08.21_실습 강의자료

플러그인 추천

code Runner

VScode에서 바로 JS코드 실행해 콘솔창을 볼 수 있습니다.

0교시

HTML 복습

CSS 복습

1교시

1-01 토끼 그리기

▼ 따옴표가 출력이 안되요!

C언어 같은 경우 “는 문자열, ‘는 문자 이러한 차이점이 있었지만

자바스크립트에서 “와 ‘는 차이가 없습니다. “, ‘를 이용해서 문자열의 시작과 끝을 결정하죠.따라서 여러분이 문자열을 쓰고 싶을때 양 끝에 “또는 ‘를 써도 동일하게 작동합니다.

하지만 주의할 점은 “와 ‘를 같은 것으로 인식하지는 않는다는 것입니다.

```
document.write("hello')
document.write('hello")
```

이런식으로 양 끝에 다른 따옴표를 쓰면 문법오류가 납니다.

▼ \ 가 출력이 안되요ㅜ

JS에서 이미 기능을 하고 있는 특수문자들이 있습니다. 예를들면 \, “, ‘, 같은 것들이죠.

그래서 JS는 이 문자들을 인식하고 해석합니다. 하지만 우리가 JS문법으로 사용하는 것이 아니라 문자열로써 \, “, ‘ 를 쓰고 싶을 때는 어떻게 할까요?

내가 문자열로써 쓸 것이다를 JS에게 알려주면 됩니다. 그 알려주는 방법이 바로 내가 쓸 문자 앞에 \(백슬래시)를 붙여주는 것입니다.

▼ 템플릿 리터럴 사용하는 방법은?

벡틱으로 불리는 `을 사용하면 더 편하게 문자열로 만들어 줄 수 있습니다. 벡틱 안에서js변수를 ${ }안에 넣으면 문자열로 변환해 줍니다.

```
const str = "test1";
console.log(`${str} test2`);
```

또한 띄어쓰기, 줄바꿈도 인식을 해주기 때문에 한번에 표현하기 편리합니다.

```
console.log("띄어쓰기는\n들여쓰기는 \t를 쓰면 되지용");

console.log(`
띄어쓰기는
들여쓰기는  	를 쓰면 되지용
`);
```

1-02 화살표 함수

▼ 함수..?

함수를 사용하는 이유는 무엇일까요?

- 함수는 몇 번이든 호출 할 수 있으므로 코드의 **재사용성** 측면에서 매우 유용하다.
- 재사용성이 높다는 것은 유**지보수의 편의성**과 **코드의 신뢰성**을 높이는 효과가 있다.
- 또한 **함수는 객체 타입의 값**으로써 식별자를 붙일 수 있는데 식별자 이름을 잘 지음으로써 **코드의 가독성**을 높일 수 있다.

여러가지 이유가 있습니다.

JS에서 함수는 매우 중요한 개념입니다. 여러분을 머리아프게 할 스코프, 실행컨텍스트, 클로저 등 핵심 개념들이 모두 함수와 밀접한 관계를 가지기 때문이죠!! 앞으로 하나씩 차근차근 배워봐요!!😊

▼ 함수만드는 방법은?

함수를 정의하는 방법으로는 함수 선언문과 함수 표현식이 있습니다.

함수 선언문

함수로서 선언을 하는 방법입니다.

호이스팅에 의해 맨 위로 올려집니다.

```
// 함수 실행
sayHi1();

// 함수 선언식
function sayHi1() {
    // 함수명 sayHi가 곧 변수명이다.
    console.log("Hello1");
}
```

함수 표현식

함수를 별도의 변수에 할당하는 방법입니다.

호이스팅이 일어나지 않습니다.

함수 표현식에서는 익명함수를 사용하는 것이 국룰이긴 하지만 기명함수가 에러가 나는 것은 아닙니다. 하지만 함수를 호출 할때는 함수의 이름이 아닌 함수를 할당해준 변수명(식별자명)을 사용해서 호출해야 합니다.

```
sayHi2(); // error

// 함수 표현식
const sayHi2 = function () {
    // 변수명 sayHi가 곧 함수명이다.
    console.log("Hello2");
};

// 함수 실행
sayHi2();
```

기업의 컨벤션에 따라 작성하시겠지만 차이점인 인지하고 사용하셔야 합니다!

