D3.js

D3 (D3.js) は JavaScript ライブラリのひとつ。無料で利用できる。D3 は Data-Driven Documents の略。データの見える化（可視化）を、ウェブ上の標準機能を利用する形で実現している。開発者によれば、このライブラリは明瞭かつ効果的に情報とコミュニケーションできるようにするのが目的であり、データ可視化はその手段にすぎないとのこと。データ可視化のためのシステムは他にもあるが、D3.js の特徴はインタラクション。マウス、キーボードなどの操作を通じて、双方向にデータをやり取りすることができる。

D3 の初版が公開されたのは 2011 年。2016 年に大規模改版が行われ、バージョン 4 となっている。機能は大幅に改善されたが、バージョン 3 との互換性が無いので、解説記事などを調べる場合は注意が必要。非互換部分は多くはないが、古いサンプルプログラムは動作しないものが多い。たとえば、バージョン 4 と 3 のライブラリを両方用意しておき、サンプルプログラムがバージョン 4 環境で動作しない場合、バージョン 3 環境下で動作することを確認してから、非互換部分を修正するなどの工夫が考えられる。

公式サイトは http://d3js.org/、日本語版サイトは http://ja.d3js.node.ws/。

公式サイトから、zip 圧縮ファイルをダウンロードし、d3.min.js ファイル、または d3.js を取り出して利用する。日本語版サイトで配布しているファイルは古いバージョンの可能性がある。通常利用するのは、圧縮されていて動作効率の良い d3.min.js ファイル。d3.js は動作は同じだが、ソースを読んで研究する場合に適している。

お勧めの参考書。ただし、d3.js version 3 ベース

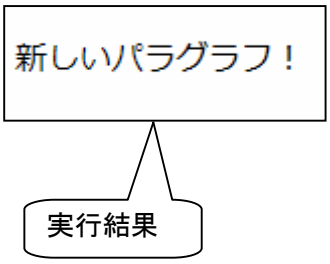
入門：「メソッドのチェイン」について

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <script src="d3.min.js"></script>
    </head>
    <body>
        <div id=graph></div>
        <script>
            d3.select("#graph").append("p").text("新しいパラグラフ！");
        </script>
    </body>
</html>
```

d3.js（ライブライ）の読み込み

表示領域の確保

d3.js を利用するプログラム



D3 は、チェイン構文と呼ばれる手法を取り入れている。メソッドをピリオドで「チェイン」する（連鎖させる）ことにより、複数の作業を一行のコードで実行できるようになっている。この手法は jQuery ライブラリでも使われている。

```
d3.select("#graph").append("p").text("新しいパラグラフ！");
```

これは、下記のように書いたのと同じ動作をする。

```
var body = d3.select("#graph");
var p = body.append("p");
p.text("新しいパラグラフ！");
```

< d3 >

D3 オブジェクトへの参照。D3 のメソッドにアクセスするためのもの。

<.select("#graph") >

select() の引数として CSS セレクタを与えると、DOM 中でセレクタにマッチする最初の要素の参照を返す。#graph は id なので、もともとひとつの要素しかないが、クラスやタグ名を与えた場合、該当する要素群の最初のものが返される。複数の要素を取り出したい場合は selectAll() を使う。ここでは、id=graph の div 要素への参照をチェインの次のメソッドに引き渡す。選択した要素のことを「セレクション」という。

<.append("p")>

append() は引数として指定した要素を新規に生成し、受け渡されたセレクションの内部の末尾に追加する。ここでは id=graph の div 要素 の内部に新しく p を生成している。append() は、生成したばかりの新しい要素への参照をチェインをセレクションとして、次のメソッドに引き渡す。

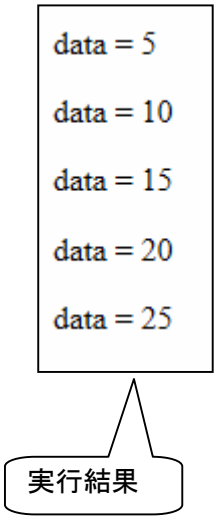
<.text("新しいパラグラフ！")>

text() は、引数で指定した文字列を、現在のセレクションの開始タグと終了タグの間に挿入する。ここでは前のメソッドから新しい p 要素への参照を受け取っているので、このコードは「新しいパラグラフ！」というテキストを と の間に挿入する。もしそこに既存のテキストがあった場合はそれを上書きする。

すべてではないが、多くの D3 メソッドは、操作したセレクションへの参照を返す。メソッドをチェインするときはその順序が重要。メソッドの出力型は、チェイン上の次のメソッドが想定する入力型と一致している必要がある。

入門の続き：「データのバインディング」について

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <script src="d3.min.js"></script>
    </head>
    <body>
        <div id=graph></div>
        <script>
            var dataset = [ 5, 10, 15, 20, 25 ];
            d3.select("#graph").selectAll("p")
                                    .data(dataset)
                                    .enter()
                                    .append("p")
                                    .text(function(d) { return "data = "+d; });
        </script>
    </body>
</html>
```



