12장. 함수

12.1 함수란?

매개변수: 함수 내부로 입력을 전달받는 변수

인수: 입력

반환값: 출력

함수호출: 인수를 매개변수를 통해 함수에 전달하면서 함수의 실행을 명시적으로 지시하는 것

12.2 함수를 사용하는 이유

함수는 몇 번이든 호출할 수 있으므로 코드의 재사용이라는 측면에서 매우 유용하다.

유지보수의 편의성을 높이고 코드의 신뢰성을 높이는 효과가 있다.

12.3 함수 리터럴

리터럴은 값을 생성하기 위한 표기법. 함수 리터럴도 평가되어 값을 생성하며 이 값은 객체

== 함수는 객체

일반 객체는 호출할 수 없지만 함수는 호출 가능

12.4 함수 정의

1. **함수 선언문**

   함수 리터럴은 함수 이름 생략 가능, 함수 선언문은 함수 이름 생략 불가능

   함수 선언문은 표현식이 아닌 문

2. **함수 표현식**

a. 일급객체: 값처럼 변수에 할당 할 수도 있고 프로퍼티 값이 될 수도 있으며 배열의 요소가 될 수도 있는데 이 값의 성질을 갖는 객체

   함수가 일급 객체이다 == 함수를 값처럼 자유롭게 사용할 수 있다.

b. 익명 함수: 함수 리터럴의 함수 이름은 생략가능

함수를 호출 할 때는 함수 이름이 아닌 함수 객체를 가리키는 식별자를 사용

함수 선언문 = 표현식이 아닌 문 / 함수 표현식 = 표현식인 문

3. **함수 생성 시점과 함수 호이스팅**

   함수 선언문을 정의한 함수와 함수 표현식으로 정의한 함수의 생성 시점이 다르기에 함수 표현식으로 정의 함수는 함수 표현식 이전에 호출 불가능

   호이스팅: 함수 선언문이 코드의 선두로 끌어 올려진 것처럼 동작하는 자바스크립트 고유의 특징

   - **함수 호이스팅 vs 변수 호이스팅**

     1. var 키워드를 사용한 변수 선언문 이전에 변수를 참조하면 변수 호이스팅에 의해 undefined로 평가
     2. 함수 선언문으로 정의한 함수를 함수 선언문 이전에 호출하면 함수 호이스팅에 의해 호출 이 가능

   따라서 함수 표현식으로 함수를 정의하면 함수 호이스팅 발생 X → 변수 호이스팅이 발생 O

4. **Function 생성자 함수**

   - **생성자 함수란?**

     객체를 생성하는 함수

   Function 생성자 함수로 생성한 함수는 클로저를 생성하지 않는 등, 함수 선언문이나 함수 표현식으로 생성한 함수와 다르게 동작함

5. **화살표 함수**

   function 키워드 대신 화살표(⇒)를 사용해 좀 더 간략한 방법으로 함수 선언 가능 == 익명 함수

   화살표 함수는 생성자 함수로 사용 불가능, this 바인딩 방식이 다르고 prototype 프로퍼티가 없고 arguments 객체 생성 하지 않음

12.5 함수 호출

함수를 가리키는 식별자와 한 쌍의 소괄호인 함수 호출 연산자로 호출한다.

함수 실행 할 때 필요한 값을 함수 외부에서 함수 내부로 전달할 필요가 있을 때 매개변수를 통해 인수를 전달(이때, 인수는 값으로 평가될 수 있는 표현식이며 함수를 호출할 때 지정)

매개변수는 함수 몸체 내부에서만 참조 가능 → 매개변수의 스코프(유효 범위)는 함수 내부

함수 특징

1. 자바스크립트 함수는 매개변수와 인수의 개수가 일치하는지 확인하지 않는다.
2. 자바스크립트는 동적 타입 언어로 매개변수의 타입을 사전에 지정할 수 없다.

→ 자바스크립트의 경우 함수를 정의할 때 적절한 인수가 전달되었는지 확인할 필요 있음

이상적인 함수는 한 가지 일만 해야 하며 가급적 적게 만들어야 한다. → 매개변수는 최대 3개 이상을 넘지 않는 것을 권장

만약 넘어간다면 하나의 매개변수 선언 후 객체를 인수로 전달하는 것이 좋음

!주의: 함수 외부에서 내부로 전달한 객체를 함수 내부에서 변경하면 함수 외부의 객체가 변경되는 부수 효과가 발생함

함수 호출 == 표현

→ 함수 호출 표현식은 return 키워드가 반환한 표현식의 평가 결과(=반환값)으로 평가

반환문 역할

1. 반환문은 함수의실행을 중단하고 함수 몸체를 빠져나간다

   → 반환문 이후에 다른 문이 존재하면 그 문은 실행되지 않고 무시된다(return문 이후에는 실행 X)

