Javascript入門

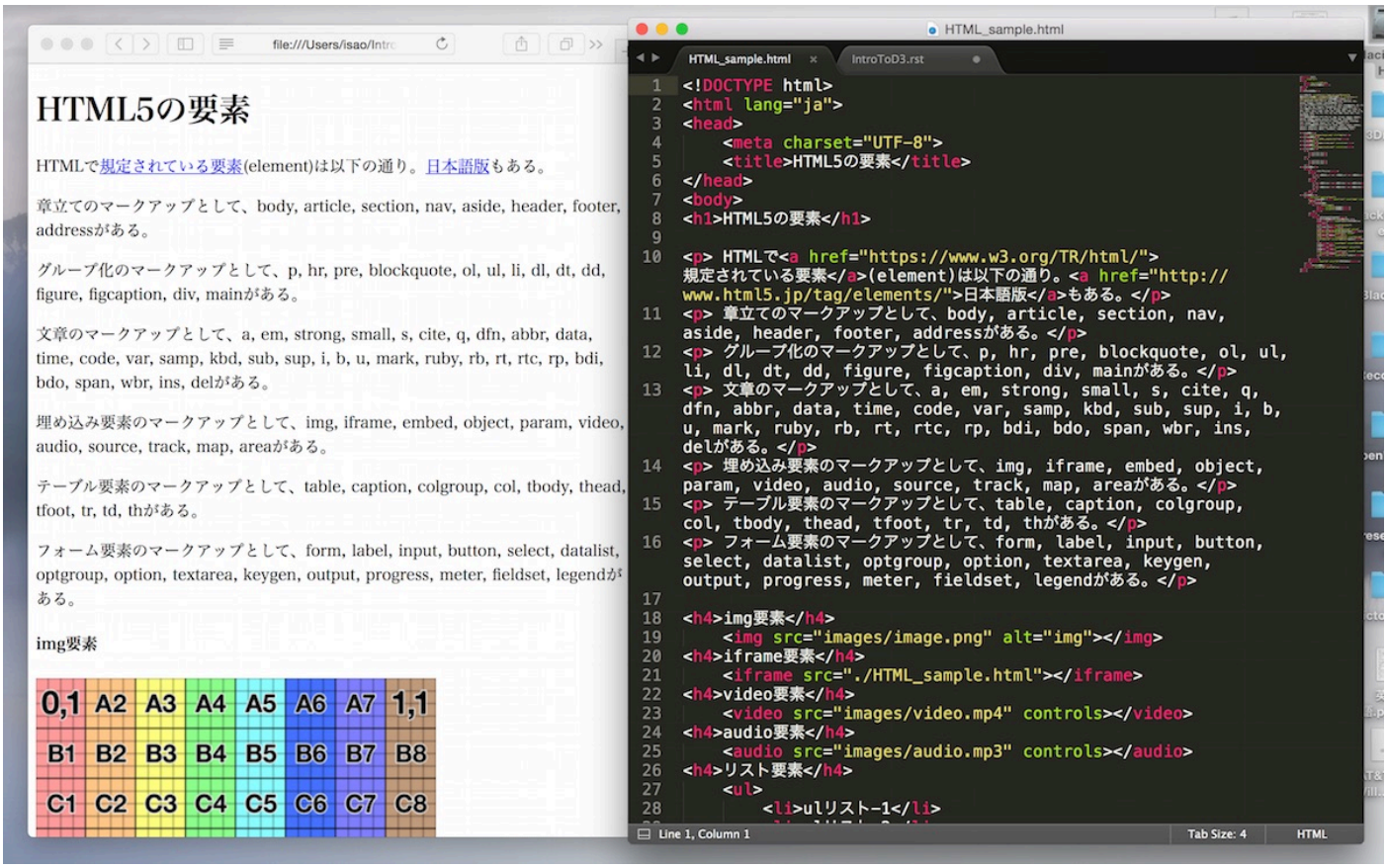
この文書は、業界標準になりつつあるd3.jsでグラフを作るコースです。「HTML/CSS入門」と「Python入門」を読んでいることを前提としています。

最終目標は、勉強会で作ったデータの可視化です。このために半年やってきたのですから。

HTMLとCSSの復習

HTML_sample.htmlをブラウザとエディタで開き、デスクトップに並べましょう。

HTMLで定義されているすべての要素（エレメント）を列挙し、主な要素の具体例が表示されています。どれほど凝ったサイトでも、使っている部品は同じです。



CSS（Cascading Style Sheet）は、Cascade(階段状の滝)のようにスタイル（外観）を指定する仕組みです。次のリストがスタイルの適用順で、上の方が優先されます。

- 作成者スタイルシート：作者の指定
    - 個別要素に対する指定
    - グループに対する指定
- ユーザースタイルシート：読者が指定するスタイル（使ったことがない！）
- デフォルトスタイルシート：ブラウザの設定

