비동기처리

동기: 순차적

콜백 함수 5초

```
setTimeout ( function () {} , 5000);
setTimeout (() => {}, 5000);   // 화살표 함수
```

: 각각 진행됨.

Callback - Hell

```
setTimout ( function() {
            function () {
            function () {},
            5000)
```

((파고드는 형태))

그다음 실행

next right after.

[Timer
  Ajax : frontend, backend 통신

☆ ↳ Promise

→ Await

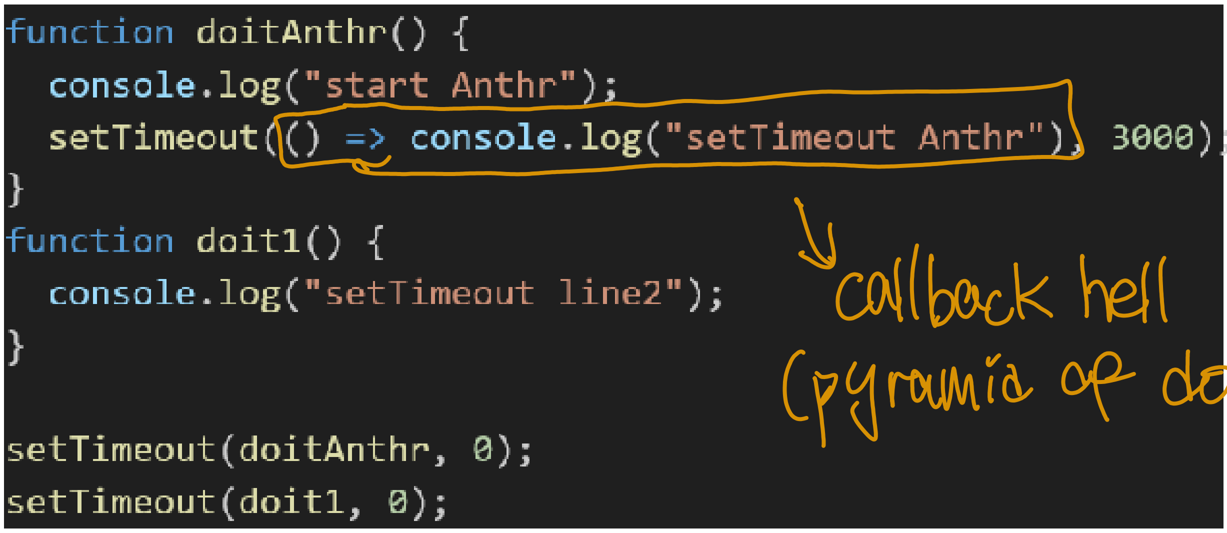
```
const mypromise = hello () { setTimeout ();
        if (    ){    }   success
        else {    }       fail
    }
```

Method Chain

It's just another syntax.

mypromise.then().catch().finally()

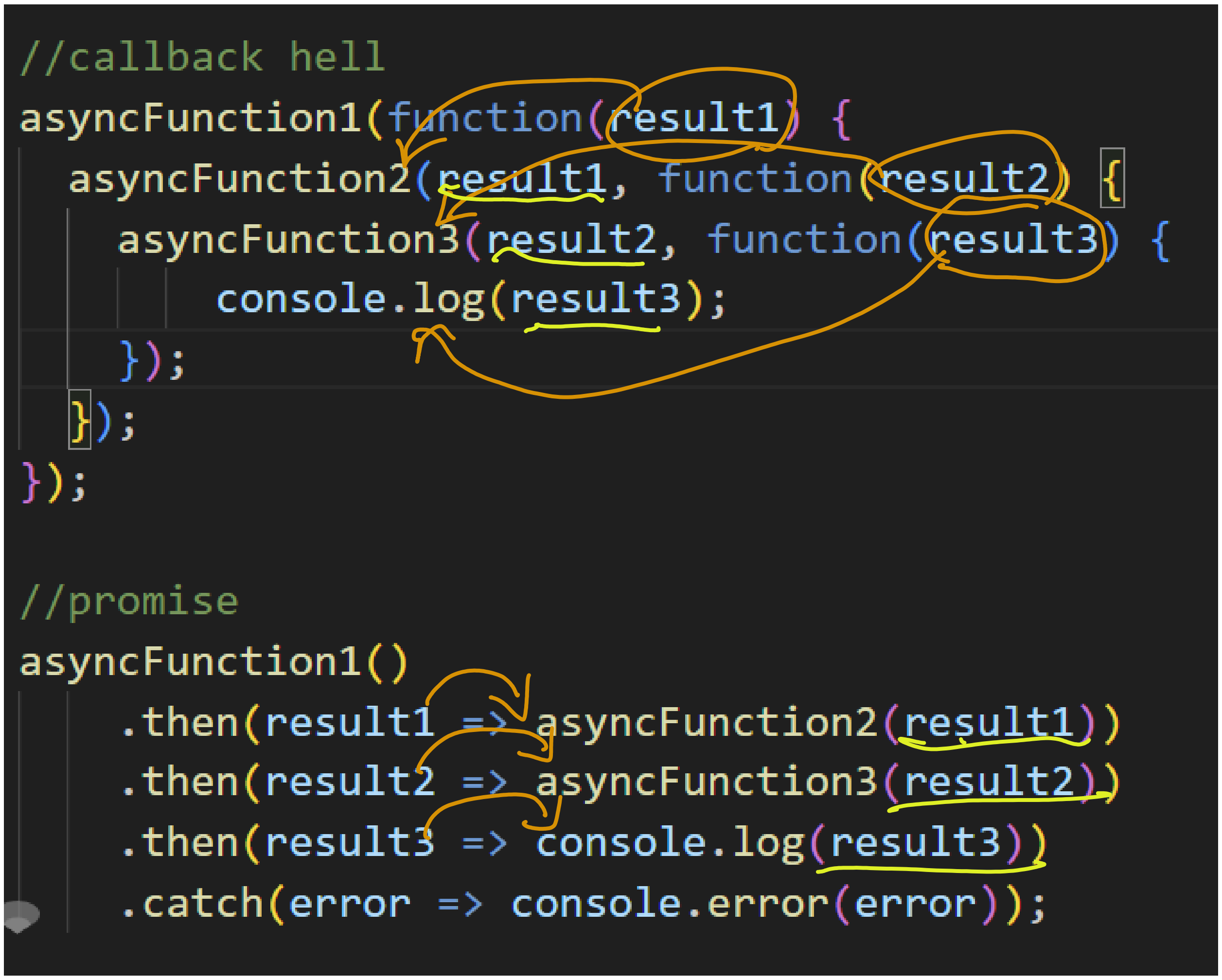
then()	catch()	finally()
resolve	reject	always
success	fail	at last



```
function doitAnthr() {
  console.log("start Anthr");
  setTimeout(() => console.log("setTimeout Anthr"), 3000);
}
function doit1() {
  console.log("setTimeout line2");
}

setTimeout(doitAnthr, 0);
setTimeout(doit1, 0);
```

callback hell
(pyramid of doom)



```
//callback hell
asyncFunction1(function(result1) {
  asyncFunction2(result1, function(result2) {
    asyncFunction3(result2, function(result3) {
      console.log(result3);
    });
  });
});

//promise
asyncFunction1()
    .then(result1 => asyncFunction2(result1))
    .then(result2 => asyncFunction3(result2))
    .then(result3 => console.log(result3))
    .catch(error => console.error(error));
```