Lab 1: Basics

Data Visualization and Analytics

The labs enable you to demonstrate a variety of skills, such as, problem solving, communication skills, time-management and so on. The labs are structured to encourage a mix of guided and active learning, innovative development, critical thinking and knowledge-based expertise of the subject.

Each lab requires you to discuss/explain and justify your implementation details (also extra check to verify that you understand concepts/reasons/limitations). The lab also offers you the chance to discuss or ask any questions while getting guided feedback on your progress/lab work.

1. Each lab must be demonstrated during lab session (multiple sessions/opportunities each week)
2. Report and code must be submitted on CANVAS (evidence) for each lab (after the demonstration to a lab helper/staff)
3. After demonstrating your work (you have until the end of the week to submit a report and code - single zip as evidence/check on CANVAS)

Note – If you fail to demonstrated/discussed/explained your work then you get 0% (even if a report is submitted). The report is a 'check' while the demonstration/discussion formulates the mark. If circumstances prevent you from demonstrating or attending the lab session, please inform the lecturer. Important, there are multiple sessions that give you an opportunity to demonstrate/complete the task early (don't leave it until the last minute).

Each lab/code must be to the highest standard (e.g., commented code, naming convention, structured, organized, tested, ...).

This lab is designed to give you an introduction to D3 Visualization Concept. Only a small subset of the operations are described in this lab but should give you an idea of the potential capabilities of the library/tools. The lab includes a number of exercises that you can work through to help you get started (i.e., they're not assessed but to aid you in working up to the assessed tasks). You're encourages to work through all the exercises and modify programs to perform similar tasks (experiment). You can practice this lab in your own time before the scheduled assessed lab sessions. During the lab sessions, once you have completed the exercises on this sheet, show it a lab helper who will mark it as completed. **This is worth 25% of the marks for this course**. Check the timetable for the scheduled sessions for this lab.

Resources:

- F20/21DV Material (https://f21dv.github.io/)
- D3 Homepage/Documentation (https://d3js.org/)

Overview, by the end of this **lab 1**, the students should have

- Setup github repository (https://pages.github.com/) (No marks for this)
- Simple graphs with d3.js (line, bar, pie, scatter plot)
- Plot multiple data sets on the same graph
- Symbol markers for line points/graphics (e.g., points and svg background of graph)
- Load data from different sources e.g., csv, json,
- Convert data between formats (maps/ranges)
- Different scales and color schemes
- Design, develop and present basic visualization graphics

FAQ

1. How should I manage my work/exercises?

As you work through the exercises in each lab you should keep a record of your progress (e.g., separate files, comments in code, name/date at the top of each file, …). You should manage your files/folders and logs.

2. What should I include in my report? What format?

Your report should be formatted to a professional standard which documents your progress/exercise results (e.g., lab report structure that detailing each exercise/results and reason/reflection). At the top of the report on the first page, you should include your name, which lab, the date and who you demonstrated your work to during the lab session

3. Do the exercises have to be completed in the order they're given?

Each exercise increases in complexity and builds upon previous tasks (hence, you need to work through them in order)

4. The answer isn't in the lab worksheet?

In addition to the lab notes, you'll also have to refer to the lectures, recommended resources/documentation/support material (i.e., not just copy and repeating each lab exercise – you also have to demonstrate active learning, understanding, problem solving and original thinking). Some exercises may require you to demonstrate basic knowledge and problem solving while other exercises more critical thinking/analysis of the task

5. Do I only get one chance to demonstrate my lab work?

There are multiple lab sessions allocated to each lab (multiple weeks), so you have several chances to complete and demonstrate your work (i.e., not a single deadline). You can demonstrate and complete the lab exercise early (don't leave it until the very end). Recommended that you start early and this way if there are any issues or problems you'll have time to address these and resolve them for your final demonstration/submission.

6. Which version of D3 do I need to use?

Your lab exercises should run using v7+ of d3.js

7. When will I get feedback/mark for my lab coursework?

After the deadline has passed, your preliminary mark will be released with your submitted report on CANVAS (i.e., you need to have demonstrated/discussed your work during the lab session).

8. How many marks are each lab worth?

25% each.

9. How many lab courseworks are there?

There are 4 lab courseworks.

Assessment Breakdown

Task

The assessed lab courseworks consist of understanding, designing, developing, implementing and testing visualization scenarios. The requirements for the courseworks are:

1. **You should attend and contribute** to the labs.
2. You need to conceptualise, develop, and test a **prototype of your visualizations**.
3. You need to plan, implement and debug **prototype of your visualizations**.
4. You need to incorporate these tested visualizations into designs and demonstrate/discuss/**presented them**.

Each assessed labs is worth 25% of your overall course mark. The concepts necessary to complete the assessment are available and covered during the lectures/support material/recommended texts.

