7장 연산자

연산자를 함수로 생각하자.

함수는 입력값과 반환값의 집합이 중요하다.

```
f: X -> Y
```

연산자는 입력받는 값과 반환되는 결과가 중요하다.

```
typeof 7 // "number" -> String

3 + 7 // 10

if (undefined) // false
```

부수효과

- 평가값 반환 이외의 효과
  - 대입연산자, 단항연산자는 변수의 저장값 번경

8장 제어문

- 코드의 실행 흐름 flow-chart
- 제어문을 지양하는 이유

들어가며

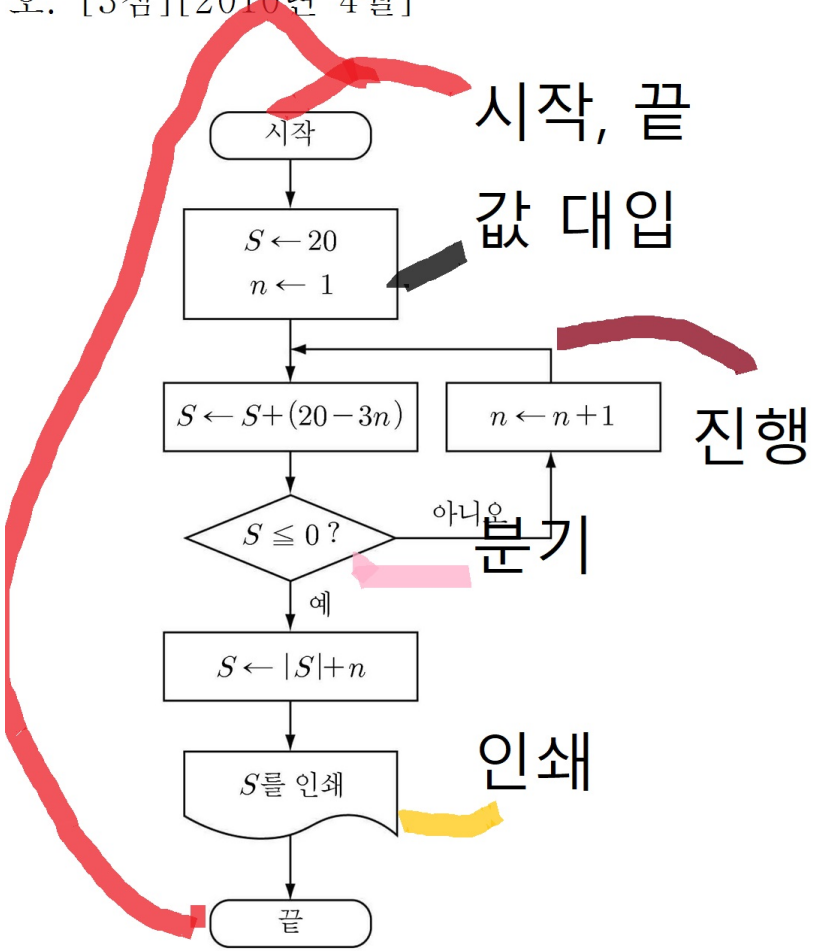
제어문은 코드의 실행 흐름을 인위적으로 제어할 수 있다.
(..) 하지만 코드의 실행순서가 번경된다는 것은 (..) 흐름을 혼란스럽게 만든다.
따라서 제어문은 코드의 흐름을 이해하기 어렵게 만들어 가독성을 해치는 단점이 있다.

(..) forEach, map, filter,reduce 같은 고차함수를 활용한
함수형 프로그래밍에서는 제어문의 사용을 억제하여 복잡성을 해결한다.

-모던 자바스크립트 Deep Dive 93p

흐름을 파악하는 도구 flowchart

다음 순서도에서 인쇄되는 S의 값을 구하시오. [3점][2010년 4월]



