코어 자바스크립트

5강 클로저

참고문헌

- 코어자바스크립트
- 모던자바스크립트 DeepDive

클로저(Closure)?

closure [klouʒə(r)]

명사

1. 폐쇄, 마감
2. 폐점, 휴업

클로저는 여러 함수형 프로그래밍 언어에 등장하는 보편적인 특징

ex) 스칼라, 하스켈, 리스프...

*** MDN에서 클로저의 정의**

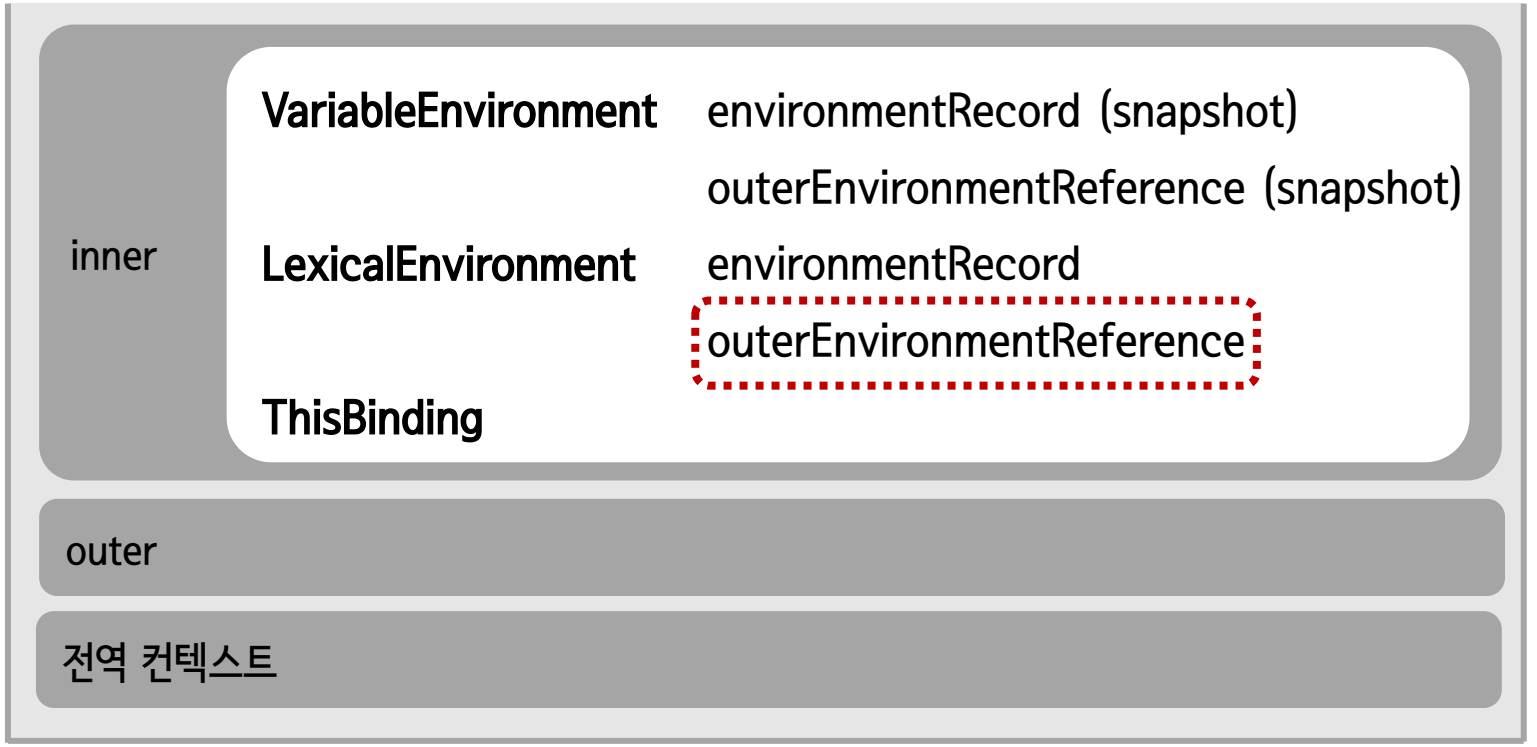
A closure is the combination of a function and the lexical environment within which that function was declared

(클로저는 함수와 그 함수가 선언될 당시의 lexical environment의 상호관계에 따른 현상)

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment).

(클로저는 주변상태(lexical environment)에 대한 참조와 함께 묶인(닫힌) 함수의 조합)