This lab coursework will be marked as following:

Sections	Description	Marks
Exercises/Visualization Prototypes	In this section your ability to complete/develop/extend exercises/prototype will be assessed.	18
Communication, Analysis, Presentation Visualization	Marks will be awarded for demonstrating creative thinking, feedback and concept used. Show understanding of the material and the appropriateness and effectiveness of associated visualisation and analysis techniques	5
Management and documentation	A couple of mark will be awarded for management, documentation and organisation of the tasks/code/exercises	2
	Total Marks F2xDV Assessment	**25**

Marking Details

Below are the marking guidelines for the lab courseworks.
There are allocated sessions for demonstrating your exercises in the timetable (lab sessions).
The demonstration sessions are run over multiple weeks, you should organise and prepare for your demonstration (i.e., organised and able to show your work/exercises/answers in addition to being able to answer questions, e.g., results, concepts used, modifications).
Note: You will need to both demonstrate your work during one of the assigned lab sessions; after this, you also need to submit a report and your code (single file zipped together) on CANVAS (evidence).
Late submissions will be subject to the normal penalties as defined in the late coursework policy. If you have any questions or queries about the assessment, please do not hesitate to contact B.kenwright@.ac.uk (Edinburgh) or Ryad Soobhany (Dubai)
Feedback and your marks: you will receive feedback for this coursework during the demonstration/discussion.

Late Submissions

The University recognises that, on occasion, students may be unable to submit coursework on the submission date or be unable to present their work on the submission date. In these cases, the University's Submission of Coursework Policy outlines are:
• No individual extensions are permitted under any circumstances.
• Standard 30% deduction from the mark awarded (maximum of five working days).
• In the case where a student submits coursework up to five working days late, and the student has valid mitigating circumstances, the mitigating circumstances policy will apply, and appropriate mitigation will be applied.
• Any coursework submitted after five calendar days of the set submission date shall be automatically awarded a no grade with no formative feedback provided.
Please contact your Personal Tutor or Councelor if you are unable to meet the deadlines or need information for Mitigating Circumstances or Temporal Suspensions of Studies.

Part 1. Getting Started – GitHub Pages

GitHub is a crucial resource which is used by many organisations around the world. A number of projects are hosted and sourced through GitHub, such as “d3.js”, “React”, “Bootstrap”, “jQuery”. GitHub and git are considered industry standard for resources and tools.

Why Learn/Use GitHub?

- Code reviews are easy to do with Github. For instance, developers can comment on other peoples’ code and approve or reject changes for pull requests. They can also request specific improvements to the code before it is merged.
- Github helps merge production-ready code from different developers. They provide automated merging based on git and you can also do many operations that would normally need the command line directly from their web-based UI.
- Tracking and searching through the version history of the code is 10x faster and easier with Github compared to other git servers/clients.
- Adding collaborators to the organization and managing access credentials for incoming and outgoing participants is particularly easy with Github.

If you’re new to Git/GitHub or haven’t had much of a chance to use/develop resources – it’s recommended for this part to setup a GitHub repository and create a ‘pages’.

See link: https://pages.github.com/

Note, it provides a secure, safe and flexible resource for managing and storing your programs (e.g., create private repository for your tasks so they’re safely backed up). Also allows you to develop good practices which can be demonstrated/showed to an employer (portfolio).

Try and maintain your repository, such as commenting code (consistent style), readme files, logical folder/file names, ..

Part 2. D3 Setup

You can use one of the many online tools to test/debug your programs (e.g., JSFiddle or CodeSandbox) – which will all run in a browser. However, be sure to keep backups of your work and track revisions/versions.

(a) Initialize D3.js (using v7) and check it's loaded (e.g., display the version number)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <script src='https://d3js.org/d3.v7.min.js'></script>
</head>
<body>

<script>
   // write your d3 code here..

  console.log('d3.version:', d3.version);
</script>
</body>
</html>
```

Exercise 1: What version number is displayed in the console output window?

(b) Document Object Model (DOM)

Demonstrates selecting objects ('select') and either adding to them or manipulating their attributes.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<p>First paragraph</p>
<p>Second paragraph</p>

<script>
   d3.select("p").style("color", "red");
</script>
```

The example above should change the 'First paragraph' text to 'red'.

Exercise 2: Change other style properties of the paragraph tag (e.g., font-size, line-height, font-family, contents, …)

There are different ways of selecting elements in the DOM (type, id or class name). The `select' option provides an efficient and compact way to select/update elements in the DOM.

For example, change all 'paragraphs' with 'class name' of 'test' to 'blue' or select a specific element by its 'id'.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<style>
   .test {
      color:'red'
   }
</style>
<p class="test">First paragraph</p>
```

```
<p>Second paragraph</p>
<p class="test">Third paragraph</p>

<script>
   d3.selectAll(".test").style('color','blue');
</script>
```

(c) Appending DOM
Use d3.selection.append() method to create a new DOM element and add it at the end of selected DOM element.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<span>Test</span>

<script>
   let newspan = d3.select("body").append('span');
   newspan.text('hello');
</script>
```

You can select the page 'body' and append 'new' elements (e.g., div, span, p, svg, …). After they've been added you can modify attributes (e.g., contents, styles, ..).

