02

고급 함수의 활용

학습내용

01 즉시 실행 함수(IIFE)

02 중첩 함수(Nested Function)

03 고차 함수(HOF)

04 콜백 함수(Callback Function)

학습목표

- 즉시 실행 함수(IIFE)의 개념과 목적을 설명하고, 직접 작성하여 실행할 수 있다.
- 중첩 함수의 구조를 이해하고, 외부 함수의 변수 접근 방식을 설명할 수 있다.
- 고차 함수의 개념을 이해하고, 함수를 인자로 받거나 반환하는 예제를 작성할 수 있다.
- 콜백 함수의 원리와 사용 사례(이벤트, 비동기 등)를 이해하고 적용할 수 있다.
- 고급 함수(고차 함수, 콜백 함수 등)의 관계를 분석하고 실용적인 코드로 응용할 수 있다.

01

즉시 실행 함수(IIFE)

즉시 실행 함수 (IIFE, Immediately Invoked Function Expression)

함수 정의와 동시에 즉시 호출되는 함수

- 반드시 그룹 연산자 괄호(...)로 감싸야 함
- 함수 이름이 없는 익명 함수를 사용하는 것이 일반적임

2) 그룹 연산자( )로 감싼 즉시 실행 함수

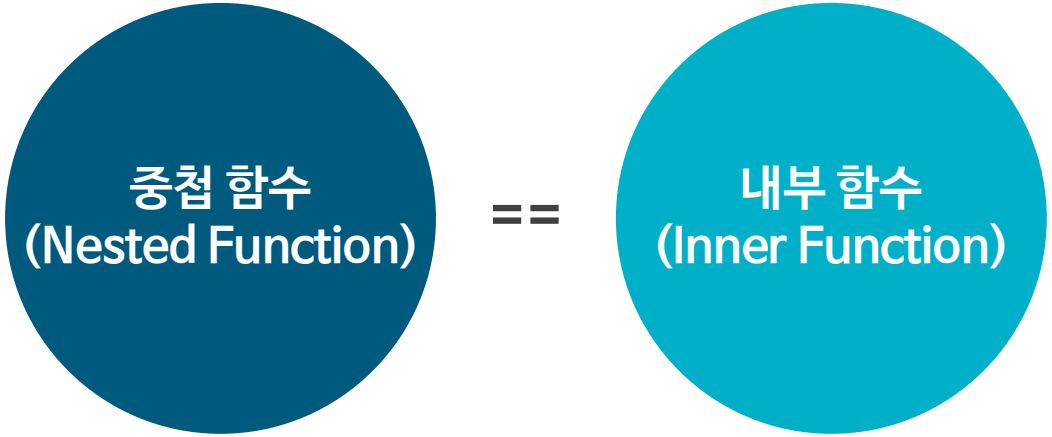

```
// 즉시 실행 함수 – 일반적인 방식
(function () {
  let msg = “Hello, IIFE!”;
  console.log(msg);
})();

//결과는 “Hello, IIFE!”가 출력됨
```

함수 전체를 괄호로 감싸서 함수 표현식으로 만들고, 바로 뒤에 ()를 붙여 즉시 호출

02

중첩 함수 (Nested Function)

1) 내부 함수(Inner Function)



함수 내부에 정의된 함수

참고

외부 함수(Outer Function) : 중첩 함수를 포함하는 함수

1) 내부 함수(Inner Function)

- 외부 함수 내부에서만 호출할 수 있음
- 자신을 포함하는 외부 함수를 돕는 헬퍼 함수(Helper Function) 역할을 함

중첩 함수

```
function outer() {
  var x = 1;

  // 중첩 함수 == 내부 함수
  function inner() {
    var y = 2;

    // 외부 함수의 변수 참조
    console.log(x + y); // 3
  }

  inner();
}
outer();
```

2) 클로저(Closure)

> “중첩 함수 중에서도 외부 함수의 변수나 상태를 참조하고,
> 그 외부 함수가 이미 종료된 이후에도
> 그 참조를 기억하고 있는 함수”

```
//클로저
function outer() {
  let msg = “Hello”;
  return  ( function inner() {
    console.log(“클로저입니다: ” + msg);
  } );
}
const closureFunc = outer();  // outer는 종료되었지만
closureFunc();                // 여전히 msg를 기억
```

VS

```
// 단순 중첩 함수
function outer() {
  let msg = “Hello”;
  function inner() {
    console.log(“중첩 함수입니다.”);
  }
  inner(); // 즉시 실행되므로 클로저 아님
}
outer();
```

클로저(Closure) 함수를 사용하는 이유

클로저는 외부 함수의 상태를 '기억'하게 함

상태 유지 · 정보 은닉 · 함수 생성 등
다양한 프로그래밍 기법에 핵심적으로 사용됨

클로저(Closure) 를 사용하는 이유

01 상태 유지(데이터 은닉)

- 외부에서 접근할 수 없는 private 변수처럼 사용 가능

예 카운터, 비밀번호 저장 등

02 함수 팩토리(커스텀마이징된 함수 만들기) 패턴 구현

- 공통 로직을 가진 다양한 함수를 동적으로 생성

예 할인율, 세율 등을 반영한 계산기 생성

클로저(Closure) 함수를 사용하는 이유

03 콜백 함수 내부에서 값 유지

- 비동기 처리나 반복문 내에서 외부 값을 안전하게 유지

예 setTimeout, addEventListener 내부에서 인덱스 기억

04 모듈 패턴 구현

- JavaScript에서 클래스 없이 모듈화/정보 은닉 가능

예 캡슐화된 라이브러리나 유틸리티 작성

2) 클로저(Closure)

