

Contents

- CHAPTER 08: 예외 처리

SECTION 8-1 구문 오류와 예외

SECTION 8-2 예외 처리 고급



CHAPTER 08 예외 처리

구문 오류와 예외를 구분하고, 예외 처리의 필요성과 예외를 강제로 발생시키는 방법을 이해

SECTION 8-1 구문 오류와 예외(1)

오류의 종류

- 구문 오류(syntax error): 프로그램 실행 전에 발생하는 오류
- 예외(exception) 또는 런타임 오류(runtime error): 프로그램 실행 중에 발생하는 오류
- 구문 오류
 - console.log() 메소드를 입력할 때 마지막에 닫는 괄호를 입력하지 않아서 괄호가 제대로 닫히지 않은 경우의 예

```
<script>
```

```
// 프로그램 시작 확인
```

```
console.log("# 프로그램이 시작되었습니다!")
```

```
// 구문 오류가 발생하는 부분
```

```
console.log("괄호를 닫지 않는 실수를 했습니다"
```

```
</script>
```

함수 뒤에 괄호를 닫지 않음

❗ 구문 오류

⊗ Uncaught SyntaxError: missing) after argument list

- 괄호가 닫히지 않았다고 바로 알려주므로 해당 위치의 괄호를 제대로 닫아주면 오류를 해결

SECTION 8-1 구문 오류와 예외(2)

◦ 오류의 종류

- 예외

- console.log() 메소드를 입력할 때 마지막에 닫는 괄호를 입력하지 않아서 괄호가 제대로 닫히지 않은 경우의 예

```
<script>  
  // 프로그램 시작 확인  
  console.log("# 프로그램이 시작되었습니다!")  
  // 구문 오류가 발생하는 부분  
  console.rog("log를 rog로 잘못 입력했습니다")  
</script>
```

→ 식별자를 잘못 입력

예외

프로그램이 시작되었습니다! → 일단 코드가 실행되었으므로 첫 번째 console.log() 메소드는 실행됨

⊗ Uncaught TypeError: console.rog is not a function

- console.rog() 줄을 읽는 순간 rog라는 식별자가 없어서 undefined인데, 이를 함수 호출 형태로 사용하니 "console.rog는 함수가 아니에요"라고 오류를 출력
- 오타자 수정을 통해서 간단하게 해결: 오타자를 고치는 것만으로는 해결할 수 없는 예외도 있음

SECTION 8-1 구문 오류와 예외(3)

- 기본 예외 처리

- `querySelector()` 메소드로 추출된 문서 객체가 없는 경우 (소스 코드 8-1-1.html)

```
01 <body>
02
03 </body>
04 <script>
05  document.addEventListener('DOMContentLoaded', () => {
06    const h1 = document.querySelector('h1')
07    h1.textContent = '안녕하세요'
08  })
09 </script>
```

오류

⊗ Uncaught TypeError: Cannot set property 'textContent' of null
at HTMLDocument.<anonymous> (test.html:7)

SECTION 8-1 구문 오류와 예외(4)

- 기본 예외 처리

- 기본 예외 처리 소스 코드 8-1-2.html)

```
01 <body>
02
03 </body>
04 <script>
05  document.addEventListener('DOMContentLoaded', () => {
06    const h1 = document.querySelector('h1')
07    if (h1) {
08      h1.textContent = '안녕하세요'
09    } else {
10      console.log('h1 태그를 추출할 수 없습니다.')
11    }
12  })
13 </script>
```

h1이 존재하면 true로 변환되고, 존재하지 않으면 false로 변환됨



실행 결과



h1 태그를 추출할 수 없습니다.

SECTION 8-1 구문 오류와 예외(5)

- 고급 예외 처리

- try catch finally 구문을 사용해서 예외를 처리

```
try {  
    // 예외가 발생할 가능성이 있는 코드  
} catch (exception) {  
    // 예외가 발생했을 때 실행할 코드  
} finally {  
    // 무조건 실행할 코드  
}
```

