코어 JS - 5단원

05. 클로저

I. 클로저의 의미 및 원리 이해

MDN - 클로저는 함수와 그 함수가 선언될 당시의 lexical environment의 상호관계에 따른 현상

- 선언할 당시의 lexical environment == outerEnvironmentReference
- 내부 함수에서 외부 변수를 참조하는 경우에 한해서만 combination(=선언될 당시의 LexicalEnvironment와의 상호관계)이 의미가 있음

→ 클로저란? 어떤 함수에서 선언한 변수를 참조하는 내부함수에서만 발생하는 현상

outer의 실행 컨텍스트가 종료된 후에도 inner 함수를 호출할 수 있게 만들면?

- inner 함수 실행 시점에 outer 함수가 이미 종료되었어도 outer 함수의 LexicalEnvironmnet에 접근 가능함 ← 가비지 컬렉터의 동작 방식때문
- 가바지 컬렉터는 어떤 값을 참조하는 변수가 하나라도 있다면 그 값은 수집 대상에 포함 X
- 호출될 가능성, 참조할 예정인 다른 실행 컨텍스트가 있는 한 실행 종료 후에도 GC되지 않음

클로저란?

- 어떤 함수 A에서 선언한 변수 a를 참조하는 내부함수 B를 외부로 전달할 경우 A의 실행 컨텍스트가 종료된 이후에도 변수 a가 사라지지 않는 현상을 말함

II. 클로저와 메모리 관리

- 필요성이 사라진 시점에는 메모리를 소모하지 않도록 해주면 됨
- 식별자에 참조형이 아닌 기본형 데이터(null, undefined 등)을 할당하면 됨

III. 클로저 활용 사례

1. 콜백 함수 내부에서 외부 데이터를 사용하고자 할 때

- 콜백 함수 내부에서 외부 변수를 참조하기 위한 세 가지 방법

```
// 1. 콜백 함수를 내부함수로 선언해서 외부변수를 직접 참조하는 방법

var fruits = ['Apple', 'Banana', 'Orange']; // 과일 리스트 정의
var alertFruit = function (fruit) {
    alert('your choice is ' + fruit);
};

var $ul = document.createElement('ul'); // ul 요소 생성

fruits.forEach(function (fruit) {
    var $li = document.createElement('li');
    $li.innerText = fruit;
    $li.addEventListener('click', function() {
        alertFruit(fruit); // 각 과일을 인자로 넘겨줌
    });
    $ul.appendChild($li);
});

document.body.appendChild($ul);
alertFruit(fruits[1]); // 리스트의 두 번째 항목(Banana)을 바로 aler
```

```
// 2. bind를 활용하는 방법
-> 클로저는 발생 X
-> but, 이벤트 객체가 인자로 넘어오는 순서가 바뀜, 함수 내부에서의 this가
...
fruits.forEach(function (fruit) {
    var $li = document.createElement('li');
    $li.innerText = fruit;
    $li.addEventListener('click', alertFruit.bind(null, fruit
    $ul.appendChild($li);
```

```
});
...
```

```
// 3. 콜백 함수를 고차 함수로 바꿔서 클로저를 적극적으로 활용한 방법
...
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);
});
...
```

2. 접근 권한 제어(정보 은닉)

- **정보 은닉**: 어떤 모듈의 내부 로직에 대해 외부로의 노출을 최소화 → 모듈간의 결합도 ↓ & 유연성을 ↑ 하는 현대 프로그래밍 언어의 중요한 개념 중 하나
- JS는 변수 자체에 이러한 접근 권한을 직접 부여하도록 설계돼 있지는 않음
- 클로저를 이용하면 함수 차원에서 public / private 한 값 구분 가능
- return한 변수들은 public member (외부에 제공하고자 하는 정보들) / return 하지 않은 변수들은 private member(내부에서만 사용할 정보들)가 됨

3. 부분 적용 함수

- bind의 실행 결과가 부분 적용 함수임 - 근데 this를 바인딩 해야 함
- 부분 함수를 사용하기 적합한 예: 디바운스
- 디바운스란? 짧은 시간 동안 동일한 이벤트가 많이 발생할 경우 이를 전부 처리 X, 처음 or 마지막에 발생한 이벤트에 대해 한 번만 처리함
- Symbol.for? 전역 심볼공간에 인자로 넘어온 문자열이 이미 있으면 해당 값을 참조하고 선언돼 있지 않으면 새로 만드는 방식 // 어디서든 접근 가능하면서 유일무이한 상수

를 만들고자 할 때 적합

4. 커링 함수

- 여러 개의 인자를 받는 함수를 하나의 인자만 받는 함수로 나눠서 순차적으로 호출될 수 있게 체인 형태로 구성한 것
- 한 번에 하나의 인자만 전달하는 것을 원칙으로 함
- 중간 과정상의 함수를 실행한 결과는 그다음 인자를 받기 위해 대기만 할 뿐 마지막 인자가 전달되기 전까지는 원본 함수가 실행되지 X (지연 실행)

  ↔ 부분 적용 함수는 여러 개의 인자를 전달할 수 있고 실행 결과를 재실행할 때 원본 함수가 무조건 실행 됨

```
1. 아래와 같은 for 루프 내에서 클로저가 마지막 값만 참조하는 문제를 해결하
for (var i = 0; i < 5; i++) {
    setTimeout(function() {
        console.log(i);
    }, 100);
}

// 위 코드는 5, 5, 5, 5, 5를 출력함.

2. 클로저의 특징(or 왜 사용하는지) 3가지 설명

3. 클로저의 메모리 누수가 발생할 수 있는 경우와 이를 방지하기 위한 방법은?
```