▼ var | const | let 뭐가 다를까?

var는 다른 두가지와 변수 선언 방식에서 차이점이 있습니다.

차이점

var는 js의 초기 변수 선언방법으로 여러가지 문제점이 있었습니다.

1. 같은 변수명으로 선언을하게 되면 덮어쓰게됨

   : 이는 코드량이 많아진다면 어디서 잘못 된건지 찾기 힘든 치명적인 문제점이 있습니다.

2. 변수가 선언되지 않아도 참조 가능 (변수 호이스팅)

: var는 function level scope이기 때문에 함수 밖에 선언된 var는 모두 전역변수가 됩니다.

이는 메모리를 잡아먹어 성능적으로 낮아지며 변수명 중복으로 인한 오류를 야기합니다.

```
// var
var a;
a = "test1";
console.log(a); // test1

var a = "test1";
console.log(a); // test1

a = "hello1";
console.log(a); // hello1

var a = 1;
console.log(a); // 1
```

이를 보안하기 위해 js 기술 규격을 정하는 Ecma 인터네셔널이라는 기구에서 2015년 es6문법을 발표했는데요 이중 const, let이라는 상수 변수를 통해 var를 대체하게 되었습니다.

const는 변수 재선언, 재할당이 안되는 변수로 바꿀수 없는 값인 상수의 역할을 가집니다. 또한 선언과 할당을 동시에 해야합니다.

```
// const
const b; // error
b = test2;

const b = "test2";
console.log(b); // test

b = "hello2";
console.log(b); // error

const b = 2;
console.log(b); // error
```

let은 변수에 재할당이 가능하며 선언과 할당을 따로할 수 있습니다. 하지만 재선언은 안됩니다.

```
// let
let c;
c = "test3";
console.log(c); // test3

let c = "test3";
console.log(c); // test3

c = "hello3";
console.log(c); // hello3
```

```
let c = 3;
console.log(c); // error
```

더 깊게 들어간다면 변수 호이스팅, function level scope VS block level scope 등의 이슈가 있으니 참고하시길 바랍니다.

참고자료

js의 역사 : https://erokuma.tistory.com/entry/자바스크립트의-역사와-ECMAScript-대해

var의 문제점 : https://happycording.tistory.com/entry/let-const-란-왜-써야만-하는가-ES6

▼ 호이스팅과 TDZ란?

호이스팅은 변수의 선언문을 유효범위 즉, 스코프 최상단으로 올려주는 것입니다.

위의 예시로 var, function, import 같은 것들이 있었죠?

이때 TDZ은 Temporal Dead Zone 즉 일시적 사각지대 라는 뜻으로 변수 선언 및 초기화 하기 전 상태로 TDZ상태에서 변수를 사용하려고 하면 ReferenceError가 나타나게 됩니다.

▼ arrow function

또 다른 함수로는 ES6에 나온 arrow function이 있습니다.

기존의 Function과 차이점을 알아보고 사용법, 주의점을 알아보겠습니다.

차이점1

함수 호이스팅

function은 호이스팅에 의해 최상단에 선언되지만 arrow function은 호이스팅되지 않습니다. 화살표 함수는 익명함수이기 때문에 호출하기 위해서는 변수에 할당해야 하는 함수 표현식이기 때문이죠.

```
// 호이스팅 차이
// function
func(); // hi~
function func() {
    return console.log("hi~");
}

// arrow function
arrFunc1(); // error
const arrFunc1 = () => console.log("hi~");

// arrFunc1(); // hi~

const arrFunc2 = () => {
    return console.log("hi~");
};
```

차이점2

this의 사용법

JS에서 this는 어떻게 호출되었느냐에 따라서 달라집니다.