→ finally 구문은 필요한 경우에만 사용

- try 구문 안에서 예외가 발생하면 더 이상 try 구문을 진행하지 않고 catch 구문을 실행

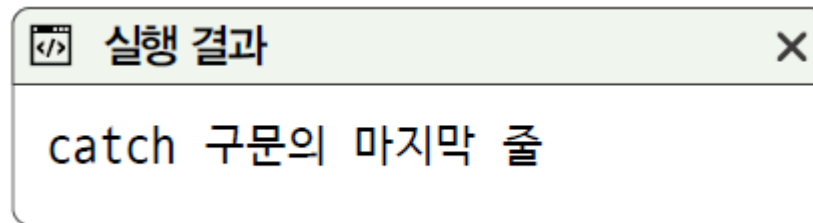
```
<script>  
    try {  
        willExcept.byeBye() → 예외를 발생  
    } catch (exception) {  
    }  
</script>
```

SECTION 8-1 구문 오류와 예외(6)

- 고급 예외 처리
 - try catch 구문의 사용 (소스 코드 8-1-3.html)

```
01 <script>
02 try {
03   willExcept.byeBye()
04   console.log("try 구문의 마지막 줄")
05 } catch (exception) {
06   console.log("catch 구문의 마지막 줄")
07 }
08 </script>
```

→ 위에서 예외가 발생하므로, 실행되지 않고 catch 구문으로 이동



SECTION 8-1 구문 오류와 예외(7)

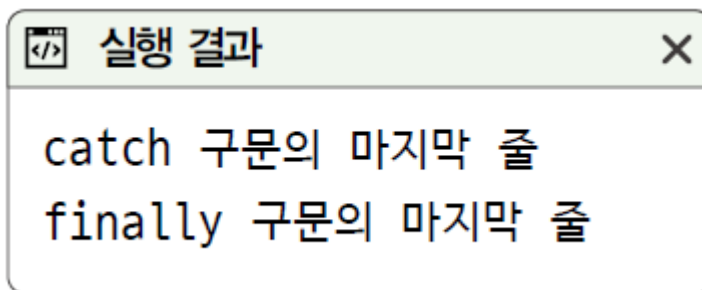
- 고급 예외 처리

- finally 구문 (소스 코드 8-1-4.html)

```
01 <script>
02 try {
03   willExcept.byeBye()
04   console.log("try 구문의 마지막 줄")
05 } catch (exception) {
06   console.log("catch 구문의 마지막 줄")
07 } finally {
08   console.log("finally 구문의 마지막 줄")
09 }
10 </script>
```

예외의 발생 여부와 상관 없이 무조건 실행됨

try 구문 내부에 있는 예외가 발생하는 코드를
지우고도 테스트 해볼 것

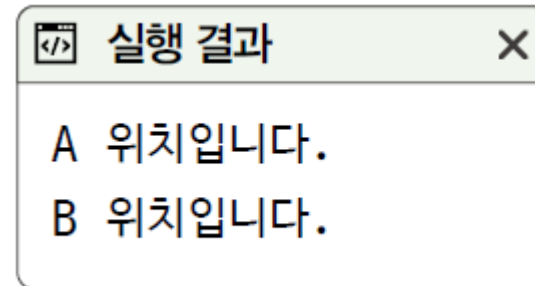


[좀 더 알아보기①] finally 구문을 사용하는 이유

- 예외 처리 구문 내부에서 return 사용하기(1) (소스 코드 8-1-5.html)

```
01 <script>
02 function test () {
03   try {
04     alert('A 위치입니다.')
05     throw "예외 강제 발생"
06   } catch (exception) {
07     alert('B 위치입니다.')
08     return
09   }
10   alert('C 위치입니다.')
11 }
12
13 // 함수를 호출합니다.
14 test()
15 </script>
```

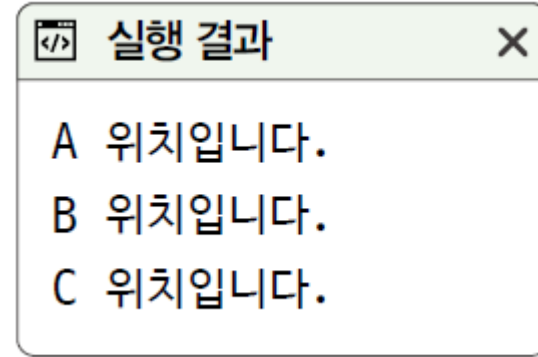
→ throw 키워드로 예외를 강제로 발생시킴
자세한 내용은 378쪽 '예외 강제 발생' 에서 학습



