

Information Visualization

D3.js - 1

Original by Jaemin Jo (<https://github.com/e->)

Modified by Dantae An (dtan@hcil.snu.ac.kr)


—

Human-Computer Interaction Laboratory
Seoul National University

-
- This image is a dense, multi-colored collage of various data visualization techniques. It features a central hexagonal grid pattern. The visual elements include:
- Maps:** Several geographical maps are scattered throughout, including a world map, a map of Africa, and a map of the Middle East.
 - Bar Charts:** Multiple bar charts in various colors (blue, green, red, yellow) are present, some with axes labeled with numbers.
 - Pie Charts:** Several pie charts and donut charts are included, showing different data distributions.
 - Network Graphs:** There are several network diagrams with nodes and connecting lines, some showing hierarchical structures.
 - Abstract Patterns:** The collage includes various abstract patterns, such as hexagonal grids, circles, and lines, in a wide range of colors.
 - Text and Labels:** Some text labels are visible, such as "KING", "Tamer", "Obama", "McCain", "Merry", "Blair", "Gore", "Bush", "30 sec", "studies", "science", and "graphy".
- The overall composition is a complex, multi-layered visual representation of data, likely generated using a generative model like GAN.

D3.js

- 저널 페이퍼로도 발간됨
 - <http://vis.stanford.edu/files/2011-D3-InfoVis.pdf>
 - Bostock, M., Ogievetsky, V., & Heer, J. (2011). D³: Data-Driven Documents. *IEEE Transactions on Visualization & Computer Graphics*, (12), 2301-2309.
- 인기 많음
 - 논문: 2700여 번 인용
 - 다양한 시각화 라이브러리의 core
 - 스타 많음

 d3 / d3 Watch






4k

 Star

93.5k

 Fork

22.2k

 Code Issues 5 Pull requests 1 Actions Wiki Security Insights

D3.js

Version & Installation

Version

- 현재 D3의 최신 버전: 6.x
 - 나온 지 한 달도 안 됨

```
d3 = require("d3@6");
```

버전 번호

- [6 vs. 5]
호환되지 않는 변화가 많지만
migrate할 수 있음
 - <https://github.com/d3/d3/blob/master/CHANGES.md#breaking-changes>

Breaking Changes

D3 6.0 introduces several non-backwards-compatible changes.

- Remove `d3.event`.
- Change `selection.on` to pass the *event* directly to listeners.
- Change `transition.on` to pass the *event* directly to listeners.
- Change `brush.on` to pass the *event* directly to listeners.
- Change `drag.on` to pass the *event* directly to listeners.
- Change `zoom.on` to pass the *event* directly to listeners.
- Remove `d3.mouse`; use `d3.pointer`.
- Remove `d3.touch`; use `d3.pointer`.
- Remove `d3.touches`; use `d3.pointers`.
- Remove `d3.clientPoint`; use `d3.pointer`.
- Remove `d3.voronoi`; use `d3.Delaunay`.
- Remove `d3.nest`; use `d3.group` and `d3.rollup`.
- Remove `d3.map`; use `Map`.
- Remove `d3.set`; use `Set`.
- Remove `d3.keys`; use `Object.keys`.
- Remove `d3.values`; use `Object.values`.
- Remove `d3.entries`; use `Object.entries`.
- Rename `d3.histogram` to `d3.bin`.
- Rename `d3.scan` to `d3.leastIndex`.
- Change `d3.interpolateTransformCss` to require absolute units.
- Change `d3.format` to default to the minus sign instead of hyphen-minus for negative values.

D3 now requires a browser that supports [ES2015](#). For older browsers, you must bring your own transpiler.

Lastly, support for [Bower](#) has been dropped; D3 is now exclusively published to npm and GitHub.

See our [migration guide](#) for help upgrading.

Version

- [5 vs. 4] 많이 바뀌지 않음 (서로 거의 호환됨)
- [5 vs. 3] 라이브러리의 모듈화 등등... 많은 변화 (사용 불가)
- <https://github.com/d3/d3/blob/master/CHANGES.md>
- 배울 때 힘든 점
 - D3 관련 서적, 웹 상의 자료 및 코드가 3 ~ 6 버전에 걸쳐 혼재함
 - 자료 참고 시 버전에 유의할 것
 - 버전 5 이하의 코드를 참고할 때에는 버전 6으로 바꿔서 사용
(다음 슬라이드 참조)
- 이번 실습에서는 6 버전을 사용!

Version Migration

• 5.x의 코드를 6.x로 바꾸는 법

- `d3 = require("d3@6");` 또는

`<script src="https://d3js.org/d3.v6.min.js"></script>` 인지 확인

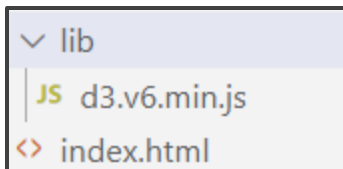
- <https://observablehq.com/d/f91cccf0cad5e9cb>
- 아래의 코드가 버전 5에 포함되어 있다면 위의 링크 내용대로 변경해야 함
 - 이벤트: d3.event ([슬라이드 66](#), [슬라이드 67](#) 참조)
 - 이벤트 위치: d3.mouse, d3.touch, d3.touches, d3.clientPoint
 - 자료구조: d3.map, d3.set, d3.keys, d3.values, d3.entries
 - 히스토그램: d3.histogram
 - 기타: d3.scan, d3.voronoi, d3.nest,
d3.interpolateTransformCss 의 길이 단위 (% , em 등의 상대적 길이 사용 불가)
- 나머지는 그대로 사용 가능

D3.js의 설치

- 방법 1: Script 태그 추가 (.html 파일 작업 시)
 - 그때그때 다운로드하여 사용 – URL 적기

```
<body>  
  <script src="https://d3js.org/d3.v6.min.js"></script>  
</body>
```

- 또는 위 주소에서 한 번 다운로드 받아 놓고 로컬에서 사용 – 경로 적기



```
<body>  
  <script src="./lib/d3.v6.min.js"></script>  
</body>
```


D3.js의 설치

• 방법 2: 패키지 매니저(npm)로 설치 (.js 또는 .ts 파일 작업 시)

1. 터미널 또는 명령 프롬프트(cmd) 실행
2. 프로젝트 폴더로 이동

```
cd "D:/infovis/exercise2"
```

3. D3 설치

```
npm install d3
```

4. .js 파일에서 다음 코드 포함

```
import * as d3 from "d3"; 또는 const d3 = require("d3");
```

JavaScript

Arrow Function

Arrow Function Expression

- Introduced in ES 6

```
let my_func = function (a, b) { return a + b; }  
my_func = (a, b) => { return a + b; }  
my_func = (a, b) => a + b;  
my_func = a => a + 1;
```

```
(param1, ..., paramN) => { statements }  
(param1, ..., paramN) => expression  
//      다음과 동일함: => { return expression; }  
  
// 매개변수가 하나뿐인 경우 괄호는 선택사항  
(singleParam) => { statements }  
singleParam => { statements }  
  
// 매개변수가 없는 함수는 괄호 필요  
() => { statements }
```

- function(){} 과는 다름!

- this 사용 시 주의!
- "lexical this"

- 영어 문서: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
- 한글 설명: <https://poiemaweb.com/es6-arrow-function#3-this>

Arrow Function Expression

• 응용(디버깅) 팁

```
array = [2, 4, 5];  
array = array.filter((d, i, a) => i % 2 === 0);
```

- 위의 코드에서 filter 함수의 인자로 넘긴
arrow function의 각 인자의 값 `(d, i, a)` 이 궁금하다면?

```
array = [2, 4, 5];  
array = array.filter((d, i, a) => {  
  console.log(d + " " + i + " " + a);  
  return i % 2 === 0;  
});
```

2	0	2,4,5
4	1	2,4,5
5	2	2,4,5

D3.js

Selection & Data Join

Selection

- D3.js에서 DOM 조작을 위해 요소를 선택하는 것
- rect 선택하기

```
<body>
  <svg height="1000" width="1000">
    <rect x="0" y="0" width="100" height="50"></rect>
    <rect x="0" y="100" width="200" height="50"></rect>
    <rect x="0" y="200" width="300" height="50"></rect>
  </svg>
  <script src="https://d3js.org/d3.v6.min.js"></script>
  <script>
    d3.select('rect')
      .attr('width', '600');
  </script>
</body>
```



d3.select()

- `d3.select(selector)`
 - 문서 전체에서 *selector*에 해당하는 첫 번째 요소를 선택한 뒤, selection을 반환
- `selection.attr(name[, value])`
 - *selection*이 갖고 있는 요소의 *name* 속성에 *value*를 지정

```
<body>
  <svg height="1000" width="1000">
    <rect x="0" y="0" width="100" height="50"></rect>
    <rect x="0" y="100" width="200" height="50"></rect>
    <rect x="0" y="200" width="300" height="50"></rect>
  </svg>
  <script src="https://d3js.org/d3.v6.min.js"></script>
  <script>
    d3.select('rect')
      .attr('width', '600');
  </script>
</body>
```

d3.selectAll()

- `d3.selectAll(selector)`
 - 문서 전체에서 *selector*에 해당하는 element(요소)를 모두 선택한 뒤, selection을 반환

```
d3.select('rect')  
  .attr('width', '600');
```



```
d3.selectAll('rect')  
  .attr('width', '600');
```



속성 및 스타일 지정

- `selection.attr()` , `selection.style()` , ...
 - 이전 시간에 배운 `map`, `forEach` 처럼 `selection` 안의 element 배열에 대해 순차적으로 실행됨
 - 이후 `selection`을 반환 (chaining 가능)

```
d3.selectAll('rect')  
  .attr('width', '600')  
  .style('fill', 'red');
```

```
d3.selectAll("p").style("font-size", "10px");
```

```
d3.selectAll("p").style("font-size", (d, i) => i * 10 + "px");
```

```
d3.selectAll("p").style("font-size", "10px").style("color", "red");
```

속성 지정

- 언제 `selection.attr()` 를 사용하는가?

