

'try..catch'와 에러 핸들링

try catch

Error

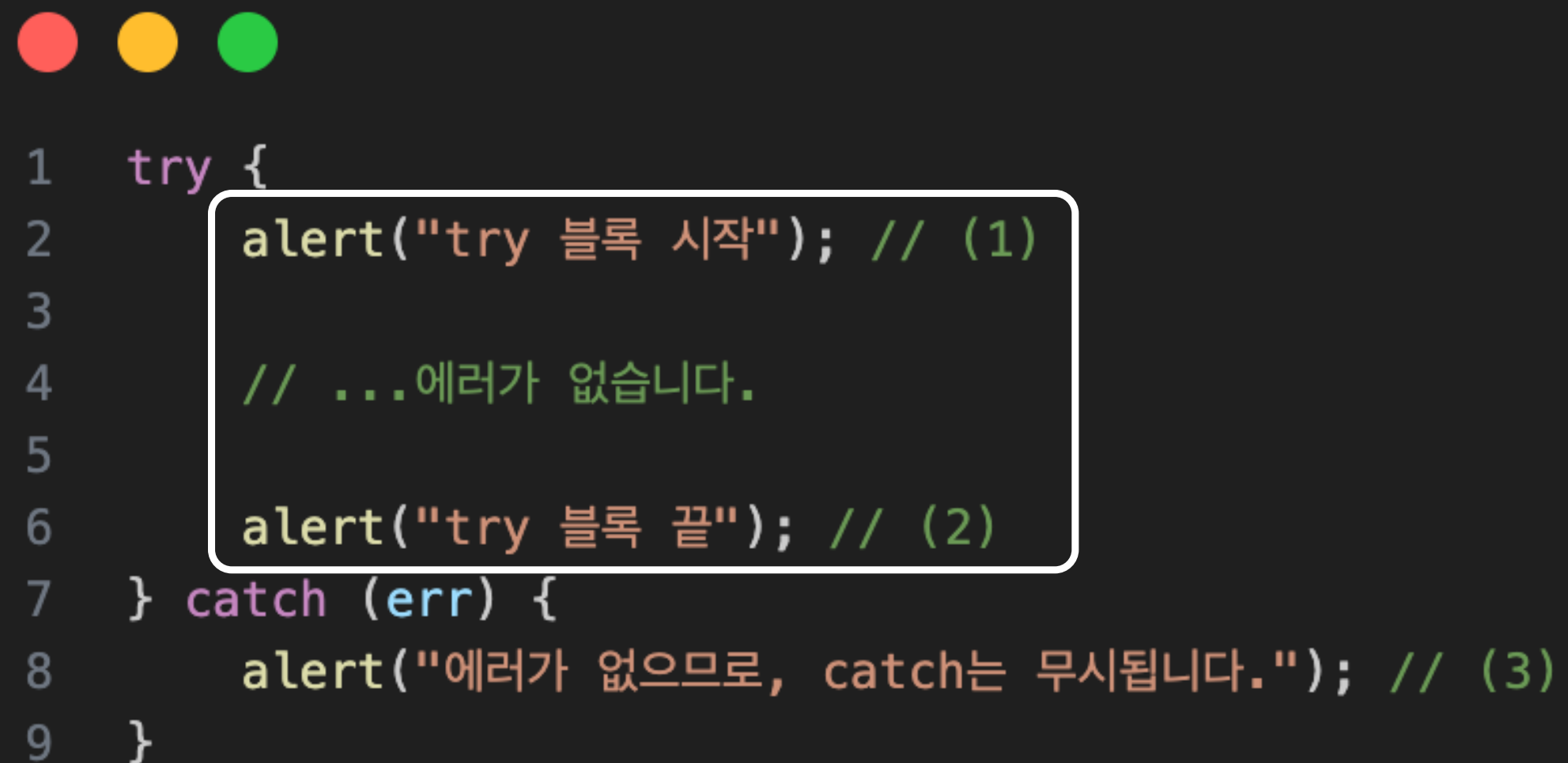
throw

try catch finally

finally와 return

try catch

에러 발생 시 스크립트가 죽는 걸 방지하고, 에러를 잡는다(catch)



