you progress further into D3.js, the code that you write will go from a few lines to potentially a couple hundred lines. Even in our example:

```
d3.select("body").append("svg").attr("width", 50).attr("height", 50).append("circle").attr("cx", 25).attr("cy", 25).attr("r", 25).style("fill", "purple");
```

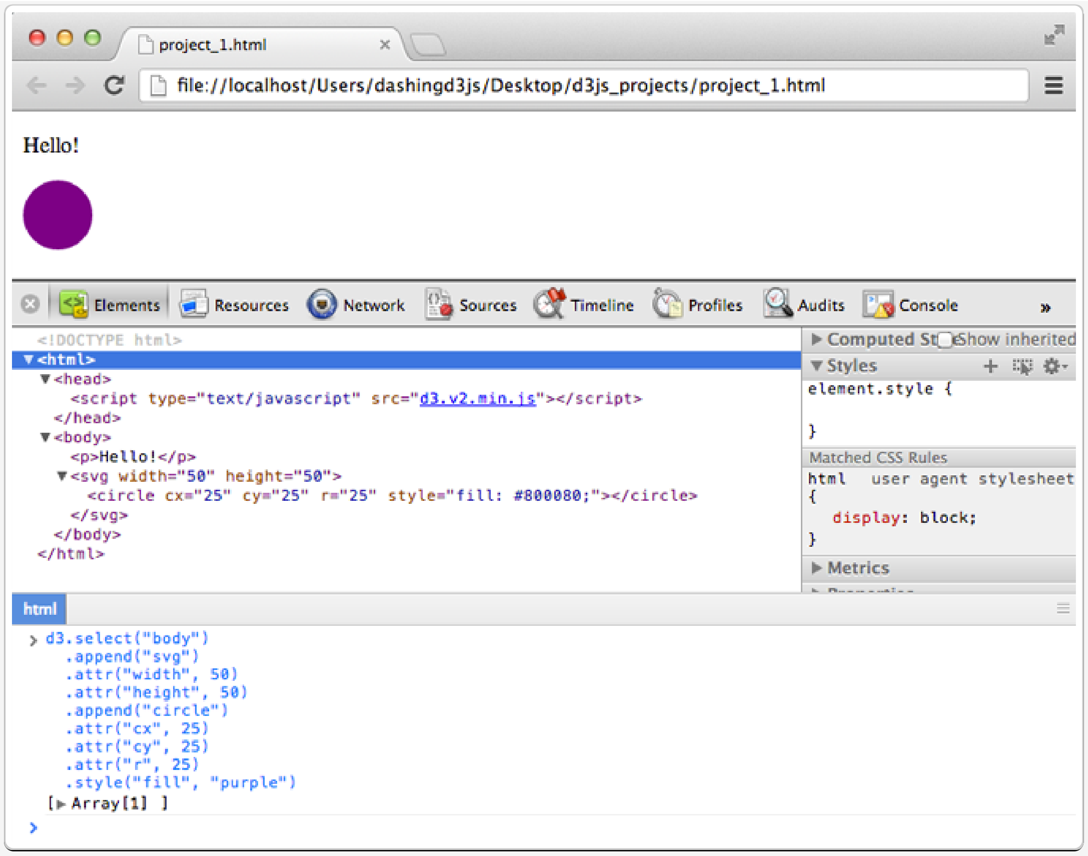
It is hard to follow what is happening all the way through. Luckily, there is hope.

JavaScript, much like HTML, does not care about white spaces or new line breaks. Which means we can rewrite our code as follows in the JavaScript Console:

```
d3.select("body")
  .append("svg")
  .attr("width", 50)
  .attr("height", 50)
  .append("circle")
  .attr("cx", 25)
  .attr("cy", 25)
  .attr("r", 25)
  .style("fill", "purple");
```

Note: to get a new line in the JavaScript Console without having the statement evaluated, hold down the **shift** key while pressing **return**.

