

Information Visualization

JavaScript

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

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

Human-Computer Interaction Laboratory
Seoul National University

일정

- Developing an interactive InfoVis interface using web technologies
- 일정
 - 9월 21일 월: JavaScript
 - 10월 5일 월: D3.js 1 (+ 과제 1)
 - 10월 7일 수: D3.js 2 (+ 과제 2)
- 기말 프로젝트 디자인 및 개발

조원 구성

- 10월 5일까지 조원 4명을 구성하여 eTL에 제출 (방법은 eTL 참고)
- 더 적은 인원 (3명 이하)으로 하고 싶으시면 그렇게 하셔도 됩니다
 - 다만 평가 기준은 4인 조와 동일하게 적용됩니다
- 조/조원을 구하기 어려워요...
 - 아래의 글에서 찾으세요!
 - <http://etl.snu.ac.kr/mod/ubboard/article.php?id=1146391&bid=495551&bwid=2214173>

HTML, CSS, JavaScript

Web Technology

Client Technologies

- HTML: HyperText Markup Language
 - 문서의 구조
 - HTML 5
- CSS: Cascading Style Sheets
 - 문서의 디자인 (표현)
 - CSS 3
- Javascript
 - 문서의 기능
 - ES 6 (ECMAScript 2015)

Client Technologies

- HTML: HyperText Markup Language
 - 문서의 구조
 - HTML 5
- CSS: Cascading Style Sheets
 - 문서의 디자인 (표현)
 - CSS 3
- Javascript
 - 문서의 기능
 - ES 6 (ECMAScript 2015)

Learning HTML & CSS

- 영어 문서
 - <https://www.w3schools.com/html>
 - <https://learn.shayhowe.com/html-css/building-your-first-web-page/>
- 한글 문서
 - <https://opentutorials.org/course/2039>

HyperText Markup Language
Scalable Vector Graphics

HTML & SVG

HyperText Markup Language (HTML)

```
1  <!DOCTYPE html>
2  ▾ <html>
3  ▾ <head>
4  ▾   <title>Animals Around the World</title>
5     </head>
6  ▾ <body>
7  ▾   <h1>The Brown Bear</h1>
8     <!-- A section that describes the brown bear -->
9  ▾   <p>The brown bear (Ursus arctos) is native to parts of northern Eurasia and North
      America. </p>
10 ▾   <a href="https://en.wikipedia.org/wiki/Brown_bear">Learn More</a>
11 ▾   <p>Here are some bear species:</p>
12 ▾   <ul>
13 ▾     <li>Arctos</li>
14 ▾     <li>Collarus</li>
15     </ul>
16 ▾   <p>The following countries have the largest populations of brown bears:</p>
17 ▾   <ol>
18 ▾     <li>Russia</li>
19     </ol>
20 ▾   <a href="#" target="_blank">
21     </a>
22   </body>
23 </html>
```

태그

- HTML에서 구조를 나타내기 위한 기본 단위
 - `<p>나의 문단</p>`
 - ``
- HTML5에는 수많은 태그들이 존재
 - <https://www.w3schools.com/tags/default.asp>
 - 태그마다 의미와 사용법이 다름

SVG

- SVG (Scalable Vector Graphics)



rect (x, y, width, height)



circle (cx, cy, r)



ellipse (cx, cy, rx, ry)



line (x1, y1, x2, y2)



path (d)



polygon (points)

SVG로 간단한 도형 그리기

- <body> </body> 사이에 아래 내용을 입력하고 결과를 확인합니다.

```
<body>
  <svg>
    <circle cx="20" cy="30" r="10" fill="red" />
    <rect x="40" y="60" width="20" height="40" fill="blue" />
  </svg>
</body>
```



SVG로 간단한 도형 그리기

- <body> </body> 사이에 아래 내용을 입력하고 결과를 확인합니다.

<pre><body> <svg> <circle cx="20" cy="30" r="10" fill="red" /> <rect x="40" y="60" width="20" height="40" fill="blue" /> </svg> </body></pre>	중심 x좌표	중심 y좌표	반지름	색
	왼쪽 위 꼭지점 x좌표	왼쪽 위 꼭지점 y좌표	가로 길이	세로 길이



SVG로 간단한 도형 그리기

- HTML로는 정적인 시각화만을 만들 수 있습니다.
- JavaScript를 이용하여 DOM에 시각적 요소를 추가하고 수정함으로써 사용자 입력에 반응하는 시각화를 만들 수 있습니다.
 - Interactive Visual Analytics

```
<body>  
  <svg>  
    <circle cx="20" cy="30" r="10" fill="red" />  
    <rect x="40" y="60" width="20" height="40" fill="blue" />  
  </svg>  
</body>
```



JavaScript

JavaScript

JavaScript란?

- HTML 문서를 interactive하게 바꿀 수 있는 경량의 프로그래밍 언어가 넷스케이프사에 의해 1995년 개발됨
 - Mocha → LiveScript → JavaScript
- Java랑 딱히 상관 없음
- 표준화 움직임 → ECMAScript
- 브라우저에 탑재되어 있는 자바스크립트 엔진이 해석 및 실행
 - 구글 크롬: V8 Engine



JS가 할 수 있는 일

- HTML Content 및 Attribute 변경

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML
content.</p>

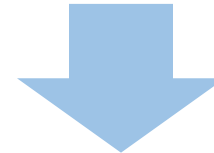
<button type="button"
onclick='document.getElementById("demo").innerH
TML = "Hello JavaScript!";'>Click Me!</button>

</body>
</html>
```

What Can JavaScript Do?

JavaScript can change HTML content.

Click Me!



What Can JavaScript Do?

Hello JavaScript!

Click Me!

JS가 할 수 있는 일

- CSS Style 변경

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change the style of
an HTML element.</p>

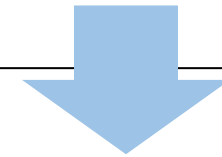
<button type="button"
onclick="document.getElementById('demo').style.
fontSize='35px'">Click Me!</button>

</body>
</html>
```

What Can JavaScript Do?

JavaScript can change the style of an HTML element.

Click Me!



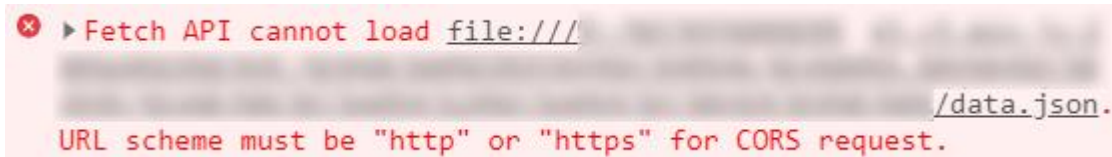
What Can JavaScript Do?

JavaScript can change the style of an HTML element.

Click Me!