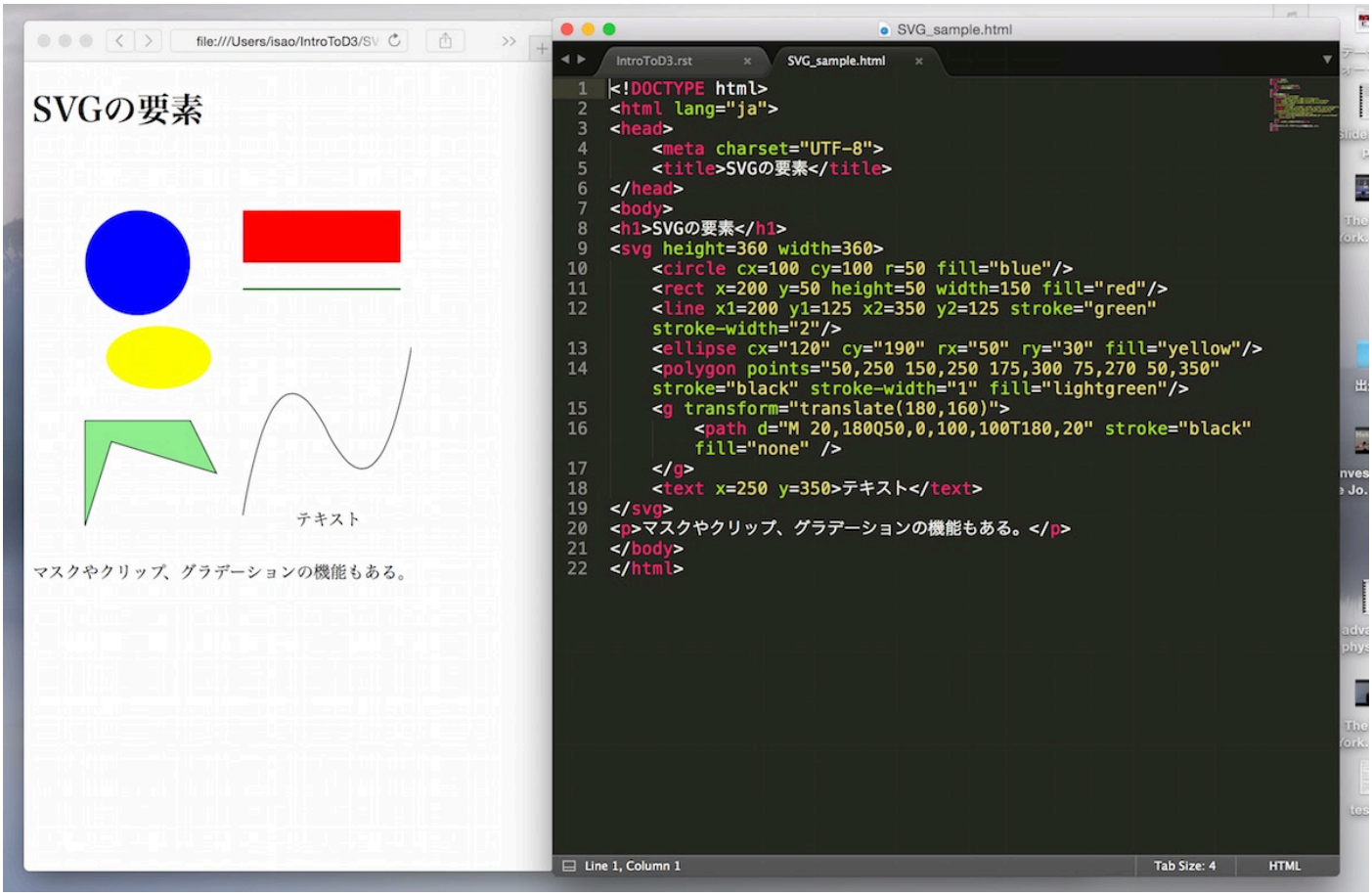
HTML_sample.htmlのソースを編集し、コメントで無効化してある部分を使って、スタイルが優先度に応じて変わっていく様子を確認してください。

SVGとは

HTMLで図形を表す方法はimg要素しかありません。コンピューターから見れば、図形は写真と同じで、静的(static)です。

動的(dynamic)な図形を表示する技術はいろいろ試みられてきました。最近ではFlash Playerが有名です。モダンブラウザ(IE11以降)の時代になってからは **SVG(Scalable Vector Graphic)** の一択です。iPhone/Androidが対応しているからです。

SVGはどんな姿をしているのでしょう？　SVG_sample.htmlをブラウザとエディタで開き、デスクトップに並べましょう。



SVGは、HTMLと同じXMLの一種です。表示されるのはcircle, ellipse, rect, line, polygon, polyline, path, textです。グループ化のためのgもよく使います。図形の位置やサイズ、色などは属性(attribute)で指定されています。