클로저 lexical environment

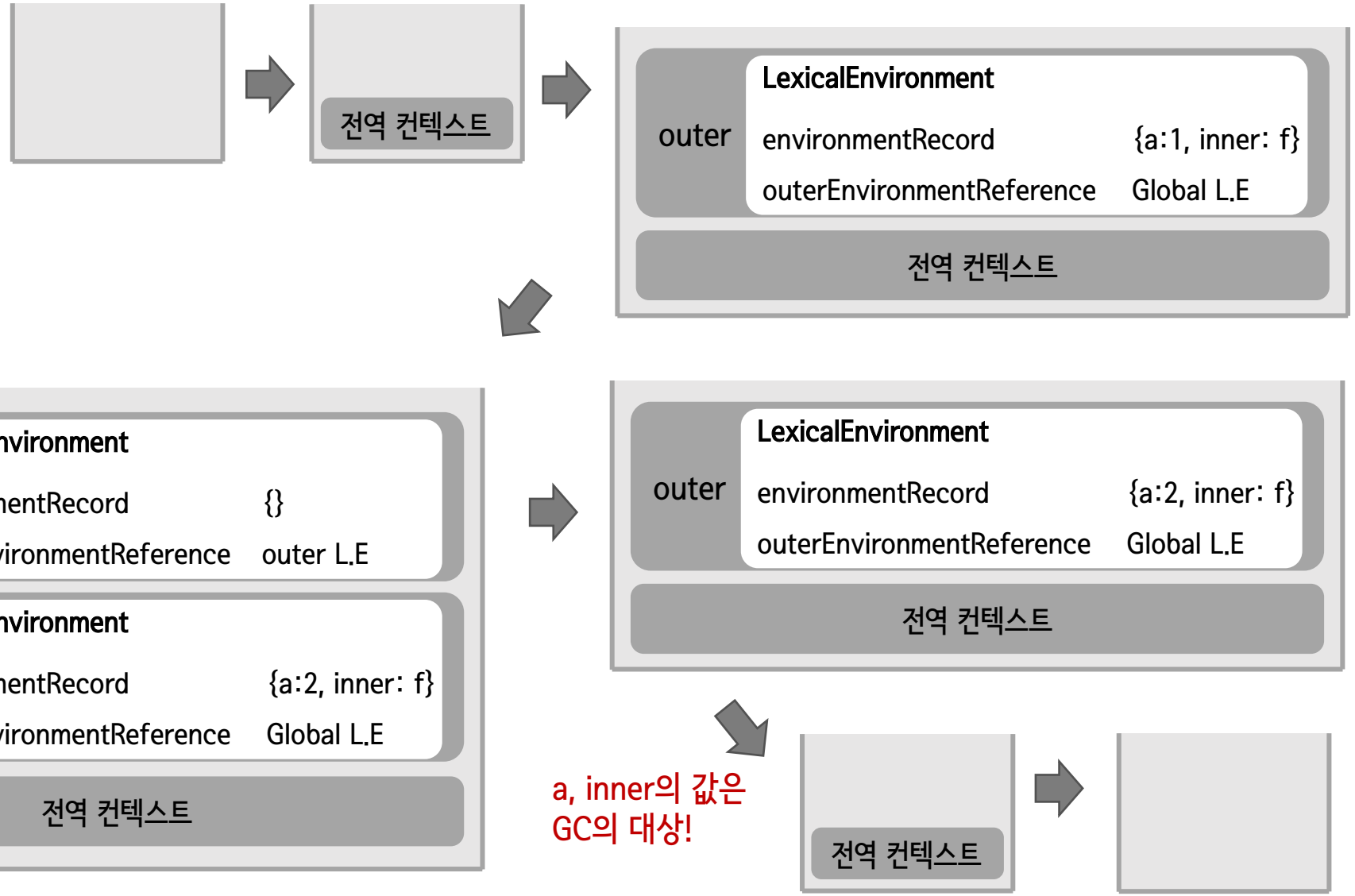


클로저의 Lexical environment는 엄밀히 말하면, outerEnvironmentReference에 해당

*outerEnvironmentReference 변수의 유효범위인 스코프가 결정되고, 스코프 체인을 가능하게 함.

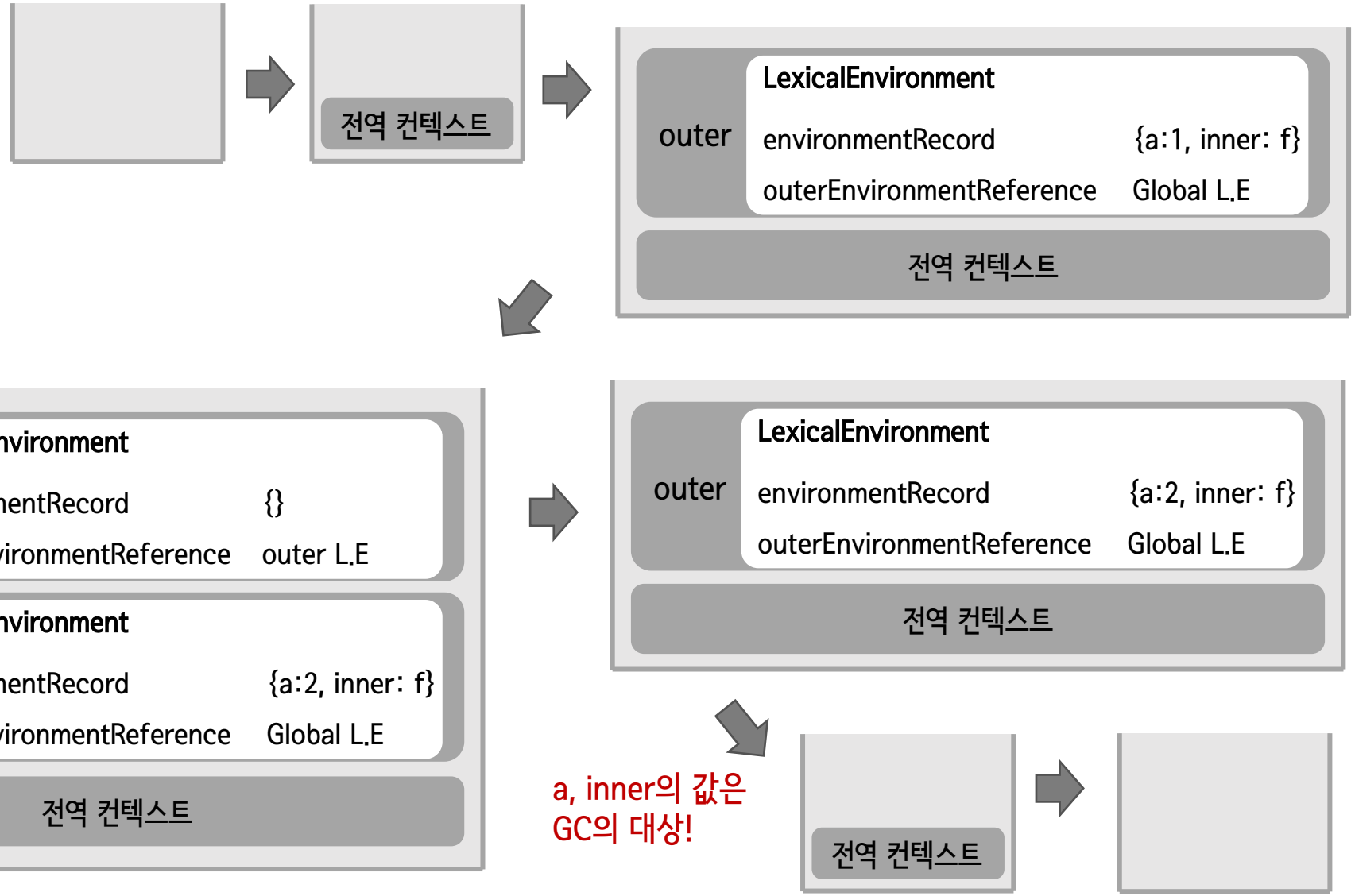
클로저 외부 함수의 변수를 참조하는 내부 함수

```
var outer = function () {
  var a = 1;
  var inner = function () {
    console.log(++a);
  };
  inner();
};
outer();
```