간단하게 정리하자면

1. 일반 함수에서 this ⇒ 전역 객체
2. 메서드 내부에서의 this ⇒ 메서드를 호출한 객체
3. 생성자 함수에서 this ⇒ 미래에 생성할 인스턴스
4. 간접 호출에서 this ⇒ 내가 할당해 줄 객체
5. 화살표함수에서 this ⇒ 가장 가까운 상위 함수(스코프체인의 상위 스코프)중 화살표 함수가 아닌 함수의 this

```
// this 차이
// function
function fun() {
    this.name = "하이";
    return {
        name: "바이",
        speak: function () {
            console.log(this.name); // 바이
        },
    };
}

const fun1 = new fun();
fun1.speak(); // 바이

// arrow func
function arrFun() {
    this.name = "하이";
    return {
        name: "바이",
        speak: () => {
            console.log(this.name); // 하이
        },
    };
}

const fun2 = new arrFun();
fun2.speak(); // 하이
```

차이점3

생성자 함수로 사용여부

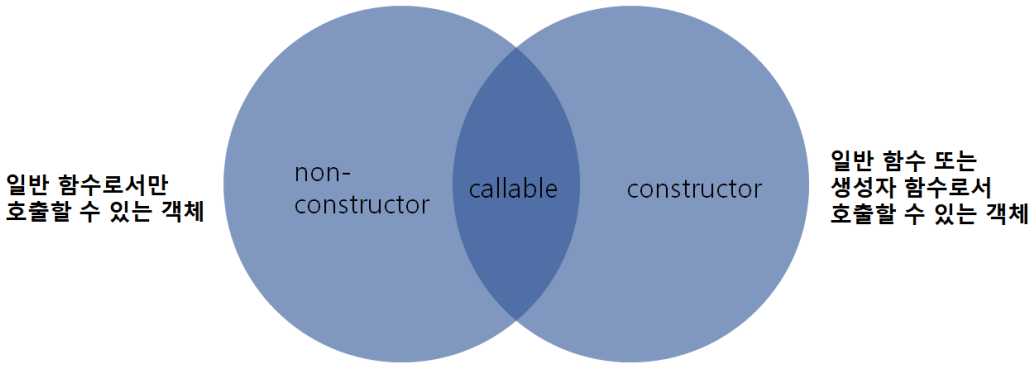
함수는 함수 객체만을 위한 내부 슬롯과 내부 메서드를 추가로 가지고있습니다.

- 내부 슬롯 : [[Environment]], [[FormalParameters]]
- 내부 메서드 : [[Call]], [[Construct]]

일반 함수로 호출했을때는 내부 메서드 [[Call]]이 호출되고, new와 함께 생성자 함수로 호출했을때는 내부 메서드 [[Construct]]가 호출됩니다.

모든 함수는 호출할 수 있으므로 callable이지만 함수중에는 생성자 함수로서 호출 할 수 없는 경우도 있다. 이러한 경우를 non-constructor라고 하죠

화살표함수가 바로 non-constructor입니다.



- constructor : 함수 선언문, 함수 표현식, 클래스
- non-constructor : ES6메서드(메서드 축약 표현 : 객체안에 f( ){ } 로 들어가 있는 메서드), 화살표 함수

```
// func
function fun() {
    this.num = 1234;
}

const funA = new fun();
console.log(funA.num); // 1234

//arrow func
const arrFun = () => {
    this.num = 1234;
};

const funB = new arrFun(); // Error
```

화살표 함수 사용법

```
// 기존의 익명함수
function (a, b) {
  return a + b + 100;
};

// Arrow function
(a, b) => a + b + 100;
```

2교시

1-03 자바스크립트의 입출력

▼ 모듈을 불러오는 방법은?

require문법과 import 문법이 있습니다.

(require / exports) 는 NodeJS에서 사용되고 있는 CommonJS 키워드이고 Ruby 언어 스타일과 비슷하다라고 볼수 있습니다.

cf. CommonJS는 JS에서 모듈화작업을 할 수 있게 만들어 줍니다.

(import / export) 는 ES6(ES2015)에서 새롭게 도입된 키워드로서 Java나 Python 언어 방식과 비슷하다.

