4강

클로저

WHATEVER YOU WANT, MAKE IT REAL.

강사 정길용

LIKE LION

{

클로저란?

클로저 용법

즉시 실행 함수

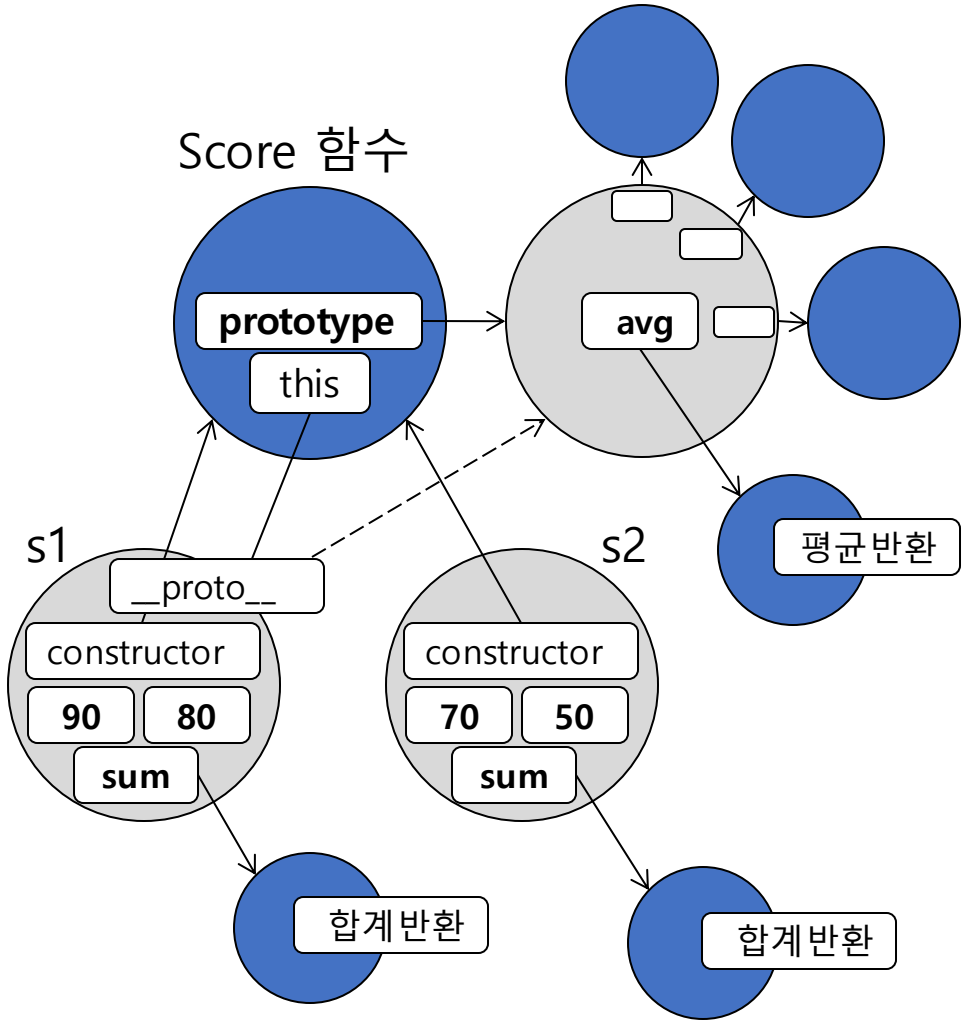
}

▶ prototype과 주변 객체의 참조관계

```
function Score(kor, eng){
  this.kor = kor;
  this.eng = eng;
  this.sum = function(){
      return this.kor+this.eng;
  };
}

Score.prototype.avg = function(){
  return this.sum()/2;
};

const s1 = new Score(90, 80);
const s2 = new Score(70, 50);
s1.sum();
s1.avg();
s2.sum();
s2.avg();
```



LIKE LION

ex04-01.js

▶ 클로저란?

- 실행이 끝난 외부 함수의 변수에 접근할 수 있는 내부 함수
- 함수가 생성되는 시점을 기준으로 접근 가능한 변수는, 그 유효범위가 사라진 후에도 접근 가능
- 클로저로 인해 유효 범위가 사라진 변수와 함수를 사용할 수 있고, 변수의 경우 그 값을 변경할 수도 있다.

```
function outer(){
  const innerVal = "outer의 지역변수";
  const innerFn = () => {
    console.log(innerVal);
  }
  return innerFn;
}
const inner = outer();
inner();
```

→ outer의 지역변수

LIKE LION

ex04-02.js

▶ 캡슐화

- 객체 내부에서만 접근 가능한 속성을 만들어 사용하고 외부에서는 해당 속성을 직접 접근하지 못하도록 만드는 객체지향 언어의 특징(C++, Java에서는 private 키워드로 지정가능)
- 함수 내부에서 선언한 지역변수는 외부에서 접근하지 못하는 반면 내부 메서드인 클로저에서는 접근 가능하다는 특징을 이용해서 구현
- ES2019에서 class 정의시 속성명이나 메서드명 앞에 #을 붙이면 해당 class 내부에서만 접근 가능한 private 속성과 메서드 정의 기능이 추가됨

```
function Counter(){
  let count = 0;
  this.ride = function(){
    count++;
  };
  this.getCount = function(){
    return count;
  };
}
const c = new Counter();
c.ride();
c.ride();
c.count += 100;
console.log(c.getCount());
```