ブラウザは、この「タグの集合体」を解釈し、画像として表示します。

SVGは、デザイン課が使うillustratorで読み取ったり、書き出すことができます。説明はしませんが、グラデーションやマスク、クリッピングもできます。

d3.jsは、この複雑なSVGをjavascriptを使って書くためのライブラリです。

プログラムの３要素

Javascript版のプログラム３要素をざっと紹介します。

1. 変数代入(assignment)

javascriptでは、変数を初めて使う場合にvarをつけます。（ただし、varをつけなくても使えます。詳しくは「スコープ」という用語を調べてください）

```
var greeting = "Hello";  // 文末に;が必要
console.log( greeting ); // 出力にはconsole.logを使います
```

2. 条件文(conditional)

   if構文は、if(論理値){ブロック}です。条件が満たされた場合にだけ{}内のコードが実行されます。読みやすいようにindent(字下げ)されていますが、pythonと違い、必須ではありません。

```
var a = 10;
if( a > 0 ){
        console.log( "aはプラスの数です" );
}
if( a < 0 ){
        console.log( "aはマイナスの数です" );
}
if( a == 0 ){
        console.log( "aはゼロです" );
}
```

3. 繰り返し(iteration)

   for構文はpythonとはかなり違います。配列(array)を使って、順番に処理します。（なお、JSにもmapという関数があります）

```
var members = ["Taro", "Jiro", "Saburo"];
for( var i = 0; i < 3; ++i ){
        console.log( "Hello, " + members[i] );
}
```

Javascriptは教科書、ウェブ上の情報が豊富です。そのため、断片的な情報は見つけやすいですが、一冊は体系的な教科書を読むことをお勧めします。

イベント駆動

Pythonの場合、コマンドラインで「python script.py」と入力し、プログラムを走らせました。ブラウザ上にあるボタンを押すと絵が動くといった、典型的なjavascriptプログラムにはそれができません。

Pythonのスクリプトのように、命令を連続的に記述する「プログラムの話法」を命令型プログラミングといいます。ボタンで動くグラフィックを命令型プログラムで作るには、「マウスの位置がボタンの上かどうかを調べ、もしクリックされているなら、絵を動かす関数を呼ぶ」という命令を毎秒数十回の速度で反復し続けるプログラムを書くことになります。気が狂いそうです。

ブラウザで動くjavascriptでは、イベント駆動型という「プログラムの話法」が使われます。「ページが読み込まれる」「ボタンが押される」「図がクリックされる」「スクロールされる」などのイベント(Event)が起きた時にこの処理を行いなさい、という指令を事前登録しておくのです。

> **命令型プログラミングを能動態とすれば、イベント駆動型は受動態** です。ボタンが押されたかどうかの判定など、面倒な処理はブラウザ側が自動でやってくれます。おかげで、javascriptのプログラムは（pythonで同じことを実現するよりは）ずっと簡単です。

ただし、面倒なことをブラウザに任せた代償はあります。ブザウザが想定していないイベントは全く分かりません。例えば、「スクロールした結果、写真がスクリーンで見えるようになる」というイベントは用意されていません。

誤解しないでほしいことですが、javascriptだからイベント駆動型、pythonだから命令型ということではありません。ターミナルから実行するようなプログラムは命令型が多く、マウスで制御するようなプログラムはイベント駆動型が多い、という程度です。

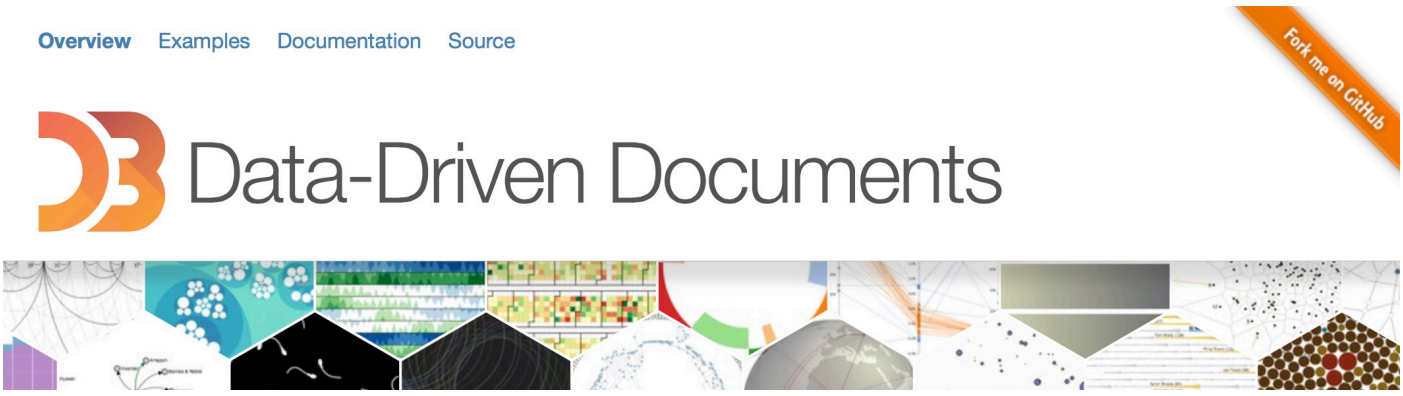
イベント駆動の例をsample01.htmlで体験してみてください。見出しをクリックすると色が変わる仕組みが、「見出しに対して『クリックされるイベントが起きた時に実行するコード』を登録する」という形式で書かれています。