차이점

1. require()는 코드를 적은 곳에서 불러와 어느 지점에서나 호출이 가능한 반면 import는 항상 맨 위로 이동하여 파일 시작부분에서만 실행됩니다.(호이스팅)

- 하지만 비동기를 사용하면 나중에 불러올 수 있습니다.

  참고 : https://inpa.tistory.com/entry/JS-📚-모듈-사용하기-import-export-정리?category=889099#브라우저(HTML)에서_모듈_사용_하기

- cf. 호이스팅이란 쉽게 생각하면 실행될때 최 상단에 선언된것으로 인식해서 먼저 실행된다고 이해하시면 됩니다.

  호이스팅 예제 : https://simplejs.gitbook.io/olaf/09.-hoisting

2. 일반적으로 import는 모듈의 필요한 부분만 불러올수 있기 때문에 더 선호되며 성능적으로 우수합니다.
3. 하지만 Import 를 사용하기 위해서는 webpack이나 Babel등의 컴파일러가 필요하다는 단점이있습니다.

- cf. 컴파일러란 우리가 작성한 언어(JavaScript)를 기계가 알아들을 수 있는 기계어(어셈블리어)로 번역해준다.

  (기본 개발 용어 알아보기 : https://www.youtube.com/watch?v=GYmuQJiPeM4)

참고 자료

require과 Import의 차이점 : https://inpa.tistory.com/entry/NODE-📚-require-⚔️-import-CommonJs와-ES6-차이-1

비동기로 import하기 : https://inpa.tistory.com/entry/JS-📚-모듈-사용하기-import-export-정리?category=889099#브라우저(HTML)에서_모듈_사용_하기

▼ readline 모듈 보충설명

readlie은 js에 내장된 모듈로 한 줄 씩 입출력을 처리할 수 있게 도와주는 모듈입니다.

먼저 require(”모듈이름”)을 통해서 모듈을 불러오고 변수에 저장합니다.

readline 안의 메서드인 createInterface()를 통해서 인터페이스 객체를 생성하고 input에 사용자의 입력을 수신하는 모듈인 process.stdin를 output에 출력을 하게 해주는 모듈인 process.stdout을 할당합니다.

```
rl.on('line', line => {
  // 입력을 받아서 실행 할 코드
  rl.close();
});
```

을 통해 입력받은 줄을 line에 저장해서 사용한 후 rl.close()를 통해 닫아줍니다.

```
rl.on('close', () => {
  // 입력이 끝난 후 실행할 코드
});
```

}

을 통해 입력이 끝난 후 하고싶은 일을 작성합니다.

참고자료

readline 모듈 : https://nodejs.org/api/readline.html

process 모듈 : https://nodejs.org/api/process.html

console.log와 process.stdout.write의 차이점 : https://www.geeksforgeeks.org/difference-between-process-stdout-write-and-console-log-in-node-js/

▼ 또다른 JS 입출력방법

많이 쓰이는 다른 입출력 모듈로는 fs모듈이 있습니다.

readline 모듈이 한 줄 씩 입력값을 받는 다면 fs모듈을 이용하면 한 번에 모든 입력값을 받을 수 있습니다.

```
const fs = require("fs");
const input = fs.readFileSync("/dev/stdin")
```

.readFileSync()를 통해서 입력값을 받아왔을 때 readline과는 다르게 입력값이 인코딩되어있는 buffer 형태로 오기 때문에 디코딩을 해줘야합니다. 그래서 utf8형태로 디코딩 기본값인 toString()을 사용해서 입력값을 디코딩 해줍니다.

```
// fs모듈 사용
const fs = require("fs");
const input = fs.readFileSync("/dev/stdin").toString();
```

cf. fs모듈의 readFileSync로 테스트 할 때는 readline의 close의 기능이 구현되어있지 않기 때문에 입력을 종료해 줘야합니다. control + d 를 통해 입력을 종료 할 수 있습니다.

공식문서 : https://nodejs.org/api/fs.html

1-04 값 입력 받기

▼ 한 줄 씩 입력받기!

이전 자료에서 봤었던 한 줄씩 입력받는 방법을 실습해봐요!

입력은 어떤 이벤트로 받는다고 했죠?