- 전역 attribute

- `.attr('class', 'my-class')`

- `.attr('transform', 'translate(50, 100)')`

- <rect> 지정 attribute

- `'x' 'y' 'width' 'height' 'rx' 'ry'`

- <circle> 지정 attribute

- `'cx' 'cy' 'r'`

모서리가 둥근 사각형을 만들 때 사용

- 기타 여러 타입의 attribute들...

스타일 지정

- 언제 `selection.style()` 을 사용하는가?
 - CSS에서 사용할 수 있는 스타일을 설정할 때 사용
- <rect>, <circle> 등의 SVG 도형 스타일 예제
 - `.style('fill', 'skyblue')`
 - `.style('stroke', 'cadetblue')`
 - `.style('stroke-width', 2)`
- <p>, <h1> 등의 글자 스타일 예제
 - `.style('color', 'red')`
 - `.style('font-family', 'consolas')`
 - `.style('font-size', '40px')`

Empty Selection

- 문서 내에 없는 요소를 선택할 경우 어떻게 될까?

```
<body>
  <svg height="1000" width="1000">
  </svg>
  <script src="https://d3js.org/d3.v6.min.js"></script>
  <script>
    d3.select('rect')
      .attr('width', '600');
  </script>
</body>
```

Empty Selection

- 문서 내에 없는 요소를 선택할 경우 어떻게 될까?

```
<body>
  <svg height="1000" width="1000">
  </svg>
  <script src="https://d3js.org/d3.v6.min.js"></script>
  <script>
    d3.select('rect')
      .attr('width', '600');
  </script>
</body>
```

- 존재하지 않는 요소를 select

→ empty selection 반환

- empty selection의 경우 attr(), style()을 수행해도 아무 일도 일어나지 않음

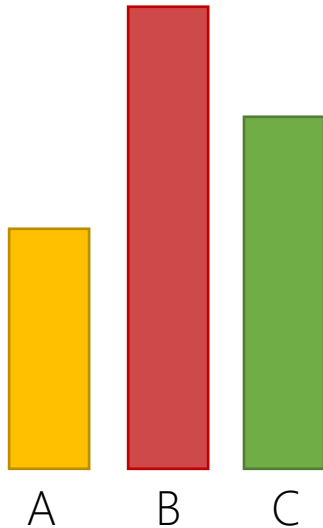
Data Join

- D3.js helps you bring data to life using HTML, SVG, and CSS.
- D3.js 의 데이터 단위는 배열
 - number, string, object array 등 모든 타입의 배열이 가능
- (예제) 내가 가진 데이터가 숫자 배열일 경우

```
<body>  
  <svg height="1000" width="1000">  
  </svg>  
  <script src="https://d3js.org/d3.v6.min.js"></script>  
  <script>  
    let my_data = [1, 2, 3];  
  </script>  
</body>
```

selection.data() 이해하기

```
let oldData = [  
  {name: 'A', value: 10},  
  {name: 'B', value: 20},  
  {name: 'C', value: 15}  
];
```



{name: 'A', value: 10}

{name: 'B', value: 20}

{name: 'C', value: 15}

각 오브젝트(데이터)가
바(요소)에 연결됨

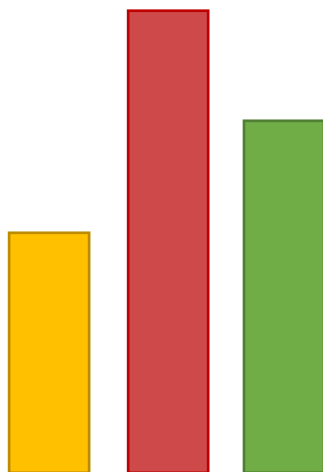
selection.data() 이해하기

```
let oldData = [
  {name: 'A', value: 10},
  {name: 'B', value: 20},
  {name: 'C', value: 15}
];
```



데이터 변경!

```
let newData = [
  {name: 'B', value: 10},
  {name: 'C', value: 15},
  {name: 'D', value: 5}
];
```



A

B

C

{name: 'A', value: 10}

{name: 'B', value: 20}

{name: 'C', value: 15}

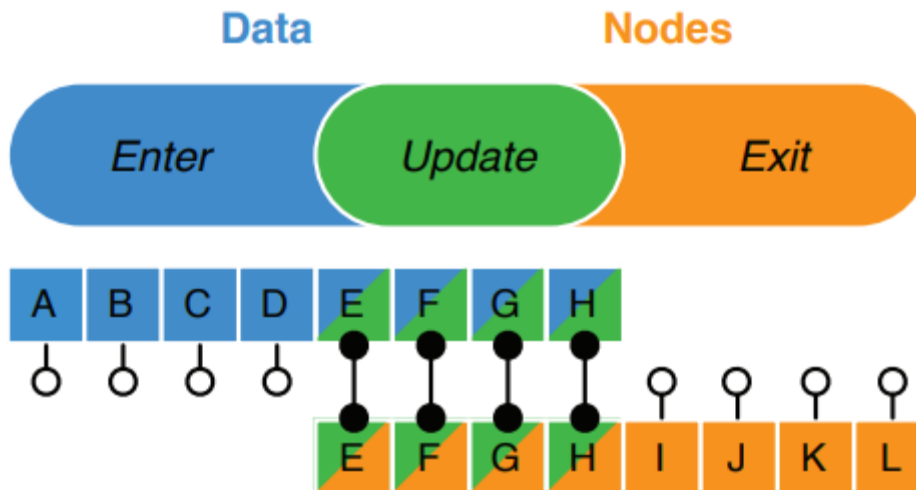


?

각 오브젝트(데이터)가
바(요소)에 연결됨

name을 key라고 했을 때,
 없어진 것: A (exit)
 변경된 것: B, C (update)
 추가된 것: D (enter)

Selection의 종류



- **Enter** selection
: 요소와 조인되지 못한 데이터들만 가지고 있다.
- **Update** selection
: 데이터와 성공적으로 조인된 요소들만 가지고 있다.
- **Exit** selection
: 데이터와 조인되지 못한 요소들만 가지고 있다.

* 요소: 시각화를 구성하는 그림

Data Join

- `selection.data([data[, key]])`

- `selection` 내의 요소들과 주어진 `data`(배열)를 조인(연결)시키고 update `selection`을 반환함

```
<script>  
  let my_data = [1, 2, 3];  
  
  d3.selectAll('rect')  
    .data(my_data);  
</script>
```

- `Update` `selection`에서 `.enter()`, `.exit()` 함수를 호출하여 각각 `enter`, `exit` `selection`을 가져올 수 있음

Common Scenario

- **Enter** selection에 있는 데이터들은 시각적 요소를 만들어줘야 함
 - 요소 추가: `selection.enter().append('rect')`
 - 요소 추가 후 크기, 위치, 색깔 등을 attr과 style 메소드를 사용해서 변경
- **Update** selection에 있는 요소들은 데이터의 값에 따라 시각적 요소의 속성을 변경해줘야 함
 - 크기, 위치, 색깔 등을 attr과 style 메소드를 사용해서 변경
- **Exit** selection에 있는 요소들은 DOM에서 제거해야 함

Common Scenario

```
let newData = [  
  {name: 'B', value: 10},  
  {name: 'C', value: 15},  
  {name: 'D', value: 5}  
];  
let svg = d3.select('svg');  
let bars = svg.selectAll('rect').data(newData, d => d.name);
```

```
/* enter selection 처리 */
```

```
bars.enter()  
  .append('rect')  
  .attr('width', 30)  
  .attr('height', d => d.value * 10)  
  .attr('transform', (d, i) => `translate(${i * 40}, 0)`);
```

```
/* update selection 처리 */
```

```
bars.attr('width', 30)  
  .attr('height', d => d.value * 10)  
  .attr('transform', (d, i) => `translate(${i * 40}, 0)`);
```

```
/* exit selection 처리 */
```

```
bars.exit().remove();
```



`translate(\${i * 40}, 0)`는
'translate(' + i * 40 + ', 0)'
와 같음 (슬라이드 50 참조)

enter된 요소와
update된 요소에
같은 변경을
가하는데,
코드 반복을
줄일 수 있을까?

selection.merge()

```
let newData = [
  {name: 'B', value: 10},
  {name: 'C', value: 15},
  {name: 'D', value: 5}
];
let svg = d3.select('svg');
let bars = svg.selectAll('rect').data(newData, d => d.name);

/* enter & update selection 처리 */
bars.enter()
  .append('rect')
  .merge(bars)
  .attr('width', 30)
  .attr('height', d => d.value * 10)
  .attr('transform', (d, i) => `translate(${i * 40}, 0)`);

/* exit selection 처리 */
bars.exit().remove();
```



- `selection.merge(other)` : 현재 *selection*과 *other* selection을 합친 새로운 selection을 반환함

FAQ

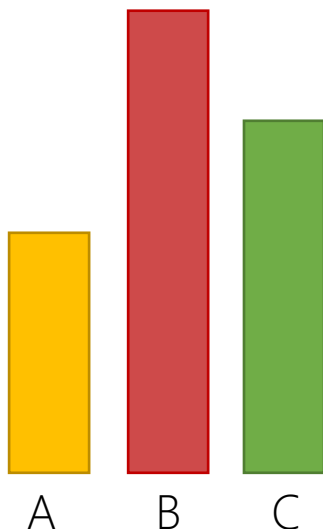
- 처음에는 어떻게 되나요?
 - 처음에는 모든 데이터가 DOM 요소를 가지고 있지 않기 때문에, 모든 데이터가 **enter** selection에 속하게 됩니다.
- 왜 굳이 이렇게 하나요? 그냥 다 지웠다가 모두 다시 그리면 안 되나요?
 - 성능, 트랜지션, 일반화, incremental processing, ...
- `svg.selectAll('rect').data(newData, d => d.name)` 에서 data 메소드의 두 번째 인자로 왜 함수(arrow function)를 넘기나요?
 - 두 번째 인자를 주지 않으면 기본적으로 index를 가지고 조인합니다.
두 번째 인자를 함수로 주면 이것의 반환 값을 key로 하여 조인합니다.