→ 102

▶ 콜백과 타이머

- 지정된 함수들이 임의의 시간 뒤에 비동기적으로 호출이 될 때 함수 외부의 데이터에 접근하는 경우

```
function setTimer(){
  const inner = 100;
  setTimeout(() => {
    console.log(inner);
  }, 1000);
}

setTimer();
```

LIKE LION

ex04-03.js

Currying

- 여러개의 인자를 받는 함수를 단일 인자를 받는 함수의 체인으로 호출하도록 바꾸는 함수형 프로그래밍 기법 중 하나
- sum(x, y) -> sum(x)(y)
- 함수형 프로그래밍 언어에 많은 공헌을 한 미국의 수학자, 논리학자인 하스켈 커리의 이름에서 따옴
- 함수의 가독성, 재사용이 좋아짐.
- 마지막 인자가 입력될 때까지 함수의 실행 타이밍을 조절할 수 있음.

```
function sum(a, b, c){
  return a + b + c;
}

const currySum = a => b => c => a + b + c;

console.log(sum(10, 20, 30));
console.log(currySum(10)(20)(30));
```

lodash의 _.curry() 함수

- 지정한 함수에 커링 기능을 추가하는 메서드

LIKE LION

ex04-04.js, ex02-17-04.js

Partial application(부분 적용 함수)

- 기존 함수의 매개변수 중 일부를 미리 채워둔 상태의 함수
- 커링된 함수를 일부 단계까지만 호출한 후 반환받은 함수를 나중에 나머지 인자를 전달해서 실행

lodash의 _.partial() 함수

- 지정한 함수에 Partial application 기능을 추가하는 함수

```
const sum = (x, y) => {
  return x + y;
}

const sum100 = _.partial(sum, 100);
console.log(sum100(200));
```

Function.prototype.bind() 메서드

- Partial application 기능을 구현한 메서드
- lodash의 _.partial() 함수와 비슷하나 미리 전달하는 인자에 this도 지정하는 기능이 추가

LIKE LION

ex04-05.js

▶ 존재하는 함수의 동작을 수정

- 클로저가 필요 없음
- 메모이제이션 예제

```
Function.prototype.memo = function(key){
  this._cache = this._cache || {};
  if(this._cache[key] !== undefined){
    return this._cache[key];
  }else{
    return this._cache[key] = this(key);
  }
};

isPrime(5); // 캐시되지 않음

isPrime.memo(1000000007); // 캐시됨
```

LIKE LION

ex04-06.js

정적 함수를 바탕으로 새로운 함수를 생성

- 클로저 이용
- 메모이제이션 예제

```
Function.prototype.memoize = function(){
  const fn = this;
  return function(){
    return fn.memo.apply(fn, arguments);
  };
};

isPrime(5); // 캐시되지 않음

isPrime = isPrime.memoize();

isPrime(5); // 캐시됨
```

▶ 즉시실행함수(Immediately-Invoked Function Expression)

- 함수 선언 후 곧바로 스스로를 호출하여 실행되는 함수

▶ 코드 실행 순서

- 함수 인스턴스를 생성한다.
- 함수를 실행한다.
- 함수를 폐기한다.(실행을 마치고 나면 더 이상 이 함수를 참조할 수 없음)

```
(function(){
        const msg = "함수 호출";
        console.log(msg);
})();
```

LIKE LION

ex04-07.js

```
const msg = '함수 호출';
console.log(msg);
```

```
const f = function(){
  const msg = '함수 호출';
  console.log(msg);
};
f();
```

```
(function(){
  const msg = '함수 호출';
  console.log(msg);
})();
```

```
const f = function(msg){
  console.log(msg);
};
f('함수 호출');
```

```
(function(msg){
  console.log(msg);
})('함수 호출');
```

▶ 임시 유효 범위와 private 변수

- 코드를 함수로 감싸고 호출하면 해당 코드의 유효 범위가 함수로 제한
- 함수 내에서 사용하는 변수는 외부에 노출되지 않으므로 외부 변수와 충돌이 발생하지 않는다.
- 즉, 외부에서 접근할 수 없는 독립적인 공간을 확보할 수 있음
- 특정 코드 블럭을 독립적인 모듈로 사용할 수 있음

```
let count = 0;
let sum = 100;
let avg = 0;
// ……
////////////////////////////////////////
(() => {
        let sum = 0;
        for(let i=1; i<=100; i++){
                sum += i;
        }
        console.log(sum);
})();
////////////////////////////////////////
// ……
```

▶ 라이브러리 래핑

- 자바스크립트 라이브러리 개발 시 임시 변수들을 즉시실행함수 내부에 묶어놓음으로써 전역 네임스페이스를 더럽히지 않는다.
- 여러 라이브러리를 로딩하면서 발생하는 이름 충돌을 막을 수 있다.

▶ 변수명 대체

- Some.long.reference.to.something 같은 복잡한 참조 관계를 짧은 변수로 대체

LIKE LION

ex04-08.html

▶ 루프에서 사용

- 루프 내부에서 하나의 변수를 사용할 경우
- 함수의 유효 범위안에서 각각의 클로저로 독립적인 변수 접근 가능

```
<button>버튼1</button>
<button>버튼2</button>
<script>
        const btn = $("button");
        for(let i=0; i<btn.size(); i++){
                ((i) => {
                        btn[i].onclick = () => {
                                alert(i);
                        };
                })(i);
        }
</script>
```