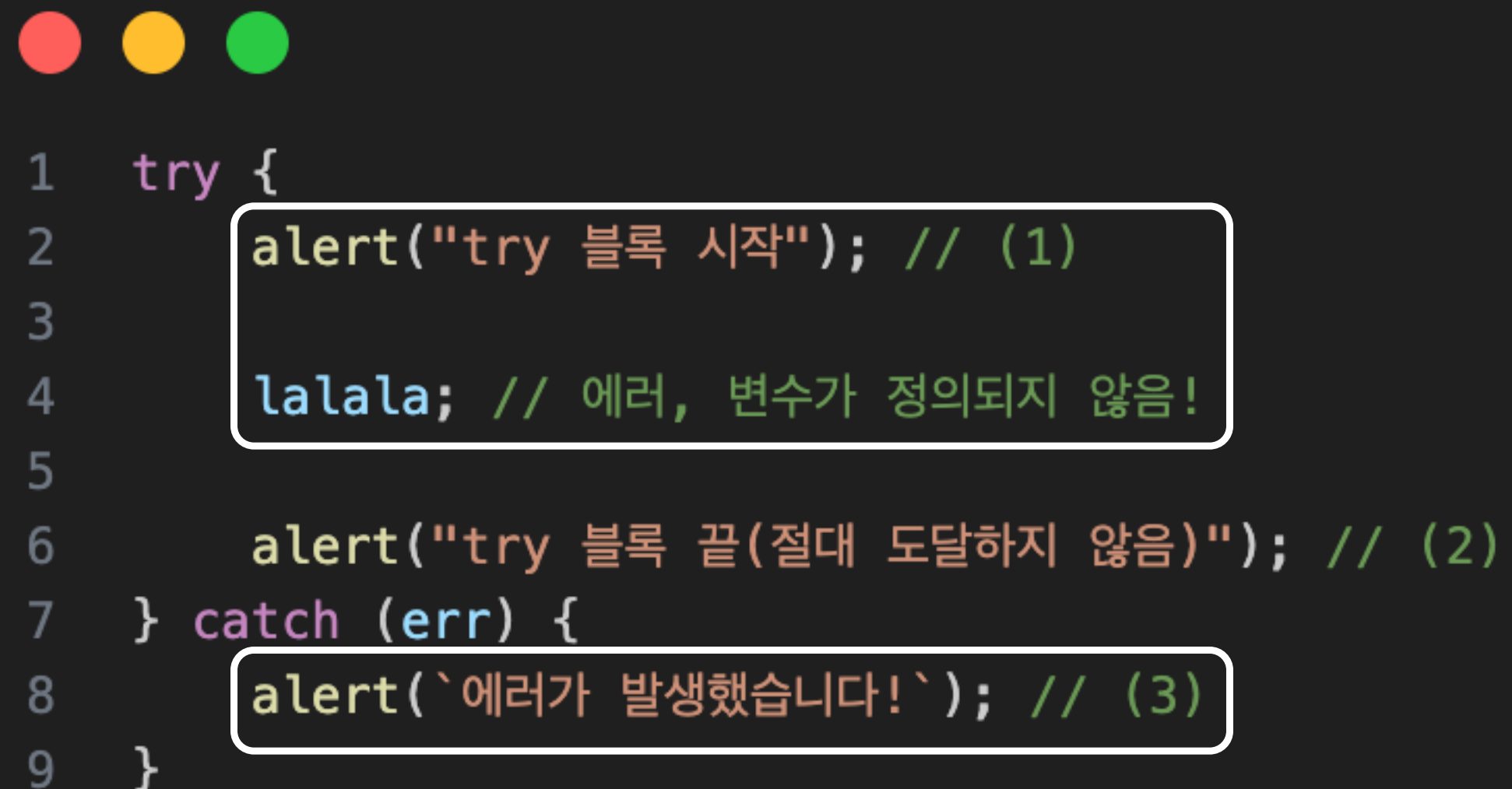
```
1  try {  
2      alert("try 블록 시작"); // (1)  
3  
4      // ...에러가 없습니다.  
5  
6      alert("try 블록 끝"); // (2)  
7  } catch (err) {  
8      alert("에러가 없으므로, catch는 무시됩니다."); // (3)  
9  }
```