こんな初歩、バカバカしいと思う人は、クリックごとに黒と赤で交互に色が変わるように修正してみてください。

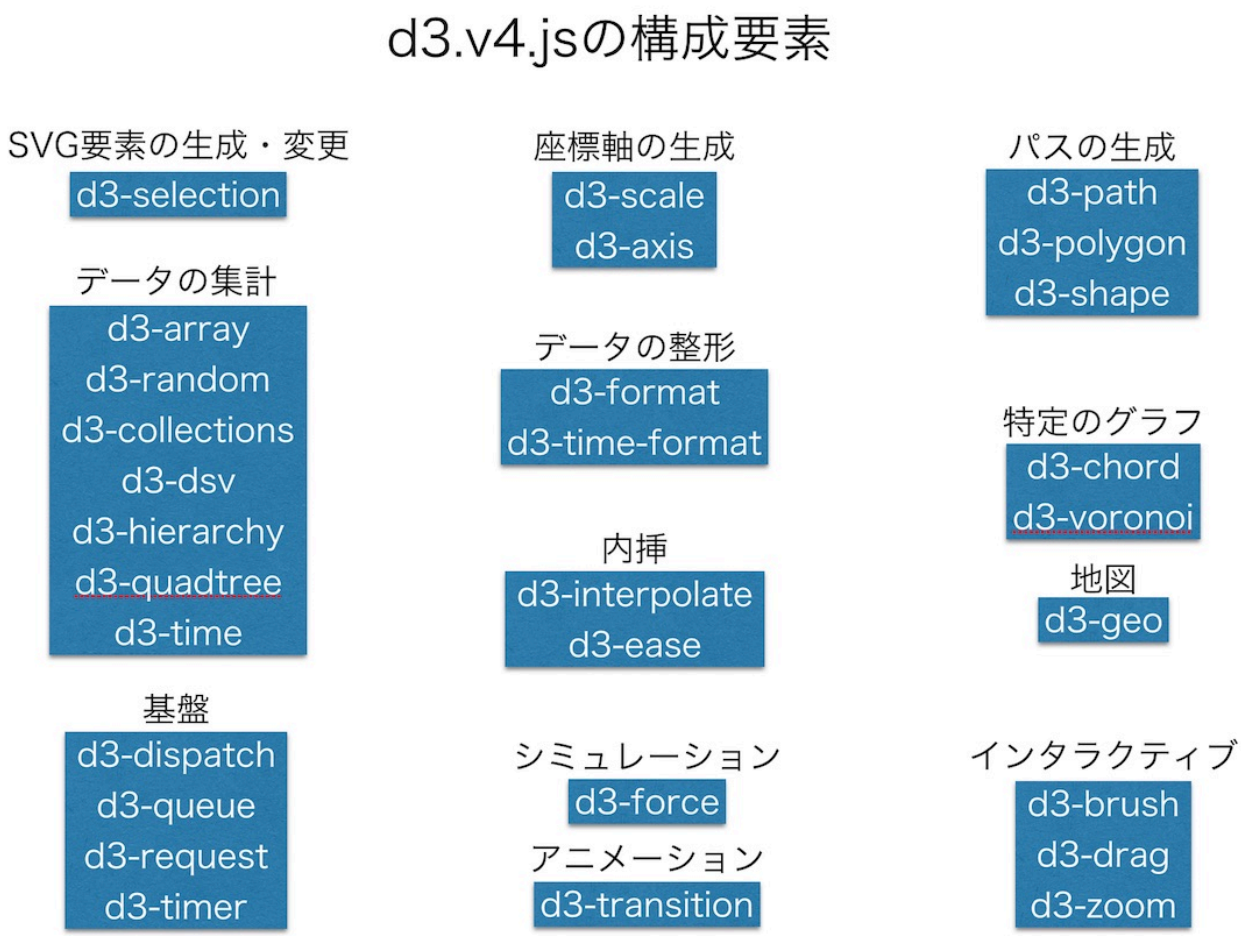
なお、ブラウザが用意しているイベントは、規約文書に定義してあります。

d3.jsとは

d3.jsは、米スタンフォード大の研究者Mike Bostockが作った、SVGのタグを生成するためのjavascriptライブラリーです。いまや、d3を使えないデザイナーはメディアの求人に応募もできないほどの「業界標準技術」になっています。(Mike BostockはNYTimesに請われてサンフランシスコ支局のグラフィックエディターを３年近く勤めました)



d3は、およそグラフを描くために必須のプログラミング部品を統合した「知恵と経験の結晶」です。部品の種類は以下のような感じで、 **密接に相互依存** しています。



