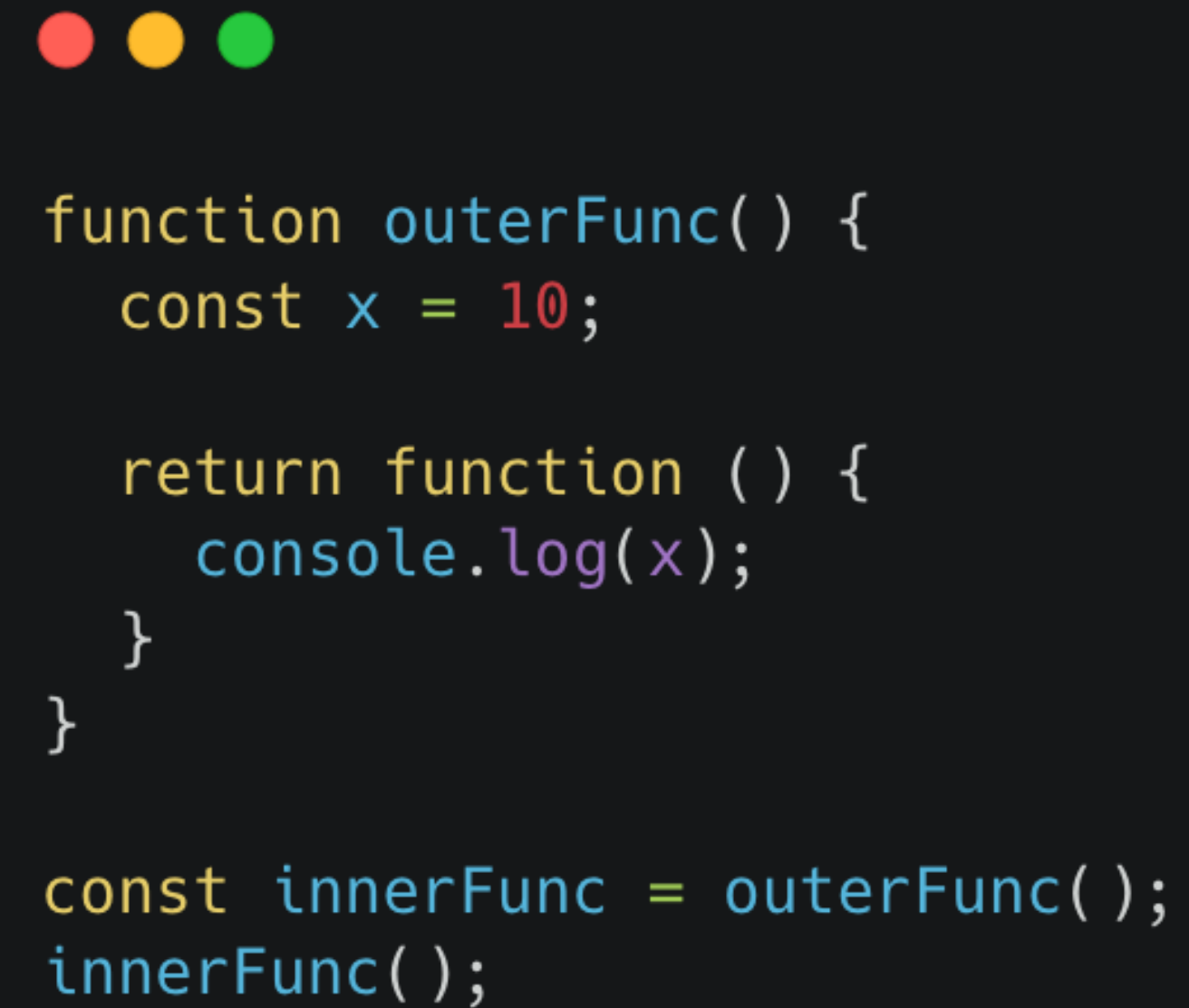


# 24장 클로저

# 클로저란

- MDN에서는 클로저를 “함수와 그 함수가 선언된 렉시컬 환경과의 조합”라고 정의한다.
- 함수는 자신이 정의된 환경을 기억한다



```
function outerFunc() {  
  const x = 10;  
  
  return function () {  
    console.log(x);  
  }  
}  
  
const innerFunc = outerFunc();  
innerFunc();
```