D3 は様々な種類のデータを扱える。数値配列、文字配列、オブジェクト配列等。JSON、GeoJSON、CSV などの記述形式のファイルを読み込む組込メソッドも備えている。

データ視覚化は、データをビジュアルにマッピングする（対応付ける）こと。データが入力で、ビジュアルのプロパティ（属性）が出力となる。対応付けの例は、数字が大きくなるほど棒を高くしたり、データの大きさが特定の値の範囲にある場合に色を変えること。このようにデータの値と、DOM 中の要素に結び付けることを「バインドする」という。データを DOM にバインドするためには、 D3 の selection.data() メソッドを用いる。

上記のプログラムは、「data = 5」から「data = 25」までの 5 つの新しいパラグラフが表示される。

「.selectAll("p")」メソッドが実行される時点では、セレクト対象のパラグラフ要素 p は、まだ存在していない。まだ存在していない要素がセレクトできるというミステリーを解く鍵は enter() にある。

<d3.select("#graph")>
DOM の中から id=graph の要素 を見つけ、その参照をチェーンの次のステップに渡す。

<.selectAll("p")>
チェーンで渡された DOM 要素内のすべてのパラグラフ要素 (p) を選択する。ここでは該当する要素が存在しないため、メソッドは空のセレクションを返す。この空のセレクションは、すぐ後に作られるパラグラフ要素を表したもの。

<.data(dataset)>
データの値の個数を数え、解析する。このデータセットには 5 つの値が含まれているので、後続するすべてのメソッドは、それぞれの値ごと、計 5 回処理を繰り返される。データを 5 回渡すというより、要素が 5 個の配列を渡すというイメージ。チェーンで渡されたセレクションへの参照も渡す。

<.enter()>

このメソッドは、渡された DOM 要素（セレクション）を調べてから、次に、受け渡されたデータを調べる。対応する DOM 要素があれば、それへの参照をチェーンの次のステップに渡す。もし該当する DOM 要素の数よりデータの値の個数のほうが多い場合は、新規に空の DOM 要素を追加生成し、その参照を渡す。DOM 要素への参照をチェーンの次のステップ（下流）に渡す際、チェーンの前ステップから受け取ったデータも対にして下流に渡す。

<.append("p")>

セレクション（参照の集まり）を受け取り、DOM に p 要素を挿入する。ここで、空の DOM 要素があった場合、p 要素に変えられる。そして生成した要素の参照をチェーンの次のステップに渡す。チェーンの上流からで受け取ったデータも、対にして下流に渡す。