Key 이해하기

```
let oldData = [
  {name: 'A', value: 10},
  {name: 'B', value: 20},
  {name: 'C', value: 15}
];
```




```
let newData = [
  {name: 'B', value: 10},
  {name: 'C', value: 15},
  {name: 'D', value: 5}
];
```



```
svg.selectAll('rect').data(newData)
```

key를 지정하지 않았다면 index를 key로 사용하므로 세 개의 rect가 모두 **update** selection에 있게 됩니다.

old key: [0, 1, 2]

 new key: [0, 1, 2]

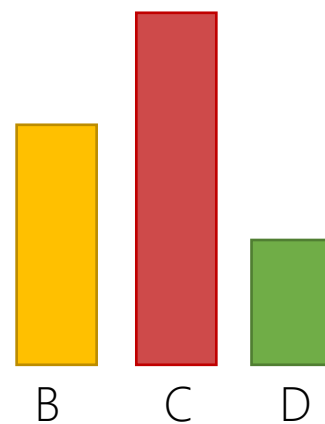
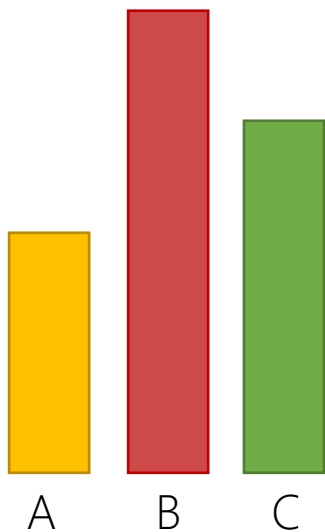
Key 이해하기

```
let oldData = [
  {name: 'A', value: 10},
  {name: 'B', value: 20},
  {name: 'C', value: 15}
];
```




데이터 변경!

```
let newData = [
  {name: 'B', value: 10},
  {name: 'C', value: 15},
  {name: 'D', value: 5}
];
```

의도한
결과가
아님!

```
svg.selectAll('rect').data(newData)
```

key를 지정하지 않았다면 index를 key로 사용하므로
세 개의 rect가 모두 update selection에 있게 됩니다.

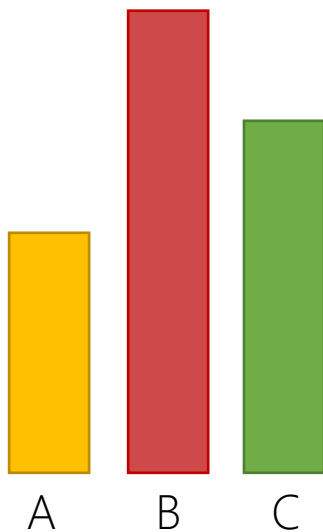
old key: [0, 1, 2]

 new key: [0, 1, 2]

Key 이해하기

```
let oldData = [
  {name: 'A', value: 10},
  {name: 'B', value: 20},
  {name: 'C', value: 15}
];
```



```
let newData = [
  {name: 'B', value: 10},
  {name: 'C', value: 15},
  {name: 'D', value: 5}
];
```



key를 지정하면
key가 같은 것끼리 같은 오브젝트로 취급합니다.
따라서 name이 B, C인 rect는 **update** selection에,
D인 rect는 **enter** selection에 있게 됩니다.

```
svg.selectAll('rect').data(newData, d => d.name)
```

old key: ['A', 'B', 'C']
new key: ['B', 'C', 'D']

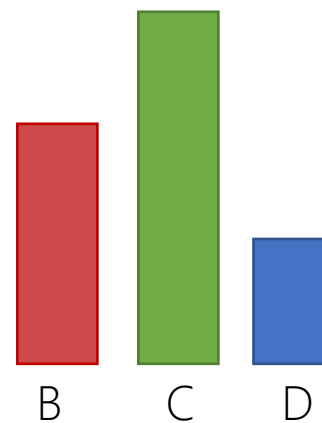
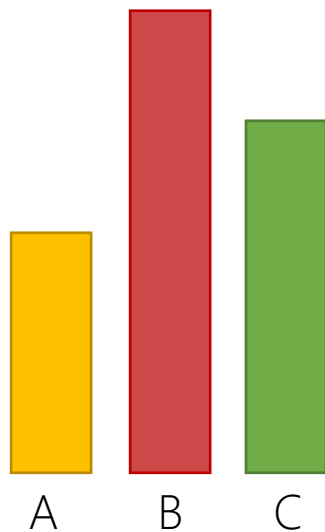
Key 이해하기

```
let oldData = [
  {name: 'A', value: 10},
  {name: 'B', value: 20},
  {name: 'C', value: 15}
];
```



데이터 변경!

```
let newData = [
  {name: 'B', value: 10},
  {name: 'C', value: 15},
  {name: 'D', value: 5}
];
```



key를 지정하면
key가 같은 것끼리 같은 오브젝트로 취급합니다.
따라서 name이 B, C인 rect는 **update** selection에,
D인 rect는 **enter** selection에 있게 됩니다.

```
svg.selectAll('rect').data(newData, d => d.name)
```

old key: ['A', 'B', 'C']
new key: ['B', 'C', 'D']

`selection.append()`

- `selection.append(type)`
 - Update selection에서 사용 시 `selection`이 갖는
 1. 각각의 요소들에 대해,
 2. `type` 이름의 새로운 요소를
 3. 마지막 자식으로 추가한다.
 - Enter selection에서 사용 시`.selectAll()`을 수행한 부모 요소에 현재 enter selection과 조인된 `type` 이름의 새로운 요소가 추가된다.

* 가능한 `type` 이름: 'rect', 'circle', 'g' 등

데이터를 반영한 그리기

- `selection.style()` , `selection.attr()` 메소드들의 두 번째 인자에 함수를 넣으면, 그 함수의 인자로는
 1. 현재 데이터 `d`
 2. 현재 요소의 `selection` 상의 인덱스 `i`
 3. `selection`이 가지고 있는 요소들의 배열 `nodes`가 순서대로 설정된다.
- 함수의 인자 이름은 바뀌서 사용할 수 있다.
- 함수의 인자를 꼭 3개 다 지정할 필요는 없다.
 - 예) 인덱스가 필요하고 요소 배열이 필요 없다면: `(d, i) => d.value * i`
- `selection.attr('width', (d, i, nodes) => d.value * i)`

D3.js

Drawing a Scatterplot

사용할 데이터

- JavaScript 배열 형태의 영화 데이터: <https://pastebin.com/7WNHLVYk>

- 통째로 복사하여 `<script> </script>` 사이에 붙여 넣는다.

- Column 정보

- title: 제목 / genre: 장르
- creative_type: 창작 유형 / source: 원작
- release: 시대 / rating: 등급 / budget: 제작비
- us_gross: 미국 수익
- worldwide_gross: 전세계 수익
- rotten_rating: 로튼토마토 평점
- imdb_rating: IMDB 평점
- imdb_votes: IMDB 평가 수

```
let data =  
[  
  {  
    "title": "Twin Falls Idaho",  
    "genre": "드라마",  
    "creative_type": "현대소설",  
    "source": "없음",  
    "release": "90년대",  
    "rating": "19세이상",  
    "budget": "5",  
    "us_gross": "9",  
    "worldwide_gross": "10",  
    "rotten_rating": "77",  
    "imdb_rating": "7.1",  
    "imdb_votes": "2,810"  
  },  
]
```

데이터 파싱(Parsing)

- string → number

```
data.forEach(function(d) {  
    d.us_gross = parseFloat(d.us_gross);  
    d.rotten_rating = parseFloat(d.rotten_rating);  
});
```

- `parseFloat(string)`

- https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/parseFloat

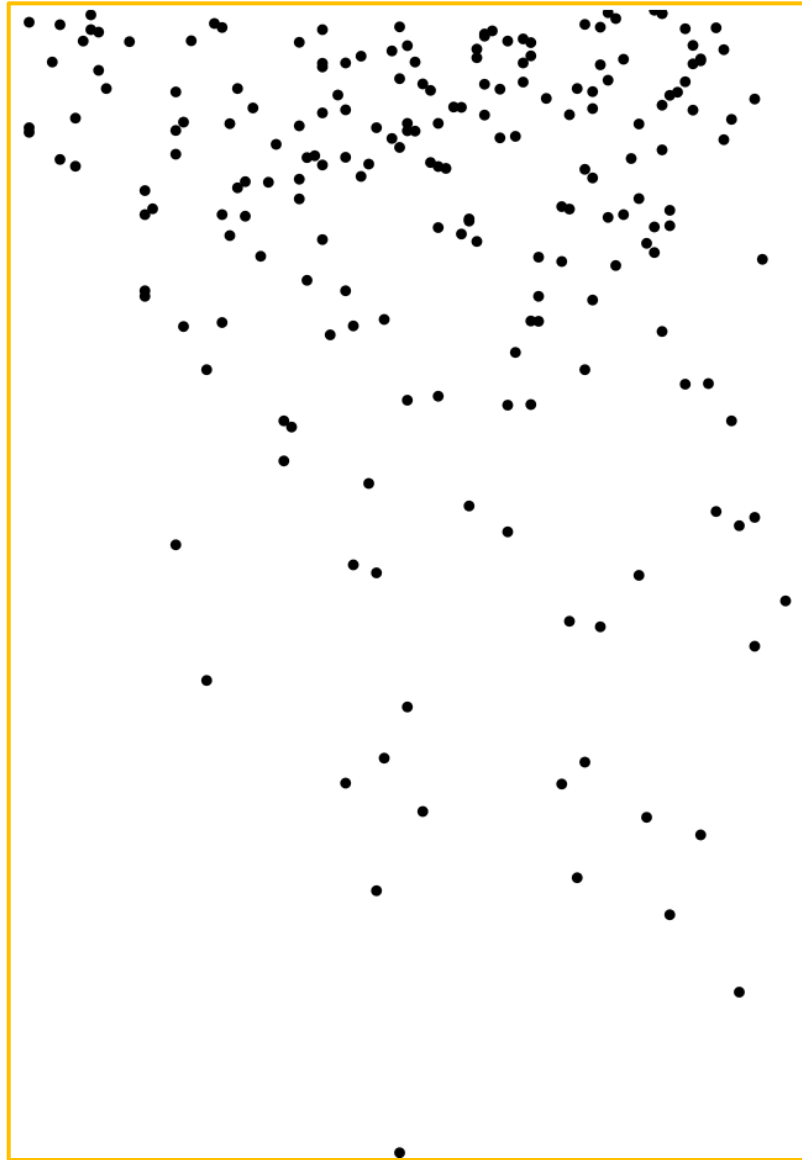
```
{  
    "title": "Beverly Hills Cop II",  
    "genre": "액션",  
    "creative_type": "현대소설",  
    "source": "없음",  
    "release": "80년대",  
    "rating": "19세 이상",  
    "budget": "200",  
    "us_gross": "1536",  
    "worldwide_gross": "2766",  
    "rotten_rating": "46",  
    "imdb_rating": "6.1",  
    "imdb_votes": "29,712"  
},
```

