```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>chaining promises</title>
</head>

<body>

    <script>
        /*
,            ***** chaining promises ****
,
,            amISexy가 Promise라고 하고 resolve를 해볼것이다.
,            const amISexy = new Promise((resolve, reject) => {
,                resolve(2)
,            });
,            amISexy.then(number => console.log(number))
,
,            반환값은 2라고 잘 나올것이다 하지만. then 이후에 또 다른 Promise가 들어가면 어떻게 될까
,            가끔 하나의 Promise가 아닌 여러개를 쓰는 경우가 있다
,            결과값이 여러개가 나와야 하는경우. 예를들어
,
,            API로 가서 data를 받는다고 가정해보자.
,            data를 받는 부분은 then으로 나타나게 되고 data를 받고 나서는
,            암호화된 부분을 풀어줘야 할것이다. data가 암호로 되어있다고 가정하면.
,            암호를 푸는데 작업시간이 좀 걸리니 그부분은 두번째 then에 넣어줄것이다.
,            암호를 푼 data를 받을때 파일로 저장해준다고 해보자
,            그럼 then을 또 써야하고 원하는 만큼 쓰면된다.
,
,            * 왜냐면 이 모든 then은 서로의 순서가 끝나기만을 기다리기 때문
,            이렇게 하는 과정을 chaining이라고 한다. 서로서로 연결되어있다 체인처럼.
,
,            amISexy
,                .then(number => {
,                    console.log(2 * 2);
,                })
,                .then(otherNumber => {
,                    console.log(otherNumber * 2);
,                });
,            반환값은
,            4
,            NAN이 나온다.
,
,            Promise resolving으로 4까지는 제대로 나왔는데
,            otherNumber가 제대로 뜨지않는다.
,            그냥 콘솔에 찍어보면 undefined가 뜬다.
,            이것이 작동이 안되는 이유는 Promise들을 엮고 싶을때는
,            **** 기존의 then에서 return값이 있어야한다.
,
,            amISexy
,                .then(number => {
,                    console.log(number * 2);
,                    return number * 2;
,                })
,                .then(otherNumber => {
,                    console.log(otherNumber * 2);
,                });
```

```
,
,
,           then에서 return값이 값을 받을때는
,           return값이 다음 then의 변수쪽에 들어가야한다.
,
,           * error
,           이제 여러개중의 하나의 then에 에러가 생기면
,           그걸 catch할 수 있다.
,
,           return값을 만들어주자. arrow function을 사용하면 바로 return할 수 있다.
,
,           const timesTwo = (number) => number * 2
,
,           amISexy
,               .then(timesTwo)
,               .then(timesTwo)
,               .then(timesTwo)
,               .then(timesTwo)
,               .then(timesTwo)
,               .then(timesTwo)
,               .then(() => {
,                   throw Error("something is wrong");
,               })
,               .then(lastNumber => console.log(lastNumber));
,               .catch(error => console.log(error))
,
,           function이고 error가 나올수 있도록 설정해보았다.
,           현재는 lastNumber까지 진행되지 않았다.
,           throw Error 부분에서 erro가 발생하게 했기 때문.
,           그래서 프로세스가 끝까지 가지를 못한것이다.
,
,           만약 이 에러가 어떤것인지 찾아내려면 catch를 하면된다
,
,           then은 원하는 만큼 넣어도 그 중에 하나만 잡으면
,           catch가 다른 모든 error를 잡아줄것이다
,           다른 모든 then이랑 function안의 에러들까지.
,
,           const amISexy = new Promise((resolve, reject) => {
,               reject(2)
,           });
,           만약 이것이 reject였다면 우리는 error라는것을 먼저 받고
,           catch에서는 error를 2로받고 2를 콘솔에 찍을것이다.
,
,           * * 정리
,
,           return값을 지정해주지 않으면 그 다음에 then을 얻었을때에
,           아무것도 얻지 못할것이다.
,
,        */

        const amISexy = new Promise((resolve, reject) => {
            resolve(2)
        });

        const timesTwo = (number) => number * 2

        amISexy
            .then(timesTwo)
            .then(timesTwo)
            .then(timesTwo)
            .then(timesTwo)
```

```
            .then(timesTwo)
            .then(timesTwo)
            .then(() => {
                throw Error("something");
            })
            .then(lastNumber => console.log(lastNumber))
            .catch(error => console.log(error))

    </script>
</body>

</html>
```