간단하게 자바스크립트 코드 작성하기

- 방법 1: 로컬에 .html 또는 .js 파일을 만들어서 웹 브라우저에서 실행
(긴 코드 작성할 때 추천)
 1. “파일이름.html” 파일 생성
 2. 이 파일을 VS Code 또는 텍스트 편집기로 열기
 3. 코드 작성 후 저장
 4. 이 파일을 Chrome 브라우저로 열어서 실행
- 만약 브라우저에서 F12 키를 눌렀을 때 [Console]에 이런 메시지가 뜬다면?



→ 세 번째 실습(D3.js 2) PPT 참조

간단하게 자바스크립트 코드 작성하기

- 방법 2: 웹 에디터 활용
 1. <https://jsbin.com> 또는 <https://jsfiddle.net> 접속
 2. html, css, js 칸에 각각 해당 코드 작성
 3. Ctrl + Enter 키 입력하여 실행
- 방법 3: 개발자 도구의 console에서 REPL 형태로 간단하게 테스트
 1. Chrome 브라우저 실행
 2. F12 키 입력 후 [Sources] – [Snippets]로 이동
 3. [+ New snippet] 버튼 클릭
 4. 오른쪽 빈 공간에 코드 작성
 5. Ctrl + Enter 키 입력하여 실행

JavaScript

Variable, Type, Operator

변수 할당

- 자바스크립트에서는 `let` 키워드를 사용하여 변수에 값을 할당합니다.
- `let (변수 이름) = (값);`

```
let x;  
let y = 2.5;  
let Condition = false;  
let under_bar_string = 'string';  
let variable1234 = null;
```

콘솔에 변수 출력

- console.log를 이용해 변수의 값을 콘솔에 출력할 수 있습니다.
- console.log(값 또는 변수 이름);

```
let x;  
let y = 2.5;  
let Condition = false;  
let under_bar_string = 'string';  
let variable1234 = null;  
  
console.log(x);  
console.log(y, Condition);  
console.log(under_bar_string);  
console.log(variable1234);
```

undefined
2.5 false
string
null

제가 산 책에서는 let이 아니라 var로 되어 있던데요?

- `var x = 1;`
 - 이제까지 써 오던 변수 정의 방식
 - `let x = 1;`
 - 새로 도입된 변수 정의 방식
-
- let과 var은 둘 다 변수를 선언하고 값을 할당합니다.
 - var의 경우 예전부터 쓰던 방식이지만, 다른 프로그래밍 언어에서의 변수 선언과 다른 scope를 가지고 있어서 헷갈릴 수 있습니다.
 - 실습에서는 모두 let을 쓰기로 합니다. 저도 let만 씁니다.

Primitive Data Type

- Number (숫자)
 - 정수, 실수 모두 가능
- String (문자열)
 - Single (' ') or double (" ") quote 사용
 - ' '과 " "에는 차이가 없음
- Boolean (논리 값)
 - 참 혹은 거짓

```
let y = 2.5;  
y = 10;
```

```
let squote = 'single';  
let dquote = "double";  
  
// 'single'이라는 문자열  
let sind = "'single'";  
let sins = '\\single\\';
```

```
let condition1 = false;
```

Primitive Data Type

- null
 - 아무런 값도 나타내지 않는 특수한 키워드

```
let empty = null;
```

- undefined (정의 되지 않음)

```
let y;
```

연산자들

- 사칙연산 (+, -, *, /, %, ++, --)
- 대입 (=, +=, -=, *=, /=, %=
 - `a = 123;`
- 비교연산 (<, <=, >, >=, ==, !=)
- 논리 연산 (&& (and), || (or), ! (not))

변수와 연산자 사용해보기

```
console.log('사칙연산');
```

```
let x;
```

```
x = 1;
```

```
x++;
```

```
console.log(x);
```

```
let y = 1.5;
```

```
x += y;
```

```
console.log(x);
```

```
console.log(10 / 3);
```

```
console.log(10 % 3);
```

```
console.log('비교 및 논리 연산');
```

```
console.log(5 < 10);
```

```
console.log(5 >= 10);
```

사칙연산

2

3.5

3.3333333333333335

1

비교 및 논리 연산

true

false

변수와 연산자 사용해보기

```

console.log('사칙연산');

let x;

x = 1;
x++;
console.log(x);

let y = 1.5;
x += y;
console.log(x);

console.log(10 / 3);
console.log(10 % 3);

console.log('비교 및 논리 연산');

console.log(5 < 10);
console.log(5 >= 10);

console.log(5 < 10 && 3 < 5);
console.log(5 < 10 && 2 > 4);
console.log(5 < 10 || 2 > 4);
console.log(true && false);
console.log(!true);

console.log('문자열 연산');

let str1 = "문자열1";
let str2 = "문자열2";

str1 += str2;
console.log(str1);
console.log(str1 + " 한번 더?");

```

사칙연산
2
3.5
3.3333333333333335
1
비교 및 논리 연산
true
false
true
false
false
문자열 연산
문자열1문자열2
문자열1문자열2 한번 더?

파이썬에서 같음을 확인할 때 `==`이 아니라 `==`을 쓰던데요?

```
let x = 1;

console.log(x == 1); // true
console.log(x === 1); // true

console.log(x == true); // true
console.log(x === true); // false

console.log(0 == []); // true
console.log('' == 0); // true
console.log(' ' == 0); // true
console.log('' == ' '); // false

console.log(0 === []); // false
console.log('' === 0); // false
console.log(' ' === 0); // false
console.log('' === ' '); // false
```

- `==`은 타입을 확인하지 않습니다. 엄밀히 할 때에는 `===`과 `!==`을 쓰도록 합시다!

JavaScript

Conditional & Loop Statement

조건문

• if-else 문

```
if (조건) { 코드; }
```

```
else if (조건) { 코드; }
```

```
else { 코드; }
```

• 코드가 한 문장이라면 중괄호 생략 가능

• 논리곱(and) / 논리합(or) / 논리부정(!) 연산자

```
if (isHero && isHulk) {  
    // 둘 다 참일 경우 실행  
}
```

```
if (isHero || isHulk) {  
    // 하나라도 참일 경우 실행  
}
```

```
if (!isHero) {  
    // 영웅이 아니면 실행  
}
```


반복문

- for loop을 사용해서 반복할 수 있다.
- for (처음실행; 반복조건; 반복 후 실행) { 반복되는 코드 }

```
let x;  
for (x = 0; x < 10; x++) {  
  console.log(x);  
}
```

0
1
2
3
4
5
6
7
8
9

실습 – 구구단 7단 출력하기

- number와 string을 더하게 되면
number를 string으로 자동으로 바꾼 후 두 개의 string을 이어 붙인다.

```
console.log("----7단----");  
  
for (let i = 1; i <= 9; i++) {  
    console.log("7*" + i + "=" + 7 * i);  
}
```

```
----7단----  
7*1=7  
7*2=14  
7*3=21  
7*4=28  
7*5=35  
7*6=42  
7*7=49  
7*8=56  
7*9=63
```