점(Point) 찍기

```
let svg = d3.select('svg');  
  
svg  
  .selectAll('circle')  
  .data(data)  
  .enter()  
    .append('circle')  
    .attr('r', 3.5)  
    .attr('cx', d => d.rotten_rating * 5)  
    .attr('cy', d => d.us_gross / 5);
```

- 데이터 배열의 원소 하나하나가 갖고 있는 평점과 수익 값을 대강 살펴본 뒤, 평점에 5를 곱하고, 수익에 5를 나누어 주는 식으로 점의 좌표를 지정하였다.

점(Point) 찍기



개선할 수 있는 점

1. 점의 위치를 구할 때 직접 계산해줄 것이 아니라 좀 더 일반적인 방법으로 구할 수 있지 않을까?
2. X, Y 축을 그리고 싶다.
3. 영화 관람 등급에 따라 분포가 어떻게 달라지는지 확인하고 싶다.
점의 색으로 영화 관람 등급을 표현하자.

개선할 수 있는 점

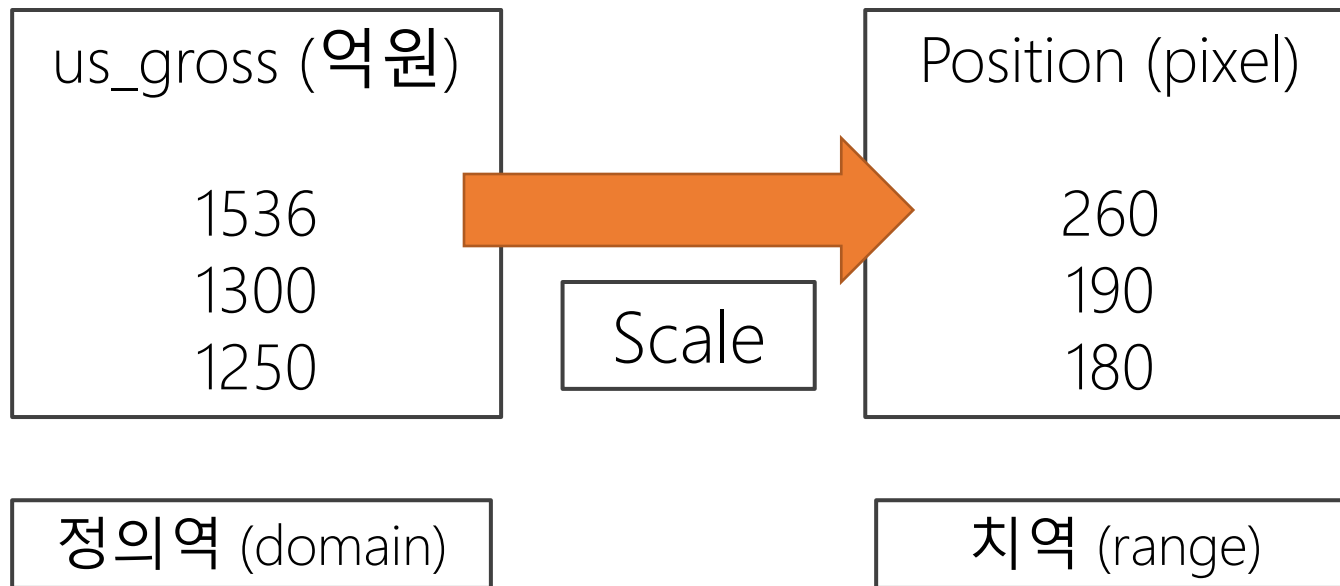
1. 점의 위치를 구할 때 직접 계산해줄 것이 아니라
좀 더 일반적인 방법으로 구할 수 있지 않을까? → `d3.scaleLinear`
2. X, Y 축을 그리고 싶다. → `d3.axisBottom` / `d3.axisLeft`
3. 영화 관람 등급에 따라 분포가 어떻게 달라지는지 확인하고 싶다.
점의 색으로 영화 관람 등급을 표현하자. → `d3.scaleOrdinal`

D3.js - Drawing a Scatterplot

Scale

Scale

- Scale이란?
→ 데이터의 값과 화면 상의 픽셀 값을 연결해주는 함수
즉, 데이터 상 값을 화면 상 위치(픽셀)에 매핑



Scale 구하기 & 적용하기

- X축의 Scale 구하기

X축의 정의역

= data 배열 내의
rotten_rating 중
최솟값 ~ 최댓값

X축의 치역

= X축의 좌표 0 부터
내가 정한 width까지

```
let svg = d3.select('svg');
let width = 500;
let height = 500;

let x = d3.scaleLinear()
  .domain([
    d3.min(data, d => d.rotten_rating),
    d3.max(data, d => d.rotten_rating)
  ])
  .range([0, width]);

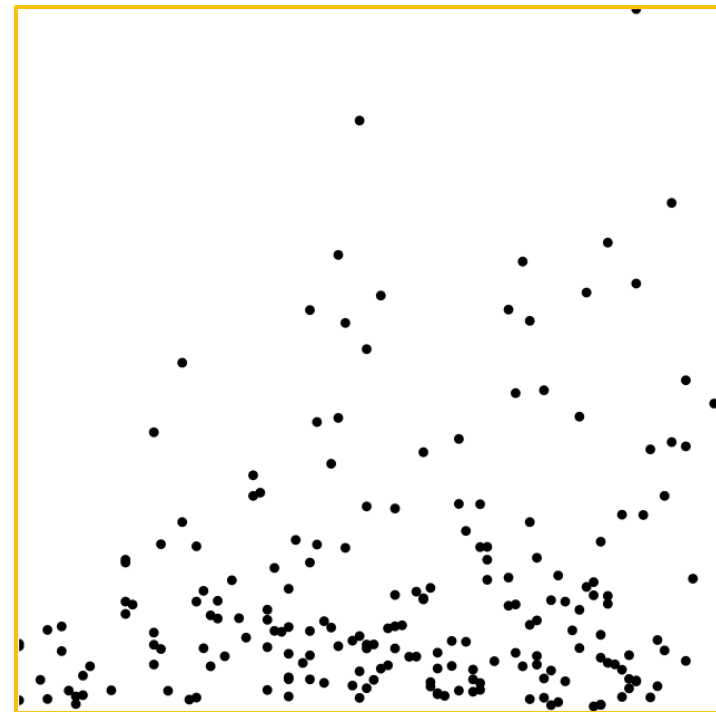
svg
  .selectAll('circle')
  .data(data)
  .enter()
  .append('circle')
  .attr('r', 3.5)
  .attr('cx', d => x(d.rotten_rating))
  .attr('cy', d => d.us_gross / 5);
```

Scale 구하기 & 적용하기

• 결과

```
let y = d3.scaleLinear()  
    .domain([  
      d3.min(data, d => d.us_gross),  
      d3.max(data, d => d.us_gross)  
    ])  
    .range([height, 0]);
```

```
svg  
  .selectAll('circle')  
  .data(data)  
  .enter()  
    .append('circle')  
    .attr('r', 3.5)  
    .attr('cx', d => x(d.rotten_rating))  
    .attr('cy', d => y(d.us_gross));
```



- 여기까지의 코드: <https://pastebin.com/5yEmitV> (버전 5이지만 상관 X)

D3.js - Drawing a Scatterplot

Axis & Margin

축(Axis) 그리기

```
let xAxis = d3.axisBottom(x);  
let yAxis = d3.axisLeft(y);
```

인자로 주어진 scale에 맞게
축을 아래쪽, 왼쪽에 그려주는 함수

```
svg  
  .append('g')  
  .attr('transform', `translate(0, ${height})`)  
  .call(xAxis);
```

현재 선택된 `<g>`에 대해, 축을 그리는 함수 호출
(`<g>`에 대한 설명은 [슬라이드 53](#) 참조)

```
svg  
  .append('g')  
  .call(yAxis);
```

- transform 속성을 `'translate(x, y)'` 으로 지정하면 (x, y) 만큼 평행이동

축(Axis) 그리기

- transform 속성에 대해...

```
><g transform="translate(0, 500)" fill="none" font-size="10"  
font-family="sans-serif" text-anchor="middle">...</g>
```

- 'translate(0, 500)' 이라는 string을 넘긴 것이다. (함수를 넘긴 게 아님)

```
let height = 500;  
/*  
 * ...  
 */  
svg  
  .append('g')  
  .attr('transform', `translate(0, ${height})`)  
  .call(xAxis);
```