▼ fs모듈을 사용하면 어떻게 할 수 있을까요?

readline모듈로 푼 문제를 fs모듈로 변경해봐요!

▼ 정답

```
const input = require('fs').readFileSync('/dev/stdin').toString();
console.log(input);
```

1-05 한 번에 여러 입력받기

▼ 한 줄에 여러개를 입력받았는데 하나씩 처리하고싶어요!

우리가 받은 입력은 readline의 on메서드에서 line이벤트를 통해 한 줄의 string으로 넘어옵니다.

그렇다면 그 string을 우리가 쪼개서 쓰면 되겠죠?

입력받은 한 줄의 string을 공백 기준으로 짤라준다면 우리가 필요한 값들로 나눌 수 있겠네요!

그러면 string을 우리가 원하는 기준으로 짜르는 방법을 알면 해결할 수 있을 것 같아요🙂

▼ split() 알아보기

split()메서드는 원하는 기준으로 문자열을 잘라 배열로 반환합니다.

str.split([separator] [, limit])

: 문자열str안에 separator값을 기준으로 나눠 arr로 반환

```
console.log("\n" + "===========" + str + "===========");
console.log(str.split(" ")); // [ '엘리스', 'AItrack', '9기', '과정', '레이서분들', '안녕하세요' ]
console.log(str.split(""));
/*
[
  '엘', '리', '스', ' ',  'A',  'I',
  't',  'r',  'a',  'c',  'k',  ' ',
  '9',  '기', ' ',  '과', '정', ' ',
  '레', '이', '서', '분', '들', ' ',
  '안', '녕', '하', '세', '요'
]
*/
console.log(str.split("SW")); // [ '엘리스 AItrack 9기 과정 레이서분들 안녕하세요' ]
console.log(str.split()); // [ '엘리스 AItrack 9기 과정 레이서분들 안녕하세요' ]
console.log(str); // 엘리스 AItrack 9기 과정 레이서분들 안녕하세요
```

split() :
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String/split

▼ 문자열을 다루는 다른 방식은?

str.substr(start[, length])

: 인덱스가 start 부터 length길이의 문자열을 반환

⇒ 사라지는 메서드!

```
const str = "엘리스 AItrack 9기 과정 레이서분들 안녕하세요";

console.log(str.substr(0, 3)); // 엘리스
```

str.substring(indexStart[, indexEnd])

: 문자열str의 indexStart로 부터 종료 인덱스 전 까지 문자열의 부분 문자열을 반환

```
console.log("\n" + "===========" + str + "===========");
console.log(str.substring(0, 3)); // 엘리스
console.log(str.substring(3)); //  AItrack 9기 과정 레이서분들 안녕하세요
console.log(str.substring(4, 10)); // AItrac
```

str.slice(beginIndex[, endIndex])

: 문자열str의 indexStart로 부터 종료 인덱스 전 까지를 추출하면서 새로운 문자열을 반환

```
console.log("\n" + "===========" + str + "===========");
console.log(str.slice(0, 3)); // 엘리스
console.log(str.slice(3)); //  AItrack 9기 과정 레이서분들 안녕하세요
console.log(str.slice(4, 10)); // AItrac
```

str.replace(regexp | substr, newSubstr | function)

: 문자열str안에 첫번째 인자로 받은 값을 두번째 인자로 받은 값으로 변경

```
console.log("\n" + "===========" + str + "===========");
console.log(str.replace("안녕하세요", "화이팅~!")); // 엘리스 AItrack 9기 과정 레이서분들 화이팅~!
console.log(str); // 엘리스 AItrack 9기 과정 레이서분들 안녕하세요
```

참고자료

substr() :
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String/substr

substring()

:
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String/substring

slice() :
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String/slice

replace() :
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String/replace

split() :
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String/split

▼ fs모듈을 사용하면 어떻게 할 수 있을까요?

readline모듈로 푼 문제를 fs모듈로 변경해봐요!

