04-프로그램_흐름제어

프로그램 구문이 무조건 순차적으로 실행되는 것이 아니라 주어진 조건을 판별하여 선택적 혹은 반복적으로 실행 여부를 결정하도록 흐름을 제어하는 코드 작성 기법

#01. 조건문

1) if문

주어진 조건식이 참인 경우에만 블록({})안을 실행하는 구문 형태.

```
if (조건식) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

a) 조건식에 따른 구분

논리값을 사용한 경우

true나 false를 저장하고 있는 변수를 조건으로 지정한다.

```
const a = true;

// 주어진 조건이 참이므로 블록을 실행함.
if (a) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

```
const b = false;

// 주어진 조건이 거짓이므로 블록을 실행하지 않음
if (b) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

숫자형 값을 사용한 경우

숫자형인 경우 0은 거짓, 0이 아닌 모든 수는 참으로 식별한다.

```
const a = 123;

// 주어진 조건이 참이므로 블록을 실행함.
```

```
if (a) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

```
const b = 0;

// 주어진 조건이 거짓이므로 블록을 실행하지 않음
if (b) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

문자열 값을 사용한 경우

빈 문자열 ""은 false, 내용이 한 글자라도 포함된 문자열은 true로 식별한다.

문자열 변수를 조건으로 사용하는 경우는 **문자열에 내용이 포함되어 있다면?** 으로 해석할 수 있다.

```
const a = "hello";

// 주어진 조건이 참이므로 블록을 실행함.
if (a) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

```
const b = "";

// 주어진 조건이 거짓이므로 블록을 실행하지 않음
if (b) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

비교식을 사용한 조건문

() 안의 식이 참인 경우 {} 안의 구문이 실행된다.

```
const a = 100 > 10;

// 주어진 조건이 참이므로 블록을 실행함.
if (a) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

```
// 주어진 조건이 거짓이므로 블록을 실행하지 않음
if (100 < 10) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

논리식을 사용한 조건문

AND(&&) 연산모든 값이 TRUE인 경우에만 결과가 TRUE

```
let a = true;
let b = true;
// 주어진 조건이 참이므로 블록을 실행함.
if (a && b) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

```
let c = true;
let d = false;
// 주어진 조건이 거짓이므로 블록을 실행하지 않음
if (c && d) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

OR(||) 연산은 하나라도 참이 포함되어 있다면 결과가 참

```
let a = true;
let b = false;
// 주어진 조건이 참이므로 블록을 실행함.
if (a || b) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

```
let c = false;
let d = false;
// 주어진 조건이 거짓이므로 블록을 실행하지 않음
if (c || d) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

! 연산자는 논리값을 부정한다.

```
let a = false;
// 주어진 조건이 참이므로 블록을 실행함.
if (!a) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

```
let b = true
// 주어진 조건이 거짓이므로 블록을 실행하지 않음
if (!b) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

문자열도 `true`, `false`를 판단할 수 있으므로 `!`로 처리 가능하다.

문자열에 대해 not 연산자를 적용할 경우 **내용이 없다면?** 으로 해석할 수 있다.

```
let a = "";
// 주어진 조건이 참이므로 블록을 실행함.
if (!a) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

```
let b = "...";
// 주어진 조건이 거짓이므로 블록을 실행하지 않음
if (!b) {
    ... 조건이 참인 경우에 실행할 명령 ...
}
```

b) if ~ else 문

그렇지 않으면~~

```
if (조건식) {
    ... 조건이 참인 경우에 실행할 명령 ...
} else {
    ... 조건이 참이 아닌 경우에 실행할 명령 ...
}
```

- if문은 조건이 참인 경우에만 수행되고 조건이 거짓인 경우는 수행되지 않는다.
- if문 뒤에 else문을 덧붙여 조건이 거짓인 경우에 수행되는 구문을 추가할 수 있다.
- else문은 if문 뒤에만 위치할 수 있고 독립적으로는 존재할 수 없다.

c) if ~ else if ~ else 문

여러가지 경우의 수를 나열하는 조건문.

순차적으로 조건을 판별하면서 **가장 처음 만나는 참인 조건의 블록을 수행**하고 빠져나간다.

```
if (조건식1) {
    ... 조건이 참인 경우에 실행할 명령 ...
} else if (조건식2) {
    ... 조건이 참인 경우에 실행할 명령 ...
} else if (조건식3) {
    ... 조건이 참인 경우에 실행할 명령 ...
} else if (조건식n) {
    ... 조건이 참인 경우에 실행할 명령 ...
} else {
    ... 조건이 참이 아닌 경우에 실행할 명령 ...
}
```

2) switch 문

기본 구문

하나의 변수나 수식에 대한 여러가지 경우의 수를 나열하는 형식.

조건에 맞는 `case` 블록부터 `break` 키워드를 만날 때 까지 실행한다.

switch의 조건을 식으로 설정하는 것도 가능하다.

마지막의 default는 모든 케이스를 충족하지 않을 경우 실행되고, 필요 없다면 생략할 수 있다.

```
switch (변수) {
    case 1:
        ... 실행할 명령 ...
        break;
    case 2:
        ... 실행할 명령 ...
        break;
    case n:
        ... 실행할 명령 ...
        break;
    default:
        ... 실행할 명령 ...
        break;
}
```

`break`가 없는 경우

중단점이 없기 때문에 그 아래 블록까지 함께 실행된다.

필요에 따라 의도적으로 `break`의 위치를 조절 할 수 있다.

#02. 반복문

1) while문

a) while문의 구문 형식

주어진 조건이 참을 충족하는 동안 수행되는 문법

```
초기식
while (조건식) {
    ... 반복적으로 수행될 구문 ...
    증감식;
}
```

실행과정

- 초기식, 조건식, 증감식의 요소를 충족해야 성립된다.
- `{}` 안에서 선언되는 변수나 상수는 그 블록 안에서만 유효하다.
- 반복문 `{}` 안에서 선언되는 변수나 상수는 그 회차에서만 유효하고 새로운 회차의 반복에서는 기존 변수는 삭제되고 새롭게 생성된다.

b) 증가량 조절하기

- 증감식을 조절하여 증가량을 조절할 수 있다.
- ex) `0`부터 `100`전(=99)까지 `**10씩 증가**`하기

2) For문

a) For문의 구문 형식

초기식, 조건식, 증감식이 하나의 괄호`()` 안에 모두 명시되는 형태.

```
for (초기식; 조건식; 증감식) {
    ...
}
```

실행과정

1. 초기식을 실행한다.
2. 조건식을 판별한다.
    - 조건식이 참인 경우 `{}`안을 1회 실행하고 증감식으로 이동한다.
        - 증감식을 실행한 후 다시 조건식으로 이동한다.
    - 조건식이 거짓인 경우 `{}` 블록을 실행하지 않고 빠져나간다.