Exercise 3: Write a loop which adds 10 'div' elements and sets the contents to the count value (i.e., 1, 2, 3, …). Also the color of the first 5 elements are red and the last 5 elements are green (lookup the attr(), property() and style() methods).

Exercise 4: 'selecting' and modifying your 'div' elements after you've created and added them (e.g., select the first div element and make its content text equal to 'start' – also change its color to 'purple'.

(d) Chaining
The "chain syntax" is an efficient and compact way of representing multiple operations (i.e., instead of storing details in temporary variables).

```
<script src='https://d3js.org/d3.v7.min.js'></script>

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

// same as

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

Exercise 5: Exercise: Add to the 'chain syntax' version for the 'hello world' example above – so it also sets the 'color' of the text to green.

Part 3. Data

The D3 methods such as append(), style(), text() etc. Each of these functions can take in a constant value or a function as a parameter. So each of these methods will be called for each of our data values bound to the DOM. Consider the following text() function.

For example, if you have an array of data (numbers), you can select elements in the DOM and update/map the data to them.

The following code takes an array of data called 'nums'. The 'selectAll' selects all the 'div' elements. The real magic is the following '.data' method. The data() function provides the array values to the selected elements, in your case below it is data from the array.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<div></div>
<div></div>
<div></div>

<script>
let nums = [10, 50, 200];

let paragraph = d3.select("body")
        .selectAll("div")
        .data(nums)
        .text(function (d, i) {
          console.log("d: " + d);
          console.log("i: " + i);
          console.log("this: " + this);

          return 'cont:' + d; // return value is used to set the 'text'
        });
</script>
```

In the code above, the parameter "d" gives you your data element, "i" gives you the index of data in the array and this is a reference of the current DOM element. In this case, it is the paragraph element.

Taking this further, you can also 'check' the data and apply different attributes/effects to the DOM elements.

The example above, 'chaining' a style function to check the value (if greater or equal to 100 use red).

Data management/conversion

Note, in the examples for this lab, you'll primarily use a fixed array of numbers, e.g., 'a=[1,2,3]', however, you can also pass an array of objects to the 'data' method, e.g., 'a=[{name:"test", val:1,}, {name:"other", val:2}]'. These can be managed and accessed using the dot notation.

As shown in the following example:

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<div></div>
<div></div>
<div></div>

<script>
let otherdata = [ {name:'test',  val:1},
```

```
            {name:'other', val:2},
            {name:'b',     val:3} ];

let paragraph = d3.select("body")
            .selectAll("div")
            .data(otherdata)
            .text(function (d, i) {
               console.log("d.name: " + d.name);
               console.log("d.val:  " + d.val);
               console.log("i: " + i);
               console.log("this: " + this);

               return 'cont:' + d.name; // return value is used to set the 'text'
            });
</script>
```

Exercise 6: Modify the example above so the 'otherdata' contains an additional variable called color (print this color value out in the 'text' method.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<div></div>
<div></div>
<div></div>
<div></div>


<script>
let num = [10, 50, 100, 200];

let paragraph = d3.select("body")
            .selectAll("div")
            .data(num)
            .text(function (d, i) {
               return 'cont:' + d; // return value is used to set the 'text'
            })
            .style("color", function(d, i) {
               if ( d >= 100 ) {
                  return "red";
               } else {
                  return "yellow";
               }
               return 'blue';
            });

</script>
```

Exercise 7: Change the bounds check so the color is red for numbers between 50 and 100.

NOTE: If you add extra numbers to the array – only the 'div' elements that currently exist are processed (you'll learn about adding/removing elements based on the data next).

Part 4. Data Binding

Bind data to DOM elements so you can not only update current elements but also 'create' or 'remove' elements based on your data.

- data() (current elements) Joins data to the selected elements
- enter() (new elements) Creates a selection with placeholder references for missing elements
- exit() (unused/deleted elements) Removes nodes and adds them to the exit selection which can be later removed from the DOM

You used 'data()' method previously to iterative over current elements and update their properties based on data. However, if you had more or less data values than DOM elements that was a problem. The 'enter()' and 'exit()' methods allow you to add and remove elements to match the data size.

If you try the following example, it will 'run' but nothing with show up. As your selectAll('p') doesn't find any paragraphs on your page.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
    var myData = ['a', 'b', 'c', 'd' ];

    var p = d3.select("body")
            .selectAll("p")
            .data(myData)
            .text(function (d, i) {
                return d;
             });
</script>
```

However, if you add the 'enter()' method, this is called each time a new data element is found that doesn't have an associated element from the selection.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
var myData = ['a', 'b', 'c', 'd' ];

var p = d3.select("body")
        .selectAll("p")
        .data(myData)
        .enter()
         .append('p')
         .text(function (d, i) {
                return d;
         });
</script>
```

Step by step what's happening:

d3.select("body")
This statement selects the HTML Body.

