프론트 엔드 개발자가 알아야 하는 컴퓨터 공학 지식

소프트웨어 공학(SW Engineering)

소프트웨어 공학 | 프론드 엔드 개발자가 알아야 하는 CS 지식

강사 나동빈

프론트 엔드 개발자가 알아야 하는 컴퓨터 공학 지식

소프트웨어 공학(SW Engineering)

함수형 프로그래밍이란?

- 순수 함수(pure function)를 조합하여 프로그램을 개발하는 방식을 말한다.
- 기존 ① 절차 지향 프로그래밍과 ② 객체 지향 프로그래밍이 각광을 받았다.

→ 함수형 프로그래밍은 기존의 방식과 다른 개발 패러다임을 제안한다.

- 대입 구문이 없는 프로그래밍이라고 이해할 수도 있다.

기존의 프로그래밍 방식과 비교

- 명령형 프로그래밍: 소프트웨어의 상태와 그것을 변경시키는 방식에 중점을 둔다.

1) **절차 지향 프로그래밍**: 소프트웨어가 위에서부터 순차적으로 처리되는 것을 중요시한다.

2) **객체 지향 프로그래밍**: 객체의 집합에 포함된 객체들이 서로 상호작용한다.

- 선언형 프로그래밍: 어떻게(how)보다는 **무엇(what)을 할 것인지**에 중점을 둔다.

1) **함수형 프로그래밍**: **순수 함수(pure function)**을 조합하여 프로그램을 개발한다.

함수형 프로그래밍의 특징

- 데이터를 처리하는 과정을 일종의 수학적 함수를 계산하는 것으로 다룬다.
- 데이터는 **불변한(immutable) 것**으로 간주한다.
- 함수형 프로그래밍에서는 문제를 함수로 **분해(decompose)**하여 해결한다.
- 예시로, 함수형 프로그래밍 언어는 조건문 및 반복문을 지원하지 않는다.

함수형 프로그래밍: 데이터의 불변성

- 함수 밖에 있는 데이터를 변경하지 않는다.
- 함수의 반환 값은 함수 내에서 수행된 작업을 반영한다.
- **Side Effect를 방지**할 수 있다.

→ 함수 내부의 동작으로 인해 함수 외부의 변수가 변경되는 것을 예방한다.

함수형 프로그래밍: 순수 함수(Pure Function)

- **입력 매개변수에만 의존**하여, Side Effect를 발생시키지 않는 함수를 의미한다.
- 즉, 인자(argument)와 반환 값(return value)이 핵심 요소로, 단순함을 유지한다.
- **예시)** 아래 함수는 함수 외부의 변수(전역 변수 value)에 접근하므로, 순수 함수가 아니다.

```
let value = 3;

function multiply(x) {
  return x * value;
}
```

함수형 프로그래밍: 순수 함수(Pure Function)

- **입력 매개변수에만 의존**하여, Side Effect를 발생시키지 않는 함수를 의미한다.
- 즉, 인자(argument)와 반환 값(return value)이 핵심 요소로, 단순함을 유지한다.
- **예시)** 아래 함수는 순수 함수다.

```
function multiply(x, y) {
  return x * y;
}
```

함수형 프로그래밍: 유지 보수의 용이성

- Side Effect가 발생하지 않기 때문에, 프로그램을 유지하고 관리하기 용이하다.

함수형 프로그래밍: 선언형 함수(Expression)

- 선언형 프로그래밍은 어떻게(how)보다는 **무엇(what)**을 할 것인지에 초점을 맞춘다.
- 절차 지향 프로그래밍에서 자주 사용되는 조건문, 반복문을 사용하지 않는다.
- **예시)** 아래 함수는 조건문, 반복문 및 대입 구문을 사용하고 있다.

```
let n = 5;
let arr = [1, 2, 3, 4, 5];

function multiply(value) {
  for (let i = 0; i < n; i++) {
    arr[i] = arr[i] * value;
  }
}
```

함수형 프로그래밍: 선언형 함수(Expression)

- 반복문을 함수형 메서드 중 하나인 map()으로 대체했다.

```
function multiply(arr, value) {
  return arr.map((num) => num * value);
}
```

함수형 프로그래밍: 1급 객체

- 순수 함수는 **1급(First Class) 객체**로 사용된다.

→ 함수 자체를 객체로 이해할 수 있으며, 유연한(flexible) 사용이 가능하다.

- 함수를 그 자체로 변수의 값으로 담아 사용할 수 있다.
- 함수를 **매개 변수**로 전달할 수 있다.
- 함수 자체를 **반환 값**으로 사용할 수 있다.

```
const add = (x, y) => x + y;
```

함수형 프로그래밍의 장점

- 전체 문제를 작은 문제를 해결하기 위한 여러 개의 함수로 분해한다.
- 가독성을 높이고, **유지 보수가 용이**하게 해준다.
- 높은 수준의 추상화 기능을 제공한다.
- 함수 단위의 **코드 재사용성**이 높다.