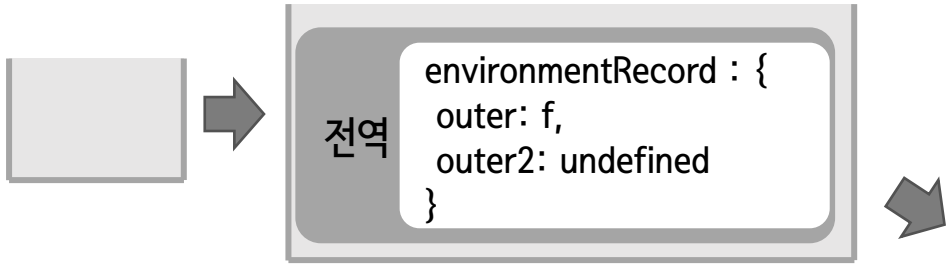
클로저 외부 함수의 변수를 참조하는 내부 함수(실행 결과 반환)

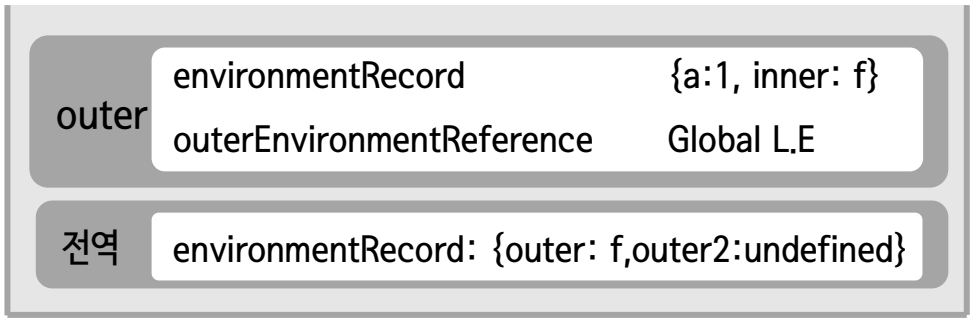
```
var outer = function () {
  var a = 1;
  var inner = function () {
    return ++a;
  };
  return inner();
};
var outer2 = outer();
console.log(outer2); //2
```

