IV. 콜백 함수

1. 콜백 함수는 다른 코드(함수 또는 메서드)의 인자로 넘겨주는 함수이다. 이때, 그 제어권도 함께 위임하며 제어권을 넘겨받은 코드는 콜백 함수 호출 시점에 대한 제어권을 가진다.

2. 콜백 함수는 함수다. 어떤 함수의 인자에 객체의 메서드를 전달하더라도 이는 결국 메서드가 아닌 함수일 뿐이다. 이 차이를 정확이 이해하는 것이 중요하다.

```
var obj = {
    vals: [1, 2, 3],
    logValues: function(v, i) {
        console.log(this, v, i);
    }
};
obj.logValues(1, 2);                // { vals: [1, 2, 3], logVa
[4, 5, 6].forEach(obj.logValues)  // Window { ... } 4 0
```

3. 콜백 함수 내부의 this에 다른 값을 바인딩 하는 방법(1)
   → self에 변수 this를 담는 전통적인 방식
   a. 다양한 상황에서 원하는 객체를 바라보는 콜백 함수를 만들 수 있는 방법이다.
   b. 함수 재활용 측면에서 유리하다.

4. 콜백 함수 내부에서 this를 사용하지 않는 경우
   a. 전통적인 방식보다 훨씬 간결하고 직관적이다.
   b. this를 이용해 다양한 상황에 재활용할 수 없게 된다.

5. 콜백 함수 내부의 this에 다른 값을 바인딩 하는 방법(2)
   →
   bind 메서드 활용

```
var obj1 = {
    name: 'obj1',
    func: function () {
        console.log(this.name);
    }
};
setTimeout(obj1.func.bind(obj1), 1000);

var obj2 = { name: 'obj2' };
setTimeout(obj1.func.bind(obj2), 1500);
```

6. CPU의 계산에 의해 즉시 처리가 가능한 대부분의 코드는 동기적인 코드이다. 반면 setTimeout, addEventListener, XMLHttpRequest 등 별도의 요청, 실행 대기, 보류 등과 관련된 코드는 비동기적인 코드이다.
7. 콜백 지옥을 해결하는 방법에는 여러가지가 있다.
   a. 기명함수로 변환
   b. Promise : 비동기 작업의 동기적 표현
   c. Generator : 비동기 작업의 동기적 표현
   d. **Promise + Async/await** : 비동기 작업의 동기적 표현