作者の方針で、各部品の使い方はコードと例で説明されます。d3のサイトには、本人とユーザーによる膨大な作例が集められています。グラフを見て可視化のアイデアを得ることができるだけでなく、サンプルを読むことで手練のプログラマーの美しいコードを勉強することができます。

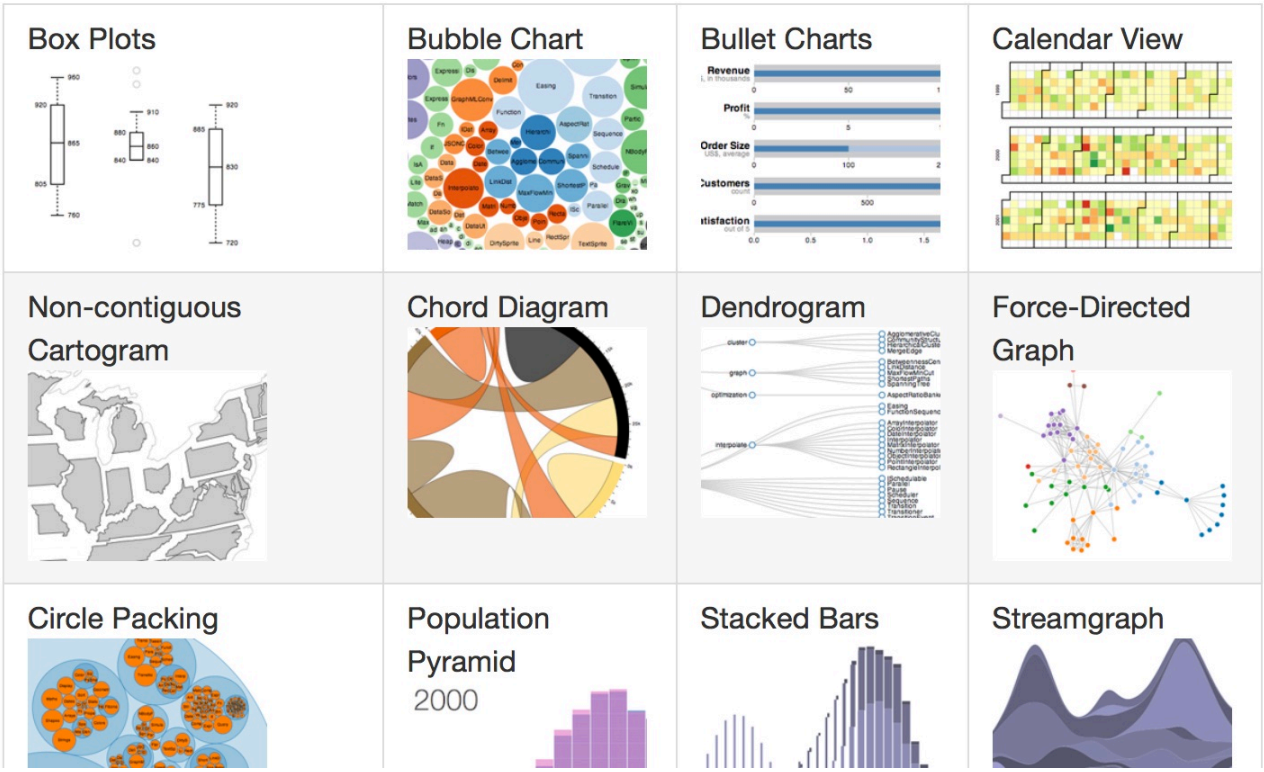
Gallery

Petr Devaikin edited this page 7 days ago · 1148 revisions

Wiki ▸ Gallery

Welcome to the **D3 gallery**! More examples are available on bl.ocks.org/mbostock. If you want to share an example and don't have your own hosting, consider using Gist and bl.ocks.org. If you want to share or view live examples try runnable.com or vida.io.

Visual Index



なお、d3は2016年6月にバージョン４に移行した結果、ネット上の情報は、**バージョン３とバージョン４が混在**するようになってしまいました。しばらく注意が必要です。