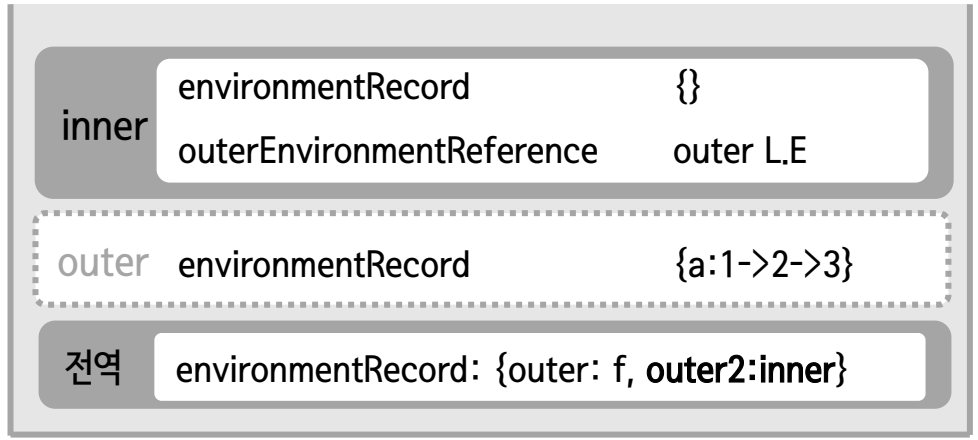


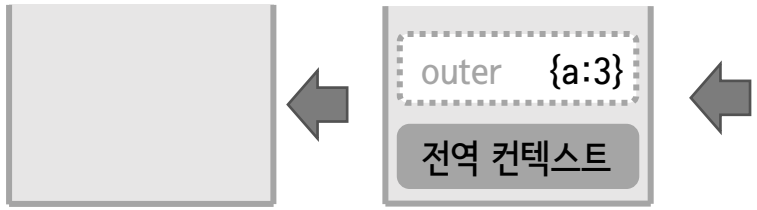
클로저 외부 함수의 변수를 참조하는 내부 함수(함수 자체를 반환)

```
var outer = function () {
  var a = 1;
  var inner = function () {
    return ++a;
  };
  return inner;
};
var outer2 = outer();
console.log(outer2()); //2
console.log(outer2()); //3
```









inner함수가 outer2에 의해 실행되면 outer의 L.E를 필요로 하게 됨으로 G.C의 대상에서 제외!

*G.C는 참조하는 변수가 하나라도 있으면 수집대상제외

*스펙상 L.E전부를 G.C하지 않지만 V8엔진은 실제 사용하는 변수만 남기고 나머지는 G.C함

클로저 정의

only return문?

클로저란 어떤 함수 A에서 선언한 변수a를 참조하는 내부함수 B를 외부로 전달한 경우 A의 실행 컨텍스트가 종료된 이후에도 변수 a가 사라지지 않는 현상

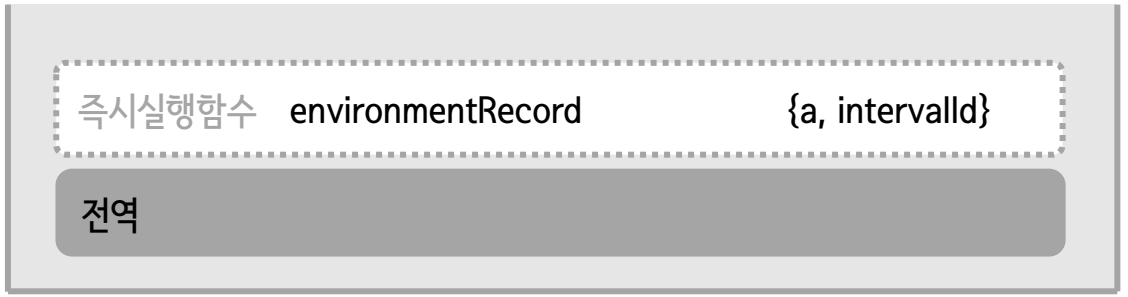
== 함수를 선언할 때 만들어지는 유효범위가 사라진 후에도 호출할 수 있는 함수

== 이미 생명주기가 끝난 외부 함수의 변수를 참조하는 함수

== 자신이 생성될 대의 스코프에서 알 수 있었던 변수들 중 언젠가 자신이 실행될 때 사용할 변수들만을 기억하며 유지시키는 함수

클로저 setInterval/setTimeout

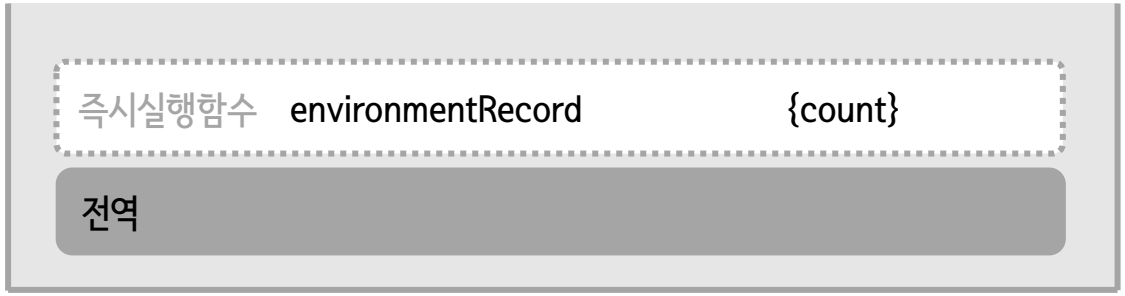
```
(function () {
  var a = 0;
  var intervalId = null;
  var inner = function() {
    if(++a >= 10){
      clearInterval(intervalId);
    }
    console.log(a);
  };
  intervalId = setInterval(inner,
1000);
})();
```



별도의 외부객체인 window의
메서드(setTimeout/setInterval)에 전달할 콜백 함수
내부에서 지역변수 참조

클로저 eventListener

```
(function () {
  var count = 0;
  var button = document.createElement('button');
  button.innerText = 'click';
  button.addEventListener('click', function () {
    console.log(++count, 'times clicked');
  })
  document.body.appendChild(button);
})();
```



별도의 외부 객체인 DOM의 메서드(addEventListner)에
등록할 handler 함수 내부에서 지역변수를 참조

클로저 메모리 관리

클로저는 메모리 누수 위험이 있다?

=> NO 메모리 소모는 클로저의 특성일 뿐!

* 메모리 누수는 개발자의 의도가 아닌데 참조카운트가 0이 아니라 GC의 수거대상이 되지 않은 것이고, 의도적으로 참조카운트가 0이 되지 않게 한 것은 누수라고 할 수 없음!