[좀 더 알아보기②] finally 구문을 사용하는 이유

- 예외 처리 구문 내부에서 return 사용하기(2) (소스 코드 8-1-6.html)

```
01 <script>
02 function test () {
03   try {
04     alert('A 위치입니다.')
05     throw "예외 강제 발생"
06   } catch (exception) {
07     alert('B 위치입니다.')
08     return
09   } finally {
10     alert('C 위치입니다.')
11   }
12 }
13
14 // 함수를 호출합니다.
14 test()
16 </script>
```



- 코드를 실행하면 예제(1)은 "A 위치입니다."와 "B 위치입니다."만 출력합니다. return 키워드를 사용해 함수를 벗어났으므로 "C 위치입니다."라는 글자를 출력하지 않는 것
- 하지만 예제(2)는 "A 위치입니다.", "B 위치입니다.", "C 위치입니다."를 모두 출력합니다. 이는 finally 구문을 반드시 실행한다는 특성 때문임

[마무리①]

◦ 4가지 키워드로 정리하는 핵심 포인트


- 구문 오류는 프로그램 실행 전에 발생하는 코드의 문법적인 문제로 발생하는 오류를 의미
- 예외는 프로그램 실행 중에 발생하는 모든 오류를 의미
- 예외 처리는 예외가 발생했을 때 프로그램이 중단되지 않게 하는 처리. 구문 오류는 예외 처리로 처리할 수 없음
- try catch finally 구문은 try 구문 안에서 예외가 발생하면 catch 구문에서 처리하고, finally 구문은 예외 발생 여부와 상관없이 실행해야 하는 작업이 있을 때 사용

SECTION 8-2 예외 처리 고급(1)

◦ 예외 객체

- 예외 객체(exception object): try catch 구문을 사용할 때 catch의 괄호 안에 입력하는 식별자. 아무 식별자나 입력해도 괜찮지만, 일반적으로 e나 exception이라는 식별자를 사용

```
try {  
} catch (exception) {  
}
```



- 예외 객체의 속성

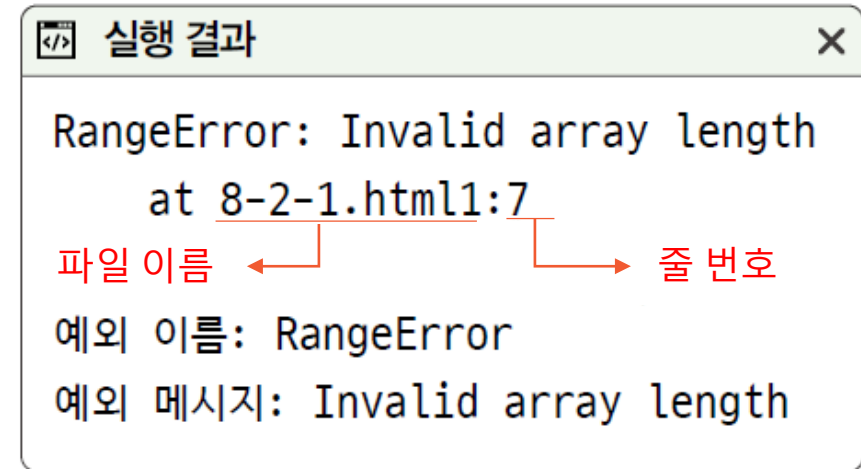
속성 이름	설명
name	예외 이름
message	예외 메시지

SECTION 8-2 예외 처리 고급(2)

◦ 예외 객체

- 자바스크립트의 배열 크기가 한정되어 있기 때문에 배열을 너무 크게 선언하면 오류를 발생하는 것을 이용해 이를 예외 처리하고, 오류를 출력해보는 코드(자바스크립트의 배열 크기는 4,294,967,295까지 가능)
- 예외 정보 출력하기 (소스 코드 8-2-1.html)

```
01 <script>
02 try {
03   const array = new Array(9999999999999999)
04 } catch (exception) {
05   console.log(exception)
06   console.log()
07   console.log(`예외 이름: ${exception.name}`)
08   console.log(`예외 메시지: ${exception.message}`)
09 }
10 </script>
```



SECTION 8-2 예외 처리 고급(3)

- 예외 강제 발생

- 예외를 강제로 발생시킬 때는 throw 키워드를 사용

```
// 단순히 예외를 발생시킵니다.  
throw 문자열  
// 조금 더 자세하게 예외를 발생시킵니다.  
throw new Error(문자열)
```