d3.jsの基本形

sample02.htmlをブラウザとエディタで開き、デスクトップに並べましょう。SVGを埋め込み、真ん中に円を描くだけです。

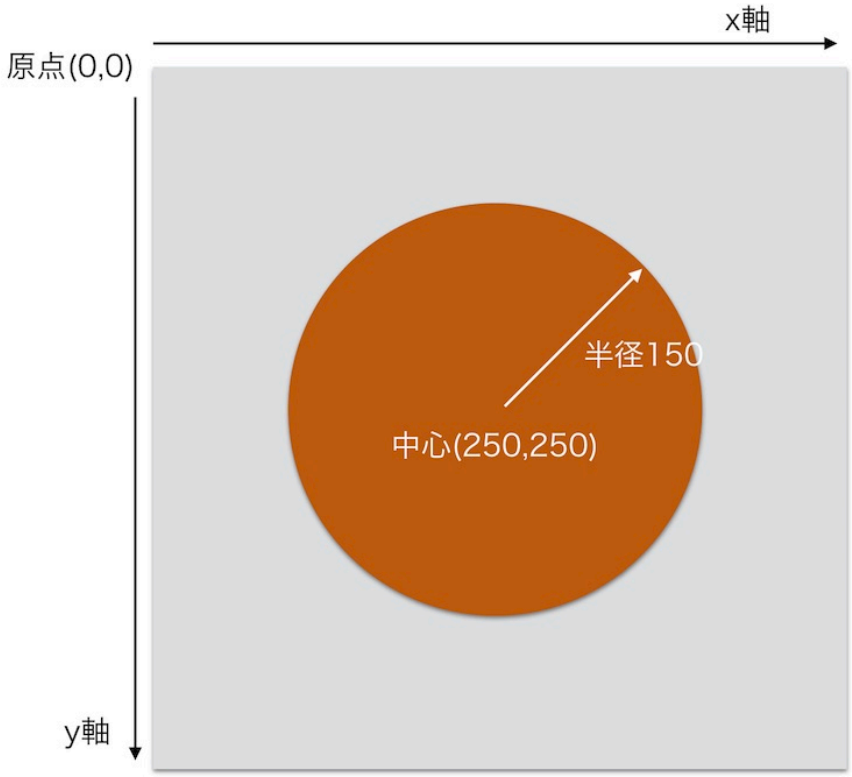
```
// イベント登録:中身が読み込まれた(load)ら実行される関数
window.onload = function(){
        // d3.selectでHTML要素を"掴む"
        var chart = d3.select("#chart");
        // HTML要素にSVG要素を追加(append)する
        var svg = chart.append("svg");
                // 幅と高さの属性を設定する
                svg.attr("width", 500);
                svg.attr("height", 500);
        // SVG要素にCIRCLE要素を追加(append)する
```

```
        var circle = svg.append("circle");
                // 位置とサイズ、色の属性を設定する
                circle.attr("cx", 250);
                circle.attr("cy", 250);
                circle.attr("r", 200);
                circle.attr("fill", "red");
};
```

SVGの各要素で設定しなければならない属性は、SVGの規約で決まっています。circleの場合、中心をcx, cy、半径をrとして指定しなければなりません。

SVG基本図形：[https://triple-underscore.github.io/SVG11/shapes.html]

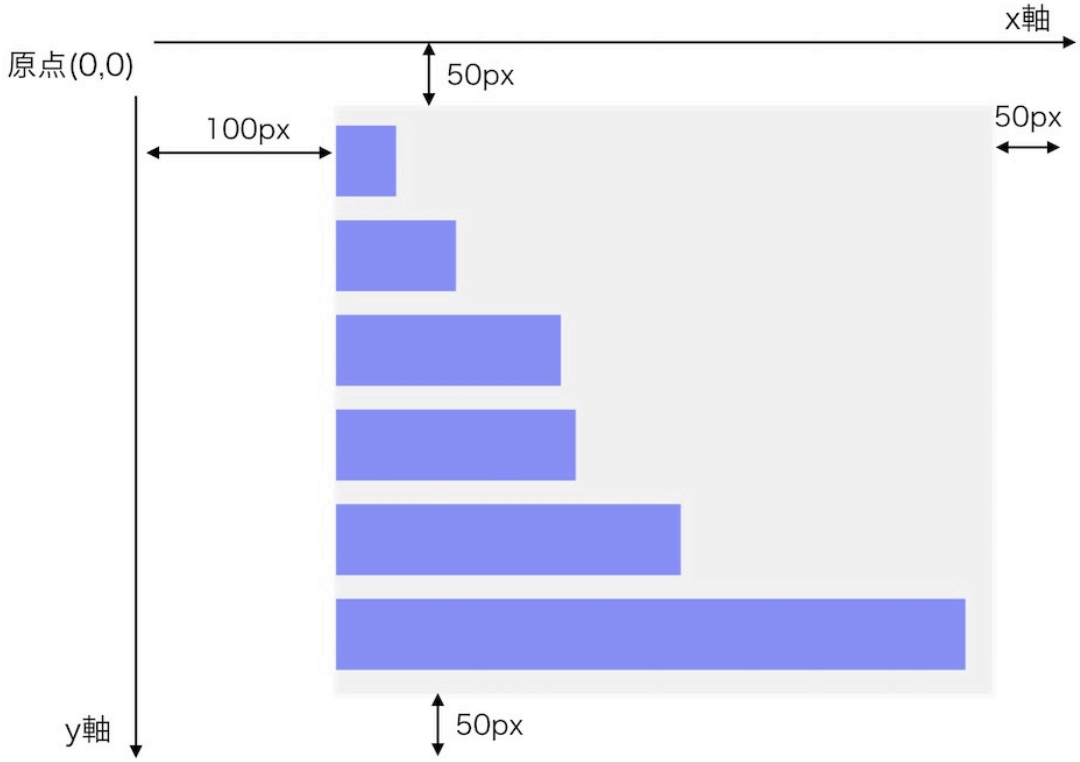
SVGのデフォルト座標系は右上が[0,0]で、Y軸は下向き、単位はピクセルです。