▼ 정답

```
const input = require('fs').readFileSync('/dev/stdin').toString().split(' ').map(Number);
console.log(input.join('\n'));
```

1-06 N번째로 큰 수 찾기

▼ 입력을 다 받은 후 처리를 하고싶다면?

readline 모듈에서 on 메서드로 이벤트를 처리 할 수 있었죠?

그 중 ‘close’라는 이벤트를 통해 입력이 끝난 후 처리할 로직을 작성해 줄 수 있습니다.

```
rl.on("close", () => {
    // 입력이 끝난 후 실행할 코드
});
```

▼ fs모듈을 사용하면 어떻게 할 수 있을까요?

readline모듈로 푼 문제를 fs모듈로 변경해봐요!

▼ 정답

```
const input = require('fs').readFileSync('/dev/stdin').toString().split('\n');

const arr = input[0].split(' ').map(Number);
const n = input[1] * 1;
arr.sort((a, b) => b - a);
console.log(arr[n - 1]);
```

1-07 충성! 입대를 명 받았습니다!

▼ sort() 알아보기

JavaScript에도 배열을 정렬해주는 sort()함수가 있습니다.

arr.sort([compareFunction])

옵셔널 파라미터 값으로 compareFunction이 있는데요 이 친구가 없다면 배열의 요소들을 문자열로 취급하여 유니코드 값 순서대로 정렬이 됩니다. 우리가 원하는 방법과 다르게 정렬될 수 있죠

```
const numbers = [10, 3, 8, 4, 1];
numbers.sort();

console.log(numbers);
// 결과 : [ 1, 10, 3, 4, 8 ]
```

위의 예시를 보면 10이 3보다 앞에 위치하는게 보이시죠? 바로 값을 비교할때 숫자 타입을 문자타입으로 형변환하여 비교하기 때문인데요 그러면 형변환 이후 첫번째 문자 기준으로 비교 하기 때문에 1이 3보다 작다고 보고 10을 3 앞에 두게 됩니다.

유니코드란?

각 나라별 언어를 모두 표현하기 위해 나온 코드 체계가 유니코드(unicode)입니다. 유니코드는 사용중인 운영체제, 프로그램, 언어에 관계없이 문자마다 고유한 코드 값을 제공하는 새로운 개념의 코드로 언어와 상관없이 모든 문자를 16비트로 표현해요

https://ko.wikipedia.org/wiki/유니코드_0000~0FFF

그러면 우리가 원하는 방식대로 정렬하기 위해서는 어떻게 해줘야 할까요? compareFunction을 이용하면됩니다.

여기서 기억하실 것은

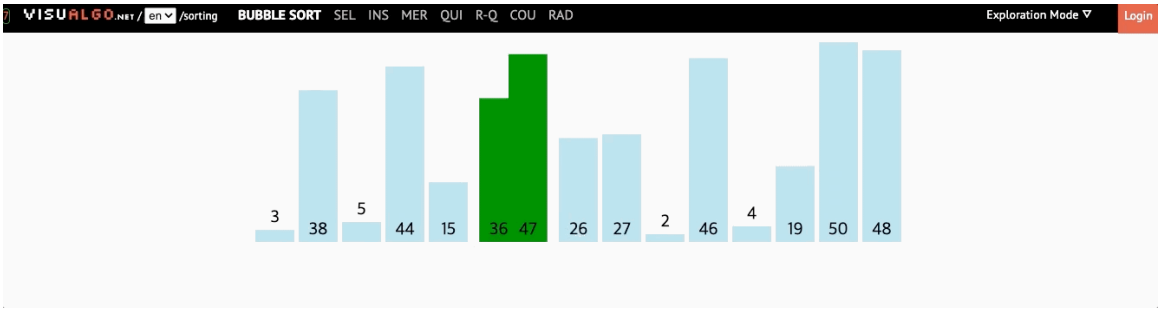
- 반환 값 < 0 : a 가 b보다 앞에 있어야 한다.
- 반환 값 = 0 : a와 b의 순서를 바꾸지 않는다.
- 반환 값 > 0 : b가 a보다 앞에 있어야 한다.

이 세가지를 생각하시며 비교함수를 작성하시면 됩니다.

