Pod init means to initialize a podfile

<https://cocoapods.org>

Sudo gem install cocoapods -> pod init

pod deintegrate

Article for installing Cocoapods for newer Macbook with M1

<https://medium.com/codex/simple-way-to-use-cocoapods-on-m1-ac9e22cf7e1c>

Image Saving and getting from Core Data

<https://betterprogramming.pub/how-to-save-an-image-to-core-data-with-swift-a1105ae2cf04>

**Error handling**

**Why we do it**

Until now, when we found errors in our apps we just fixed them. There was always a certain outcome from our functions because we were the ones plugging in the data.

**What happens when we need data from a website or an API?**

Sometimes, internet connections are intermittent. Sometimes requests time out. There are all kinds of reasons why a function might break, so sometimes we need to "handle" any errors that might occur.

**What it looks like**

do {

try myFunction(){

// ... you can also try assignments ( ie: try let x = someValueThatMightError )

}

} catch specificError1 {

// ... code for dealing with specificError1

} catch specificError2 {

// ... code for dealing with specificError2

} catch {

// ... code for dealing with general errors. we must be exhaustive

}

This way, we can tell our applications what to run when they encounter errors. Our applications can dynamically handle varying outcomes from user input and API requests without crashing.

**Basic Examples**

Here's a function that returns a Bool, but throws an error if something goes wrong first

func doUnsafeThings( badStuffHappened: Bool, somethingUnlikely: String? ) throws -> Bool {

if ( badStuffHappened == true ) {

throw Error

}

// if you made it here, you can return:

return true

}

Here, we TRY to call this function and set CATCH operations for what to do if it fails:

do {

let theResult = try doUnsafeThings()

} catch error {

// ... deal with error bc dangerousStuff might not work out

}

**Handling Multiple Error Types**

Say we scaled our "doUnsafeThings" function, and we have many types of errors that could potentially occur.

Here's how we could define multiple error types with an **enum**:

enum BadStuffError: Error { // error type can be extended to our specific case/need

case Bad

case Worse

case Terrible

}

**Structured Example**

func doUnsafeStuff() throws -> Bool {

if badStuffHappened == true {

throw BadStuffError.Bad

}

if worseStuffHappened == true {

throw BadStuffError.Worse

}

if TerribleStuffHappened == true {

throw BadStuffError.Terrible

}

return true

}

We now have a function that will "throw" errors under certain conditions.

**Catching thrown errors**

do {

let result = try doUnsafeStuff()

} catch BadStuffError.Bad {

// ... code to execute

} catch BadStuffError.Worse {

// ... code to execute

} catch BadStuffError.Terrible {

// ... code to execute

}

**The Guard Statement**

Writing clean, comprehensible code often saves us time and errors. The **guard** keyword allows us to write simpler conditionals

// if/else syntax

func foo() {

if let value = anotherValue {

// ... code for successful assignment

}

// ... code for unsuccessful assignment

return

}

// guard syntax

func fooGuard() {

guard let value = anotherValue else {

// ... deal with unsuccessful assignment, throw an error, return out of function, etc.

return

}

// just keep coding, just keep coding ...

}

This way we can do our checks up front and keep organized ( and error free ).

# CocoaPods

In this lesson, we will learn about CocoaPods. **CocoaPods** is a dependency manager for Swift and Objective-C Cocoa projects.

Please watch the video below and install CocoaPods.

Before using CocoaPods, please navigate to <https://cocoapods.org/> and install CocoaPods.