## || && 논리연산자

여러가지 조건을 결합할 때 사용한다.

&& 는 둘 다 일치할때만 실행된다. (AND)

```
const myName = "로미오";
const yourName = "줄리엣";
if (myName === "로미오" && yourName === "줄리엣") {
  console.log("&& 실행");
}
```

|| 는 둘 중 하나라도 일치할 때 실행된다. (OR)

```
const myName = "로미오";
const yourName = "앨리스";
if (myName === "로미오" || yourName === "줄리엣") {
  console.log("|| 실행");
}
```

## ? :  삼항연산자

if else 조건문을 짧게 쓴 것.

```
const myName = "로미오";
const yourName = "줄리엣";

myName === "로미오" || yourName === "줄리엣" ? console.log("||실행") : console.log("error!");
```

조건 / true일 경우 / false일 경우

참 좋은 것 배웠으니, if else 는 이제 필요없는것일까?
만약 조건에 따른 실행결과가 단순히 console.log() 같이 짧지 않고 길 경우,
그때는 if else 를 쓰는 게 훨씬 좋다.

# 배열(array)

여러 개의 data를 대괄호 [ ] 를 사용해 하나로 묶은 것을 의미한다.

배열을 쓰지 않을 경우부터 살펴보자. 요일 변수를 7개 선언하고, 출력해보았다.

```
const mon = "월";
const twe = "화";
const wed = "수";
const thu = "목";
const fri = "금";
const sat = "토";
const sun = "일";

console.log(mon, twe, wed, thu, fri, sat, sun);
```

하지만 이렇게 하나하나 적는 건 비효율적.
왜? 7가지 모두 “요일” 이라는 공통된 특징이 있기 때문.

배열을 사용하면 다음과 같이 쓸 수 있다.

```
const daysOfWeek = ["월", "화", "수", "목", "금", "토", "일"];

console.log(daysOfWeek);
```

즉, 배열은 여러개의 data를 대괄호 [ ] 를 사용해 하나로 묶은 것이다.

배열을 제대로 활용하려면 요소(element)와 인덱스(index) 의 개념에 대해 알아야한다.

요소(element)는 배열을 이루는 각각의 data를 의미한다.
인덱스(index)는 각 요소에 부여된 번호이다.

daysOfWeek 의 첫번째 요소인 “월”을 가져와보자.

```
const daysOfWeek = ["월", "화", "수", "목", "금", "토", "일"];

console.log(daysOfWeek[0]); // 월
console.log(daysOfWeek[1]); // 화
console.log(daysOfWeek[6]); // 일
console.log(daysOfWeek[7]); // undefined
```

다음과 같이, 배열변수 옆에 대괄호를 쓴 숫자를 넣어주면 된다.
“월”의 인덱스는 1이 아니라 0이다. 컴퓨터는 숫자를 셀 때 0부터 세기 때문이다.
만약, 없는 인덱스 값을 입력하면 undefined라는 결과가 뜬다.
이것은, 정의되지 않았다는 뜻이다. 우린 인덱스 7, 인간 기준으로 8번째 데이터를 정의한 적 없다.

비슷한걸로 null 이라는것이 있는데, undefined와 null 은 둘 다 조건문에서 false로 인식한다.
그러나, 아주 큰 차이가 있는데,
undefined는 고의가 아니다. 즉, 의도치 않은 결과다.
null은 고의다. 프로그래머가 작정하고 “일부러” 비워둔 것이다.
왜 일부러 null을 쓸까?
개발자가 의도적으로 비워두고 싶은 경우가 있기 때문이다.

배열의 길이를 구할 땐 length 를 사용한다.

```
const daysOfWeek = ["월", "화", "수", "목", "금", "토", "일"];

console.log(daysOfWeek.length); // 7
```

length 는 배열뿐만 아니라 문자열의 길이도 구할 수 있다.

```
const name1 = "조교행님";
console.log(name1.length); // 4
```

배열의 요소는 const 로 선언되었더라도 변경 가능하다.

```
const daysOfWeek = ["월", "화", "수", "목", "금", "토", "일"];

daysOfWeek[2] = "ㅋㅋㅋ";

console.log(daysOfWeek);
```

배열에 요소를 추가할때는 push() 를 사용한다.

```
const daysOfWeek = ["월", "화", "수", "목", "금", "토", "일"];

daysOfWeek.push("야호");

console.log(daysOfWeek); // 맨 뒤에 "야호" 추가된 배열 출력됨
```

그러나, const로 선언된 배열을 다시 선언할 순 없다.

```
const daysOfWeek = ["월", "화", "수", "목", "금", "토", "일"];

daysOfWeek = ["짜장면", "짬뽕", "탕수육"];  // error
```

이건 당연하다. const 이기 때문이다.

배열 안에 배열도 가능하다. 이차원 배열이라고 한다.
프로그래밍에서 행렬을 표현할 때 주로 사용되기 때문에 matrix 라고도 한다.

```
const matrix = ["김치찌개", "햄버거", ["짜장면", "짬뽕", "탕수육"]];

console.log(matrix[2][1]);  // 짬뽕
```

# 객체(object)

JavaScript에서 가장 중요한 두 가지는 다음과 같다.
객체(object) 와 함수(function)
이 둘은 JavaScript의 정체성이라고 할 수 있다.