グラフに必要なすべてのSVG要素を、プログラムによって追加し、属性をうまく設定して、自分が作りたい図を生成するのが、d3の作法です。

棒グラフをつくる

sample03.htmlをブラウザとエディタで開き、デスクトップに並べましょう。 最小構成の棒グラフです。

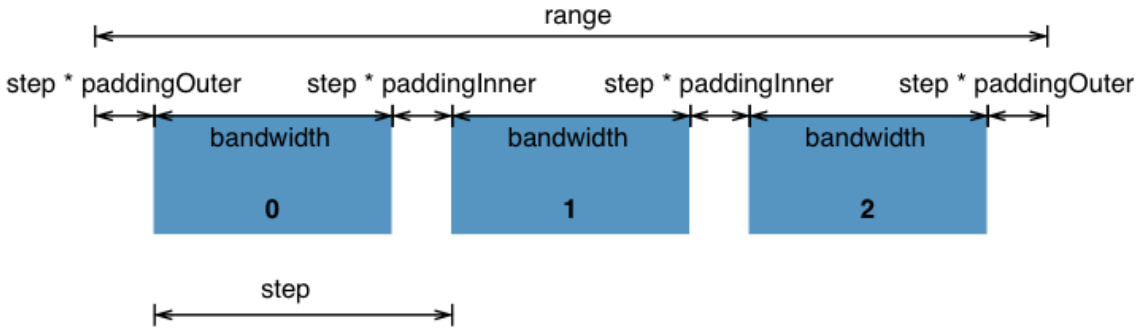



```
var xScale = d3.scaleLinear() // x軸用scaleを作る
        .domain([0, 50])          // 入力範囲
        .range([100, 450]);       // 出力範囲

var yScale = d3.scaleBand()   // y軸用scaleを作る
        .domain([0,1,2,3,4,5])    // 入力範囲
        .range([50, 350])
        .padding(0.25);           // 棒の間隔の比率
```

scaleLinearは、データを座標に変換する関数です。ここでは、データの範囲[0,50]を 座標[100, 450]に対応させる関数を作ります。

scaleBandも、データを座標に変換する関数です。ただし、scaleLinearのように線形関係ではなく、棒グラフを作る際に必要なrectの位置を計算することに特化した関数です。ここでは、データの背番号[0,1,2,3,4,5]を座標[50, 350]に対応させる関数を作ります。



これらのscale関数を事前に作っておくと、SVG要素の属性を計算する時に極めて簡潔に書けるようになります。

```
var bars = svg
        .selectAll("rect")        // rect要素を全部集める(最初はゼロ個)
        .data(mydata);            // データと一対一対応させる
// dataメソッドによって、以下の3種類が作られる
```

```
var bars_exist = bars;        // データもrectもあるもの(最初はゼロ)
var bars_enter = bars.enter();// データはあるがrectがないもの
var bars_exit  = bars.exit(); // データはあるがrectがないもの(最初はゼロ)

bars_enter.append("rect")
        .attr("x", xScale(0))     // 右上のx座標
        .attr("y", function(d,i){return yScale(i);})     // 右上のy座標
        .attr("width", function(d,i){return xScale(d) - xScale(0);})
        .attr("height", function(d,i){return yScale.bandwidth();})
        .attr("fill", "blue");
```

selectAll()とdata()の部分は、d3で最も難しい「データバインディング」という仕組みです。詳細を説明すると日が暮れるので省略しますが、この結合によって、データとSVG要素が 一対一 で結びつけられます。

その結果、「SVG要素は自分が表示すべきデータを知っている」状態になります。属性yを設定する行で、function(d,i)という関数が記述されていますが、この時のdには「自分が表示すべきデータ」が、iには「自分が表示すべきデータの位置」が自動で設定されます。

そのd,iと、事前に用意したscale関数とを使えば、rectの属性(x,y,width,height)が簡潔に計算できます。

ここで理解できなくても悲観しないでください。d3用にエンジニアを採用する会社があるほど、奥が深いノウハウです。

それほど難しいd3をなぜ勧めるのか。

sample04.htmlをブラウザとエディタで開き、デスクトップに並べましょう。２箇所修正し、４行追加されています。

```
// axisは、scaleの設定を読み取って自動で軸を作る関数
var xAxis = d3.axisTop(xScale);
var yAxis = d3.axisLeft(yScale);
// axisを適用する
svg.append("g").attr("transform", "translate(0,30)").call(xAxis);
svg.append("g").attr("transform", "translate(100,0)").call(yAxis);
```

これだけでグラフの軸を自動生成してくれます。自分で軸の線やラベルを作ることを想像してみてください。また、表示データやscaleの設定を自由に変更してみてください。