에러가 없을 경우

1. try문 안에 있는 **(1)과 (2)**만 실행
2. catch문 안의 (3)은 무시

try catch

에러 발생 시 스크립트가 죽는 걸 방지하고, 에러를 잡는다(catch)



```
1  try {
2      alert("try 블록 시작"); // (1)
3
4      lalala; // 에러, 변수가 정의되지 않음!
5
6      alert("try 블록 끝(절대 도달하지 않음)"); // (2)
7  } catch (err) {
8      alert(`에러가 발생했습니다!`); // (3)
9  }
```

에러가 있을 경우

1. try문 안의 에러발생 전 **(1)**만 실행
2. error 발생
3. 바로 catch문 안의 **(3)** 실행

try catch

```
1  try {  
2      setTimeout(function () {  
3          noSuchVariable; // 스크립트는 여기서 죽습니다.  
4      }, 1000);  
5  } catch (e) {  
6      alert("작동 멈춤");  
7  }
```

try catch문은 동기적으로 동작한다



비동기 함수 안의 에러를 잡아내지 못한다

try catch

```
1  setTimeout(function () {  
2      try {  
3          noSuchVariable; // 이제 try..catch에서 에러를 핸들링 할 수 있습니다!  
4      } catch {  
5          alert("에러를 잡았습니다!");  
6      }  
7  }, 1000);
```

try catch문은 동기적으로 동작한다



비동기 함수 안의 에러를 잡아내지 못한다



스케줄 된 함수 내부의 예외를 잡으려면
try..catch를 반드시 함수 내부에 구현


error

```
1  try {  
2    lalala; // 에러, 변수가 정의되지 않음!  
3  } catch (err) {  
4    alert(err.name); // ReferenceError  
5    alert(err.message); // lalala is not defined  
6    alert(err.stack); // ReferenceError: lalala is not defined at ... (호출 스택)  
7  
8    // 에러 전체를 보여줄때 에러 객체는 "name: message" 형태의 문자열로 변환  
9    alert(err); // ReferenceError: lalala is not defined  
10 }
```

에러 객체

- **name** - 에러 이름. 정의되지 않은 변수 때문에 발생한 에러라면 "ReferenceError"가 이름이 됩니다.
- **message** - 에러 상세 내용을 담고 있는 문자 메시지
- **stack** - 현재 호출 스택. 에러를 유발한 중첩 호출들의 순서 정보를 가진 문자열로 디버깅 목적으로 사용됩니다.

error



```
1 let error = new Error(message);  
2 // or  
3 let error = new SyntaxError(message);  
4 let error = new ReferenceError(message);
```

표준 에러 객체의 종류

- **SyntaxError** - 언어 사양을 따르지 않는 코드 ex- { 를 빼먹은 경우
- **ReferenceError** - 정의되지 않은 변수 ex- 00 is undefined
- **TypeError** - 함수 또는 변수의 값이 예상치 못한 유형 ex- sharks.map is not a function

throw

문법적으로 잘못되진 않았지만,
스크립트 내에서 사용 중인 필수 프로퍼티 name을 가지고 있지 않다면?

```
1 let json = '{ "age": 30 }'; // 불완전한 데이터 (name 프로퍼티가 없다)
2
3 try {
4     let user = JSON.parse(json);
5     //name 프로퍼티가 없으면 오류 만들어서 던지기
6     if (!user.name) {
7         throw new SyntaxError("불완전한 데이터: name 없음");
8     }
9     alert(user.name);
10 } catch (e) {
11     alert("JSON Error: " + e.message); // JSON Error: 불완전한 데이터: name 없음
12 }
```


throw

throw 연산자와 에러 생성 연산자를 사용해 새로운 에러 객체를 만들어 던질 수 있다.

```
1 let json = '{ "age": 30 }'; // 불완전한 데이터 (name 프로퍼티가 없다)
2
3 try {
4   let user = JSON.parse(json);
5   //name 프로퍼티가 없으면 오류 만들어서 던지기
6   if (!user.name) {
7     throw new SyntaxError("불완전한 데이터: name 없음");
8   }
9   alert(user.name);
10 } catch (e) {
11   alert("JSON Error: " + e.message); // JSON Error: 불완전한 데이터: name 없음
12 }
```

try catch finally

finally안의 코드는 어떤 상황이라도 마지막에는 실행된다.



```
1  try {  
2      console.log("try 블록 시작");  
3      if (confirm("에러를 만드시겠습니까?")) 이상한_코드();  
4  } catch (e) {  
5      console.log("catch");  
6  } finally {  
7      console.log("finally");  
8  }
```