<.text(function(d) { return "data = "+d; })>

p 要素への参照とデータを受け取り、”deta = “ という文字列にデータを文字列化したものをつないでテキスト生成し、p 要素に代入する。

プログラムを下記のように改良すると、データが 15 を超えた場合に表示を赤く変えることができる。

```
<body>
    <div id=graph></div>
    <script>
        var dataset = [ 5, 10, 15, 20, 25 ];
        d3.select("body").selectAll("p")
                                .data(dataset)
                                .enter()
                                .append("p")
                                .text(function(d) { return "data = "+d; })
                                .style("color", function(d) {
                                      if (d > 15) { return "red"; }
                                      else        { return "black"; }
                                 });
    </script>
</body>
```

data = 5

data = 10

data = 15

data = 20

data = 25

応用1： 要素を使った棒グラフ

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <style>
            div.bar {
                display: inline-block;
                width: 20px;
                height: 75px;
                background-color: teal;
            }
        </style>
        <script src="d3.min.js"></script>
    </head>
    <body>
        <div id=graph></div>
        <script>
            var dataset = [ 5, 10, 15, 20, 25 ];
            d3.select("#graph").selectAll("div")
                                .data(dataset)
                                .enter()
                                .append("div")
                                .attr("class", "bar")
                                .style("height", function(d) { return d + "px"; });
        </script>
    </body>
</html>
```

この height 属性は
プログラムで書き換える

実行結果

さらに発展。データの最大値が画面いっぱいになるように調節。棒グラフ内にデータを文字表示。

```
        <div class="graph"></div>
        <script>
var data = [4, 8, 15, 16, 23, 42];
var width = 420;
var x = d3.scaleLinear()
            .domain([0, d3.max(data)])
            .range([0, width]);
d3.select('.graph')
            .selectAll('div')
            .data(data)
            .enter().append('div')
            .style('width', function(d) {
                        return x(d) + 'px';
            })
            .text(function(d) { return d;  });
        </script>
```

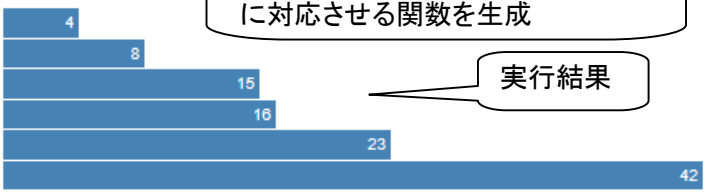
< d3 version 3 >
d3.scale.linear()

d3 version 3
の場合

d3 scaleLog() もある

表示が画面いっぱいになるような
伸縮関数 x(d) を作る

0 ～ 42(max)を 0 ～ 420(width)
に対応させる関数を生成



実行結果

データを文字表示

応用2：svg を使った棒グラフ

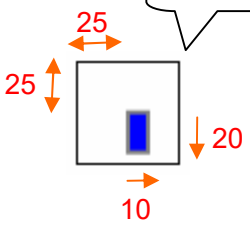
① svg は、ブラウザにもともとある、図形表示機能。まず、d3.js とは関係なく使ってみる。

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <svg width="50" height="50">
        <circle cx="25" cy="25" r="22"
                fill="blue" stroke="gray" stroke-width="2"/>
    </svg>
  </body>
</html>
```

実行結果

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>svg</h1>
    <svg width="50" height="50" style="border:solid 1px">
        <rect x="25" y="25" width="10" height="20"
                fill="blue" stroke="gray" stroke-width="2"/>
    </svg>
  </body>
</html>
```

実行結果



② d3.js から svg を出力してみる

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <script src="d3.min.js"></script>
    </head>
    <body>
        <div id=graph></div>
        <script>
     var canvas = d3.selectAll("#graph");
     var svg = canvas.append("svg").attr("width",200).attr("height",200);
     var circle = svg.append("circle")
           .attr("cx",100)
           .attr("cy",100)
           .attr("r",80)
           .attr("fill","red")
           .attr("stroke","orange")
           .attr("stroke-width",10);
        </script>
    </body>
</html>
```