This will give you the following:



```
<!DOCTYPE html>
▼<html>
  ▼<head>
      <script type="text/javascript" src="d3.v2.min.js"></script>
    </head>
  ▼<body>
      <p>Hello!</p>
    ▼<svg width="50" height="50">
        <circle cx="25" cy="25" r="25" style="fill: #800080;"></circle>
      </svg>
    </body>
</html>
```

```
element.style {

}
```

```
html    user agent stylesheet
{
   display: block;
}
```

```
> d3.select("body")
    .append("svg")
    .attr("width", 50)
    .attr("height", 50)
    .append("circle")
    .attr("cx", 25)
    .attr("cy", 25)
    .attr("r", 25)
    .style("fill", "purple")
  [▶ Array[1] ]
>
```

More legible code for the same functionality.

D3.js Style Operator

The very last line of the JavaScript code is **.style("fill", "purple")**.

The Style Operator, if a *name* and *value* is specified, sets the CSS style property for the given selection with the given specified value.

The *selection* given in our example was **everything that came before the statement**.

The *CSS style property* given in our example was **"fill"**.

The *specified value* given in our example was **"purple"**.

This powerful because we can apply any CSS style property to any type of selection.

D3.js Chain Syntax

If we look at our code again, you will notice a **period** in front of every method and operator:

```
d3.select("body").append("svg").attr("width", 50).attr("height", 50).append("circle").attr("cx", 25).attr("cy", 25).attr("r", 25).style("fill", "purple");
```

This syntax is called *chain syntax*.

If you have used the jQuery JavaScript Librarybefore , the *chain syntax* should be familiar to you. The D3.js chain syntax works much the same way.

If you have not used jQuery before, let us see what is happening:

We start at the left and apply each *.method* or *.operator* to it.

So *.select* is applied to *d3*

.append is applied to *d3.select("body")*

etc...

The longer the chain the more things are being applied to that which is to the left of it.

The reason this works is because the operator method returns a selection.

Which means that not only does *d3.select("body")* return a selection, so does *d3.select("body").append("svg")*!

Because we can apply an operator method to selections, we can continue to apply operators to operators that have returned selections.

Note - not all D3.js methods return a selection. However, most do so we can use chaining syntax to keep the code clean.

Looking at our code again:

```
d3.select("body").append("svg").attr("width", 50).attr("height", 50).append("circle").attr("cx", 25).attr("cy", 25).attr("r", 25).style("fill", "purple");
```

You can see that we move from left to right. A selection is returned. An operator is applied, which returns a selection. Then an operator is applied to that selection... so on and so forth.

This is important to realize because it means **order matters**.

Selections as JavaScript Variables

One of the great things about D3.js returning selections is that we can assign those selections to a JavaScript Variable. Thus we can go from this:

```
d3.select("body")
  .append("svg")
  .attr("width", 50)
  .attr("height", 50)
  .append("circle")
  .attr("cx", 25)
  .attr("cy", 25)
  .attr("r", 25)
  .style("fill", "purple");
```

To this:

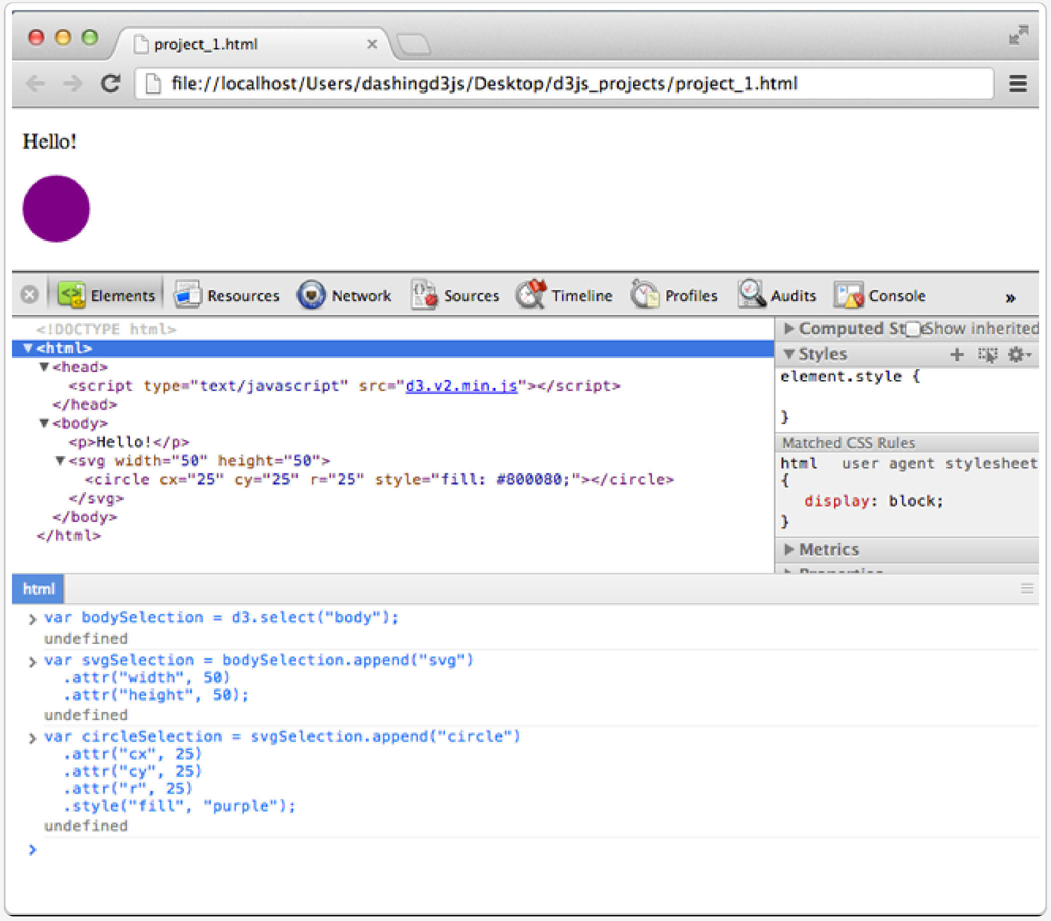
```
var bodySelection = d3.select("body");

var svgSelection = bodySelection.append("svg")
      .attr("width", 50)
      .attr("height", 50);

var circleSelection = svgSelection.append("circle")
      .attr("cx", 25)
```

```
        .attr("cy", 25)
        .attr("r", 25)
        .style("fill", "purple");
```

Which when typed into the JavaScript Console gives us this screenshot:



```
<!DOCTYPE html>
▼<html>
  ▼<head>
     <script type="text/javascript" src="d3.v2.min.js"></script>
   </head>
  ▼<body>
     <p>Hello!</p>
    ▼<svg width="50" height="50">
       <circle cx="25" cy="25" r="25" style="fill: #800080;"></circle>
     </svg>
   </body>
</html>
```

```
element.style {

}
html    user agent stylesheet
{
  display: block;
}
```

```
> var bodySelection = d3.select("body");
  undefined
> var svgSelection = bodySelection.append("svg")
    .attr("width", 50)
    .attr("height", 50);
  undefined
> var circleSelection = svgSelection.append("circle")
    .attr("cx", 25)
    .attr("cy", 25)
    .attr("r", 25)
    .style("fill", "purple");
  undefined
>
```

Which is the same as before. How you decide to write the D3.js code is up to you. Organizing the code by variables makes it easier to read and understand what the code does.

Want to better understand this topic? Check out this step-by-step course => Introductory D3 Training

← Adding a DOM Element

Binding Data to DOM Elements →

Learn D3.js

D3 Tutorial
D3 Screencasts
D3 Mapping Training
D3 Introductory Training
D3 Intermediate Training
D3 Advanced Training
D3 Corporate Training

DashingD3js.com

Blog
About
Hire Me
D3 Examples
D3 Resources
D3 & Data Viz Newsletter Archive

Data Visualization & D3.js Weekly Newsletter

Get D3.js and Data Visualization news, articles, jobs and more delivered to your inbox every Tuesday:

E-mail

Get the Newsletter

Did you sign up for the newsletter? :)