에러가 없을 경우

- try 실행이 끝난 후

에러가 있을 경우

- catch 실행이 끝난 후

try catch finally

finally안의 코드는 실행 결과에 상관없이 실행하고 싶을 경우 사용됩니다.



```
1  try {  
2      console.log("try 블록 시작");  
3      if (confirm("에러를 만드시겠습니까?")) 이상한_코드();  
4  } catch (e) {  
5      console.log("catch");  
6  } finally {  
7      console.log("finally");  
8  }
```

"에러를 만드시겠습니까?"에

'OK'로 답한 경우

try -> catch -> finally

'NO'로 답한 경우

try -> finally

finally와 return

```
1 function func() {  
2   try {  
3     return 1;  
4   } catch (e) {  
5     /* ... */  
6   } finally {  
7     alert("finally");  
8   }  
9 }  
10  
11 alert(func());
```

finally 절은 try..catch 절을 빠져나가는 어떤 경우에도 실행
return을 사용해 명시적으로 빠져나가려는 경우도 마찬가지!

finally와 return

```
1 function func() {  
2   try {  
3     return 1;  
4   } catch (e) {  
5     /* ... */  
6   } finally {  
7     alert("finally");  
8   }  
9 }  
10  
11 alert(func());
```

finally 절은 try..catch 절을 빠져나가는 어떤 경우에도 실행
return을 사용해 명시적으로 빠져나가려는 경우도 마찬가지!

이 페이지 내용:

finally

이 페이지 내용:

1

확인

finally 안의 alert가 실행되고 난 후, return 실행

finally를 쓰는 이유

try..catch에 '빠져나오게 하는' 코드가 있다면

finally O

```
1 function f() {  
2   try {  
3     console.log("시작");  
4     return "결과";  
5   } catch (e) {  
6     /// ...  
7   } finally {  
8     console.log("초기화!");  
9   }  
10 }  
11  
12 const result = f();  
13 console.log(result);
```

시작	VM14:3
초기화!	VM14:8
결과	VM14:13

finally X

```
1 function f() {  
2   try {  
3     console.log("시작");  
4     return "결과";  
5   } catch (e) {  
6     /// ...  
7   }  
8   console.log("초기화!");  
9 }  
10  
11 const result = f();  
12 console.log(result);
```

시작	VM18:3
결과	VM18:13

**finally 절을 붙여줘야
초기화가 보장!**

단순히 f의 끝에 붙였다면,
위와 같은 상황일 때
초기화 코드가 실행 X

finally를 쓰는 이유

try..catch에 '빠져나오게 하는' 코드가 있다면

finally O

```
1 function f() {
2   try {
3     console.log("시작");
4     throw new Error("에러 발생!");
5   } catch (e) {
6     /// ...
7     if ("에러를 핸들링 할 수 없다면") {
8       throw e;
9     }
10  } finally {
11    console.log("초기화!");
12  }
13 }
14
15 f();
```

시작	VM42:3
초기화!	VM42:11
✖ ▶ Uncaught Error: 에러 발생!	VM42:8
at f (<anonymous>:4:15)	
at <anonymous>:15:1	

finally X

```
1 function f() {
2   try {
3     console.log("시작");
4     throw new Error("에러 발생!");
5   } catch (e) {
6     /// ...
7     if ("에러를 핸들링 할 수 없다면") {
8       throw e;
9     }
10  }
11  console.log("초기화!");
12 }
13
14 f();
```

시작	VM46:3
✖ ▶ Uncaught Error: 에러 발생!	VM46:8
at f (<anonymous>:4:15)	
at <anonymous>:14:1	

finally 절을 붙여줘야
초기화가 보장!

단순히 f의 끝에 붙였다면,
위와 같은 상황일 때
초기화 코드가 실행 X

throw로 에러를 던지는 것도 같은 경우

THANK YOU