- 클로저(Closure) 함수를 사용하는 이유
  - 상태 유지 예제(카운터)

```
//클로저
function createCounter() {
  let count = 0;
  return (function () {
    count++;
    return count;
  });
}
const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2
```

count는 외부에서 접근 불가(지역변수)하지만, 내부 함수는 계속 기억하고 증가시킴

클로저(Closure) 함수를 사용하는 이유

함수 팩토리 예제

특정 설정 값을 기반으로, 새로운 함수를 생성해 반환하는 함수

함수를 만들어내는 함수 → 함수의 재사용성과 유연성이 높아지는 구조

```
function makeMultiplier(x) {
  return function (y) {
    return x * y;
  };
}

const double = makeMultiplier(2);
console.log(double(5)); // 2(x) * 5(y)=10
```

x (공통로직)값을 고정시킨 함수를 동적으로 생성

클로저(Closure) 함수를 사용하는 이유

반복문 + 클로저 예제

```
for (var i = 0; i < 3; i++) {
  setTimeout( ( function(index) {
                    return function() {
                        console.log("i:", index);
                    }; // 클로저
        })(i), // for문이 순식간에 3번 돌면서 setTimeout 3개 등록
  1000); // 1 초 후 callback 함수 수행
}
```

클로저를 쓰지 않으면 i 값이 3으로만 찍힐 수 있음

클로저 없이 ➡ for문 끝나고 i=3 하나만 기억 ➡ 3, 3, 3 출력

클로저 사용 ➡ 각 반복 시점의 i값(0,1,2)을 따로 기억 ➡ 0,1,2 출력

03

고차 함수(HOF)

고차 함수(HOF, Higher-Order Function)

함수를 인자로 받거나 함수를 결과로 반환하는 함수

- 고차 함수의 2가지 조건

고차 함수는 다음 중 하나 이상을 만족해야 함

01 다른 함수를 인자로 받음 ➡ 콜백 함수가 여기에 해당

02 다른 함수를 반환함 ➡ 클로저가 여기에 해당

2) 고차 함수와 클로저

고차 함수가 반환한 함수가 클로저가 되는 경우가 많음

고차 함수는 클로저를 만들어내는 대표적인 패턴

```
def make_power(n):
    def power(x):
        return x ** n  # n은 클로저로 캡처됨
    return power

square = make_power(2)
print(square(4))  # 16
```

make_power는 함수를 반환하는 고차함수로, 내부 함수 power가 외부 변수 n을 기억하는 클로저로 동작
➡ n=2를 기억

04

콜백 함수 (Callback Function)

콜백 함수(Callback Function)

다른 함수에 인자(arguments)로 전달되어,
특정 시점에 자동으로 실행되는 함수

참고 매개변수로 다른 함수를 전달받아 실행하는 함수는
고차 함수(HOF, Higher-Order Function)에 해당

1) 개념


```
function greet(name, callback) {
  console.log("안녕하세요, " + name + "님!");
  callback(); // 콜백 함수 호출
}
function afterGreeting() {
  console.log("방문해 주셔서 감사합니다.");
}
// greet (고차함수)에 afterGreeting(콜백함수)을 인자로 전달
greet("철수", afterGreeting);
```

afterGreeting은 greet 함수가 호출한 후 실행되는 콜백 함수

2) 원리

- 콜백 함수는 고차 함수 내부에서 호출(실행)됨

- 함수는 일급 객체(값처럼 다루는 객체)이므로
  함수의 매개변수를 통해 함수 전달이 가능함
  - ➡ 함수는 더 이상 내부 로직에 강력히 의존하지 않고,
    외부에서 로직의 일부분을 함수로 전달받아 수행하므로
    유연한 구조를 가짐

- 고차 함수는 필요에 따라 콜백 함수에 인수(Argument)를
  추가로 전달할 수 있음
  - ➡ 그렇기 때문에 고차 함수에 콜백 함수 전달 시 콜백 함수를
    호출하지 않고, 함수 자체를 전달해야 함

```
function getUserData(callback) {
  fetch("https://jsonplaceholder.typicode.com/users/1")
    .then(response => response.json()) //응답처리
    .then(data => {  //비동기 작업이 성공했을 때 실행
      console.log("사용자 데이터:", data);
      callback(); // 콜백 함수 호출
    });
}
function afterFetch() {
  console.log("데이터 요청이 완료되었습니다.");
}
// 실행
getUserData(afterFetch);
```

```
<button id="myBtn">클릭하세요</button>

<script>
  function handleClick() {
    alert("버튼이 클릭되었습니다!");
  }

  const button = document.getElementById("myBtn");
  //"click" 이벤트가 발생하면 실행할 동작을 등록 ➡ 실행 중 Click 발생 시 콜백 함수 실행
  button.addEventListener("click", handleClick); // 콜백으로 handleClick 전달
</script>
```

세종사이버대학교

05주. 나만의 기능 만들기 : 여러 가지 함수

03

함수 활용 실습