.selectAll("p")
At this point, there are no paragraph elements in the body. So this will return an empty array.

.data(data)
You provide your data array to the data() function. Since your array has four elements, the code after this will run four times for every element in the array.

.enter()
The enter() function checks for <p> elements corresponding to your elements. Since it does not find any, it will call this for each 'p' that needs to be created (i.e., each of the array elements).

.append("p")
And this will append above to the body element.

.text(function(d) { return d + " "; });
And finally our text function adds each of the values from the data array as text to each of our paragraph selections and prints them onto the screen!

Exercise 8: Modify the above code, use the following data:
var myData = ['a', 4, 1, 'b', 6, 2, 8, 9, 'z'];
Instead of a paragraph 'p' use a 'span'. If the data element is a 'character' display it as 'blue', if it's a number 'display it as green' (note you'll need to 'chain' a '.style' method after you've appended the new elements).

The 'enter()' method is called when new elements are need, while the 'exit()' method is called when elements need removing.

In the following example, there are 3 paragraph elements, but only 2 values in the data array. The first two paragraph values get updated to match the 'data'. While the last one, is passed to the 'exit()' method. In the example below, it's 'removed()' from the DOM (not needed anymore).

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<p>1</p>
<p>2</p>
<p>3</p>

<script>
   var myData = [3, 4];

   var p = d3.select("body")
            .selectAll("p")
            .data(myData)
            .text(function (d, i) {
               return d;
            })
            .exit()
            .remove();
</script>
```

Part 5. Loading Data

D3 provides the following methods to load different types of data from external files

- d3.csv() Sends http request to the specified url to load .csv file or data and executes callback function with parsed csv data objects.
- d3.json() Sends http request to the specified url to load .json file or data and executes callback function with parsed json data objects.
- d3.tsv() Sends http request to the specified url to load a .tsv file or data and executes callback function with parsed tsv data objects.
- d3.xml() Sends http request to the specified url to load an .xml file or data and executes callback function with parsed xml data objects.

d3.csv() signature:

```
d3.csv(url[, row, callback]);
```

Take the following CSV file (passenger details for the Titanic). First line defines the column names. The csv function, will call your callback function for each line as it processes the csv.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>

let titaniccsv = 'https://raw.githubusercontent.com/dsindy/kaggle-titanic/master/data/test.csv';
/*
PassengerId,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked
892,3,"Kelly, Mr. James",male,34.5,0,0,330911,7.8292,,Q
893,3,"Wilkes, Mrs. James (Ellen Needs)",female,47,1,0,363272,7,,S
894,2,"Myles, Mr. Thomas Francis",male,62,0,0,240276,9.6875,,Q
895,3,"Wirz, Mr. Albert",male,27,0,0,315154,8.6625,,S
*/

d3.csv(titaniccsv, function(data) {
   console.log( data.Name );
});
</script>
```

Exercise 9: For the example above, to count how many of the names include 'Mr.' and 'Mrs' (or other). Also print out other details using other column header information, such as, how many passengers are 'male' and how many 'female'.

For the following example, it provides a list of details on patients who suffered heart failure. Write code to determine how many patients had heart failure over the age of '50'.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>

let heartfailurecsv = 'https://raw.githubusercontent.com/akmand/datasets/master/heart_failure.csv';
/*
age,anaemia,creatinine_phosphokinase,diabetes,ejection_fraction,high_blood_pressure,platelets,serum_creatini
ne,serum_sodium,sex,smoking,time,DEATH_EVENT
75,0,582,0,20,1,265000,1.9,130,1,0,4,1
55,0,7861,0,38,0,263358.03,1.1,136,1,0,6,1
```

```
65,0,146,0,20,0,162000,1.3,129,1,1,7,1
50,1,111,0,20,0,210000,1.9,137,1,0,7,1
*/

d3.csv(heartfailurecsv, function(data) {
   //console.log( data.age);

   // todo
});

</script>
```

Exercise 10: Exercise: Write an update to the example above, so extra elements are added to the window to display information. For instance, display paragraphs for the total patients with heart failure between 1-30, 31-40, 41-60, 61-100. Process the data, store it in an array then pass that array to 'selectAll()', 'data()' as discussed in previous sections.

Part 6. SVG

The Scalable Vector Graphics (SVG) is just one of the web standards you can use with the D3 library for outputting visual information to the screen.

You can read more about the official specification/standard for SVG here: https://www.w3.org/TR/SVG/

As with other DOM elements, you can use d3 to add them to your webpage.

The following example shows you how to create and add an svg to your page, then to append a 'line' component.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
   //Create SVG element
   var svg = d3.select("body")
   .append("svg")
   .attr("width",  400)
   .attr("height", 400)
   .style("border", '1px solid green');

   //Create line element inside SVG
   svg.append("line")
     .attr("x1", 100)
     .attr("x2", 200)
     .attr("y1", 50)
     .attr("y2", 150)
     .attr("stroke", "blue")
</script>
```

What's happening in the above example:

You create an SVG with the specified dimensions. (400x400). You set the style and attributes. Note the style border is set to 1px solid green so you should see a green border around your SVG.