2. 반환문은 return 키워드 뒤에 오는 표현식을 평가해 반환한다

   → return 키워드 뒤에 반환값으로 사용할 표현식을 명시적으로 지정하지 않으면 undefined 반환

반환문은 생략할 수 있으며, 함 수 몸체 마지막 문까지 실행한 후 암묵적으로 undefined를 반환한다

반환문은 함수 몸체 내부에서만 사용할 수 있으며 전역에서 반환문을 사용하면 문법 에러가 발생

12.6 참조에 의한 전달과 외부 상태의 변경

값에 의한 호출, 참조에 의한 호출: 함수를 호출하면서 매개변수에 값을 전달하는 방식

1. 원시값

   함수 외부에서 함수 몸체 내부로 전달한 원시 값은 원본을 변경하는 어떠한 부수 효과 발생 X

2. 참조값

   함수 외부에서 함수 몸체 내부로 전달한 참조 값에 의해 원본 객체가 변경되는 부수 효과 발생 O

   → 객체의 상태 변경이 필요한 경우에는 객체의 방어적 복사를 통해 원본 객체를 완전히 복제(깊은 복제)를 통해 새로운 객체를 생성하고 재할당을 통해 교체를 통해 외부 상태가 변경되는 부수 효과를 없앨 수 있다

12.7 다양한 함수의 형태

- **즉시 실행 함수란?**

  함수 정의와 동시에 즉시 호출되는 함수, 즉시 실행 함수는 단 한 번만 호출되며 다시 호출 불가능

  기명 함수도 사용할 수 있지만, 보통 익명 함수를 사용하는 것이 일반적

  But. 그룹 연산자(...) 내의 기명 함수는 함수 선언문이 아니라 함수 리터럴로 평가되어 함수 이름은 함수 몸체에서만 참조할 수 있는 식별자 → 즉시 실행 함수를 다시 호출 할 수 없다

함수 선언문은 함수 이름을 생략 불가능

- **재귀함수란?**

함수가 자기 자신을 호출하는 것, 재귀 호출을 수행하는 함수

→ 재귀 함수는 반복되는 처리를 위해 사용 ex) 팩토리얼

```
//일반 함수
function countdown(n) {
    for (var i = n; i >= 0; i--) console.log(i);
}
countdown(10);


//재귀 함수
function countdown(n) {
    if (n < 0) return;
    console.log(n);
    countdown(n - 1) //재귀 호출
}
countdown(10);
```

재귀 함수는 자신을 무한 재귀 호출한다 → 재귀 호출을 멈출 수 있는 탈출 조건을 반드시 만들어야 함

만약 탈출 조건이 없으면 함수가 무한한 호출되어 **스택 오버플로** 에러가 발생

- **중첩함수란?**

  함수 내부에 정의된 함수(=내부 함수), 중첩함수를 포함하는 함수(=외부 함수)

  중첩 함수는 외부 함수 내부에서만 호출가능

  보통 중첩함수는 자신을 포함하는 외부 함수를 돕는 헬퍼 함수의 역할을 함

- **repeat 함수란?**

  어떤 일을 반복 수행함, 매개변수를 통해 전달받은 숫자만큼 반복

- **콜백 함수란?**

  함수의 매개변수를 통해 다른 함수의 내부로 전달되는 함수

  → 매개 변수를 통해 함수의 외부에서 콜백 함수를 전달받은 함수 == 고차 함수

중첩 함수가 외부 함수를 돕는 헬퍼 함수의 역할 (단, 중첩 함수는 고정되어있어 교체 X)

== 콜백 함수도 고차 함수에 전달되어 헬퍼 함수의 역할을 함

즉, 고차 함수는 콜백 함수를 자신의 일부분 합성한다.

고차 함수는 매개변수를 통해 전달받은 콜백 함수의 호출 시점을 결정해서 호출

!정리: 콜백 함수는 고차 함수에 의해 호출, 이때 고차 함수는 필요에 따라 콜백 함수에 인수를 전달 할 수 있어야 함

- **순수 함수란?**

  부수 효과가 없는 함수(함수형 프로그래밍에서 어떤 외부 상태에 의존하지도 않고 변경하지도 않는 함수)

  동일한 인수가 전달되면 언제나 동일한 값을 반환(=상수), 외부 상태에 따라 반환값이 달라짐

  ex) 전역 변수, 서버 데이터, 파일, Console, DOM ...

  → 어떤 외부 상태에도 의존하지 않으며 외부 상태를 변경하지도 않는 함수

- **비순수 함수란?**

  부수 효과가 있는 함수(외부 상태에 의존하거나 외부 상태를 변경하는 함수)

  함수의 외부 상태에 따라 반환값이 달라지는 함수

  함수가 외부 상태를 변경하면 상태 변화를 추적하기 어려움 → 순수함수 사용이 좋다

함수형 프로그래밍: 순수 함수와 보조 함수의 조합을 통해 외부 상태를 변경하는 부수 효과를 최소화해서 불변성을 지향하는 프로그래밍 패러다임

목표: 로직 내 존재하는 조건, 반복문을 제거해서 복잡성 해결 및 변수 사용 억제를 통해 생명주기를 최소화하여 상태 변경을 피해 오류를 최소화하기