클로저에서 메모리 관리는?

=> 필요성이 사라진 시점에 메모리를 소모하지 않도록 참조 카운트를 0으로 만든다!

* 식별자에 참조형이 아닌 기본형 데이터(보통 null이나 undefined) 할당

클로저 메모리 관리 예시

return에 의한 클로저의 메모리 해제

```
var outer = (function () {
  var a = 1;
  var inner = function () {
    return ++a;
  };
  return inner;
})();
console.log(outer());
console.log(outer());
outer = null; //outer 식별자의 inner 함수 참조를 끊음
```

setInterval에 의한 클로저의 메모리 해제

```
(function (){
  var a = 0;
  var intervalId = null;
  var inner = function () {
    if(++a >= 10) {
      clearInterval(intervalId);
      inner = null; //inner 식별자의 함수 참조를 끊음
    }
    console.log(a);
  }
  intervalId = setInterval(inner, 1000);
})();
```

eventListener에 의한 클로저의 메모리 해제

```
(function (){
  var count = 0;
  var button = document.createElement('button');
  button.innerText = 'click';

  var clickHandler = function () {
    console.log(++count, 'times clicked');
    if(count >= 10){
      button.removeEventListener('click',clickHa
ndler);
      clickHandler = null; //clickHandler 식별자의
함수 참조를 끊음
    }
  };
  button.addEventListener('click', clickHandler);
  document.body.appendChild(button);
})();
```

클로저 활용 사례 콜백 함수 내부에서 외부 데이터를 사용하고자 할 때

```
var fruits = ['apple', 'banana', 'peach'];
var $ul = document.createElement('ul');

fruits.forEach(function (fruit) {
  var $li = document.createElement('li');
  $li.innerText = fruit;
  $li.addEventListener('click', function () {
    alert('your choice is ' + fruit);
  })
  $ul.appendChild($li);
});
document.body.appendChild($ul);
```

$li.addEventlistener에 넘겨준 콜백함수에서 fruit라는 외부 변수를 참조하고 있으므로 클로저가 있음.

클로저 활용 사례 콜백 함수 내부에서 외부 데이터를 사용하고자 할 때

```
var fruits = ['apple', 'banana', 'peach'];
var $ul = document.createElement('ul');

var alertFruit = function (fruit) {
  alert('your choice is ' + fruit);
};
fruits.forEach(function (fruit) {
  var $li = document.createElement('li');
  $li.innerText = fruit;
  $li.addEventListener('click', alertFruit);
  $ul.appendChild($li);
});
document.body.appendChild($ul);
alertFruit(fruits[1]);
```

alert를 띄우는 콜백 함수를 다른 곳에서도 사용한다면, fruit를 인자로 받아 출력하는 형태의 함수를 구현하는 게 반복을 줄일 수 있음

But 다음과 같이 구현하면 콜백함수에는 [Object MouseEvent]라는 결과가 출력하는데 addEventListener는 콜백 함수를 호출할 때 첫 번째 인자에 이벤트 객체를 주입하기 때문!

클로저 활용 사례 콜백 함수 내부에서 외부 데이터를 사용하고자 할 때

```
var fruits = ['apple', 'banana', 'peach'];
var $ul = document.createElement('ul');

var alertFruit = function (fruit) {
  alert('your choice is ' + fruit);
};
fruits.forEach(function (fruit) {
  var $li = document.createElement('li');
  $li.innerText = fruit;
  $li.addEventListener('click',
alertFruit.bind(null, fruit));
  $ul.appendChild($li);
});
document.body.appendChild($ul);
alertFruit(fruits[1]);
```

***bind 활용**

1)이벤트 객체의 인자 순서가 바뀜!

이전에는 arguments의 첫번째가 PointerEvent였으나, 지금은 첫번째 인자가 fruit가 되고, 두번째 인자가 PointerEvent임

2)함수 내부의 this가 원래와 달라질 수 있음!

bind함수에 첫번째인자는 thisArg로 this값을 명시해주는 것인데, 여기에 null로 인자를 주면 this값은 항상 전역객체로 바뀜

*참고 사항 고차 함수

고차 함수(Higher-Order-Function)는 함수를 인수로 전달받거나 함수를 반환하는 함수.

자바스크립트에서 함수는 일급 객체여서 함수를 값처럼 인수로 전달할 수 있고 반환 할 수 있다.

* 일급 객체란?

1. 무명의 리터럴 생성 가능. 즉, 런타임에 생성 가능
2. 변수나 자료구조(객체, 배열 등)에 저장 가능
3. 함수의 매개변수에 전달 가능
4. 함수의 반환값으로 사용 가능

```
const increase = function (num) {
  return ++num;
}//1, 2
const decrease = function (num) {
  return --num;
}//1, 2
const auxs = { increase, decrease };//2
function makeCounter(aux) {
  let num = 0;
  return function () { //4
    num = aux(num);
    return num;
  };
}
const increaser =
makerCounter(auxs.increase);//3
const decreaser =
makerCounter(auxs.decrease);//3
```

클로저 활용 사례 콜백 함수 내부에서 외부 데이터를 사용하고자 할 때

```
var fruits = ['apple', 'banana', 'peach'];
var $ul = document.createElement('ul');
var alertFruitBuilder = function (fruit) {
  return function() {
    alert('your choice is ' + fruit);
  };
};
fruits.forEach(function (fruit) {
  var $li = document.createElement('li');
  $li.innerText = fruit;
  $li.addEventListener('click', alertFruitBuilder(fruit));
  $ul.appendChild($li);
});
document.body.appendChild($ul);
alertFruitBuilder(fruits[1])();
```

***고차 함수 + 클로저 활용**

alertFruitBuiler는 alert를 띄우는 익명함수를 반환함.

addEventListener의 콜백함수는 alertFruitBuilder에 인자를 fruit로 넘겨주고 반환된 함수

alertFruitBuilder의 environmentRecord에는 fruit가 GC의 수거대상이 되지 않고 남아있음.