You then append a line element to your SVG and provide it with the x1, y1 x2, y2 and stroke attributes using attr() function.

Exercise 11: Exercise: Modify the code so the example draws a 'square shape' (4 lines) – each side of the square a different color.

Next, you'll add other graphics to the SVG, such as, 'rectangle', 'circle', 'ellipse', and 'text'. Notice with the SVG you specify the 'position' of each element.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
   //Create SVG element
   var svg = d3.select("body")
   .append("svg")
   .attr("width",  400)
   .attr("height", 400)
   .style("border", '1px solid green');

   //Create and append rectangle element
   svg.append("rect")
        .attr("x", 0)
```

```
        .attr("y", 0)
        .attr("width", 200)
        .attr("height", 100)
        .attr("fill", 'pink');

  //Append circle
  svg.append("circle")
    .attr("cx", 250)
    .attr("cy", 50)
    .attr("r", 50);

  svg.append("ellipse")
    .attr("cx", 250)
    .attr("cy", 50)
    .attr("rx", 150)
    .attr("ry", 50);

  //Create and append text element into group
  svg.append("text")
   .attr("x", 150)
   .attr("y", 50)
   .attr("stroke", "#fff")
   .text("This is some text!");
</script>
```

Exercise 12: Build an SVG scene which is created from an external file. You need to create a csv with the information about the shapes. You should include columns in your csv file for the type of shape (circle, rectangle, ellipse, line), its dimensions and position, and color. Your program reads the data and creates and displays the shapes to the screen.

Exercise 13: Extend the example to include the 'enter' and 'exit' concepts. So that the svg elements are updated, created or removed based on the csv data from your csv file.

Part 7. Bar Chart

Simple bar charts – which are popular visual graphs for displaying quantities.

Styles are added to the page instead of hardcoding them into the script (makes updating/customizing visuals easier). However, all of the visuals in the following example, could be 'scripted'. The example extends the previous concepts, but uses the 'rectangle' and 'text' svg elements to draw a simple bar chart.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<style>
svg rect {
        fill: blue;
}

svg text {
        fill:white;
        font: 10px sans-serif;
        text-anchor: end;
}
</style>

<script>
var data = [5, 10, 12, 6];

var width      = 200;
var scaleFactor = 10;
var barHeight   = 20;

var graph = d3.select("body")
    .append("svg")
    .attr("width", width)
    .attr("height", barHeight * data.length);

var bar = graph.selectAll("g")
    .data(data)
    .enter()
    .append("g")
    .attr("transform", function(d, i) {
      return "translate(0," + i * barHeight + ")";
    });

bar.append("rect")
  .attr("width", function(d) {
    return d * scaleFactor;
  })
  .attr("height", barHeight - 1);

bar.append("text")
  .attr("x", function(d) { return (d*scaleFactor); })
  .attr("y", barHeight / 2)
  .attr("dy", ".35em")
  .text(function(d) { return d; });

</script>
```

Exercise 14: Extend the simple bar chart example to display the heart failure data you processed in Part 5 (Part 5 - Loading Data) from the csv file. (i.e., age ranges for people with heart failure).

Exercise 15: Modify the simple bar chart to use color more (i.e., values over a certain threshold are displayed in ‘red’).

Part 8. Circle Chart

Instead of rectangles, another example create circles dynamically from data and styles them.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
const width  = 500;
const height = 500;

const data   = [10, 15, 20, 25, 30];
// Note different valid ways of specifying color
const colors = ['#ffffcc','red','rgb(0,255,0)','#31a354','#006837'];

const svg = d3.select("body")
          .append("svg")
          .attr("width", width)
          .attr("height", height);

const g = svg.selectAll("g")
          .data(data)
          .enter()
          .append("g")
          .attr("transform", function(d, i) {
             return "translate(0,0)";
          })

g.append("circle")
  .attr("cx", function(d, i) {
    return i*100 + 50;
  })
  .attr("cy", function(d, i) {
    return 100;
  })
  .attr("r", function(d) {
    return d*1.5;
  })
  .attr("fill", function(d, i){
    return colors[i];
  })

