Overview

Our goal is to create interactive visualizations viewable in your, or anyone's, web browser. To get content into the browser we will use **HTML** documents, to make them interactive we will include **JavaScript** in these documents. Finally, to tend to our graphical needs, we will use **D3**, a JavaScript library, for creating visualizations.

For this tutorial, we will keep our use direct of HTML, CSS, and JavaScript to a minimum and focus on D3.

Some examples of what is possible with D3 can be found here:

- http://bl.ocks.org/mbostock
- http://christopheviau.com/d3list/

HTML (Hypertext Markup Language)

Most of what we well be doing today will happen in a HTML document. HTML uses tags to structure content and define how they are presented. Usually they look somewhat like:

<tagname> content </tagname>

Tags affect the content written between the tag opening and the tag closing (identifiable by the '/").

Common tags are for example <b></b> for **bold** text, <i></i> for *italic* text, or <p></p> for paragraphs.

Every HTML documents needs to include some boilerplate code:

```
<!DOCTYPE HTML>
<html lang="en">
        <head>
                <meta charset="UTF-8">
                <title></title>
        </head>
        <body>
        </body>
</html>
```

Most things will happen within the body tag.

Mini-Exercise

Now create a HTML file including some text in the body utilizing at least one tag.

To do so:

- Create a new file on your computer
- Give it the ending .html
- Paste the boilerplate code
- Fill in some content
- See if it works in your browser

IDs and Classes

Later in this tutorial we want to be able to address specific elements or element groups. To achieve this we can assign attributes to tags that give them a certain identity. Attributes are defined in the first part of the tag, after its name.

```
<div id="myID">This div has a unique ID. No other element can have the same. </div>

<div class="myClass">This div has member of a class. The class can be shared by multiple elements.
</div>
```

The name of the class or ID is a string that can be decided by you, the developer.

A list of tags and their effect can be found here:
https://developer.mozilla.org/en-US/docs/Web/HTML/Element

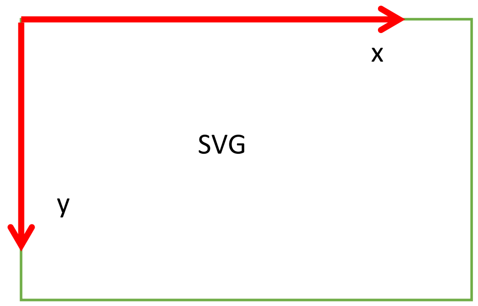
More information regarding HTML is available here:
http://www.w3schools.com/html/

SVG (Scalable Vector Graphics)

SVGs are part of the HTML5 standard and provide us with the means of creating graphical elements that can be displayed in the browser.

A main concept about SVGs is that we do not define our graphical elements pixel by pixel but rather *what* element we want, e.g. a circle, and what *properties* it should have, e.g. size, color, … One advantage of this approach is that resizing of the elements does not decrease their quality.

The coordinate system of SVGs has its origin at the top left corner and has its x axis to the right and its y axis downwards.



Another important thing to note is that the depth ordering of elements is defined by the order in which they are drawn, superimposing the new over the old ones.

To use SVGs in we first have to define an SVG element, and give it a size:

```
<svg width="400" height="200"> </svg>
```

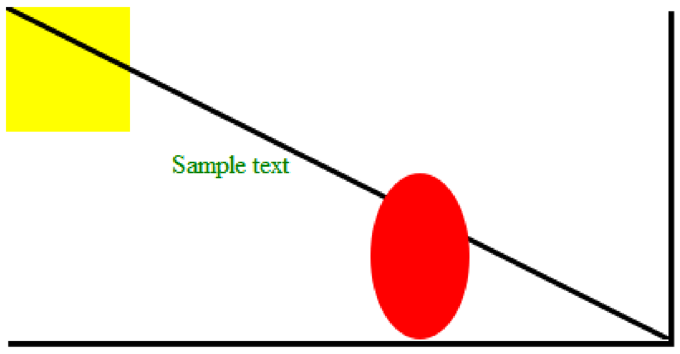
You can think of this as a canvas.

Within the svg tag we can define graphical elements, e.g. circles, ellipses, rectangles, lines, text, …

For example:

```
<svg width="400" height="200">
        <rect x="0" y="0" width="75" height="75" fill="yellow" />
        <line x1="0" y1="0" x2="400" y2="200" stroke="black" stroke-width="3" />
        <ellipse cx="250" cy="150" rx="30" ry="50" fill="red" />
        <text x="100" y="100" fill="green">Sample text</text>
</svg>
```

Results in:



A list of SVG elements can be found here:
https://developer.mozilla.org/en-US/docs/Web/SVG/Element

D3

To dynamically add SVG shapes we will use JavaScript and D3.
First we have to load the D3 library. To do that:

- Download d3: https://d3js.org/
- Unzip the folder
- Take d3.min.js and put it in the same folder you .html file resides in.

Now you can load the d3 library from within your HTML document by simply adding the following line:

```
<script src="d3.min.js"></script>
```

We can now use D3 from within a script tag. This library offers a wealth of methods to dynamically modify the HTML document, and provides the *d3* object which is needed to call them.

Select

To choose which objects to manipulate, D3 provides the *select("mySelector")* method, which returns the first element in your document to match the selector, and the *selectAll("mySelector")* method that returns all elements matching the selector.
You can select elements by various criteria, the most important ones being by tag, by class, and by id (we have discussed class and id earlier):

```
d3.select("div")  <!—Targets the first "div" tag -->
d3.select(".class") <!—Targets the first tag with the class "class" -->
d3.select("#id") <!—Targets the tag with the id "id" -->
```

More about select here: https://github.com/mbostock/d3/wiki/Selections

Append

Another important method is *append("elementToAppend")*. This method can be called on the return value of the *select()* or *selectAll()* and simply adds new elements as children to the selected nodes.