- 자바스크립트 콘솔에서 throw 구문을 사용

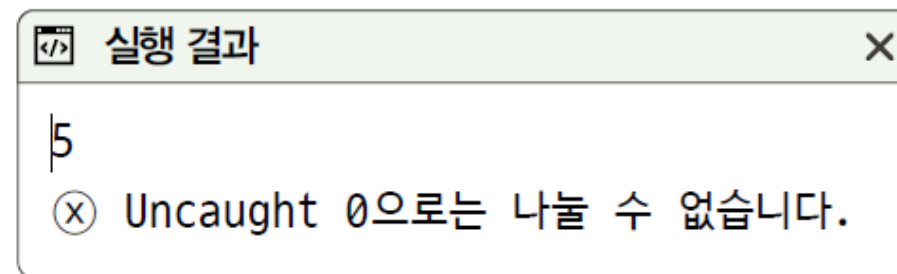
```
> throw '문자열'  
ⓧ Uncaught 문자열  
  
> throw new Error('문자열')  
ⓧ Uncaught Error: 문자열  
  at 파일 이름:줄 번호
```

SECTION 8-2 예외 처리 고급(4)

◦ 예외 강제 발생

- divide() 함수 예시: 함수 내부에는 0으로 나눌 때 '0으로는 나눌 수 없습니다'라는 오류를 발생하도록 작성
- 예외 강제로 발생시키고 잡기 (소스 코드 8-2-2.html)

```
01 <script>
02 function divide(a, b) {
03   if (b === 0) {
04     throw '0으로는 나눌 수 없습니다.'
05   }
06   return a / b
07 }
08
09 console.log(divide(10, 2))
10 console.log(divide(10, 0))
11 </script>
```

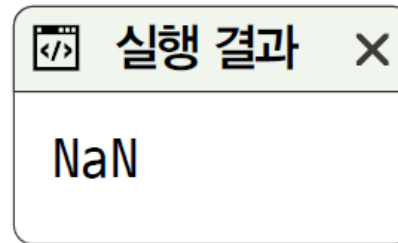


SECTION 8-2 예외 처리 고급(5)

- 예외 강제 발생

- 예외를 강제로 발생시키기 소스 코드 (8-2-3.html)

```
01 <script>
02  function test(object) {
03    console.log(object.a + object.b)
04  }
05
06  test({})
07 </script>
```

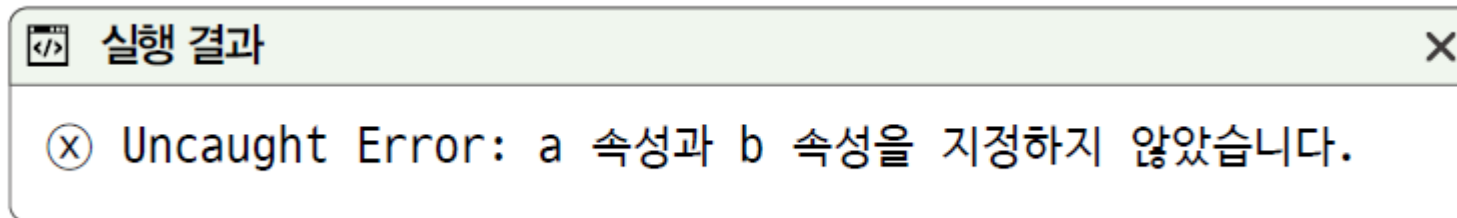


- 자바스크립트는 undefined와 NaN이라는 값이 있어서 다른 프로그래밍 언어에 비해서 예외를 많이 발생하지는 않음
- 그렇기 때문에 사용자에게 함수를 잘못 사용했다는 것을 강제로라도 인지시켜줄 필요가 존재함

SECTION 8-2 예외 처리 고급(6)

- 예외 강제 발생
 - 예외를 강제로 발생시켜서 사용 유도하기 (소스 코드 8-2-4.html)
 - 이렇게 코드를 작성하면 사용자가 코드의 문제점을 인지하고 해결할 수 있음

```
01 <script>
02  function test(object) {
03    if (object.a !== undefined && object.b !== undefined) {
04      console.log(object.a + object.b)
05    } else {
06      throw new Error("a 속성과 b 속성을 지정하지 않았습니다.")
07    }
08  }
09
10  test({})
11</script>
```



[마무리①]

- 2가지 키워드로 정리하는 핵심 포인트
 - 예외 객체는 예외와 관련된 정보를 담은 객체를 의미
 - throw 구문은 예외를 강제로 발생시킬 때 사용하는 구문