g.append("text")
 .attr("x", function(d, i) {
    return i * 100 + 40;
 })
 .attr("y", 105)
 .attr("stroke", "teal")
 .attr("font-size", "12px")
 .attr("font-family", "sans-serif")
 .text(function(d) {
    return d;
 });
</script>
```

Exercise 16: Add additional shapes to the concept (draws both circles and squares based on the data).

Part 9. Scales, Domain, Range

1. Scales

D3 Scales provide a convenient way to map your data values to values that would be better represented in visualizations. For example: d3.scaleLinear(), d3.scaleLog(), ..

2. Domain

Domain denotes minimum and maximum values of your input data. In our data [100, 400, 300, 900, 850, 1000], 100 is minimum value and 1000 is maximum value.

So, your domain is [100, 1000]

3. Range

Range is the output range that you would like your input values to map to.

You may not have enough space to display a bar chart for the above values, if you map your data values to pixels. Let's say you want to display a chart in SVG within 500 px width. So, you would like your output range between 50 to 500, where minimum value will be mapped to 50 and maximum value will be mapped to 500 that is [50, 500]. That would mean, an input value of 100 would map to an output value of 50. And an input value of 1000 would map to an output value of 500. It means scaling factor is 0.5 and the data will be represented in pixels as: data value * 0.5.

100 -> 50

1000 -> 500

So, now if our input value is 300, the output value would be 150.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
var data = [0, 400, 300, 900, 850, 1000];

var scale = d3.scaleLinear()
        .domain([d3.min(data), d3.max(data)])
        .range([0, 10]);

console.log( 'scale 0:',    scale(0) );    // 0 maps to 0
console.log( 'scale 1000:', scale(1000) ); // 1000 maps to 10
console.log( 'scale 200:',  scale(200)  );
</script>
```

Combining the concept with the 'bar chart' example, it means the data can be mapped to the size of the svg elements on screen (so they scale exactly to the desired size).

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
const data = [50, 400, 300, 900, 250, 1000]

const width     = 500;
const barHeight = 20;
const margin    = 1;

var scale = d3.scaleLinear()
```

```
      .domain([d3.min(data), d3.max(data)])
      .range([50, 500]);

var svg = d3.select("body")
       .append("svg")
       .attr("width", width)
       .attr("height", barHeight * data.length);

var g = svg.selectAll("g")
       .data(data)
       .enter()
       .append("g")
       .attr("transform", function (d, i) {
         return "translate(0," + i * barHeight + ")";
       });

g.append("rect")
  .attr("width", function (d) {
    return scale(d);
  })
  .attr('fill', 'blue')
  .attr("height", barHeight - margin)

g.append("text")
  .attr("x", function (d) { return (scale(d)); })
  .attr("y", barHeight / 2)
  .attr("dy", ".35em")
  .style('text-anchor', 'end')
  .text(function (d) { return d; });
</script>
```

Exercise 17: Modify the example above so the bars are green if below 100 and red if above 500.

Exercise 18: Extend the example, so the 'bar chart' data is displayed from an external file (e.g., csv).

Exercise 19: Put the code in a 'function' so the bar chart is only displayed when the function is called. Also if the function is called twice, then it will show the bar chart twice on screen. Extend this function so it takes a 'csv' file name as the input argument. Call it twice and it displays two different bar charts using different data on screen.

Part 10. Axis

The axes renders human-readable reference marks for scales. Graphs have two axes: the horizontal axis or the x-axis and the vertical axis or the y-axis.

D3 provides the following functions to draw axes.

Axis Method	Description
• d3.axisTop()	Creates top horizontal axis.
• d3.axisRight()	Creates vertical right-oriented axis.
• d3.axisBottom()	Creates bottom horizontal axis.
• d3.axisLeft()	Creates left vertical axis.

Simple example that draws a simple 'horizontal' axis:

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
const width  = 400;
const height = 100;

const data = [10, 12, 20, 25, 32];

// Append SVG
var svg = d3.select("body")
    .append("svg")
    .attr("width", width)
    .attr("height", height);

// Create scale
var scale = d3.scaleLinear()
      .domain([d3.min(data), d3.max(data)])
      .range([0, width - 100]);

// Add scales to axis
var x_axis = d3.axisBottom()
       .scale(scale);

//Append group and insert axis
svg.append("g")
  .call(x_axis);

</script>
```

Next you'll draw both a horizontal and vertical axis (left and bottom):

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
const width  = 400;
const height = 300;

var data = [10, 15, 20, 25, 30];
var svg = d3.select("body")
   .append("svg")
   .attr("width", width)
```

```
   .attr("height", height);

var xscale = d3.scaleLinear()
   .domain([0, d3.max(data)])
   .range([0, width - 100]);

var yscale = d3.scaleLinear()
       .domain([0, d3.max(data)])
       .range([height/2, 0]);

var x_axis = d3.axisBottom()
       .scale(xscale);

var y_axis = d3.axisLeft()
       .scale(yscale);

   svg.append("g")
      .attr("transform", "translate(50, 10)")
      .call(y_axis);

var xAxisTranslate = height/2 + 10;

   svg.append("g")
         .attr("transform", "translate(50, " + xAxisTranslate  +")")
         .call(x_axis)

