

Error Handling

❶ Throwing errors

A number of functions in SwiftUI are marked with the `throws` keyword. This means that they can “throw” an error at you when you call them.



Error Handling

❶ Throwing errors

A number of functions in SwiftUI are marked with the **throws** keyword. This means that they can “throw” an error at you when you call them.

The code inside them looks something like this ...

```
func attendLecture() throws {  
    if sleptIn {  
        throw CS193pError.missedLecture  
    }  
    askQuestions() // not done if you sleptIn because throw exits attendLecture()  
    . . .  
}  
  
enum CS193pError: Error {    // thrown things conform to Error (empty) protocol  
    case lateHomework(daysLate: Int)  
    case missedLecture  
}
```



Error Handling

- You must try

When you call a function that can throw (i.e. it is marked with throws), you must “try” it.



Error Handling

- You must try

When you call a function that can throw (i.e. it is marked with throws), you must “try” it.

Imagine a function that reads some data from a url (and throws when there's an i/o error) ...

```
func read(from: URL) throws
```

To call this, you must use the try keyword ...

```
try read(from: url)
```

There are different ways to try depending on how you want to handle an error thrown at you ...

Error Handling

- Choosing not to handle an error throw at you

There are three ways to try which don't handle any error thrown at you ...



Error Handling

- Choosing not to handle an error throw at you

There are three ways to try which don't handle any error thrown at you ...

`try?` ignores any error that is thrown and returns `nil` instead

```
if let imageData = try? Data(contentsOf: url) { ... }
```



Error Handling

- Choosing not to handle an error throw at you

There are three ways to try which don't handle any error thrown at you ...

`try?` ignores any error that is thrown and returns `nil` instead

```
if let imageData = try? Data(contentsOf: url) { ... }
```

```
try? someFunctionThatCanThrowAndDoesntReturnSomething()
```

Error Handling

● Choosing not to handle an error throw at you

There are three ways to try which don't handle any error thrown at you ...

`try?` ignores any error that is thrown and returns nil instead

```
if let imageData = try? Data(contentsOf: url) { ... }
```

```
try? someFunctionThatCanThrowAndDoesntReturnSomething()
```

`try!` crashes your program if an error is thrown

```
try! data.write(to: url)
```



Error Handling

• Choosing not to handle an error throw at you

There are three ways to try which don't handle any error thrown at you ...

`try?` ignores any error that is thrown and returns `nil` instead

```
if let imageData = try? Data(contentsOf: url) { ... }
```

```
try? someFunctionThatCanThrowAndDoesntReturnSomething()
```

`try!` crashes your program if an error is thrown

```
try! data.write(to: url)
```

`try` inside a function that is itself marked as `throws` (this rethrows any error that is thrown)

```
func foo() throws {
```

```
    try somethingThatThrows()
```

```
}
```

In this last scenario, any code that calls `foo()` must do so with `try` (e.g. `try foo()`).

Error Handling

- Actually handling a thrown error

Sometimes you actually want to find out what error was thrown at you, of course.



Error Handling

• Actually handling a thrown error

Sometimes you actually want to find out what error was thrown at you, of course.

You do this by wrapping a `do { } catch` around your code that is going to try something.

```
do {  
    try functionThatThrows()  
} catch let error {  
    // handle the thrown error here (e.g. CS193pError.missedLecture)  
}
```

The `"let error"` can be left off if you want (Swift assumes it if you leave it off).

Or you can change it to `"let foo"` in which case the error variable inside the catch will be `foo`.



Error Handling

- **Ways to catch**

You can be more specific about the kinds of errors you want to handle.



Error Handling

• Ways to catch

You can be more specific about the kinds of errors you want to handle.

You do this by adding catch phrases to your `do { } catch { } ...`

```
do {  
    try somethingThatThrows()  
    print("hello") // will only happen if somethingThatThrows() does not throw anything  
} catch CS193pError.lateHomework(let daysLate) {  
    // notice how daysLate is grabbing some error-specific enum associated data  
} catch let cs193pError where cs193pError is CS193pError {  
    // handle all CS193pErrors other than .lateHomework (which is handled above)  
} catch {  
    // handle all other (non-CS193pErrors, since those are all handled above)  
}  
print("keep going") // will execute next after any caught error is handled
```