実行結果

③ svg で棒グラフを描いてみる

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <script src="d3.min.js"></script>
    </head>
    <body>
        <div id=graph1></div>
        <script>
     var darr = [ 10,20,15,25,28,19,20,15 ];
     var scaleX = d3.scaleLinear()
            .domain([0, darr.length])
            .range([0, 200]);
     var scaleY = d3.scaleLinear()
            .domain([0, d3.max(darr, function(d){return d;})])
            .range([0,180]);
     var canvas = d3.selectAll("#graph1");
     var svg = canvas.append("svg")
            .attr("width",200)
            .attr("height",200);
      svg.selectAll("rect")
            .data(darr)
            .enter()
            .append("rect")
            .attr("x", function(d, i){return scaleX(i) + 25/darr.length})
            .attr("y", function(d, i){return 190 - scaleY(d)})
            .attr("width", function(d, i){return 150/darr.length})
            .attr("height", function(d, i){return scaleY(d)})
            .attr("fill", function(d, i){return "blue"});
        </script>
    </body>
</html>
```

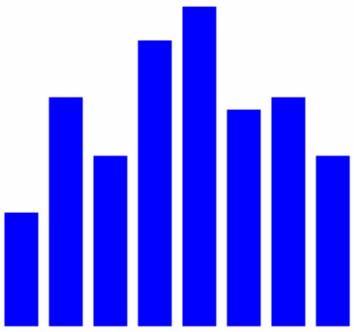
d3 version 3 の場合

< d3 version 3 >
d3.scale.linear()

横軸の伸縮関数

縦軸の伸縮関数

実行結果



応用3：csv 形式のファイルから入力

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="d3.min.js"></script>
  </head>
  <body>
    <div id="result"></div>
    <script>
d3.csv("./data.csv", function(error, list){
    d3.select("#result")
        .append("table")
        .selectAll("tr")
        .data(list)
        .enter()
        .append("tr")
        .append("td")
        .text(function(d){
            return d["value"];
        })
});
    </script>
  </body>
</html>
```

```
value
200
100
150
200
160
180
160
```

こういう内容の
ファイル
data.csv
を用意する

実行結果

200
100
150
200
160
180
160

div タグ内に出力されているのは、html で、

```
<table>
<tr><td>200</td></tr>
<tr><td>100</td></tr>
<tr><td>150</td></tr>
<tr><td>200</td></tr>
<tr><td>160</td></tr>
<tr><td>180</td></tr>
<tr><td>160</td></tr>
</table>
```

応用4：csv 形式ファイルから入力して、svg で棒グラフ

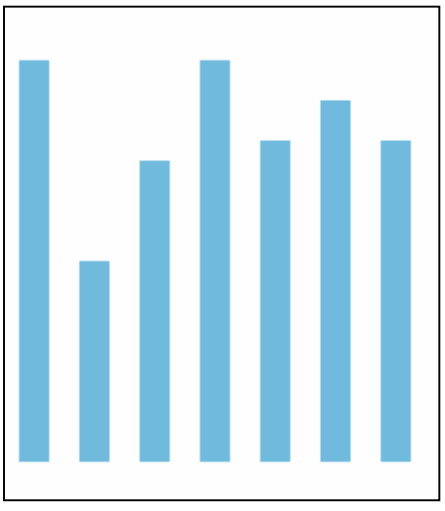
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="d3.min.js"></script>
  </head>
  <body>
    <svg width="500" height="300" id=sv></svg>
d3.csv('data.csv', function(csvdata) {
    var dataset = [];
    for (var i = 0; i < csvdata.length; i++) {
        dataset.push(csvdata[i]['value']);
    };
    make(dataset);
});
function make(dataset) {
    d3.select('svg').selectAll('rect')
    .data(dataset)
    .enter()
    .append('rect')
    .attr("x",function(d, i){ return i * 30; })
    .attr("y",function(d){ return 300 - d; })
    .attr("width",15)
    .attr("height",function(d){ return d; })
    .attr("fill",'#6fbadd');
}
    </script>
  </body>
</html>
```

```
value
200
100
150
200
160
180
160
```

こういう内容の
ファイル
data.csv
を用意する

d3 version 3
の場合