</script>
```

Exercise 20: Update the example so an axis is drawn on all sides (axis on the left, right, top and bottom). Make the top and right axis the color blue (text and lines are blue in color).

Exercise 21: Add an 'axis' to the bar chart example (bottom and left axis for the bar chart).

Part 12. Line Chart

The following example brings together the svg line drawing details, scales and the axis information.

This example below, generates 100 points which follow a 'sine wave' signal.

The code may seem long, but it brings together all of the details discussed up to now.

Exercise 22: Modify the code so it's contained within a function (pass the data to the function, so you're able to draw sine wave, cosine, or other type).

Exercise 23: Load in some test data from a csv and plot the line (instead of 'generating' the data you load it from an external file).

Exercise 24: Exercise: Draw multiple lines on the same chart (e.g., sinewave and a cosine wave). Make one blue and the other green.

```
<script src='https://d3js.org/d3.v7.min.js'></script>

<script>
// Set Dimensions
const xSize  = 600;   const ySize  = 600;
const margin = 40;
const xMax   = xSize - margin*2;
const yMax   = ySize - margin*2;

// Create Random Points
const numPoints = 100;
const data      = [];
for (let i = 0; i < numPoints; i++)  {  data.push( {x: i/100, y: Math.sin( 6.2 * i/100 ) } );   }

// Get the 'limits' of the data - the full extent (mins and max)
// so the plotted data fits perfectly
const xExtent = d3.extent( data, d=>{ return d.x } );
const yExtent = d3.extent( data, d=>{ return d.y } );

// Append SVG Object to the Page
const svg = d3.select("body")
          .append("svg")
          .attr('width',  xSize  )
          .attr('height', ySize  )
          .append("g")
          .attr("transform","translate(" + margin + "," + margin + ")");

// X Axis
const x = d3.scaleLinear()
        .domain([ xExtent[0], xExtent[1] ])
        .range([0, xMax]);

// bottom
svg.append("g")
  .attr("transform", "translate(0," + yMax + ")")
  .call(d3.axisBottom(x))
  .attr('color', 'green'); // make bottom axis green

// top
svg.append("g")
  .call(d3.axisTop(x));

// Y Axis
const y = d3.scaleLinear()
        .domain([ yExtent[0], yExtent[1] ])
        .range([ yMax, 0]);

// left y axis
svg.append("g")
```

```
  .call(d3.axisLeft(y));

// right y axis
svg.append("g")
  .attr("transform", `translate(${yMax},0)`)
  .call(d3.axisRight(y));

// Add the line
svg.append("path")
  .datum(data)
  .attr("fill", "none")
  .attr("stroke", "steelblue")
  .attr("stroke-width", 1.5)
  .attr("d", d3.line()
            .x(function(d) { return x(d.x)  })
            .y(function(d) { return y(d.y)  })
  );
</script>
```

Part 13. Markers

Add extra visuals to the graphs to show data points.

```
svg.selectAll("dot")
  .data(data)
  .enter()
  .append("circle")
  .attr("cx", function (d) { return x(d.x) } )
  .attr("cy", function (d) { return y(d.y) } )
  .attr("r", 5)
  .style("fill", "red");
```

Exercise 25: Add a ‘circle point’ to the line graph, so that each data point is displayed on the graph as circle.

Exercise 26: Plot two lines on the same graph, and mark the data points as circles for the first line and triangles for the second line.

Exercise 27: Add ‘text’ to certain points on the line plot (e.g., next to the ‘circle’ dot also write the number of the data value). Limit the text information to only a few points so it doesn’t get over crowded.

Part 14. Colors

Color is an underappreciated component that is vital to any visualization.

You can represent color in different formats, such as::

- string (blue)
- html/hex (#3323ba)
- rgb (rgb(20,2,200)
- rgba (rgba(2,10,200,0.1) opacity

Taking advantage of the d3 color methods

- Sequential color scale
- Categorical color scale

Examples

```
var myColor1 = d3.scaleLinear().domain([1,10]).range(["white", "blue"]);

var myColor2 = d3.scaleSequential().domain([1,10]).interpolator(d3.interpolatePuRd);

var myColor3 = d3.scaleSequential().domain([1,10]).interpolator(d3.interpolateViridis);

var myColor4 = d3.scaleOrdinal().domain(data).range(["gold", "blue", "green", "yellow"]);

var myColor5 = d3.scaleOrdinal().domain(data).range(d3.schemeSet3);
```

You can test out the colors generated, for example, the 'scaleLinear', setting the range from 'white' to 'blue'. Takes a number and returns a color value, as shown below:

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<script>
var myColor0 = d3.scaleLinear().domain([1,10]).range(["white", "blue"]);

console.log( 'col 0:', myColor0(0) );
console.log( 'col 2:', myColor0(2) );
console.log( 'col 9:', myColor0(9) );
</script>
```

The following code example, dynamically adds 'divs' to the page – setting each 'div' background style color. You should be able to see the range of colours.