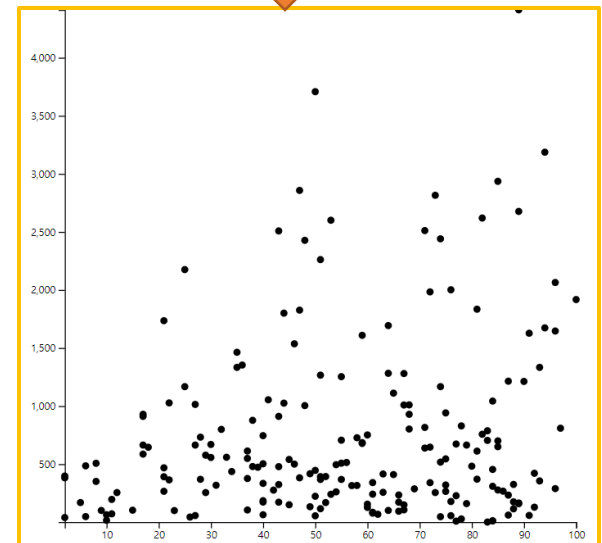
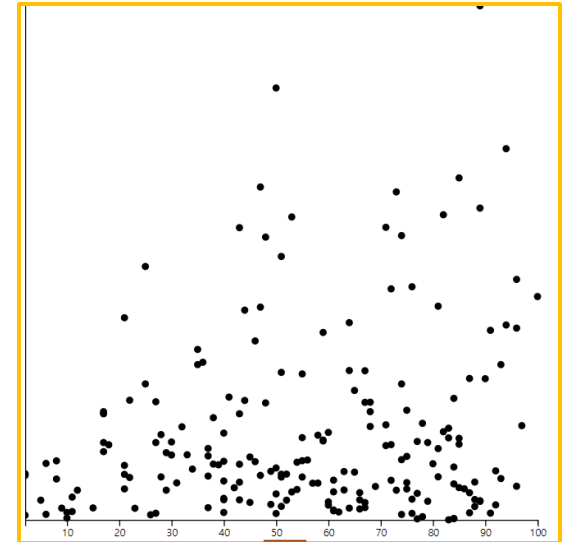
template literal: 백틱(``, 키보드의 ~와 같은 위치의 키)을 사용한 문자열
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Template_literals

Margin 적용하기

- y축이 잘 보이지 않는 문제가 있다.
 - 점들의 치역을 50씩 오른쪽으로,
 - 축을 50만큼 오른쪽으로 평행이동하자.

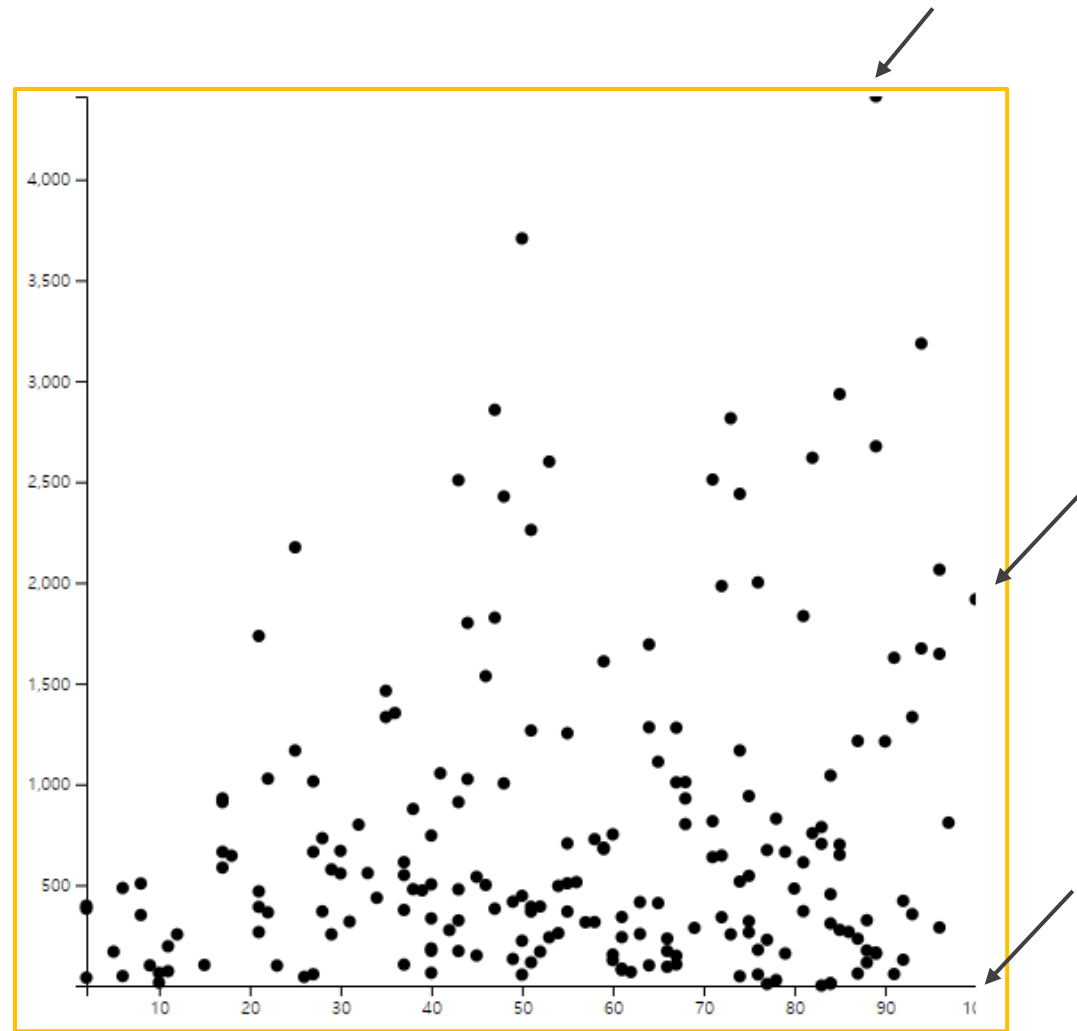
```
let x = d3.scaleLinear()  
  .domain([  
    d3.min(data, d => d.rotten_rating),  
    d3.max(data, d => d.rotten_rating)  
  ])  
  .range([50, width + 50]);
```

```
svg  
  .append('g')  
  .attr('transform', translate(50, 0))  
  .call(yAxis);
```



Margin 적용하기

- 문제점
 - 여전히 잘릴 수 있다.
 - 더욱 일반적인 방법?



Margin 적용하기

- 새로운 그룹 요소 `<g>`를 만들고, 거기에 차트를 그리자.

```
<body>
  <div id="plot"></div>
  <!--
    ...
  -->
```

`<g>`: SVG에서 여러 요소들을 묶을 수 있는 그룹 요소.
그룹화하면 그룹 전체를 한번에 조작할 수 있어
계층 구조를 만들 때 유용하다.

```
let svgWidth = 550;
let svgHeight = 550;
let margin = {top: 20, right: 10, bottom: 30, left: 40};
let width = svgWidth - margin.left - margin.right;
let height = svgHeight - margin.top - margin.bottom;
let svg = d3.select('#plot')
  .append('svg')
  .attr('width', svgWidth)
  .attr('height', svgHeight)
  .append('g')
  .attr('transform', translate(margin.left, margin.top));
```

- 그리고 [슬라이드 51](#)에서 건드렸던 변경사항을 이전으로 되돌려야 한다.

Margin 적용하기

- SVG와 g의 관계

`<svg>`

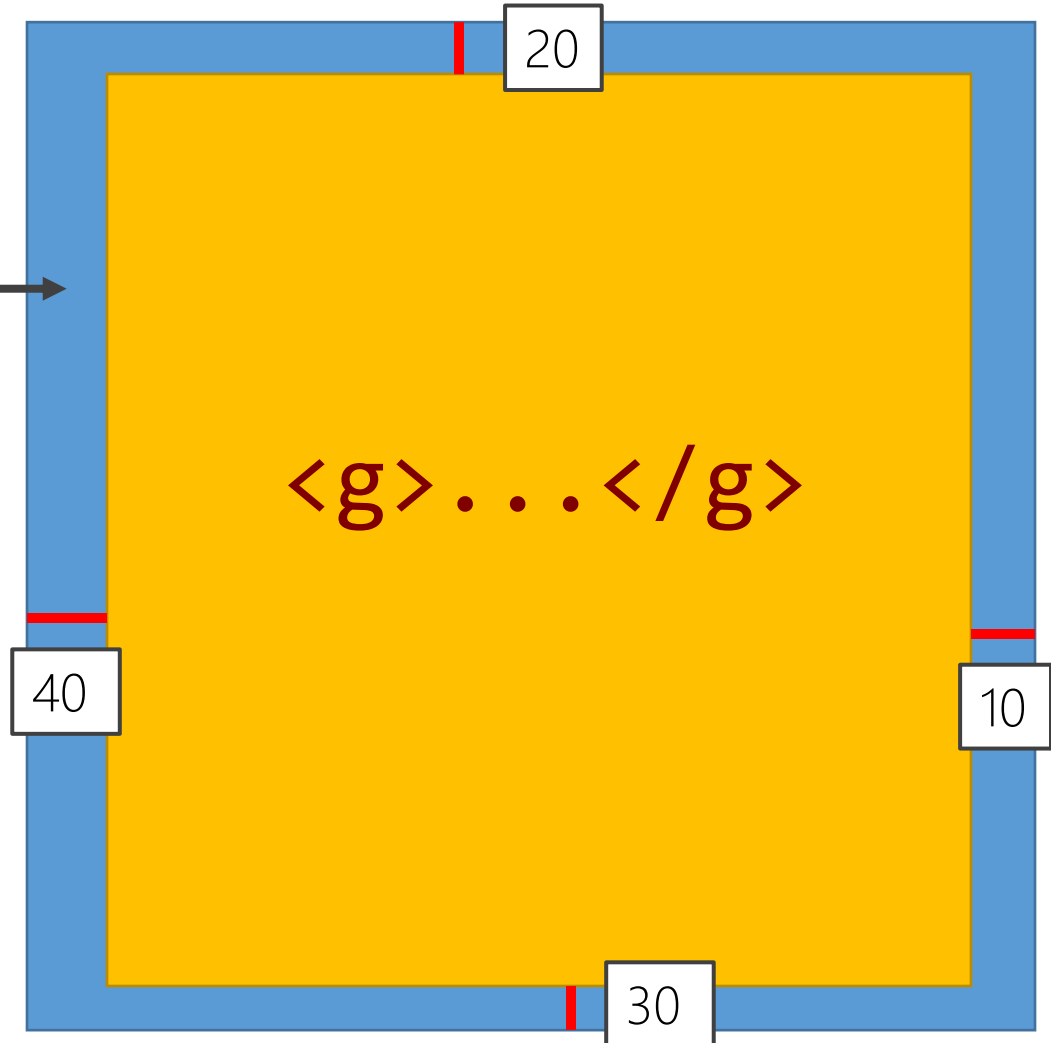
...