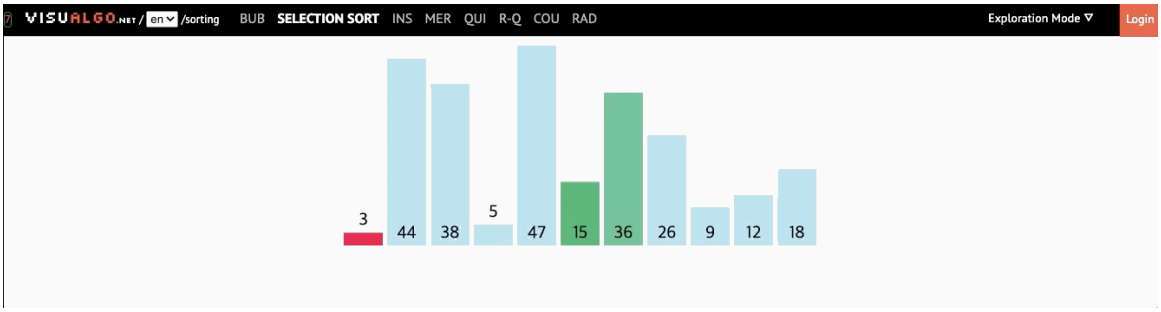
예를들어 오름차순으로 정렬하고 싶다면 compareFunction 자리에 (a,b)⇒ a-b 를 넣어주면 됩니다.

▼ 정렬하는 알고리즘들은 뭐가 있을까?

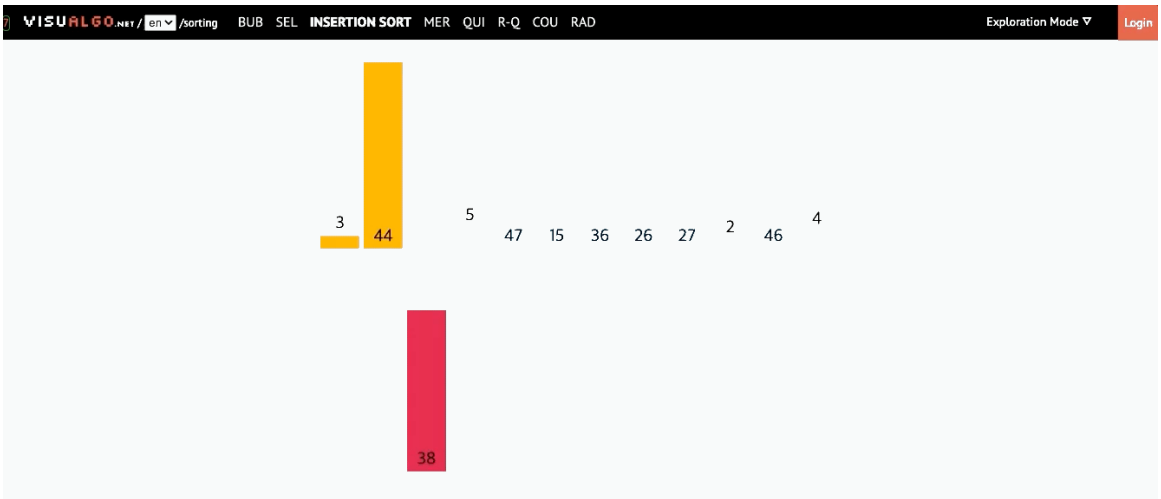
버블 정렬 (Bubble Sort)



선택 정렬 (Selection Sort)



삽입 정렬 (Insertion Sort)



여러가지 정렬 법 시각화로 보기 : https://visualgo.net/en/sorting?slide=1

추가내용

▼ JS의 타입

js에서 타입은 크게 두종류가 있습니다. 원시 타입과 객체 타입이 있는데요

원시 타입(기본 타입)으로는

1. 숫자(number)
   - int, long, float, double과 같이 다양한 숫자 타입이 존재하는 C언어와는 다르게 자바스크립트에서는 단 하나의 숫자 타입만 존재합니다. 따라서 모든 숫자를 실수로 처리하기 때문에 나누기 연산을 할 경우 C언어와 다르게 소수점까지 출력하므로 주의하여야 합니다.

```
// number
let num = 9 / 2;

console.log(num); // 4.5
```