```
< d3 version 3 >
.attr({
    x : function(d, i){ return i * 30; },
    y : function(d){ return 300 - d; },
    width : 15,
    height : function(d){ return d; },
    fill : '#6fbadd'
})
```



実行結果

応用5：json 形式ファイルの読み込み

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="d3/d3.min.js"></script>
  </head>
  <body>
    <div id="data"></div>
  </body>
    <script>
d3.json('data.json', function(error, root) {
    d3.select('#data')
        .append('div')
        .selectAll()
        .data(root)
        .enter()
        .append('div')
        .text(function (d) {
            return d['kanji']
            + ' :' + d['kana']
            + ' :' + d['english'];
        });
});
   </script>
</html>
```

Data.json ファイルの内容

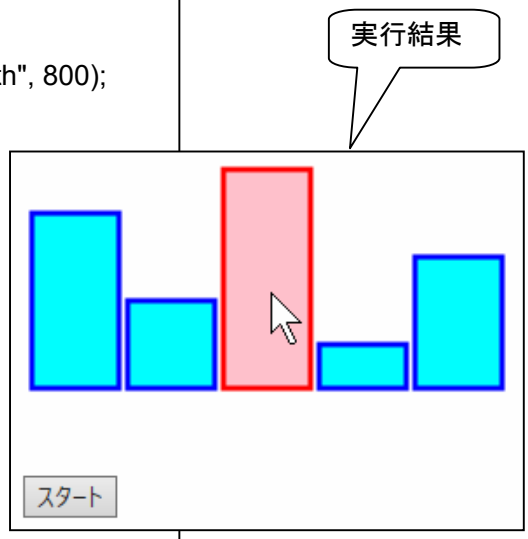
```
[
    {
        "kanji": "犬",
        "kana": "いぬ",
        "english": "dog"
    },
    {
        "kanji": "猫",
        "kana": "ねこ",
        "english": "cat"
    },
    {
        "kanji": "鳥",
        "kana": "とり",
        "english": "bird"
    }
]
```

実行結果

犬：いぬ：dog
猫：ねこ：cat
鳥：とり：bird

応用6：イベントとアニメーション

```
<script src="d3.min.js"></script>
<svg id="graph"></svg><br>
<button id="aButton">スタート</button>
<script type="text/javascript">
    var dataset = [ 20, 10, 25, 5, 15 ];
    var width = 50;
    var height = 150;

    var svg = d3.select("#graph").attr("height", 200).attr("width", 800);

    svg.selectAll("rect")
    .data(dataset)
    .enter()
    .append("rect")
    .attr("width", width)
    .attr("height", function(d) { return 0 })
    .attr("x", function(d, i) { return i*(width+5)+5 })
    .attr("y", function(d) { return height })
    .attr("fill", "cyan")
    .attr("stroke", "blue")
    .attr("stroke-width", "3")
    .on("mouseover", function() { d3.select(this)
                .attr("fill", "pink")
                .attr("stroke", "red");
    })
    .on("mouseout", function(d) {
            d3.select(this)
            .attr("fill", "cyan")
            .attr("stroke", "blue")
    });

    d3.select("#aButton").on("click", function() {
            svg.selectAll("rect")
           .data(dataset)
           .attr("height", function(d) { return 0 })
           .attr("y", function(d) { return height })

            svg.selectAll("rect")
           .data(dataset)
           .transition().delay(0).duration(2000)
           .attr("y", function(d) { return height-d*5 })
           .attr("height", function(d) { return d*5 })
    });
</script>
```