`</svg>`



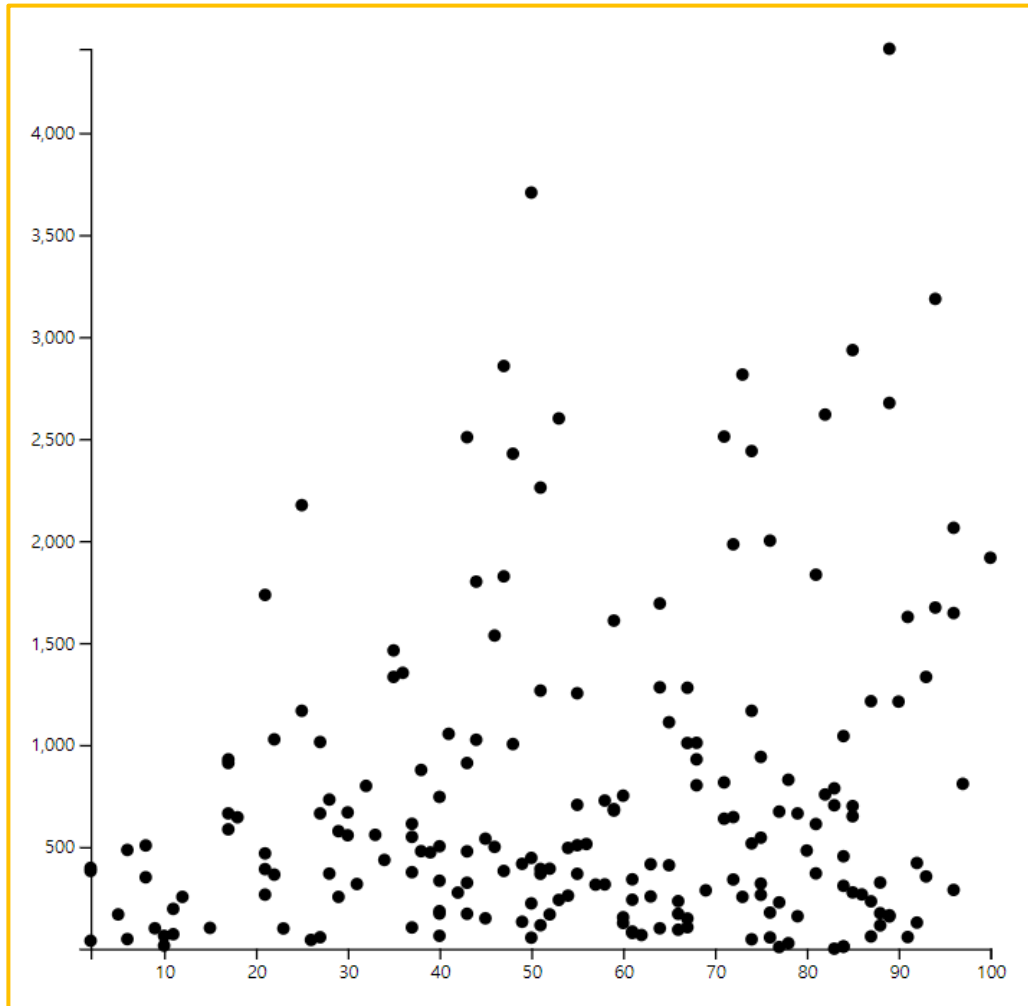
`<g>...</g>`

```
let margin = {  
  top: 20,  
  right: 10,  
  bottom: 30,  
  left: 40  
};
```



Margin 적용하기

- 결과 (<https://pastebin.com/PBb2VGHd>)



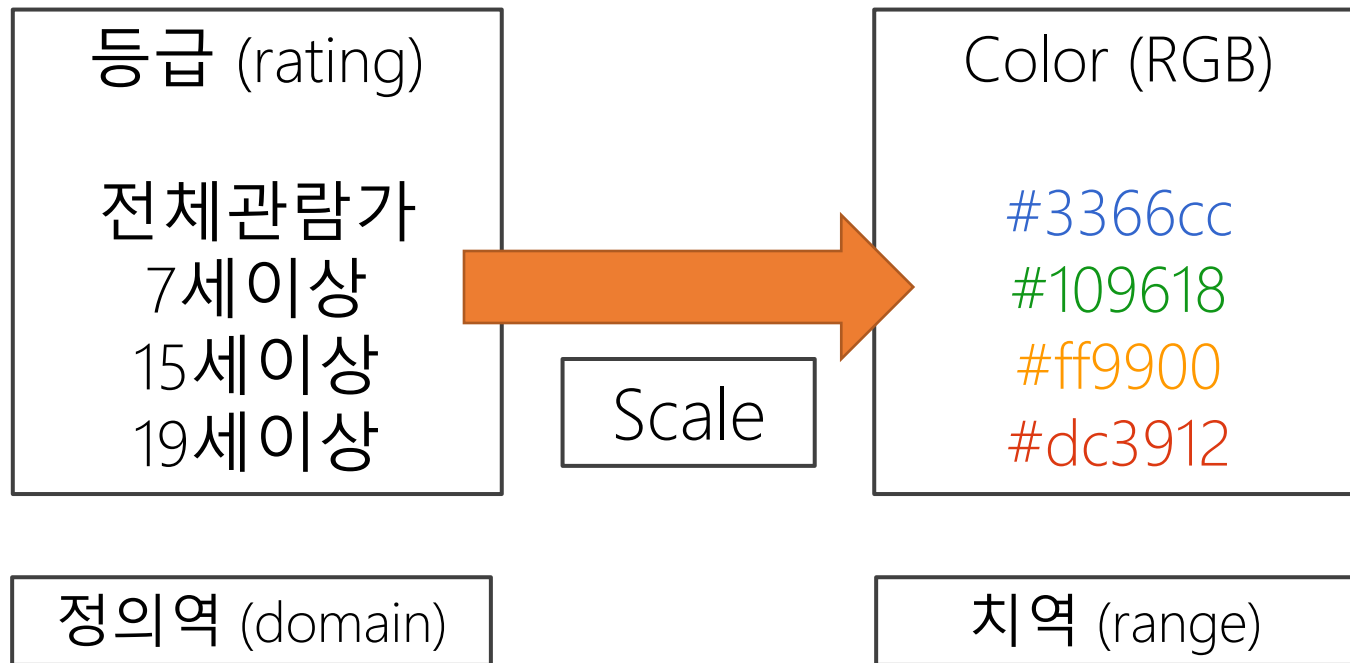
D3.js - Drawing a Scatterplot

Color Scale

Color Scale

- Color Scale?

→ 데이터의 값과 화면 상의 컬러코드를 연결해주는 함수



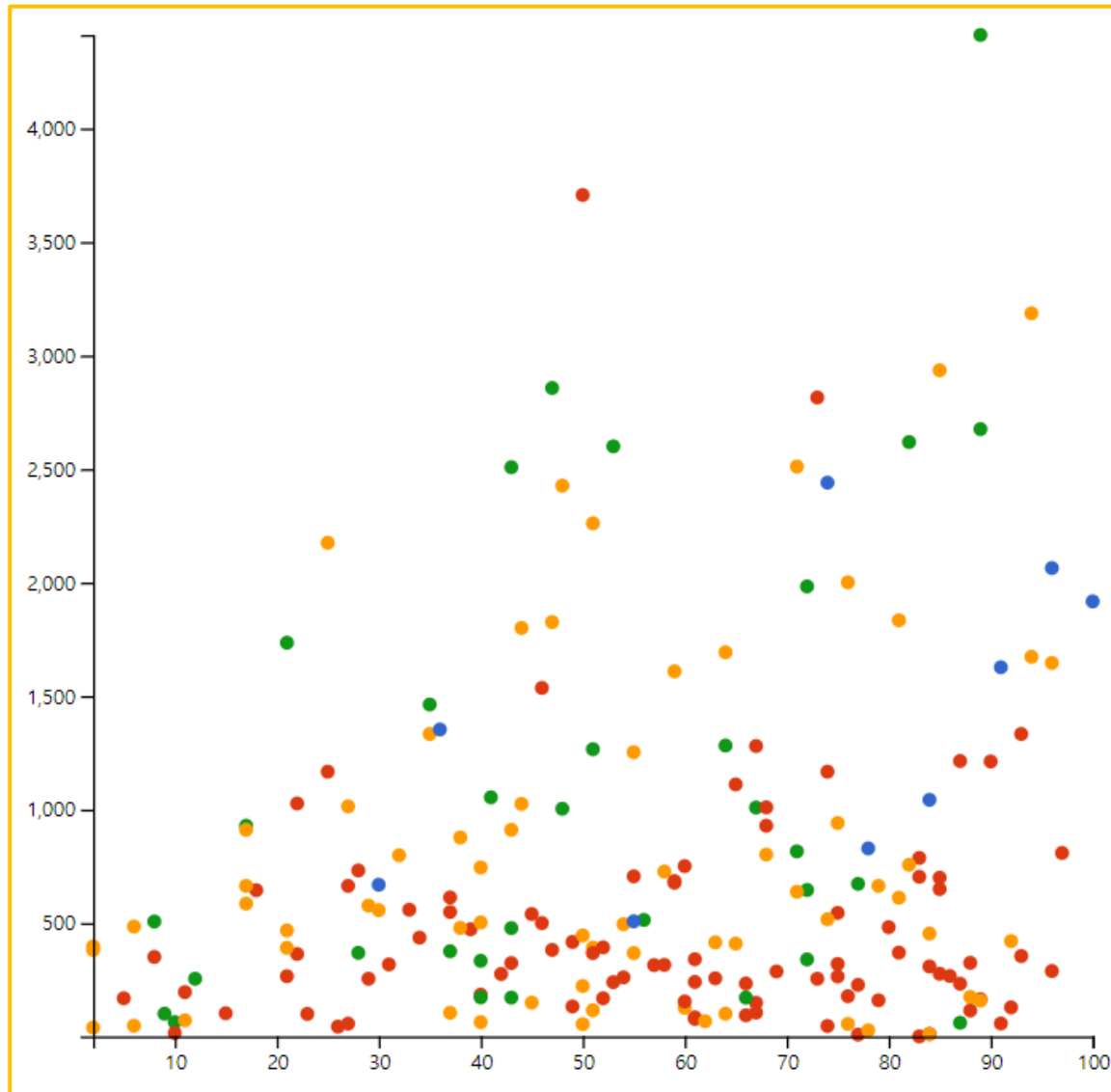
Color Scale 구하기 & 적용하기

```
let color = d3.scaleOrdinal()  
    .domain(['전체관람가', '7세 이상', '15세 이상', '19세 이상'])  
    .range(['#3366cc', '#109618', '#ff9900', '#dc3912']);
```

```
svg  
    .selectAll('circle')  
    .data(data)  
    .enter()  
        .append('circle')  
        .attr('r', 3.5)  
        .attr('cx', d => x(d.rotten_rating))  
        .attr('cy', d => y(d.us_gross))  
        .style('fill', d => color(d.rating));
```