2. 문자열(string)
   - C언어에서는 문자 하나만을 저장하는 char 타입이 존재하는 반면, 자바스크립트에서는 문자열의 길이에는 상관 없이 string이라는 데이터 타입으로 값이 생성됩니다.

```
// string
let str = "dev";

str[0] = "D";
console.log(str); // dev
```

3. 불린값(boolean)
   - true와 false 값을 나타내는 불린 타입이 존재합니다.

```
// boolean
let flag = true;

console.log(typeof flag); // boolean
```

4. undefined, null
   - 자바스크립트에서의 undefined와 null은 모두 값이 비어있음을 나타내는 데이터 타입입니다. 하지만 두 타입에는 차이점이 있습니다.
     - 값이 할당되지 않은 변수는 undefined 타입

- 명시적으로 값이 비어있을 때는 null 타입

```
// undefined, null
let tmp;

console.log(typeof tmp); // undefined
console.log(tmp); // undefined

tmp = null;
console.log(typeof tmp); // object
console.log(tmp); // null
```

5. 심벌(symbol)
   - ECMA Script 6에서 등장한 새로운 데이터 타입으로, 고유한 값를 만들기 위한 데이터 타입입니다.

```
// symbol
let tmp1 = Symbol("tmp");
let tmp2 = Symbol("tmp");

console.log(typeof tmp); // symbol
console.log(tmp1 === Symbol("tmp")); // false
console.log(tmp1 === tmp2); // false
```

객체 타입(참조타입)으로는

1. 객체
   - 자바스크립트에서 기본 타입을 제외한 모든 값은 객체로 취급됩니다. 따라서 배열이나 함수 등도 모두 객체로 표현됩니다. 주로 key-value 쌍의 데이터를 저장합니다.

```
/* 객체타입  */
let obj = {
    name: "elice AI",
    age: 6,
    isSmart: true,
    getInfo: function () {
        return this.name + "_" + this.age + "기는 smart합니까? : " + this.isSmart;
    },
};

console.log(obj.name, obj.age, obj.isSmart); // elice AI 6 true
console.log(obj.getInfo()); // elice AI_6기는 smart합니까? : true
```

가 있습니다.

▼ String + Number가 가능하다고요?

자바스크립트는 가능합니다. +연산의 경우 String의 우선순위를 더 높게 치기 때문에 String + Number 같은 경우 string으로 형을 변환하여 계산합니다.

```
//  + 연산자는 string이 더 우선 순위가 높음
console.log(1 + 1);
console.log(1 + "1");
console.log("1" + "1");

// - 연산자는 number가 더 우선순위가 높음
console.log(1 + 1 - 1);
console.log(1 + 1 - "1");
console.log(1 + "1" - "1");
```

JS는 웬만하면 에러를 내지 않게끔 설계된 언어기 때문에 알아서 형을 변환하고 처리를 해줍니다.

이런 유연함은 JS의 장점임과 동시에 단점이 되어 헬버깅의 원인이 됩니다. 개발자의 의도치 않은 값이나 실수도 잘 작동되어 나중에 찾기 힘들어진다는 점 때문이죠. 이러한 것들로 인해 TypeScript가 나오게 되었습니다.

▼ ==, === 차이점은?

==는 Equal Operator이고, ===는 Strict Equal Operator입니다.

==는 a == b 옆에, a와 b의 값이 같은 판단해서, 같으면 true, 다르면 false라고 합니다.(값만 같으면 true)

===는 엄격한 Equal 연산자로 data type까지 비교를 하며 같으면 true, 다르면 false라고 합니다.

```
// ==
console.log(10 == 10); // true
console.log(10 == "10"); // true
console.log(true == 1); // true
console.log(true == "1"); // true
console.log(true == "true"); // false
console.log(null == undefined); // true

// ===
console.log(10 === 10); // true
console.log(10 === "10"); // false
console.log(true === 1); // false
console.log(true === "true"); // false
console.log(null === undefined); // false
```