실습 - 2중 for문으로 별 출력하기

- for문을 사용하여 Console에 다음의 별들을 출력한다.

```
let n = 5;  
for (let i = 0; i < n; i++) {  
  let str = '';  
  for (let j = 0; j <= i; j++) {  
    str += '*';  
  }  
  console.log(str);  
}
```

```
*  
**  
***  
****  
*****
```

JavaScript

Array & Object

배열(array)과 오브젝트(object)

- 배열: 값들의 리스트
- 오브젝트: (키, 값) 의 묶음
- 각각 []와 {}로 초기화할 수 있다.
 - 배열 리터럴, 오브젝트 리터럴 (literal)

실습 - 배열 기초

```
let empty_array = [];  
let array = [1, 2, 3, 4];
```

```
console.log(array);  
console.log(array[0]);  
console.log(array.length);  
console.log(array[4]);
```

```
array[2] = -1;  
array.push(5);
```

```
console.log(array);  
console.log(array.length);
```

```
for (let i = 0; i < array.length; i++) {  
  console.log(array[i]);  
}  
let array2 = [1, 1.5, false, 'string', []];  
console.log(array2);
```

▶ (4) [1, 2, 3, 4]

1

4

undefined

▶ (5) [1, 2, -1, 4, 5]

5

1

2

-1

4

5

▶ (5) [1, 1.5, false, "string", Array(0)]

실습 - 오브젝트 기초

```
let empty_object = {};  
let obj = {x: 10, y: 10.5};  
  
console.log(obj); // {x: 10, y: 10.5}  
console.log(obj.x); // 10  
console.log(obj.y); // 10.5  
  
obj.x = -1;  
obj.z = 100;  
delete obj.y;  
  
for (let key in obj) {  
    console.log(key, obj[key]);  
}  
// "x" -1  
// "z" 100  
  
let my_key = 'x';  
console.log(obj[my_key]); // -1;  
  
my_key = 'z';  
console.log(obj[my_key]); // 100;
```

배열과 오브젝트, 어디에 쓸까?

- 오브젝트

- 키와 값을 매핑할 수 있음
- 현실에서 어떤 객체를 정의하는데 사용할 수 있음
 - 가령, 수강생

```
let john = {  
  name: 'John',  
  birthday: '11/02',  
  gender: 'male'  
}
```

- 배열

- 값들의 순서 있는 집합
- 오브젝트의 배열을 만들어서 객체들의 집합을 나타낼 수 있음
 - 가령, 수강생들

전개(spread) 구문

- ECMAScript 2018 문법 (최신 브라우저에서 동작)
- 이미 존재하는 배열을 일부로 하는 새 배열을 만드는 방법은?

```
let parts = ['shoulders', 'knees'];  
let lyrics = ['head', ...parts, 'and', 'toes'];  
// ["head", "shoulders", "knees", "and", "toes"]
```

- 오브젝트를 복사하는 방법은?

```
let obj = { x: 10, y: 10.5 };  
let clonedObj = { ...obj }; // {x: 10, y: 10.5}
```

- 오브젝트의 일부 키의 값만 수정하거나 새 키를 추가하는 방법은?

```
let obj1 = { x: 10, y: 10.5, z: "hello" };  
let obj2 = { x: 9, z: "Hi", w: true };  
let mergedObj = { ...obj1, ...obj2 };  
// {x: 9, y: 10.5, z: "Hi", w: true}
```

- https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/Spread_syntax

JavaScript Object Notation

JSON

JavaScript Object Notation (JSON)

- 현대 웹에서 데이터를 나타내기 위해 **가장** 널리 쓰이는 포맷
 - 자바스크립트의 오브젝트, 배열 문법과 굉장히 흡사

```
{  
  "employees": [  
    {"firstName": "John", "lastName": "Doe"},  
    {"firstName": "Anna", "lastName": "Smith"},  
    {"firstName": "Peter", "lastName": "Jones"}  
  ],  
  "last_updated": "2020/03/01"  
}
```

JSON 사용하기

```
let server_response = '{ \
  "employees":[ \
    {"firstName":"John", "lastName":"Doe"}, \
    {"firstName":"Anna", "lastName":"Smith"}, \
    {"firstName":"Peter", "lastName":"Jones"} \
  ], \
  "last_updated":"2020/03/01" \
}';

console.log(server_response);

console.log('parsing');

let data = JSON.parse(server_response);
console.log(data);

let students = [
  {id: 1, name: 'Sally'},
  {id: 2, name: 'John'}
];

console.log(JSON.stringify(students));
```

- `JSON.parse(string)` : 로컬 파일/서버 응답에서 온 JSON 텍스트를 메모리 상의 오브젝트로 바꾸기
- `JSON.stringify(object)` : 메모리 상의 오브젝트를 텍스트로 바꾸기 (서버 전송을 위해)

JavaScript

Function

함수 (Function)

- 특정 일을 하기 위해 묶여진 코드 조각
 - 최솟값을 구하는 함수
 - 평균을 구하는 함수
 - 서버에서 데이터를 받아오는 함수
 - 차트를 그리는 함수
- 라이브러리: 특정 목적을 위해 필요한 함수들을 모아서 재사용 가능하게 해 둔 것
 - 예) D3.js: 차트를 그리는 라이브러리

실습 - 아주 간단한 함수 만들어 보기


- 아래 코드를 입력한 다음 결과를 확인해 봅시다.
- <https://jsbin.com/?js,console>

```
function sum(a, b) {  
  return a + b;  
}  
  
console.log(sum(1, 3)); // 4  
  
let a = 3, b = 5;  
  
console.log(sum(a, b)); // 8
```

- 더하기 한 번만 하면 되는 걸 굳이?

함수 뜯어보기


함수 선언임을 표시



```
function mean(a, b) {  
  let m;  
  
  m = (a + b) / 2;  
  
  return m;  
}
```


함수 뜯어보기


함수의 이름 (mean)



```
function mean(a, b) {  
  let m;  
  
  m = (a + b) / 2;  
  
  return m;  
}
```

함수 뜯어보기

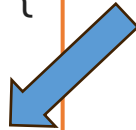
함수의 인자 (a와 b)



```
function mean(a, b) {  
  let m;  
  
  m = (a + b) / 2;  
  
  return m;  
}
```

함수 뜯어보기

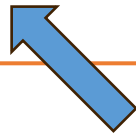
```
function mean(a, b) {  
  let m;  
  
  m = (a + b) / 2;  
  
  return m;  
}
```



호출되었을 시 실행되는 코드 (바디)

함수 뜯어보기

```
function mean(a, b) {  
  let m;  
  
  m = (a + b) / 2;  
  
  return m;  
}
```



결과 값

함수 뜯어보기

```
function mean(a, b) {  
  let m;  
  
  m = (a + b) / 2;  
  
  return m;  
}
```

```
let v = mean(10, 5);
```