- `d3.scaleOrdinal()` 을 이용

지금까지 그린 산점도



D3.js

Event

Event 등록하기

• 뼈대 코드

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title>Event</title>
</head>
<body>
  <svg height="500">
    <rect transform="translate(100, 100)" width="50" height="50"></rect>
  </svg>
  <script src="https://d3js.org/d3.v6.min.js"></script>
  <script>

    // 여기서부터 작성

  </script>
</body>
```

Event 등록하기

• Event 등록 순서

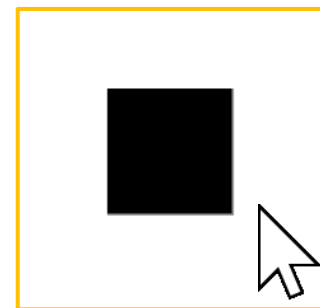
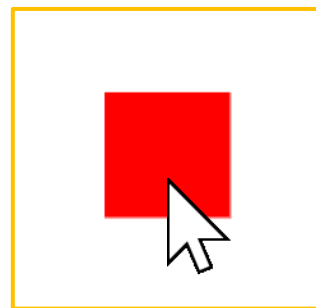
1. Select한다.

`d3.selectAll('type')`

2. Selection에 Event를 연결한다.

`selection.on('event_type', func)`

```
d3.selectAll('rect')  
  .on('mouseenter', function() {  
    d3.select(this)  
      .style('fill', 'red');  
  })  
  .on('mouseleave', function() {  
    d3.select(this)  
      .style('fill', 'black');  
  });
```



Event의 종류

- DOM Event Type
 - https://developer.mozilla.org/en-US/docs/Web/Events#Standard_events

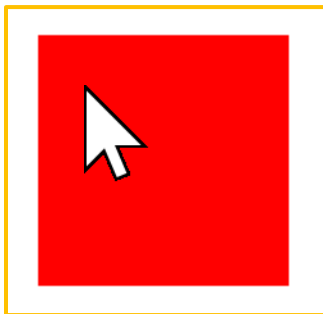
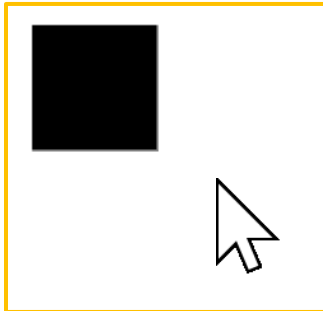
- 자주 쓰이는 이벤트

```
selection
.on('mouseenter', function() { }) // 커서를 올릴 때
.on('mouseleave', function() { }) // 커서를 바깥으로 뺄 때
.on('mousemove', function() { }) // 커서를 올린 상태에서 움직일 때마다
.on('click', function() { }) // 클릭할 때
.on('mouseenter', null)
```

- null 로 지정할 경우 등록된 이벤트가 사라진다.

Multiple Callbacks

- 한 이벤트에 여러 함수 등록
 - 'event_type.name'



```
d3.selectAll('rect')
  .on('mouseenter.e1', function() {
    d3.select(this)
      .style('fill', 'red');
  })
  .on('mouseenter.e2', function() {
    d3.select(this)
      .attr('width', 100)
      .attr('height', 100);
  })
  .on('mouseleave.e1', function() {
    d3.select(this)
      .style('fill', 'black');
  })
  .on('mouseleave.e2', function() {
    d3.select(this)
      .attr('width', 50)
      .attr('height', 50);
  });
```


주의사항

- function으로 정의한 이벤트 핸들러(두 번째 인자) 내부에서 this 키워드는 현재 이벤트가 발생한 요소를 가리킨다.
- 핸들러로 arrow function을 쓰면 this의 의미가 다르므로 제대로 동작하지 않는다(lexical this).

```
selection.on('click', function() {  
  d3.select(this).attr('color', 'red');  
});  
  
selection.on('click', () => {  
  d3.select(this); // wrong!  
});
```

버전 관련 주의사항

- 이벤트 핸들러의 인자는 버전 5와 버전 6에서 사용 문법이 다르다!

- <https://observablehq.com/@d3/d3v6-migration-guide#events>

- 버전 5의 코드

```
selection.on('click', function(d, i, nodes) {  
  console.log(d, i, nodes);  
});
```



d: 현재 요소에 조인된 데이터
i: 현재 요소의 *selection* 상의 인덱스
nodes: *selection*이 가지고 있는 요소들의 배열

- 버전 6에서 똑같이 동작하게 바꾸려면... (function 사용)


```
selection.on('click', function(event, d) {  
  const nodes = selection.nodes();  
  const i = nodes.indexOf(this);  
  console.log(d, i, nodes);  
});
```

event: 일어난 이벤트 정보
d: 현재 요소에 조인된 데이터

버전 관련 주의사항

- 버전 5에서는 이벤트 핸들러로 arrow function을 사용할 때 this의 기능을 대신하는 코드를 다음과 같이 짤 수 있었다.

```
selection.on('click', (d, i, nodes) => {  
  console.log(nodes[i]);  
});
```



- 버전 6에서는 이벤트 핸들러로 arrow function을 사용할 때 this의 기능을 대신하는 코드를 다음과 같이 짤 수 있다.

```
selection.on('click', (event, d) => {  
  console.log(event.currentTarget);  
});
```

D3.js

Transition

Transition 사용하기

- Transition: 시각적 요소에 애니메이션을 주는 것
- Transition을 통해 할 수 있는 것
 - 어떤 속성을
 - 어떤 값(상태)으로
 - 어떻게(몇 초 후에, 몇 초 동안) 바꿀지

Transition 사용하기

- 간단한 예제 (출처: <https://bl.ocks.org/d3noob/c3cbb8af554eb848d09ab97306bb5583>)

```
<!DOCTYPE html>
<meta charset="utf-8">
<body>
<script src="https://d3js.org/d3.v6.min.js"></script>
<script>
var svg = d3.select("body") // Select the body element
    .append("svg")           // Append an SVG element to the body
    .attr("width", 960)       // make it 960 pixels wide
    .attr("height", 500)      // make it 500 pixels high
    .append("circle")         // append a circle to the svg
    .attr("fill", "blue")     // fill the circle with 'blue'
    .attr("r", 20)            // set the radius to 10 pixels
    .attr('cx', 40)           // position the circle at 40 on the x axis
    .attr('cy', 250)          // position the circle at 250 on the y axis
    .transition()             // apply a transition
    .duration(4000)           // apply it over 4000 milliseconds
    .attr('cx', 920);         // new horizontal position at 920 on x axis
</script>
</body>
```

실습 – Circle에 간단한 Transition 적용하기

```
<body>
  <svg height='600' width='600'></svg>
  <script src="https://d3js.org/d3.v6.min.js"></script>
  <script>
    let svg = d3.select('svg');
    let circle = svg.append('circle')
      .attr('cx', 50)
      .attr('cy', 50)
      .attr('fill', 'red')
      .attr('r', 10);