주의사항이 있다.
JavaScript의 객체(object)는

C, C++, C#, Java, Python 의 객체와는 완전히 다른 개념이다.
JavaScript 에선 클래스를 만들지 않고도 객체를 만든다.

객체는 키(key) 와 값(value) 로 이루어진 프로퍼티(property) 모음이다.
가장 비슷한 개념은 Python의 딕셔너리 라고 할 수 있다.

내 개인정보를 배열로 저장했다고 가정해보자.

```
const myInfo = [
    "조교행님",
    28,
    true,
    [
        "아빠",
        "엄마",
        "멍멍이"
    ]
];
```

참고로, JavaScript 는 독특하게도 서로 다른 타입끼리도 배열을 만들 수 있다.

이 경우, 두 가지 점에서 불편하다.
1. 각각의 요소가 무엇을 의미하는지 예상해야한다.
2. 데이터를 다룰 때 인덱스를 전부 다 기억해야 한다.

만약 이 각각의 요소에 “이름” 을 달 수 있다면 편하지 않을까?

```
const myInfo = {
    name: "조교행님",
    age: 28,
    isGrilfriend: true,
    family: [
        "아빠",
        "엄마",
        "멍멍이"
    ]
};
```

이 경우 개선된 것을 보자.
뭐가 뭔지 예상할 필요가 없다. 이름으로 구분이 가능하기 때문이다.

인덱스를 기억할 필요도 없어졌는데, 인덱스라는 개념이 빠진 대신 해당 데이터가 무엇인지 알려주는 "이름"이 들어갔기 때문이다.

여기서,

앞에 있는게 key (따옴표 안 씀. 영어만 허용)
뒤에 있는게 value
둘을 합쳐서 property (프로퍼티)
여러 개면 , (콤마) 로 구분

객체는 인덱스가 존재하지 않기 때문에, key 를 통해 접근한다.

```
const myInfo = {
    name: "조교행님",
    age: 28,
    isGrilfriend: true,
    family: [
        "아빠",
        "엄마",
        "멍멍이"
    ]
};

console.log(myInfo.family[0]); // 아빠
```

여기서, 마침표 . 를 통해 데이터에 접근할 수 있음을 알 수 있다.

인덱스가 없다는 건, 순서가 따로 없다는 뜻이다.

물론 잘못된 key 를 입력하면 undefined 이다.

그럼 우리가 흔히 썼던 console.log() 의 정체도 알 수 있다.
즉, console 객체 안에 있는 log() 라는 함수다.

객체의 프로퍼티는 const 로 선언되었더라도 변경 가능하다.

```
const myInfo = {
    name: "조교행님",
    age: 28,
    isGrilfriend: true,
    family: [
        "아빠",
        "엄마",
        "멍멍이"
    ]
};

myInfo.isGrilfriend = false;
console.log(myInfo);
```

그러나, const 로 선언된 객체를 다시 선언할 순 없다.

```
const myInfo = {
    name: "조교행님",
    age: 28,
    isGrilfriend: true,
    family: [
        "아빠",
        "엄마",
        "멍멍이"
    ]
};

myInfo = {
    name: "Jony",
    age: 16
}
```

객체는 대체 왜 중요할까?
프론트엔드(클라이언트)와 백엔드(서버) 는 객체를 주고받아 통신하기 때문이다.
나중에 배우게 될 JSON 이 바로 그것이다.

자, 그럼 객체가 배열보다 무조건 나을까? 그렇다고 할 순 없다.
이 둘은 JavaScript 에서 제공하는 자료구조일 뿐이다.
개발자는 상황에 따라서, 둘 중 알맞은 타입을 선택해야 한다.

빈 배열 [ ] 과 빈 객체 { } 는 false 가 아니다.

```
const a = {};  // 또는 []
if(a) {
    console.log(true);
} else {
    console.log(false);
}
```

즉, 둘은 false 가 아니라 true 이다.
빈 배열과 빈 객체의 조건을 정확히 비교하고 싶으면
if(a === [ ]) 식으로 써야.

같아보이는 배열과 객체는 비교 불가능이다.

```
const a = [1, 2, 3];
const b = [1, 2, 3];

if (a === b) {
    console.log("같다");
} else {
    console.log("다르다");
}
```

객체도 마찬가지이다.

```
const c = {
    myName: "조교행님",
    age: 28
}
const d = {
    myName: "조교행님",
    age: 28
}

if (c === d) {
    console.log("같다");
} else {
    console.log("다르다");
}
```

배열, 객체의 비교연산은 JavaScript 에서 제공하지 않기 때문이다.
만약 이런 기능을 원한다면 lodash 를 배워보길 추천한다.

자, 직접 실습해보는 시간이다.
나의 정보를 객체로 만들어보자.
이름, 나이, 직업, 학과, 사는곳
좋아하는것 (3개 이상의 배열)
객체와 객체 안의 배열 요소에 접근해 원하는대로 변경해보라.