클로저 2부

선행학습

- 스코프 : 모든 식별자(변수 이름, 함수 이름, 클래스 이름등) 자신이 선언된 위치에 의해 다른 코드가 식별자 자신을 참조할 수 있는 유효 범위가 결정 된다. 즉, 식별자가 유효한 범위
  - 렉시컬스코프 : 함수를 어디서 호출하는지가 아닌 어디에 선언하였는지에 따라 결정됨.
  - 함수스코프 : var
  - 블록스코프 : let, const
- this 바인딩
  - web에서 전역 객체는 window
  - node에서 module.exports, exports
  - 함수선언식, arrow function
    - 함수선언식 안에서 this는 항상 global객체를 이미 한다. 함수선언식은 함수가 호출, 실행될 때 this가 동적으로 결정된다. (apply, call, bind 메서드 사용해 this 바인딩 하기)
    - arrow function 은 this 바인딩 객체가 선언할 때 정적으로 결정 된다. 즉 상위 스코프의 this를 가르킨다. Lexical this 라한다.

- 즉시실행함수
  - 즉시 실행 함수를 사용하는 이유
    - 초기화 : 변수를 전역으로 선언하는 것을 피하기 위해, 전역에 변수를 추가하지 않아도 되기 때문에 코드충돌 없이 구현 할 수 있어, 플러그인이나 라이브러리 등을 만들때 많이 사용 됩니다.

```
(function(){
  // statements
})();

// 기명 즉시 실행 함수
(function square(x){
  console.log(x);
})(2);

(function square(x) {
  console.lof(x*x);
}(2));

// 익명 즉시 실행 함수
(function (x) {
  console.log(x);
})(2);

(function (x) {
  console.log(x);
}(2));

// 변수에 즉시 실행 함수 저장
var mysquare;
(mysquare = function(x) {
    console.log(x * x);
})(2);
mysquare(3);

// 변수에 즉시 실행 함수의 리턴 값 저장도 가능
var mySquare = (function(x) {
  return x * x;
})(2);
console.log(mySquare);
```

예제

카운트를 증가 시키는 함수

```
const count = (function(){
    var i = 0;
```

```
    return function() {
        return i++;
    }
}());
```

토글 함수

```
const toggle = (function () {
    var togg = false;
    return function () {
        togg = !togg;
        return togg;
    }
}());
```

클래스같이 사용하는 경우

```
function Member() {
    var name = '클로저이름';
    return {
        getName: function() {
            return name;
        }
    }
}
```

클로저, this 바인딩

```
function Person() {
    this.name = "Object this 인가?"; // this -> window
    return {
        name: this.name, // window this.name -> object name
        getName: function() {
            console.log(this); // this -> object name
        },
        getname: () => {
            console.log(this); // this -> window
        }
    }
}
```