함수이름(인자1, 인자2, ...)로 호출

실습 - 간단한 함수 만들어 보기

```
function sum(a, b) {  
    return a + b;  
}  
  
function mean2(a, b) {  
    return sum(a, b) / 2;  
}  
  
function arrayMean(arr) {  
    let sum = 0;  
    for (let i = 0; i < arr.length; i++)  
        sum += arr[i];  
  
    return sum / arr.length;  
}  
  
let a = 10, b = 5;  
console.log(sum(a, b));    // 15  
console.log(mean2(a, b)); // 7.5  
  
let arr = [a, b, 2];  
console.log(arrayMean(arr)); // 5.666666667
```

리턴 값이 없는 함수와 값에 의한 호출 (call by value)

- 논리 값, 수, 문자열의 경우 인자로 넘겨질 때 새로 복사된다.
 - 함수 내에서 값을 바꿔도 외부에는 변화가 없음

```
function increment(a) {  
  a++;  
}  
  
increment(3);  
console.log(increment(3));  
  
let c = 1;  
increment(c);  
console.log(c);
```

출력 결과:
3? 4? undefined?

출력 결과:
1? 2? undefined?

리턴 값이 없는 함수와 값에 의한 호출 (call by value)

- 논리 값, 수, 문자열의 경우 인자로 넘겨질 때 새로 복사된다.
 - 함수 내에서 값을 바꿔도 외부에는 변화가 없음

```
function increment(a) {  
  a++;  
}  
  
increment(3);  
console.log(increment(3));  
  
let c = 1;  
increment(c);  
console.log(c);
```

출력 결과:
3? 4? undefined?

출력 결과:
1? 2? undefined?

undefined

1

참조에 의한 호출 (call by reference)

- 배열, 오브젝트의 경우 인자로 넘겨질 때 원본의 **참조**가 넘어간다.
 - 함수 내에서 값을 바꾸면 외부에서도 변화가 생김

```
function append(array, value) {  
  array.push(value);  
}
```

```
let arr = [1, 2];  
append(arr, 3);  
console.log(arr);
```

출력 결과:

[1, 2]?

[1, 2, 3]?

undefined?

참조에 의한 호출 (call by reference)

- 배열, 오브젝트의 경우 인자로 넘겨질 때 원본의 **참조**가 넘어간다.
- 함수 내에서 값을 바꾸면 외부에서도 변화가 생김

```
function append(array, value) {  
  array.push(value);  
}
```

```
let arr = [1, 2];  
append(arr, 3);  
console.log(arr);
```

출력 결과:

[1, 2]?

[1, 2, 3]?

undefined?

▶ (3) [1, 2, 3]

함수와 오브젝트의 차이?

- 함수: 호출이 가능한 오브젝트 (callable object)

```
function my_func() { console.log(1); }  
// let my_func = {};  
  
my_func.x = -1;  
my_func.y = 100;  
  
console.log(my_func.x); // -1  
  
delete my_func.x;  
  
for (let key in my_func) {  
  console.log(key, my_func[key]);  
}  
// "y" 100  
  
my_func(); // prints 1
```

실습 - 함수와 클로저(closure)

- 함수의 바디 안에서 접근할 수 있는 이름들은
 - 바디 안에서 선언한 이름들
 - 인자로 넘어온 이름들
 - 함수가 **정의**될 때 존재했던 이름들
 - Closure

```
let n = 1;
function foo() { n++; }
function print() { console.log(n); }

console.log(n); // 1
foo();
console.log(n); // 2
foo();
console.log(n); // 3

function foo2(n) { n++; } // 인자 n

foo2();
console.log(n); // 3
foo2(n);
console.log(n); // 3
{
  let n = 10;
  console.log(n); // 10
  print(); // prints 3
}
```

실습 - 함수와 클로저(closure)

- 함수의 바디 안에서 접근할 수 있는 이름들은
 - 바디 안에서 선언한 이름들
 - 인자로 넘어온 이름들
 - 함수가 **정의**될 때 존재했던 이름들
 - Closure

```
let n = 1;
function foo() { n++; }
function print() { console.log(n); }

console.log(n); // 1
foo();
console.log(n); // 2
foo();
console.log(n); // 3

function foo2(n) { n++; } // 인자 n

foo2();
console.log(n); // 3
foo2(n);
console.log(n); // 3
{
  let n = 10;
  console.log(n); // 10
  print(); // prints 3
}
```

인자의 개수가 모자라거나 많을 때에는?

- 함수가 **정의**될 때 인자의 개수와 **호출**될 때의 인자의 개수가 달라도 에러가 발생하지 않는다.
 - 호출 시 인자를 적게 넘기면 없는 인자들은 undefined
 - 호출 시 인자를 많이 넘기면 남는 인자들은 무시
 - 엄밀하게는, 바디 내에서 arguments라는 이름으로 접근할 수 있다.

```
function test(a, b) {  
  console.log('a=' + a + ', b=' + b);  
  console.log(arguments)  
}
```

```
test(1, 2);  
test(1, 2, 3);  
test(1);
```

a=1, b=2

► Arguments(2) [1, 2, callee: f, Symbol(Symbol.iterator): f]

a=1, b=2

► Arguments(3) [1, 2, 3, callee: f, Symbol(Symbol.iterator): f]

a=1, b=undefined

► Arguments [1, callee: f, Symbol(Symbol.iterator): f]

실습 - 함수 가지고 놀기 (무한히 호출할 수 있는 함수?)

- 함수의 리턴 값으로 함수를 돌려줄 수도 있다.
 - 함수는 특별한 것이 아니라, 호출 가능한 오브젝트임을 유념

```
let n = 0;
function callme() {
  n++;
  return callme;
}

callme();
console.log(n); // 1

callme()()()();
console.log(n); // 5

callme()()()()()()()()()()()()()();
console.log(n); // 17
```

실습 – 함수 가지고 놀기 (다른 함수를 부르는 함수?)

- 함수의 인자 값으로 함수를 넘겨줄 수도 있다.
 - 함수는 특별한 것이 아니라, 호출 가능한 오브젝트임을 유념

```
function add(a, b) { return a + b; }  
function sub(a, b) { return a - b; }  
  
function calculator(op, a, b) { return op(a, b); }  
  
console.log(calculator(add, 2, 3)); // 5  
console.log(calculator(sub, 5, 1)); // 4
```


이름이 없는 함수 (익명 함수)

- `function foo(a, b) { return 10; }`

- 인자가 없는 함수? → 가능

- `function foo() { return 10; }`

- 리턴 값이 없는 함수? → 가능

- `function foo(a, b) { }`

- 그렇다면 이름이 없는 함수? → 가능

- `function (a, b) { return 10; }`

✖ Uncaught SyntaxError: Function statements require a function name

???