    circle.transition()
      .delay(500)
      .duration(3000)
      .attr('cx', 300)
      .attr('cy', 300)
      .attr('r', 100)
      .attr('fill', 'black');
  </script>
</body>
```

.delay(500) : transition 시작 전 0.5초 대기

.duration(3000) : 3.0초 동안 transition

오늘 배운 것

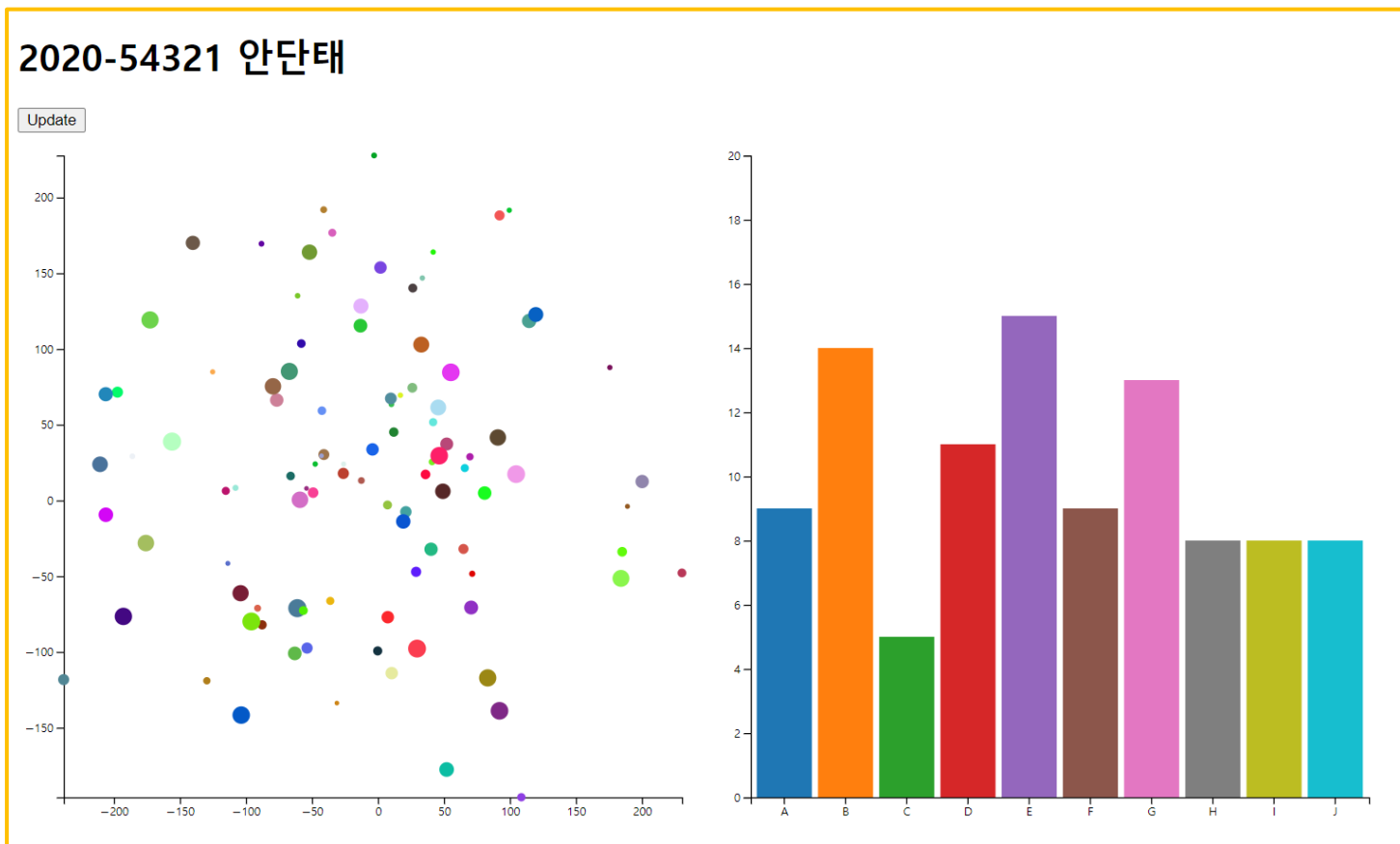
- Selection: `d3.select` `d3.selectAll`
`selection.select` `selection.selectAll`
- Data join: `selection.data` `selection.enter` `selection.exit`
`selection.merge`
- Drawing: `selection.append` `selection.attr` `selection.style`
- Scale: `selection.scaleLinear` `selection.scaleOrdinal`
- Axis: `selection.axisLeft` `selection.axisBottom` `selection.call`
- Margin / Color
- Event: `selection.on`
- Transition: `selection.transition` `selection.delay` `selection.duration`

D3.js

Homework 1

과제 개요

- 산점도와 바 차트를 그려보자.
- 과제 의도: JavaScript와 d3.js를 사용하기 위한 전반적인 환경 연습

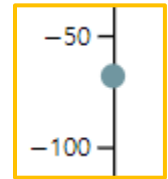


필수 스펙 - 데이터

- 8가지 키를 갖는 오브젝트를 **100개** 생성하여 데이터 배열을 만든다.
 - `"id", "cx", "cy", "radius", "red", "green", "blue", "class"`
 - `"id"` 는 이 오브젝트만의 **고유한 정수** 값이다. 새로 생성한 오브젝트의 `"id"` 는 언제나 기존의 모든 오브젝트의 `"id"` 보다 **더 커야 한다**.
 - `"cx", "cy"` 는 각각 평균이 0, 표준편차가 100인 Gaussian 분포를 따르는 무작위 **실수** 값이다.
 - `"radius"` 는 2 이상 8 미만의 uniform 분포를 따르는 무작위 **실수** 값이다.
 - `"red", "green", "blue"` 는 각각 0 이상 255 **이하**의 uniform 분포를 따르는 무작위 **정수** 값이다.
 - `"class"` 는 A ~ J(10가지)의 uniform 분포를 따르는 무작위 **문자열** 값이다.
- **힌트:** `Math.random()` 또는 d3-random을 이용한다.

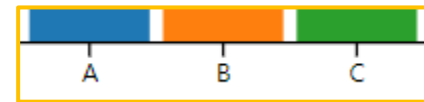
필수 스펙 - 산점도 (Scatterplot)

- 데이터의 모든 오브젝트(100개)를 산점도로 그린다.
 - `"cx"`, `"cy"` 는 원의 중심 좌표, `"radius"` 는 원의 반지름, `"red"`, `"green"`, `"blue"` 는 원의 색깔(RGB 코드)에 대응된다.
 - x축과 y축을 데이터의 `"cx"`, `"cy"` 의 최솟값 ~ 최댓값 범위에 맞게 그린다.
 - 작은 값일수록 왼쪽 하단에 위치해야 한다.
 - 점이 축에 의해 가려지지 않게, 점이 축보다 더 앞으로 나와야 한다.
- 점(원) 위에 마우스를 올리면 즉시 점의 색깔이 `rgb(0, 0, 0)`으로 변하고 반지름이 원래 반지름의 **2배**로 늘어나야 한다. (visual feedback 제공)
- 점 밖으로 마우스를 빼면 즉시 점이 원래 상태로 돌아가야 한다.
 - Update 기능으로 점이 transition하는 동안 마우스를 점 위에 올리거나 점 밖으로 빼는 경우는 고려하지 않는다.
- 축이나 점이 SVG 경계에서 잘리지 않도록 한다.



필수 스펙 – 바 차트 (Bar chart)

- 데이터에서 `"class"` 에 따른 오브젝트의 개수를 세로 바 차트로 그린다.
 - A ~ J의 각 `"class"` 는 막대의 x좌표,
해당 `"class"` 에 속하는 오브젝트의 개수는 막대의 높이에 대응된다.
 - **힌트:** `"class"` 개수(10개)만큼의 길이를 갖는 새로운 배열을 만든다.
 - x축을 항상 10개의 범주(A, B, C, D, E, F, G, H, I, J)를 갖도록 그린다.
 - x축의 모든 눈금의 x좌표는 각 막대의 중심의 x좌표와 일치해야 한다.
 - y축을 0 이상 $\text{Max}(18, \text{Max}(\text{각 } \code{"class"} \text{에 속하는 오브젝트의 개수}))$ 이하의 범위를 갖도록 그린다.
 - 작은 값일수록 하단에 위치해야 한다.
 - 이웃한 두 막대 사이에는 약간의 공백이 있어야 한다.
 - **힌트:** D3에서 범주형 데이터를 다루는 scale을 찾아본다.
 - 막대의 색깔은 `"class"` 에 따라 서로 다른 10개의 색상을 지정한다.
 - **힌트:** `d3.schemeCategory10` 을 이용한다.
 - 축이나 막대가 SVG 경계에서 잘리지 않도록 한다.



필수 스펙 – Update 기능

- “Update” 버튼을 누르면 다음을 모두 수행한다.
 - 데이터
 - `"id"`가 가장 작은 오브젝트 **25개 제거**
 - 남은 오브젝트들의 다음 키의 값을, 각각의 생성 시 분포를 따르도록 무작위로 **변경**
→ `"cx", "cy", "radius", "red", "green", "blue"`
 - 새 오브젝트 **25개**를 **생성**하여 데이터에 추가
 - 산점도
 - 제거된 오브젝트의 점들 즉시 제거
 - 변경된 오브젝트의 점들, x축, y축이 **3초** 동안 동시에 transition하면서 변화
 - 위의 transition이 끝난 후, 추가된 오브젝트의 점들이 **1초** 동안 fade in transition하면서 등장
 - **힌트:** `selection.style("opacity", ...)` 을 이용한다.
 - 바 차트
 - 산점도에서 transition하기 시작할 때, 동시에 막대 높이, y축이 **2초** 동안 transition하면서 변화
 - Transition 진행 중에는 Update 버튼을 누를 수 없게 **비활성화**하고, 끝나면 활성화한다.
 - **힌트:** 버튼에 `"disabled"` attribute가 있으면 비활성화, 없으면 활성화된다.

필수 스펙 - 기타

- 두 차트를 하나의 HTML 페이지에 구현한다.
 - 페이지 맨 위에 학번과 이름을 표시한다.
 - 학번보다 아래에, 두 차트보다 위에 Update 버튼이 하나 놓여 있어야 한다.
 - 왼쪽에 산점도, 오른쪽에 바 차트를 수평으로 배치한다.
 - 가능하면 페이지 전체를 1295px * 864px (한 화면) 안에 표시하도록 한다.
 - 스크롤바가 생기지 않도록
- D3 버전은 6을 사용해야 한다.
 - `<script src="https://d3js.org/d3.v6.min.js"></script>`
- 여기에 명시되지 않은 스펙은 자유롭게 정해도 무방하다.
 - 단, 조교가 채점할 수 있어야 한다.

제출 방법

- 하나의 .html 파일만 eTL을 통해 제출
 - .js, .json 파일을 비롯한 다른 파일은 포함시키지 말 것
 - D3.js도 웹에서 불러오도록 할 것
 - 압축하지 말 것
- 채점은 최신 Chrome 브라우저에서 진행
- 스펙 관련 문의는 eTL의 질의응답 게시판을 이용
- 제출 기한: ~ 10월 14일 (수) 23:59