The parameter we pass to the append() method is a string defining the type of element we want to add. E.g.: div, p, svg, a, h1, …
Some elements might be addressed via their namespace-prefix, e.g.: svg:circle

More about *append* here: https://github.com/mbostock/d3/wiki/Selections#append

We can now, for example, append a new div and fill it with text:

```
<script>
		var body = d3.select("body");
		var myDiv = body.append("div");
		myDiv.text("Hello World!");
</script>
```

JavaScript allows us to do that in an even more compact way, via *method chaining*. This enables us to call multiple functions (on the same object) consecutively:

```
d3.select("body").append("div").text("Hello World!");
```

Method calls are connected with periods and the output of each method is used as the input for the next one.

Styles and Attributes

Each element you select can be modified with certain *attributes*, like class, id, etc., and CSS *styles*, like color, stroke, opacity, etc.

To set these values D3 offers the *.attr(name, value)* and the *.style(name, value)*:

Fr example we can give all our divs the same class and change their text color:

```
d3.selectAll("div").attr("class","myDivClass");
d3.selectAll(".myDivClass").style("color","orange");
```

More about *.attr():*

https://github.com/mbostock/d3/wiki/Selections#attr

More about *.style():*

https://github.com/mbostock/d3/wiki/Selections#style

A list of styles:

http://www.w3schools.com/cssref/default.asp

Excursion: Debugging

Something very helpful when working with HTML and JavaScript are the inspection and debugging capabilities of your browser. Those differ from browser to browser, but the most popular ones all have the same basic tools to aid you.

For example in Firefox:
The **Inspector** (Extras -> Web-Developer -> Inspector) lets you look at the current elements on your page.

This is helpful as it allows you to see if elements have been successfully created and what attributes they have. If you cannot see an element that should be there, it might be that it has been created, but some attributes area not defined or set to values that make the object not appear on the screen.

The **Console** (Extras -> Web-Developer -> Inspector) displays you error messages and warnings that might occur in your JavaScript part.

If you prefer a different browser, it should be easy to find the pendants to these tools by googling.

Exercise

In your HTML file:

- Include D3
- Using HTML: Create an SVG
- Using d3: Add a blue circle of radius 60 to your SVG and make sure it's completely visible
- Using d3: Add a paragraph containing green text to your document – all in one line.

Data

For data visualization it is, of course, necessary to be able to access data. For now we will work with static data we hard-code in our HTML file. Inside the script tag, define a JavaScript array:

```
var data = [1, 5, 15, 20, 25];
```

We can bind this data to HTML elements using the *.data(data)* method after the selection.

```
d3.select("body").selectAll("p").data(data).text("Sample text");
```

So now if we have the same number of items in our array as we have paragraphs, each of the paragraphs get its text set.

Usually we don't want to create our elements beforehand so that they match the number of data-items in our array. That's why d3 provides the *.enter()* method that tells the browser what to do if there are not enough HTML-elements for the number of data items. (.enter() will be explained more thoroughly in the next tutorial)

Consequently, we can use the following code to create paragraphs on the fly if we do not already have enough in our document.

```
d3.select("body").selectAll("p").data(data).enter().append("p").text("Sample text");
```

This is an important concept in D3, as it lets us select elements, which do not yet exist.

But we also want our changes to be dependent on the different data-items. For this we can use *functions*, a concept you should be familiar with from other languages. In D3 we often use *anonymous functions* that only differ from normal functions by not having a name, and which thus cannot be reused.

```
d3.select("body").selectAll("div").data(data).enter().append("div").text(
  function(d){
    return d;
  });
```

The parameter d is provided as first argument to anonymous functions by D3 and holds the data elements. Optionally, you can also pass the index of the data-item as second parameter to your functions.

Storing Elements

Of course you can also store the elements you have created in variables so you can access them even easier than with selections. For your assignments, you will probably want to save the SVG in a variable.

```
var mySvg = d3.select("body").append("svg");
```

Exercise

- Create an array of at least 4 numbers
- Create an SVG element and store it in a variable
- For each data-item in your array create an SVG-shape with its position being a function of the index of the respective data-item and its size depending on the value of the data-item.
- Check your results in the browser and make sure that all circles are at least partially visible.

Load Data

D3 also allows us to load real data in various formats; in this tutorial we will use a .csv file.

D3 provides the *.csv("datafile.csv", function(data){ })* method. It takes two parameters, a string with the path to the file, and an aonymous *callback* function.

```
d3.csv("fruit.csv", function(data) {
        console.log(data);
});
```

D3 loads data asynchronously, so that interaction with the browser is not hampered while loading (scripts not related to the data can still run in the meantime). To signal that the loading is complete, D3 calls the callback function.

This is an important concept, as it means for developers that code which depends on the data should only exist in the callback function.

Conversion

As D3 loads all our data as strings, we have to convert it to numbers to work with it. This can be done with the "+" operator.

```
data.price = +data.price;
```

But as this has to be done for every item in the data set, you have to iterate over all items, either by using a loop or with the following statement:

```
data.forEach(function(d){ d.price = +d.price });
```

Exercise

- Download the data http://vda.univie.ac.at/Teaching/Vis/16s/data/fruit.csv
- Load the data
- Convert the price to numbers
- Create a simple barchart with the bar length corresponding to the price, each bar labelled by the fruit name and the bar color encoding the pit-type.

Acknowledgements

Many thanks to Michael Opperman for all the help and material he provided.