イベント

イベント

アニメーション

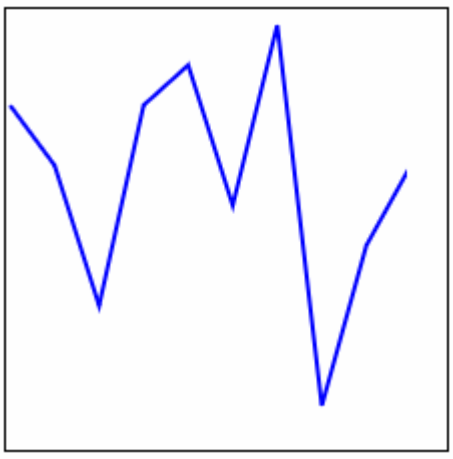
応用7：折れ線グラフ

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <style>
          #graph { width: 220px; height: 220px; border: solid 1px }
      </style>
   </head>
   <body>
      <script src="d3/d3.min.js"></script>
      <div id=graph></div>
      <script>
var dataset =[ 15,12,5,15,17,10,19,0,8,12 ];

var    pathinfo = [];
var b_x = 200 / (dataset.length - 1);
for (var i=0; i<dataset.length; i++) {
   pathinfo.push({x:b_x*i, y:(200 - dataset[i]*10) });
}

var d3line = d3.line()
   .x(function(d){return d.x;})
   .y(function(d){return d.y;})
   .curve(d3.curveLinear);

var svg = d3.select("#graph")
   .append("svg")
   .attr("width", 200)
   .attr("height", 200)
   .append("path")
   .attr("d", d3line(pathinfo))
   .style("stroke-width", 2)
   .style("stroke", "blue")
   .style("fill", "none");
      </script>
   </body>
</html>
```

< version 3 >
d3.svg.line()

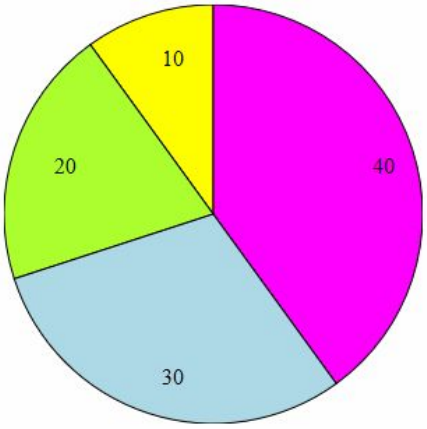
シャープな折れ線

< version 3 >
.interpolate("linear")



応用8：円グラフ

円グラフは英語で pie chart という。




```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <style>svg { width:300px; height:300px }</style>
        <script src="d3/d3.min.js"></script>
    </head>
    <body>
        <svg id="chart"></svg>
        <script>
var width = 300;
var height = 300;
var data = [ 10, 40, 20, 30 ];

var pie = d3.pie().value(function(d){ return d; });

var g = d3.select("#chart")
      .selectAll("g.arc")
      .data(pie(data))
      .enter()
      .append("g")
      .attr("class", "arc")
      .attr("transform", "translate(" + (width / 2) + "," + (height / 2) + ")");

var arc = d3.arc().innerRadius(0).outerRadius(width / 2);
var color = d3.scaleOrdinal()
      .range(["yellow", "magenta","greenyellow", "lightblue"]);

g.append("path")
      .attr("d", arc)
      .style("fill", function(d) { return color(d.data); })
      .attr("stroke", "black");

var labelArc = d3.arc()
      .outerRadius(width/2 - 30)
      .innerRadius(width/2 - 30);

g.append("text")
        .attr("transform", function(d) { return "translate(" + labelArc.centroid(d) + ")"; })
        .attr("dy", "0.5em")
        .text(function(d) { return d.data; });
        </script>
    </body>
</html>
```

<version 3>　<version 4>
d3.layout.pie()　-> d3.pie()
d3.svg.arc()　-> d3.arc()

pie() は、データを降順にならべかえ、 データごとに扇形（パイ形）の開始角、終了角情報を含むオブジェクトを生成するメソッド

g はグループタグ

表示位置を調整する

<version 3>
d3.scale.ordinal()

扇の形を生成

var data = [10, 40, 20, 30];に対する色

扇と扇の間に黒線を入れる

扇の中に書き込むテキストの位置

扇にテキストを挿入