클로저 활용 사례 접근 권한 제어(정보 은닉)

정보 은닉(information hiding)은 어떤 모듈의 내부 로직에 대해 외부로의 노출은 최소화해서 모듈 간의 결합도를 낮추고 유연성을 높이는 것.

public	외부에서 접근 가능
protected	외부에서 접근 불가능. 해당 클래스를 상속받은 클래스나 해당 클래스에서만 접근 가능
private	내부에서만 사용 가능 *ES2019부터 선두에 #을 붙여서 private 필드를 정의할 수 있다

클로저 활용 사례 접근 권한 제어(정보 은닉)

```
var outer = function () {
  var a = 1;
  var inner = function () {
    return ++a;
  };
  return inner;
};
var outer2 = outer();
console.log(outer2()); //2
console.log(outer2()); //3
```

outer함수를 종료할 때, inner함수를 반환하여 outer함수의 지역변수인 a의 값을 외부에서 읽을 수 있음. 이와 같이 클로저를 활용하면 함수 내부 변수들 중 선택적으로 일부의 변수에 대한 접근 권한을 부여할 수 있음!

outer함수는 외부인 전역 스코프와 격리된 공간이고, 외부에서는 outer함수가 return한 정보에만 접근 가능.
return되지 않은 변수들은 private으로, return된 변수들은 public으로 볼 수 있음!

클로저 활용 사례 접근 권한 제어(정보 은닉)

```
var createCar = function () {
  var fuel = Math.ceil(Math.random() * 10 + 10);
  var power = Math.ceil(Math.random() * 3 + 2);
  var moved = 0;
  return {
    get moved(){
      return moved;
    },
    run: function(){
      var km = Math.ceil(Math.random() * 6);
      var wasterFuel = km / power;
      if(fuel < wasterFuel){
        console.log('이동 불가');
        return;
      }
      fuel -= wasterFuel;
      moved += km;
      console.log(`${km}km 이동 (총 ${moved}km). 남은 연료: ${fuel}`);
    }
  };
}
var car = createCar();
```

클로저를 활용해서 외부에서는 moved값을

확인하거나, run메서드를 실행하는 것만 가능!

fuel나 power에 접근하는 것이 불가능

*fuel, power는 private

```
car.run = function(){
      console.log('이동 불가');
}
```

와 같이 car.run이 외부에서도 수정 가능

클로저 활용 사례 접근 권한 제어(정보 은닉)

```
var createCar = function () {
  var fuel = Math.ceil(Math.random() * 10 + 10);
  var power = Math.ceil(Math.random() * 3 + 2);
  var moved = 0;
  var publicMembers = {
    get moved(){
      return moved;
    },
    run: function(){
      var km = Math.ceil(Math.random() * 6);
      var wasterFuel = km / power;
      if(fuel < wasterFuel){
        console.log('이동 불가');
        return;
      }
      fuel -= wasterFuel;
      moved += km;
      console.log(`${km}km 이동 (총 ${moved}km). 남은 연료: ${fuel}`);
    }
  };
  Object.freeze(publicMembers);
  return publicMembers;
}
var car = createCar();
```

car.run = function(){

console.log('이동 불가');

}

와 같이 car.run을 외부에서 수정 불가능

* Object.freeze(obj)를 사용하면 객체를 동결함. 동결된 객체에 새로운 속성을 추가하거나, 존재하는 속성을 제거하는 것을 방지.

클로저 활용 사례 부분 적용 함수

부분 적용 함수(partially applied function)는 n개의 인자를 받는 함수에 미리 m개의 인자만 넘겨 기억시켰다가, 나중에 (n-m)개의 인자를 넘기면 원래 함수의 실행결과를 얻을 수 있는 함수.

bind 메서드 부분 적용 함수

```
var add = function () {
  var result = 0;
  for(var i = 0; i < arguments.length; i++){
    result += arguments[i];
  }
  return result;
};
var addPartial = add.bind(null, 1, 2, 3, 4, 5);
console.log(addPartial(6, 7, 8, 9, 10)); //55
```

BUT this에 임의로 null을 넣어줬기 때문에 원래의 this와 달라지게 됨!

클로저 활용 사례 부분 적용 함수

클로저 부분 적용 함수

```
var partial = function () {
  var originalPartialArgs = arguments;
  var func = originalPartialArgs[0];
  if(typeof func !== 'function'){
    throw new Error('첫번째 인자가 함수가 아닙니다.');
  }
  return function() {
    var partialArgs = Array.prototype.slice.call(originalPartialArgs, 1);
    var resArgs = Array.prototype.slice.call(arguments);
    return func.apply(this, partialArgs.concat(resArgs));
  };
};
```

클로저 부분 적용 함수(spread operator)

```
...(중략)
  return function() {
    var partialArgs = [...originalPartialArgs].slice(1);
    var resArgs = arguments;
    return func.apply(this, [...partialArgs, ...resArgs]);
  };
```

BUT 인자는 반드시 앞에서부터 차례로 전달할 수 밖에 없다!