실습 - 이름이 없는 함수 (익명 함수)

- 단, 익명 함수는 이름이 없기 때문에 이후에 참조할 수 없다.
 - 정의하자마자 바로 호출하거나
 - let 키워드를 이용하여 이름에 할당하거나 (변수 할당)
 - 다른 함수에 넘겨서 인자로 받아야 사용 가능하다.

```
function (x) { return x + 1; } // error!  
  
(function (x) { return x + 1; })(2);  
let inc = function (x) { return x + 1; };  
my_func(function (x) { return x + 1; });
```

이름이 없는 함수가 실제로 사용되는 시나리오는 무엇이 있나요?

1. 함수의 클로저(closure)를 활용하기 위해서

- (예제) 내부 상태를 가지고, 호출할 때마다 상태를 1씩 증가시켜 반환

// 클로저를 활용하지 않은 코드는

```
let n = 0;  
function counter() { return ++n; }
```

```
console.log(counter()); // 1  
console.log(counter()); // 2
```

// 아래에 올 코드가
// n에 접근할 수 있기 때문에 위험

```
n = 5;  
console.log(counter()); // 6
```

// 하지만 클로저를 활용한 코드라면

```
let counter2 = (function() {  
  let n = 0;  
  return function() { return ++n; }  
})();
```

```
console.log(counter2()); // 1  
console.log(counter2()); // 2
```

```
n = 5;  
console.log(counter2()); // 3
```

이름이 없는 함수가 실제로 사용되는 시나리오는 무엇이 있나요?

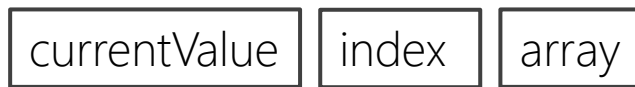
1. 함수의 클로저(closure)를 활용하기 위해서
 2. 배열을 더 잘 활용하기 위해
- 함수를 인자로 받는 자주 쓰이는 배열의 메소드
 - `let array = [1, 2, 3];`
 - `array.map(func)`
 - `array.filter(func)`
 - `array.forEach(func)`

배열의 메소드 활용하기

- `array.map(func)`
 - 배열의 각 값을 인자로 *func*를 호출한 후 *func*가 돌려주는 값으로 새로운 배열을 만든다.
 - *func*는 최대 3개의 인자 (currentValue, index, array)를 받아서 **임의의 값을 반환**하는 함수이어야 한다.
- `array.filter(func)`
 - 배열의 각 값을 인자로 *func*를 호출한 후 *func*가 돌려주는 값이 true인 값들만 모아 새로운 배열을 만든다.
 - *func*는 최대 3개의 인자 (currentValue, index, array)를 받아서 **논리 값(Boolean)을 반환**하는 함수이어야 한다.
- `array.forEach(func)`
 - 배열의 각 값을 인자로 *func*를 호출한다.
 - *func*는 최대 3개의 인자 (currentValue, index, array)를 받아서 일을 처리하고(side effect) **아무것도 반환하지 않는** 함수이어야 한다.

배열의 메소드 활용하기

- *func*의 인자
 - 첫 번째 인자(currentValue): 현재 요소. 순회하고 있는 데이터 하나.
 - 두 번째 인자(index): 현재 요소의 배열 안에서의 인덱스.
 - 세 번째 인자(array): 메소드를 호출한(순회하고 있는) 배열.
- 인자 이름은 바뀌서 사용할 수 있다.
- 인자를 꼭 3개 다 지정할 필요는 없다.



```
let arr = [3, 4, 5];  
arr.forEach(function(x, i, array) {  
  console.log('arr[' + i + ']=' + x + ' / ' + array);  
});
```

arr[0]=3	/	3,4,5
arr[1]=4	/	3,4,5
arr[2]=5	/	3,4,5

실습 - 배열의 메소드 활용하기

```
let arr = [0, 1, 2];  
let even = arr.map(function(x) { return 2 * x; });  
let odd = arr.map(function(x) { return 2 * x + 1; });
```

```
console.log(even);  
console.log(odd);
```

```
let small = even.concat(odd).filter(function(x) {  
    return x < 3;  
});
```

```
console.log(small);
```

```
small.forEach(function(x, i) {  
    console.log('arr[' + i + ']=' + x);  
});
```

```
small.filter(function(x) { return x === 0; });  
console.log(small);
```

```
small = small.filter(function(x) { return x === 0; });  
console.log(small);
```

```
▶ (3) [0, 2, 4]  
▶ (3) [1, 3, 5]  
▶ (3) [0, 2, 1]  
arr[0]=0  
arr[1]=2  
arr[2]=1  
▶ (3) [0, 2, 1]  
▶ [0]
```

실습 - 메소드 체인

- 배열을 리턴 하는 메소드를 호출하고, 결과 배열에 또 호출하고 ...

```
let students = [  
  {name: "John", scores: [10, 20, 30]},  
  {name: "Jane", scores: [40, 10, 20]},  
  {name: "David", scores: [20, 30, 55]}  
];
```

```
students  
  .map(function(student) {  
    let sum = 0;  
    student.scores.forEach(function(s) { sum += s; });  
    return {  
      name: student.name,  
      avg: sum / student.scores.length  
    };  
  })  
  .sort(function(a, b) { return b.avg - a.avg; })  
  .forEach(function(student, rank) {  
    console.log('rank' + (rank + 1) + ': ' + student.name);  
  });
```

각 학생의
점수의 평균
계산하여
데이터 가공

점수가 높은
학생 순으로
정렬

```
rank1: David  
rank2: Jane  
rank3: John
```


JavaScript

Drawing a Bar Chart

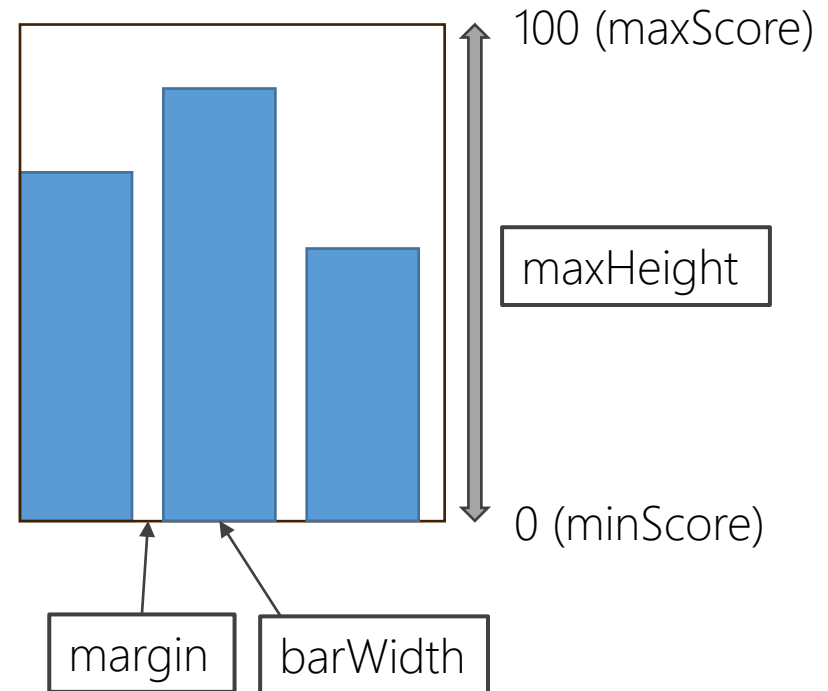
실습 - 세로 바 차트 그리기

- SVG와 데이터 준비

```
<h1>학생 별 점수</h1>
<svg id="chart" width="200" height="200"
  style="border:black 1px solid"
></svg>
```

```
<script>
let students = [
  {"name": "Jane", "score": 80},
  {"name": "John", "score": 90},
  {"name": "Bob", "score": 60}
];

let minScore = 0;
let maxScore = 100;
let barWidth = 50;
let maxHeight = 200;
let margin = 10;
</script>
```