このように、d3は、グラフを描くために毎回必要な手続きを部品化し、面倒な計算を引き受けてくれます。

インタラクティブにする

デジタルコンテンツならではの機能として、インタラクティブなグラフを作ってみましょう。ここでは、過去３年分のセ・リーグの成績を表示します。

なお、最近の潮流として、インタラクティブなグラフは、積極的な理由がない限り、あまり使われなくなっています。

sample05.htmlをブラウザとエディタで開き、デスクトップに並べてください。

```
var mydata = [ // 2014-16のデータ
        {name:"広島", color:"#FF0000", wins:[74, 69, 89]},
        {name:"巨人", color:"#FFA500", wins:[82, 75, 71]},
        {name:"DeNA",color:"#6495ED", wins:[67, 62, 69]},
        {name:"阪神", color:"#FFFF00", wins:[75, 70, 64]},
        {name:"ヤクルト", color:"#0000CD", wins:[60, 76, 64]},
        {name:"中日", color:"#0000FF", wins:[67, 62, 58]}
];
```

このように、javascriptでは、データをObject型でパッケージ化して扱うことができます。

```
// もう一度selectAllで選択しなおす
bars_exist = svg.selectAll("rect");

// 再描画のための関数
function redraw(){
        // 棒の長さをtransition付きで更新する
        bars_exist.transition().duration(500)
                .attr("width", function(d,i){return xScale(d.wins[selectIndex]) - xScale(0);})
}
```

関数redrawは、年が選択された時に実行される「描き直し」です。transition()を設定するだけで、自動でアニメーションになります。

```
// ボタンの設定
d3.select("#year2014").on("click", function(){
        selectIndex = 0;
        redraw();
});
```

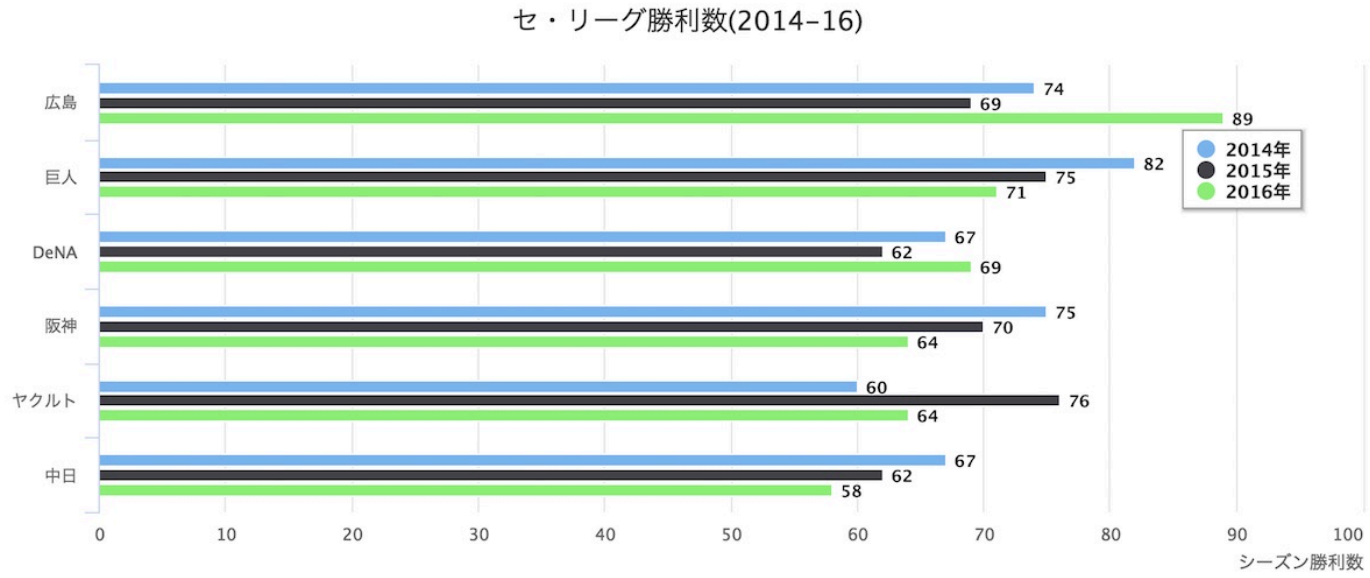
最後にボタンを設定します。d3ではon()でイベントを登録します。

このように、アニメーションが簡単に実現できることもd3の強みです。

外部サービスを利用する例

d3.jsを使ったグラフ作成ライブラリ（有料）として、HighChartがあります。 sample06.htmlをブラウザとエディタで開き、デスクトップに並べてください。

HighChart版



外部ライブラリを利用するメリット・デメリットは以下の通りです。

- データとオプションを設定するだけで、高機能なグラフが実現できる
- データの形式を含め、ライブラリの仕様に合わせなければならない
- javascript/d3の学習機会は失われる

グラフ表現にこだわりがないなら、d3を使う必要はありません。しかし、汎用性を追求しているライブラリにオリジナリティはありません。何を求めるか、会社・個人で判断しなければなりません。

最後に

これまでの勉強会で展開してきたデータを使い、可視化の愚案を紹介します。