클로저 활용 사례 부분 적용 함수

클로저 부분 적용 함수

```
var partial = function () {
  var originalPartialArgs = arguments;
  var func = originalPartialArgs[0];
  if(typeof func !== 'function'){
    throw new Error('첫번째 인자가 함수가 아닙니다.');
  }
  return function() {
    var partialArgs = Array.prototype.slice.call(originalPartialArgs, 1);
    var resArgs = Array.prototype.slice.call(arguments);
    return func.apply(this, partialArgs.concat(resArgs));
  };
};
```

```
var add= function(){
  var result = 0;
  for(var i=0; i < arguments.length; i++){
    result += arguments[i];
  }
  return result;
};
var addPartial = partial(add, 1,2,3,4,5);
console.log(addPartial(6,7,8,9,10));
```

```
var dog = {
  name: '강아지',
  greet: partial(function(prefix, suffix)
{
    return prefix + this.name + suffix;
  }, '왈왈, ')
};
dog.greet('입니다!');
```

클로저 활용 사례 부분 적용 함수

클로저 부분 적용 함수

```
Object.defineProperty(window,'_',{
  value: 'EMPTY_SPACE',
  writable: false,
  configurable: false,
  enumerable : false
});
var partial2 = function() {
  var originalPartialArgs = arguments;
  var func = originalPartialArgs[0];
  if (typeof func !== 'function') {
    throw new Error('첫 번째 인자가 함수가 아닙니다.');
  }
  return function () {
    var partialArgs = Array.prototype.slice.call(originalPartialArgs, 1);
    var restArgs = Array.prototype.slice.call(arguments);
    for(var i = 0; i < partialArgs.length; i++) {
      if (partialArgs[i] === _){
        partialArgs[i] = restArgs.shift();
      }
    }
    return func.apply(this, partialArgs.concat(restArgs));
  }
}
```

* Object.defineProperty 객체에 새로운 속성을 직접 정의하거나 이미 존재하는 속성을 수정한 후, 해당 객체를 반환하는 메서드.

전역 객체에 '_'라는 이름을 갖는 속성을 정의하는데, 값은 EMPTY_SPACE라는 string이고, 할당 연산자로 속성값 변경 불가능(writable), 속성값 변경 및 삭제 불가능(configurable), 객체 속성 열거 시 속성이 비노출(enumerable)한 속성을 갖음.

_라는 전역 객체의 속성을 정의하여, 부분 적용 함수에 원하는 위치에 인자를 미리 넣고, 빈자리는 '비어있음'을 표시하는 _를 사용해 나중에 넘어온 인자들이 차례대로 끼어넣어지게 함!

클로저 활용 사례 부분 적용 함수

클로저 부분 적용 함수

```
var _ = Symbol.for('EMPTY_SPACE');
var partial2 = function() {
  var originalPartialArgs = arguments;
  var func = originalPartialArgs[0];
  if (typeof func !== 'function') {
    throw new Error('첫 번째 인자가 함수가 아닙니다.');
  }
  return function () {
    var partialArgs = Array.prototype.slice.call(originalPartialArgs, 1);
    var restArgs = Array.prototype.slice.call(arguments);
    for(var i = 0; i < partialArgs.length; i++) {
      if (partialArgs[i] === _){
        partialArgs[i] = restArgs.shift();
      }
    }
    return func.apply(this, partialArgs.concat(restArgs));
  }
}
```

Object.defineProperty를 사용하면 전역공간을 침범하므로, ES6에서 나온 Symbol.for메서드를 사용하는 것이 좋음! Symbol.for메서드는 전역 심볼공간에 인자로 넘어온 문자열이 있으면 해당 값을 참조하고, 선언되어 있지 않으면 새로 만든다. 어디서든 접근 가능하면서 유일무이한 상수가 됨.

부분 적용 함수 활용 사례 디바운스

디바운스(debounce)란 짧은 시간 동안 동일한 이벤트가 많이 발생할 경우 이를 전부 처리하지 않고 처음 또는 마지막에 발생한 이벤트에 대해 한 번만 처리하는 것.

프론트엔드 성능 최적화에 도움을 주는 기능 중 하나

scroll, wheel, mousemove, resize 등에 적용하기 좋음

Lodash* 등의 라이브러리를 이용하면 디바운스 사용 가능

최소한의 기능(마지막 이벤트만 처리해도 됨, 시간 지연 상관 없음)을 구현하고자 하면 간단.

부분 적용 함수 활용 사례 디바운스

```
var debounce = function (eventName, func, wait) {
  var timeoutId = null;
  return function (event) {
    var self = this;
    console.log(eventName, 'event 발생');
    clearTimeout(timeoutId);
    timeoutId = setTimeout(func.bind(self,event), wait);
  };
};
var moveHandler = function (e) {
  console.log('move event 처리');
};
var wheelHandler = function (e) {
  console.log('wheel event 처리');
};
document.body.addEventListener('mousemove', debounce('move', moveHandler,500));
document.body.addEventListener('mousewheel', debounce('wheel', wheelHandler,700));
```

디바운스 함수는 eventName과 실행할 함수(func), 대기시간(ms)를 받는다. 클로저로 처리되는 변수는 앞의 3개의 인자와 timeoutId이다. 마지막으로 발생한 이벤트를 처리하기 위해, 대기시간 동안 새로운 이벤트가 발생하게 되면 clearTimeout으로 대기열을 초기화하기 때문에 마지막으로 발생한 이벤트만 처리된다.