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<style>
div {
  display: inline-block;
  width  : 10px;
  height : 10px;
  margin:  1px;
  padding: 10px;
}
</style>

<script>
var body = d3.select('body');

var myColor0 = d3.scaleLinear().domain([1,10]).range(["white", "blue"]);
var myColor1 = d3.scaleSequential().domain([1,10]).interpolator(d3.interpolatePuRd);
var myColor2 = d3.scaleSequential().domain([1,10]).interpolator(d3.interpolateViridis);

/*
d3.range(5)
```

```
[0, 1, 2, 3, 4]
*/
var data = d3.range(5);

var myColor3 = d3.scaleOrdinal().domain(data).range(["gold", "blue", "green", "yellow"]);
var myColor4 = d3.scaleOrdinal().domain(data).range(d3.schemeSet3);

var cols = [myColor0,
        myColor1,
        myColor2,
        myColor3,
        myColor4 ];

for (let k=0; k<cols.length; k++)
{
  for (let i=0; i<20; i++)
  {
    body.append('div').style( 'background-color', cols[k](i) );
  }
  body.append('br');
}

console.log('ready..');
</script>
```

Experiment with the colors so that you have an idea of their capabilities (e.g., different color schemes and how they map values to colors).

Exercise 28: Take one of the d3 color methods and apply it to the bar chart example (from earlier). So the color is related to the value (not just fixed).

Exercise 29: Take one of the d3 color methods and apply it to the line chart example (from earlier).

Part 15. Pie Chart

The d3.pie() function takes in a dataset and creates handy data for you to generate a pie chart in the SVG. It calculates the start angle and end angle for each wedge of the pie chart. These start and end angles can then be used to create actual paths for the wedges in the SVG.

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<script>
   var data = [2, 4, 8, 10];
   var pie = d3.pie()
   console.log(pie(data))
</script>
```

In the above example, you defined a d3.pie() object and provided your data to the pie function. This function calculated certain fields for the pie chart like `startAngle' and `endAngle` along with the data values.

You usually use the d3.pie() with the d3.arc() method. The d3.arc() generates an arc. These are the paths that will create your pie's wedges. Arcs need an inner radius and outer radius. If the inner radius is 0, the result will be a piechart, otherwise the result will be a donut chart. You need to supply these generated arcs to your SVG path elements.

```
<script type='text/javascript' src='https://d3js.org/d3.v7.min.js'></script>

<script>
var data = [3, 4, 8, 12];

const xSize  = 400;   const ySize  = 400;
const margin = 40;
const xMax   = xSize - margin*2;
const yMax   = ySize - margin*2;

// Append SVG Object to the Page
const svg = d3.select("body")
          .append("svg")
          .attr('width',  xSize  )
          .attr('height', ySize  )
          .append("g")
          .attr("transform","translate(" + xSize/2 + "," + ySize/2 + ")");

const radius = Math.min(xSize, ySize) / 2;

var color = d3.scaleOrdinal(['#4daf4a','#377eb8','#ff7f00','#984ea3','#e41a1c']);

// Generate the pie
var pie = d3.pie();

// Generate the arcs
var arc = d3.arc()
    .innerRadius(0)
    .outerRadius(radius);

//Generate groups
var arcs = svg.selectAll("arc")
    .data(pie(data))
    .enter()
```

```
    .append("g")
    .attr("class", "arc")

//Draw arc paths
arcs.append("path")
  .attr("fill", function(d, i) {
    return color(i);
  })
  .attr("d", arc);
</script>
```

Step by step overview of the example above:

1. var pie = d3.pie(); This will generate your pie values like startAngle and endAngle as seen in the previous example.

2. Next, you define your arc with an inner radius of 0 and outer radius as the radius calculated earlier. This will be used to give paths to your pie wedges.

3. Then, you create group elements for each of your data values. The group element will hold your individual paths or wedges.

4. Finally, you add a path element for each of our wedges. You provide the arc generated earlier and fill it with a color from your color scale.

Exercise 30: Add more data values (e.g., 12 different numbers)

Exercise 31: You've already learned about svg text items (previous sections). Add a text item to each 'arc' (e.g., draw the values for the data on the pie chart).

Part 16. SVG Graphics

You can insert graphics (images) within the SVG output to add extra detail
(for the exercise go online and find free open source svg/png image which you can use for testing)

For example:

```
d3.select("p")
  .append("svg:image")
  .attr("xlink:href", "img/icons/sun.svg")
  .attr("width", 40)
  .attr("height", 40)
  .attr("x", 228)
  .attr("y",53);
```

Exercise 32: Add a graphical image (e.g., png/jpg) to the background of one of the graphs (e.g., bar chart or line plot). Scale the image to fit the size of the svg bounds (covers background).