실습 - 세로 바 차트 그리기

- 학생 한 명 당 하나의 차트 막대 만들어 보기
 - <script> 태그 안에 이어서 작성

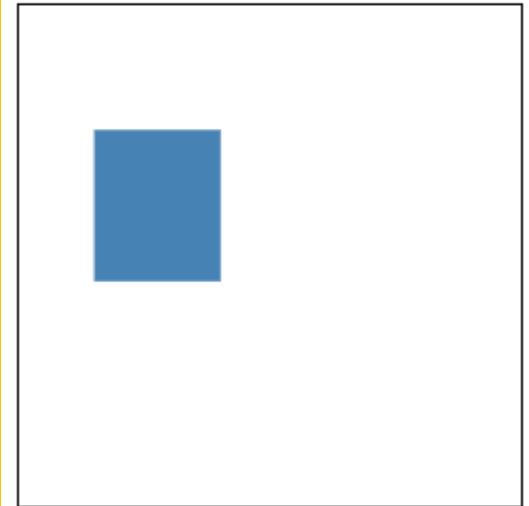
```
let svg = document.getElementById("chart");
let spec = "http://www.w3.org/2000/svg";

students.forEach(function(student, i) {
  let rect = document.createElementNS(spec, "rect");

  rect.setAttribute("width", 50);
  rect.setAttribute("height", 60);
  rect.setAttribute("x", 30);
  rect.setAttribute("y", 50);
  rect.style.fill = "steelblue";

  svg.appendChild(rect);
});
```

학생 별 점수



실습 - 세로 바 차트 그리기

- 학생 한 명 당 하나의 차트 막대 만들어 보기
 - <script> 태그 안에 이어서 작성

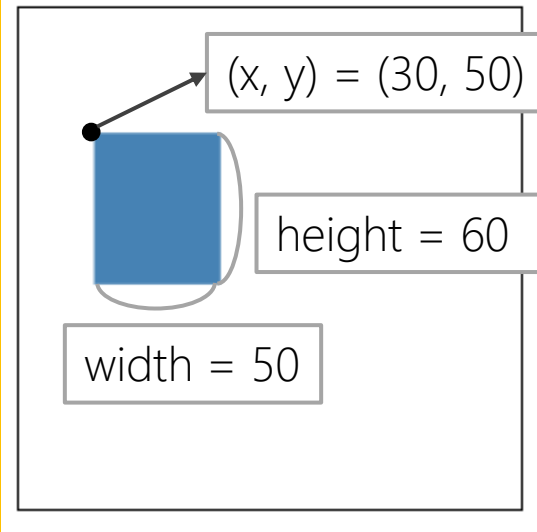
```
let svg = document.getElementById("chart");
let spec = "http://www.w3.org/2000/svg";

students.forEach(function(student, i) {
  let rect = document.createElementNS(spec, "rect");

  rect.setAttribute("width", 50);
  rect.setAttribute("height", 60);
  rect.setAttribute("x", 30);
  rect.setAttribute("y", 50);
  rect.style.fill = "steelblue";

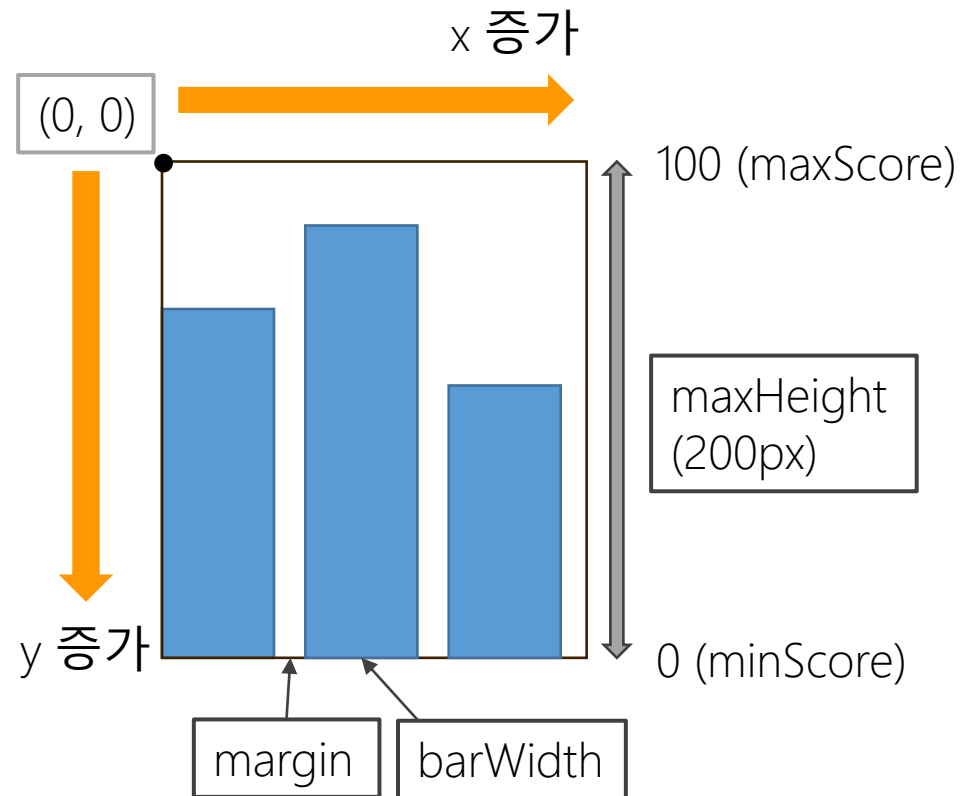
  svg.appendChild(rect);
});
```