클로저 활용 사례 커링 함수

커링 함수(currying function)이란 여러 개의 인자를 받는 함수를 하나의 인자만 받는 함수로 나눠서 순차적으로 호출될 수 있게 체인 형태로 구성한 것
부분 적용 함수와 유사하나, 한 번에 하나의 인자만 전달하는 것과 마지막 인자가 전달되기 전까지 원본 함수가 실행되지 않는 차이가 있음

```
var curry3 = function (func) {
  return function (a) {
    return function (b) {
      return func(a, b);
    };
  };
};

var getMaxWith10 = curry3(Math.max)(10);
console.log(getMaxWith10(8)); //10
console.log(getMaxWith10(25)); //25

var getMinWith10 = curry3(Math.min)(10);
console.log(getMinWith10(8)); //8
console.log(getMinWith10(25)); //10
```

클로저 활용 사례 커링 함수

인자가 많아지면 들여쓰기가 깊어져서 가독성이 떨어짐!
ES6 화살표 함수로 표현하면 간결하게 표현 가능
*마지막 단계 호출 전까지 인자들은 GC의 수거대상이 되지 않음!

```
var curry5 = function (func) {
  return function (a) {
    return function (b) {
      return function (c) {
        return function (d) {
          return function (e) {
            return func(a, b, c, d, e);
          };
        };
      };
    };
  };
};
var getMax = curry5(Math.max);
console.log(getMax(1)(2)(3)(4)(5));
```

```
var curry5 = func => a => b => c => d => e => func(a, b, c, d, e);
var getMax = curry5(Math.max);
console.log(getMax(1)(2)(3)(4)(5));
```

커링 함수 활용 사례 지연 실행

지연 실행(laze execution)은 당장 필요한 정보만 받아서 전달하고 또 필요한 정보가 들어오면 전달하는 식으로

마지막 인자가 넘어갈 때까지 함수 실행을 미루는 것.

지연 실행에 커링 함수를 활용할 수 있다!

* 자주 쓰이는 함수의 매개변수가 항상 비슷하고 일부만 바뀔 때도 커링 함수 사용 가능

커링 함수 활용 사례 지연 실행

```
var getInformation = function (baseUrl) {
  return function (path) {
    return function (id) {
      return fetch(baseUrl + path + '/' + id);
    };
  };
};
//var getInformation = baseUrl => path => id => fetch(baseUrl + path + '/' + id);

var imageUrl = 'http://imageAddress.com/';

var getImage = getInformation(imageUrl); //http://imageAddress.com/
var getEmoticon = getImage('emoticon'); //http://imageAddress.com/emoticon
var getIcon = getImage('icon'); //http://imageAddress.com/icon

var emoticon1 = getEmoticon(100); //http://imageAddress.com/emoticon/100
var icon1 = getIcon(205); //http://imageAddress.com/icon/205
```

fetch함수는 url을 받아 해당 url에 HTTP요청을 함. 이때 REST API를 이용하면 baseUrl은 고정이고 path나 id값은 매우 많을 수 있다!
이때 baseUrl부터 공통적인 요소는 기억시켜두고 특정한 값만 서버 요청을 수행하는 함수를 만들어 두면 효율성이나 가독성을 향상시킬 수 있음!

커링 함수 활용 사례 Redux의 미들웨어

Redux는 자바스크립트의 상태관리 라이브러리.
* store는 상태가 관리되는 하나의 공간
* action은 앱에서 스토어에 운반할 데이터

```
const logger = store => next => action => {
  console.log('dispatching', action);
  console.log('next state', store.getState());
  return next(action);
};
const thunk = store => next => action => {
  return typeof action === 'function' ? action(dispatch, store.getState) :
next(action);
};
```

QnA 지수님 스터디 참고

- 클로저란 무엇인가요?
- 내부 함수에서 외부 함수의 변수를 참조하는 과정을 실행 컨텍스트를 이용하여 설명해주세요.
- 클로저에서 외부 함수의 실행 컨텍스트가 종료된 상태인데, 어떻게 내부 함수가 외부 함수에 접근할 수 있는지 설명해주세요.
- 내부 함수를 외부로 전달하는 예시에는 어떠한 것들이 있나요?
- 클로저는 곧 메모리 소모이기도 합니다. 이를 잘 관리하는 방법에는 어떠한 것이 있나요?
- 클로저의 활용 사례에는 어떠한 것들이 있나요?
- 콜백 함수 내부에서 외부 데이터를 사용하는 방법에는 무엇이 있고, 각 방법의 장, 단점을 설명해주세요.
- 클로저를 활용하여 변수를 보호하는 방법에 대해서 설명해주세요.
- 부분 적용 함수와 커링 함수의 차이점을 설명해보세요.
- 부분 함수를 사용하기에 적합한 기능에 대해 설명해보세요.
- 콜백함수를 만들 때 일급객체여야 된다는 조건이 있는데 일급객체란 무엇인가요?
- 클로저가 왜 중요할까요?
- 클로저와 Lexical scope와 연관지어서 설명해주세요