- 컴퓨터의 문제 처리 순서를 단계화
- 기호와 흐름선을 사용
- 제어문이 많을수록 그림이 복잡해짐

제어문과 함수형 비교 > 배열순회 코드

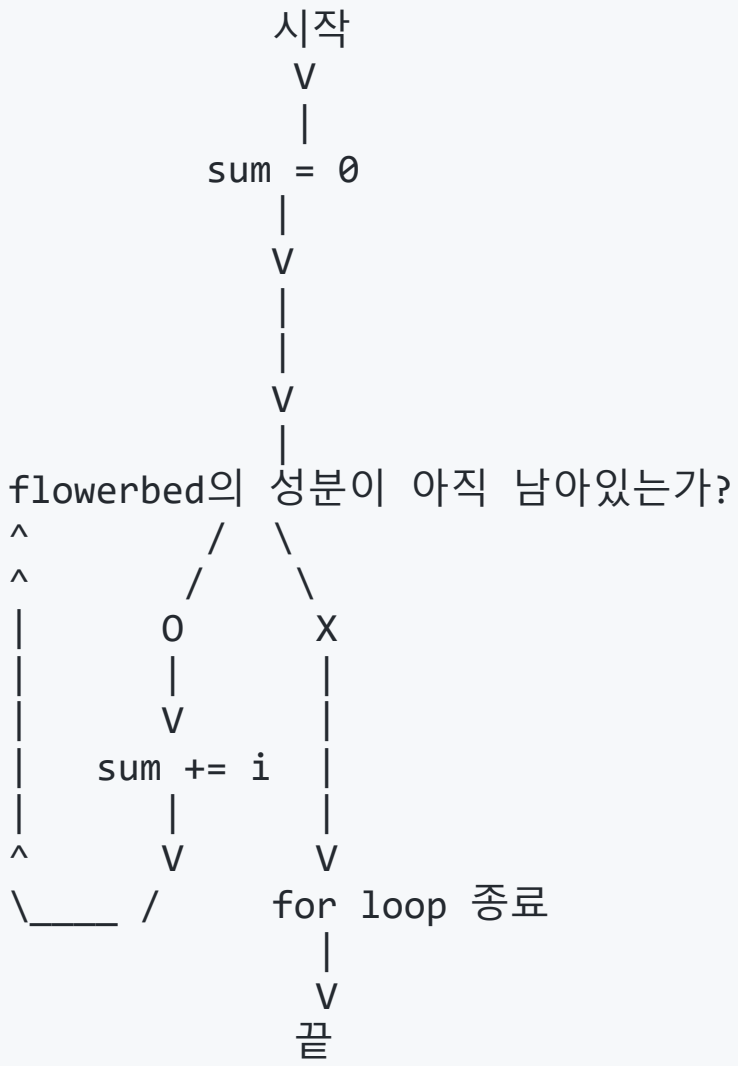
제어문 코드

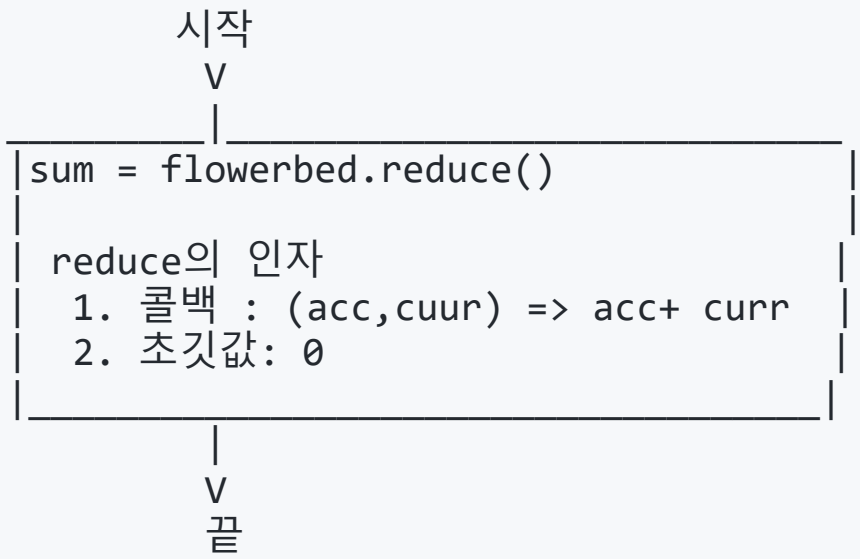
```
let sum = 0;

for (i of flowerbed) {
    sum += i;
}
```

함수형 코드

```
const sum = flowerbed.reduce((acc, curr) => acc + curr, 0);
```

제어문과 함수형 비교 > 배열순회 코드 > FlowChart





함수형은 조건이 많으면 진가가 살아난다.

