

# COMP1531

## ✓ Correctness - Exceptions

### Lecture 5.3

Author(s): Hayden Smith



[\(Download as PDF\)](#)

# In This Lecture

- **Why?** 🤔
  - Finding more graceful ways to deal with errors makes your program more robust
- **What?** 📄
  - Exceptions
  - Raising & Catching Exceptions



# Dealing With Problems

The simplest way to deal with problems at run-time...

## Just crash

```
1 import prompt from 'prompt-sync';
2 const promptFn = prompt();
3
4 function sqrt(x: number) {
5   if (x < 0) {
6     console.error('Error Input < 0');
7     process.exit(1);
8   }
9   return Math.pow(x, 0.5);
10 }
11
12 const input = promptFn('Please enter a number: ');
13 console.log(sqrt(parseInt(input)));
```

[5.3\\_just\\_crash.ts](#)

Not very clean though.

# Dealing With Problems

However, if we throw an exception 🤪 we start to get into a new territory of programming.

```
1 import prompt from 'prompt-sync';
2 const promptFn = prompt();
3
4 function sqrt(x: number) {
5   if (x < 0) {
6     throw new Error('Error Input < 0');
7   }
8   return Math.pow(x, 0.5);
9 }
10
11 const input = promptFn('Please enter a number: ');
12 console.log(sqrt(parseInt(input)));
```

5.3\_exception1.ts


# Dealing With Problems

However, if we throw an exception 🤪 we start to get into a new territory of programming.

```
1 import prompt from 'prompt-sync';
2 const promptFn = prompt();
3
4 function sqrt(x: number) {
5   if (x < 0) {
6     throw new Error('Error Input < 0');
7   }
8   return Math.pow(x, 0.5);
9 }
10
11 const input = promptFn('Please enter a number: ');
12 console.log(sqrt(parseInt(input)));
```

5.3\_exception1.ts

# Dealing With Problems

However, if we throw an exception  we start to get into a new territory of programming.

```
1 import prompt from 'prompt-sync';
2 const promptFn = prompt();
3
4 function sqrt(x: number) {
5   if (x < 0) {
6     throw new Error('Error Input < 0');
7   }
8   return Math.pow(x, 0.5);
9 }
10
11 const input = promptFn('Please enter a number: ');
12 console.log(sqrt(parseInt(input)));
```

5.3\_exception1.ts

Let's take a step back...

# Exceptions

An **exception** is an action that disrupts the normal flow of a program. This action is often representative of an error being thrown. Exceptions are ways that we can elegantly recover from errors.

# Exceptions

Exceptions are a particular method of ensuring **software safety**. Different languages have different conventions for managing unexpected runtime events.

Javascript relies on Exceptions for the majority of error handling. Unlike C, which has no exceptions





# Easier To Ask Forgiveness Than Permission

- EAFP is the javascript convention for handling errors.
- It encourages you to assume something will work and just have an exception handler to deal with anything that might go wrong
- Pros:
  - Can simplify the core logic
  - Multiple different sorts of errors can be handled with one except block
- Cons:
  - Makes code non-structured
  - Harder to reason what code will be executed.



# Look Before You Leap

- LBYL is a convention for avoiding errors in popular languages like C
- Unlike EAFP it encourages you to check that something can be done before you do it
- Pros:
  - Doesn't require exceptions
  - Code is structured and therefore easier to reason about
- Cons:
  - Core logic can be obscured by error checks

# Exception Examples

This program is good in that it throws an exception, but we aren't handling it.

```
1 import prompt from 'prompt-sync';
2 const promptFn = prompt();
3
4 function sqrt(x: number) {
5   if (x < 0) {
6     throw new Error('Error Input < 0');
7   }
8   return Math.pow(x, 0.5);
9 }
10
11 const input = promptFn('Please enter a number: ');
12 console.log(sqrt(parseInt(input)));
```

5.3\_exception1.ts

# Exception Examples

This program is good in that it throws an exception, but we aren't handling it.

```
1 import prompt from 'prompt-sync';
2 const promptFn = prompt();
3
4 function sqrt(x: number) {
5   if (x < 0) {
6     throw new Error('Error Input < 0');
7   }
8   return Math.pow(x, 0.5);
9 }
10
11 try {
12   const input = promptFn('Please enter a number: ');
13   console.log(sqrt(parseInt(input)));
14 } catch (err) {
15   console.error(`Error when inputting! ${err}`);
16   const input = promptFn('Please enter a number: ');
17   console.log(sqrt(parseInt(input)));
18 }
```

5.3\_exception2.ts

# Exception Examples

Or we could make this even more robust

```
1 import prompt from 'prompt-sync';
2 const promptFn = prompt();
3
4 function sqrt(x: number) {
5   if (x < 0) {
6     throw new Error('Error Input < 0');
7   }
8   return Math.pow(x, 0.5);
9 }
10
11 let success = false;
12 while (!success) {
13   try {
14     const input = promptFn('Please enter a number: ');
15     console.log(sqrt(parseInt(input)));
16     success = true;
17   } catch (err) {
18     console.error(`Error when inputting! ${err}`);
19   }
20 }
```

5.3\_exception3.ts

# Exception Examples

- Key points:
  - Exceptions carry data
  - When exceptions are thrown, normal code execution stops

```
1 function sqrt(x: number) {
2   if (x < 0) {
3     throw new Error('Error Input < 0');
4   }
5   return Math.pow(x, 0.5);
6 }
7
8 if (process.argv.length === 3) {
9   try {
10    console.log(sqrt(parseInt(process.argv[2])));
11    console.log('Never called if error!');
12  } catch (err) {
13    console.error(`Error when inputting! ${err}`);
14  }
15 }
```

5.3\_throw\_catch.ts



# Testing With Exceptions

We can use jests `toThrowError` function to test if functions are appropriately throwing exceptions.

```
1 function sqrt(x: number) {
2   if (x < 0) {
3     throw new Error('Error Input < 0');
4   }
5   return Math.pow(x, 0.5);
6 }
7
8 export { sqrt };
```

5.3\_sqrt.ts

```
1 import { sqrt } from './5.2_sqrt';
2
3 describe('sqrt correctness', () => {
4   test('deals with valid bases', () => {
5     expect(sqrt(4)).toEqual(2);
6     expect(sqrt(2)).toBeCloseTo(1.414213, 5);
7   });
8   test('throws error on negatives', () => {
9     // Note that these require a function, not result
10    expect(() => sqrt(-2)).toThrow('Error: Input < 0');
11    expect(() => sqrt(-5)).toThrowError('Error: Input < 0');
12  });
13 });
```

5.3\_catch.test.ts

# Feedback



Or go to the [form here](#).