학생 별 점수



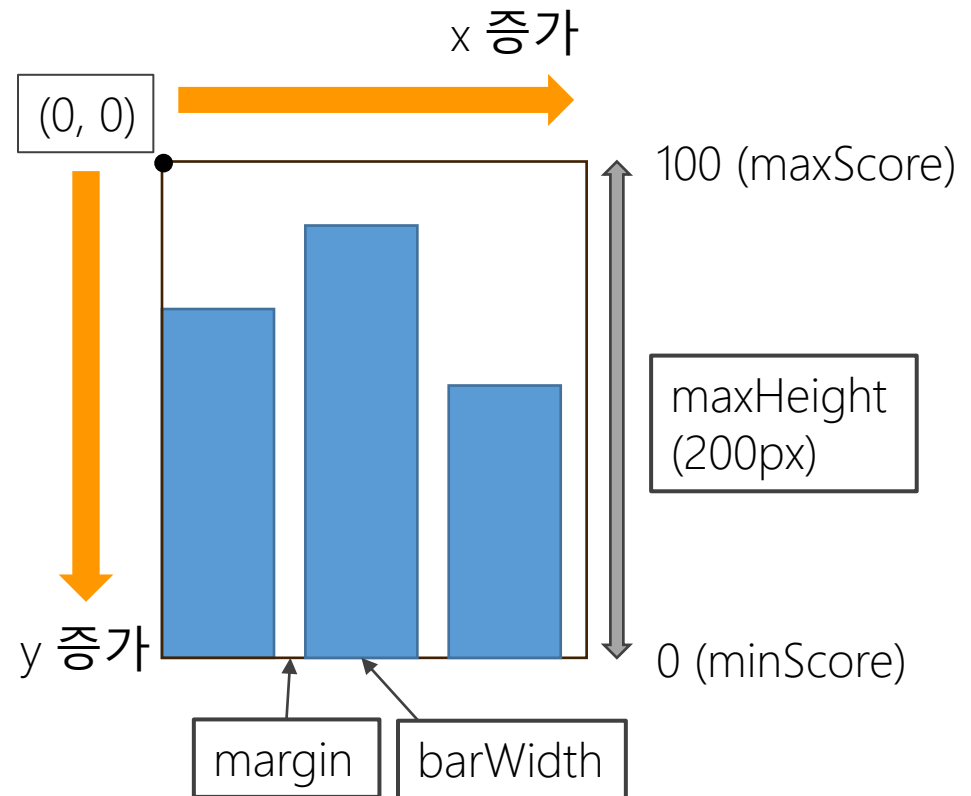
i번째 바의 너비, 높이, 위치는 어떻게 될까?

- i번째 바의 점수를 score라고 할 때 ($i = 0, 1, 2$)
- 바의 너비: ?
- 바의 왼쪽 위 x좌표: ?
- 바의 높이: ?
- 바의 왼쪽 위 y좌표: ?



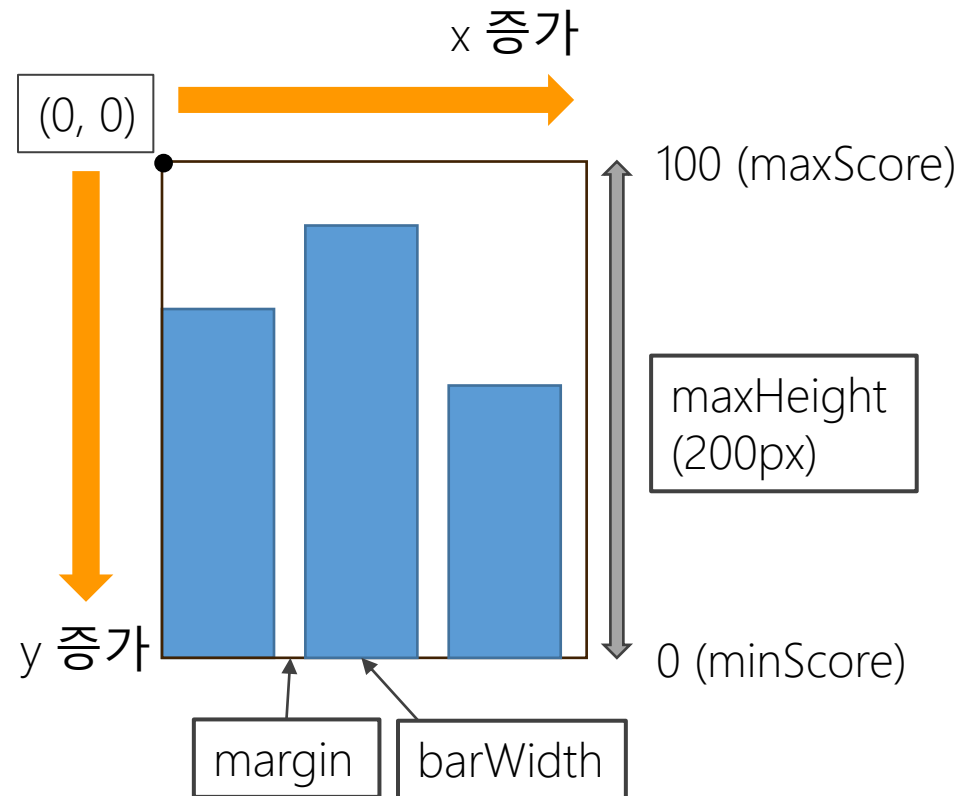
i번째 바의 너비, 높이, 위치는 어떻게 될까?

- i번째 바의 점수를 score라고 할 때 ($i = 0, 1, 2$)
- 바의 너비: `barWidth` (고정)
- 바의 왼쪽 위 x좌표: ?
- 바의 높이: ?
- 바의 왼쪽 위 y좌표: ?



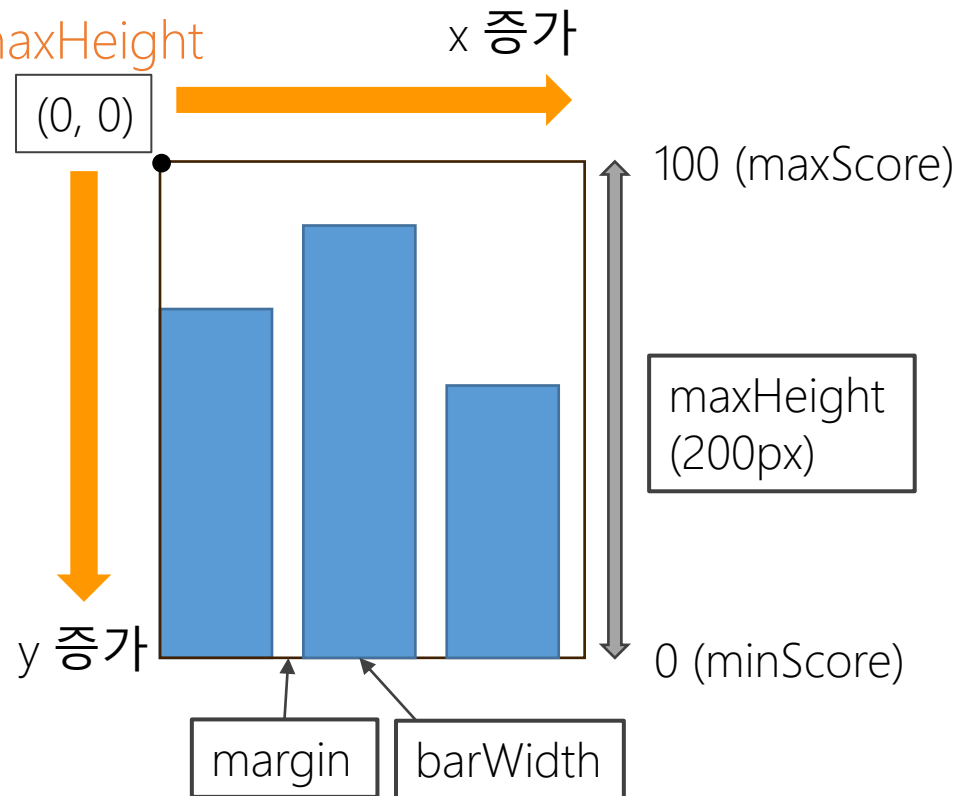
i번째 바의 너비, 높이, 위치는 어떻게 될까?