# 클로저와 렉시컬 환경

- outerFunc 함수를 호출하면 중첩 함수 innerFunc 를 반환하고, 생명 주기를 마감한다. (가바지 콜렉터)
- 즉, outerFunc의 실행 컨텍스트는 실행 컨텍스트 스택에서 제거되고, 변수 x 또한 생명 주기를 마감한다.
- 따라서 지역 변수인 x는 더는 유효하지 않게 되는 것처럼 보인다.
- 그러나, 외부 함수보다 중첩함수가 더 오래 유지되는 경우 중첩 함수는 이미 생명 주기가 종료된 외부 함수의 변수를 참조할 수 있다. 이러한 중첩 함수를 클로저라고 부른다.

```
function outerFunc() {  
  const x = 10;  
  
  return function () {  
    console.log(x);  
  }  
}  
  
const innerFunc = outerFunc();  
innerFunc();
```

# 가비지 콜렉터

- 가비지 콜렉터는 애플리케이션이 할당된 메모리 공간을 주기적으로 검사하여 더 이상 사용되지 않는 메모리를 해제하는 기능을 말한다.
- 더이상 사용하지 않는 메모리란 간단히 말하자면 어떤 식별자도 참조하지 않는 메모리 공간을 의미한다.
- 자바스크립트는 가비지 콜렉터를 내장하고 있는 언어로서 이를 통해 메모리 누수를 방지한다.
- 클로저는 외부 함수의 변수를 참조하기 때문에 변수는 가비지 콜렉션의 대상이 되지 않게 된다.

# 클로저와 렉시컬 환경

- 중첩 함수 innerFunc는 외부 함수 outerFunc보다 더 오래 생존하기 때문에, 외부 함수의 생존 여부와 상관없이 자신이 정의된 위치에 의해 설정된 상위 스코프를 기억한다.
- 이처럼 중첩함수 inner의 내부에서는 상위 스코프를 참조할 수 있으므로 상위 스코프의 식별자를 참조할 수 있고, 값을 변경할 수 있다.

```
function outerFunc() {  
  const x = 10;  
  
  return function () {  
    console.log(x);  
  }  
}  
  
const innerFunc = outerFunc();  
innerFunc();
```

# 클로저의 활용

- 클로저를 주로 상태를 안전하게 변경하고 유지하기 위해 사용한다.
- 즉, 상태를 은닉하고 특정 함수에게만 상태 변경을 허용하기 위해 사용한다.

# 클로저의 활용

- 카운트 상태를 전역 변수를 통해 관리되고 있기 때문에 언제든지 접근하고 변경할 수 있다. (암묵적 결합)
- 따라서 의도치 않게 상태가 변경될 수 있다는 것.

오류 가능성을 내포하는 코드 (클로저 X)

```
let num = 0;

const increase = function () {
  return +num;
}

console.log(increase()); // 1
console.log(increase()); // 2
console.log(increase()); // 3
```

# 클로저의 활용

- getIncrease 함수를 호출하면 increase 함수를 반환한다.
- Increase 함수는 클로저로, 자신이 정의된 환경을 기억한다. 즉, num 변수에 접근 가능하다.
- 카운트 상태(num 변수)에는 increase 함수만이 접근 가능하기 때문에 상태가 외부에서 접근하여 변경될 가능성이 없다.

오류 가능성을 제거한 코드 (클로저 O)

```
const getIncrease = function () {  
  let num = 0;  
  
  return function increase() {  
    return ++num;  
  }  
}  
  
const increase = getIncrease();  
  
console.log(increase()); // 1  
console.log(increase()); // 2  
console.log(increase()); // 3
```



# 클로저의 활용

```
function createCounter (initialValue) {  
  let num = initialValue;  
  
  return {  
    increase () {  
      return ++num;  
    },  
    decrease () {  
      return --num;  
    }  
  }  
}  
  
const counter = createCounter(5);  
counter.increase(); // 6  
counter.decrease(); // 5
```