# Error Handling

## Error Protocol

* Error is basically a protocol that you have to adopt.
* You can adopt it in structs
  + **enum PrinterError: Error {**
  + **case outOfPaper**
  + **case noToner**
  + **case onFire**
  + **}**

## Throw

* Use throw to throw an error and throws to mark a function that can throw error
  + **func send(job: Int, toPrinter printerName: String) throws -> String {**
  + **if (printerName == "Never Has Toner")**
  + **throw PrinterError.noToner**
  + **}**
  + **return "Job sent"**
  + **}**

## Catch

* First way is to use a do catch
  + **do {**
  + **let printerResponse = try send(job: 1040, toPrinter: "Bi Sheng")**
  + **print(printerResponse)**
  + **} catch {**
  + **print(error)**
  + **}**
* You can also use multiple catch
  + **do {**
  + **let printerResponse = try send(job: 1440, toPrinter: "Gutenberg")**
  + **print(printerResponse)**
  + **} catch PrinterError.onFire {**
  + **print("I'll just put this over here, with the rest of the fire.")**
  + **} catch let printerError as PrinterError {**
  + **print("Printer error: \(printerError).")**
  + **} catch {**
  + **print(error)**
  + **}**
* You can use try? to convert the result to an optional. If the function throws an error then error is discarded and nil is saved and if no error is thrown then the result is an optional that contains the value returned by the function.
  + **let printerSuccess = try? send(jo: 1884, toPrinter: "NeverHasToner")**