- i번째 바의 점수를 score라고 할 때 ($i = 0, 1, 2$)
- 바의 너비: barWidth (고정)
- 바의 왼쪽 위 x좌표: $(\text{barWidth} + \text{margin}) * i$
- 바의 높이: ?
- 바의 왼쪽 위 y좌표: ?



i번째 바의 너비, 높이, 위치는 어떻게 될까?

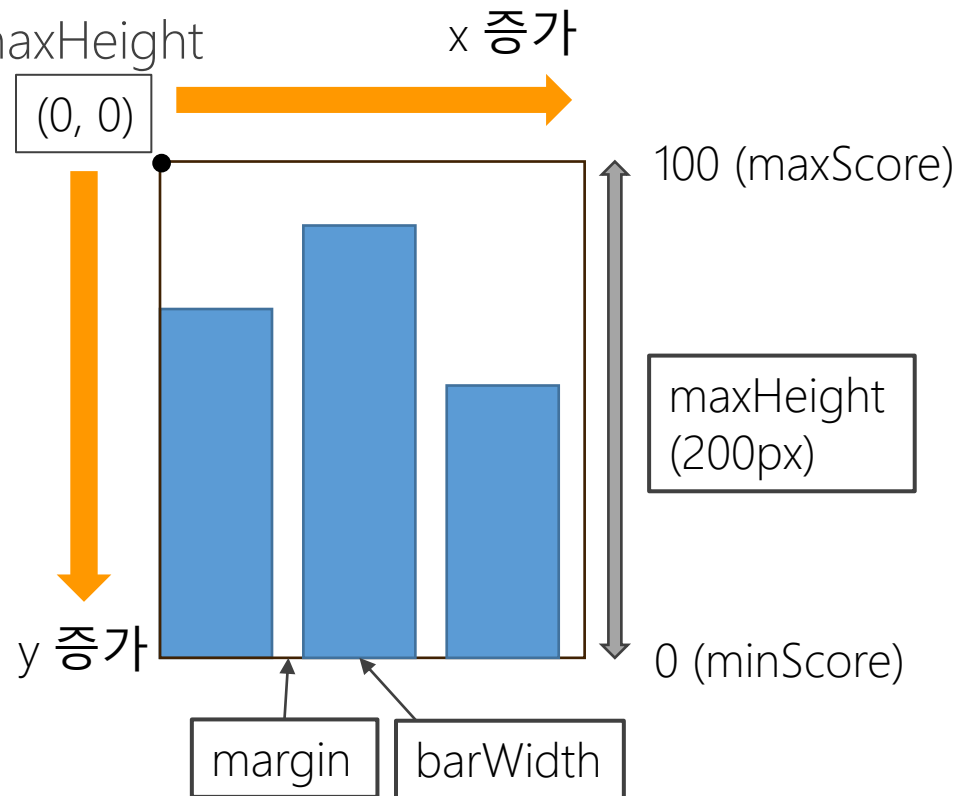
- i번째 바의 점수를 score라고 할 때 ($i = 0, 1, 2$)
- 바의 너비: barWidth (고정)
- 바의 왼쪽 위 x좌표: $(\text{barWidth} + \text{margin}) * i$
- 바의 높이: $\text{score} / \text{maxScore} * \text{maxHeight}$
- 바의 왼쪽 위 y좌표: ?



i번째 바의 너비, 높이, 위치는 어떻게 될까?

- i번째 바의 점수를 score라고 할 때 ($i = 0, 1, 2$)
- 바의 너비: barWidth (고정)
- 바의 왼쪽 위 x좌표: $(\text{barWidth} + \text{margin}) * i$
- 바의 높이: $\text{score} / \text{maxScore} * \text{maxHeight}$
- 바의 왼쪽 위 y좌표:

$\text{maxHeight} - (\text{현재 바의 높이})$



실습 - 막대 위치 설정하기

- 학생 한 명 당 하나의 차트 막대 만들어 보기
 - students 데이터를 사용하도록 앞 슬라이드의 코드 수정

```
let svg = document.getElementById("chart");
let spec = "http://www.w3.org/2000/svg";

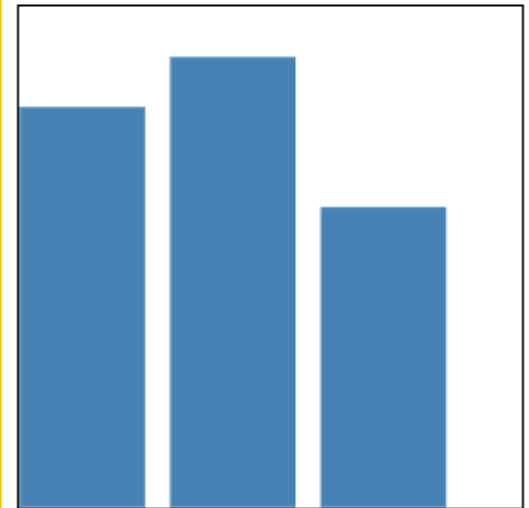
students.forEach(function(student, i) {
  let rect = document.createElementNS(spec, "rect");

  let height = student.score / maxScore * maxHeight;

  rect.setAttribute("width", barWidth);
  rect.setAttribute("height", height);
  rect.setAttribute("x", (barWidth + margin) * i);
  rect.setAttribute("y", maxHeight - height);
  rect.style.fill = "steelblue";

  svg.appendChild(rect);
});
```

학생 별 점수



결론

- 코드가 너무 많고, 절차적이다. (imperative)
- x, y 좌표 구하는 부분 등은 일반화하여 재사용할 수 있을 것 같은데...
 - 바 차트, 라인 차트, 산점도 등 position 채널에 인코딩하는 차트들...
- D3.js: **시각화**를 만들 때 자주 쓰이는 코드를 재사용 가능한 선언적 문법으로 만들어 둔 **라이브러리**