- 함수의 합성 f(g(x))을 점으로 표현.
  - 여러 메서드를 단일문으로 호출가능

```
targetArray
    .filter(i => typeof i === "number")
    .map(i => i % 2)
    .reduce((acc, current) => acc + current, 0);
```

제어문

외부전역변수, 배열의 길이, 순회 증감식, 무수한 {}, 공통된 변수 i

```
let sum = 0;
for (let i = 0; i < targetArray.length; i++) {
    if (typeof targetArray[i] === "number") {
        sum += targetArray[i] % 2;
    }
}
```

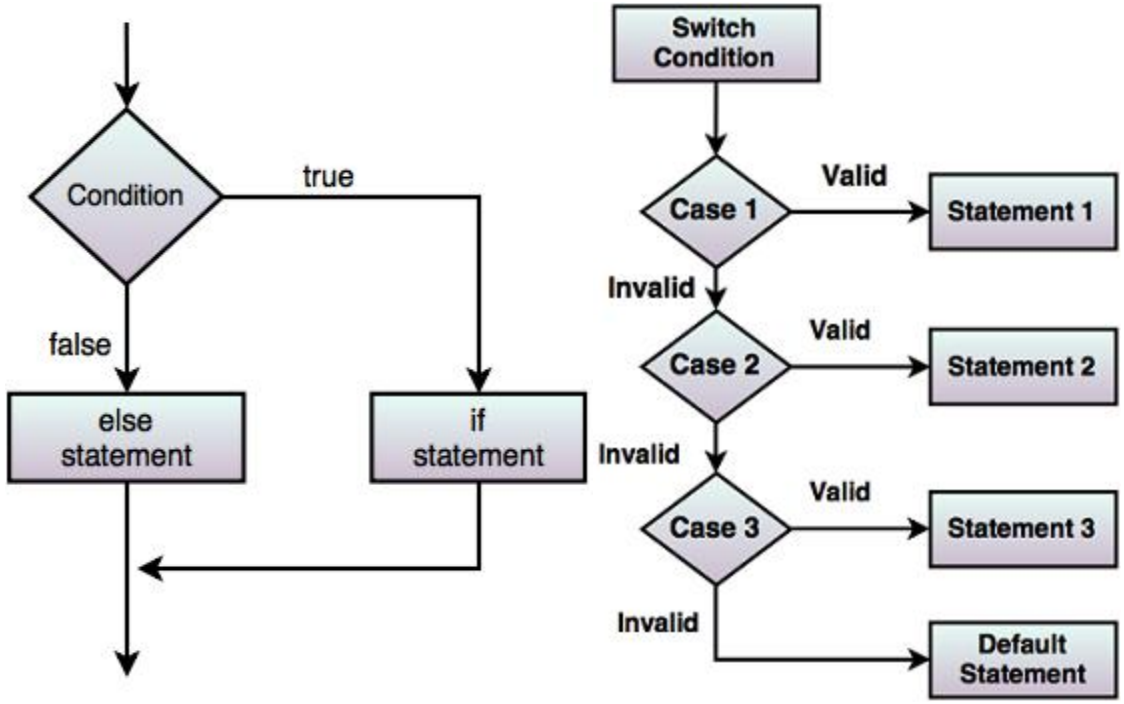
함수형

다양한 메서드

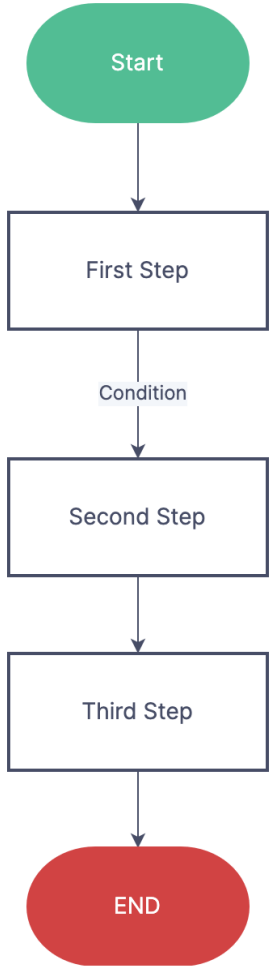
```
const sum = targetArray
                    .filter(item => typeof item === "number")
                    .map(num => num % 2)
                    .reduce((acc, current) => acc + current, 0);
```

FlowChart 비교

제어문



함수형



그래도 제어문 사랑합시다

함수형의 단점

- 고차함수가 존재하는 언어에서만 사용가능
- 불변객체는 메모리 이슈를 발생시킴
- 필요한 선수 지식이 많고, 언어별로 다름.

FlowChart 사이트

https://draw.io/