

# MOBILE DEVELOPMENT

## DELEGATES, SEGUES, ANYOBJECT

*Kishin Manglani*

---

# INTRO TO SWIFT

---

## AGENDA

- Recap
- Delegates
- Segues
- AnyObject

---

**INTRO TO SWIFT**

---

**RECAP**

# COMPUTED PROPERTIES

- We can use computed properties when both getting and setting a variable

```
class Square {  
    var sideLength = 100  
  
    var area: Int {  
        get {  
            return sideLength * 2  
        }  
        set(newArea) {  
            sideLength = newArea / 2  
        }  
    }  
}
```

# INTRO TO SWIFT

---

```
func someFunction(firstParameter: Int, secondParameter: Int) {  
    //body  
}  
someFunction(2, secondParameter: 3)
```

```
func exampleFunction(firstName firstParameter: Int, secondParameter: Int) {  
    //body  
}  
exampleFunction(firstName: 4, secondParameter: 9)
```

```
func similarFunction(firstName firstParameter: Int, secondName secondParameter: Int) {  
    //body  
}  
similarFunction(firstName: 9, secondName: 9)
```

```
func anotherFunction(firstParameter: Int, _ secondParameter: Int) {  
    //body  
}  
anotherFunction(2, 3)
```

# STRUCTS

```
class SomeClass {  
    // class definition goes here  
}  
  
struct SomeStructure {  
    // structure definition goes here  
}
```

# ENUMS

# INTRO TO SWIFT

---

## ENUMS

```
enum Month {  
    case January  
    case February  
    case March  
    case April  
    case May  
    case June  
    case July  
    case August  
    case September  
    case October  
    case November  
    case December  
}
```

```
enum Season {  
    case Winter, Summer, Spring, Fall  
}
```



## INTRO TO SWIFT

---

# ENUMS

```
enum Month: Int {  
    case January  
    case February  
    case March  
    case April  
    case May  
    case June  
    case July  
    case August  
    case September  
    case October  
    case November  
    case December  
}  
Month.January.rawValue  
Month.February.rawValue
```

0

1

## INTRO TO SWIFT

---

# ENUMS

```
//Declaring an enum
enum Month: Int {
    case January = 1
    case February = 2
    case March = 3
    case April = 4
    case May = 5
    case June = 6
    case July = 7
    case August = 8
    case September = 9
    case October = 10
    case November = 11
    case December = 12
}
Month.January.rawValue
Month.February.rawValue
```

1  
2

## INTRO TO SWIFT

---

# ENUMS

```
enum Month: Int {  
    case January = 1  
    case February  
    case March  
    case April  
    case May  
    case June  
    case July  
    case August  
    case September  
    case October  
    case November  
    case December  
}  
Month.January.rawValue  
Month.February.rawValue
```

1

2

## INTRO TO SWIFT

---

# ENUMS

```
enum Coin: Int {  
    case Penny = 1, Nickel = 5, Dime = 10, Quarter = 25  
}
```

```
enum TaskPriority: Int {  
    case Low = 0  
    case Medium = 1  
    case High = 2  
}
```

## INTRO TO SWIFT

---

# ENUMS

- A `rawType` gives an enum an initializer
- We can then initialize an enum with a `rawValue`
- These are “failable” initializers: if something goes wrong (like the `rawValue` is not compatible) the initializer can return `nil`

```
TaskPriority(rawValue: 0)
```

Low

```
TaskPriority(rawValue: 909)
```

nil

---

## INTRO TO SWIFT

---

# CLASS, STRUCT, ENUM

- Enums should represent a finite number of possibilities
- We can think of these as named labels
- Structs are a base type and cannot be subclassed,
- All functions and properties pertain to the instance
- Should not manipulate properties, etc. of other objects
- Is not the complete definition or model of your app
- Classes should be the primary actor/target of your app
- They represent the data model of your app
- Manipulate other objects properties

---

## INTRO TO SWIFT

---

# CLASS, STRUCT, ENUM

- Remember, Classes, Structs and Enums create data types
- In other words, we can make variables or properties take on the types that we create if they are a class, struct or enum

---

**INTRO TO SWIFT**

---

# STATIC & CLASS



## INTRO TO SWIFT

---

# STATIC & CLASS

```
class Dog {  
    static let dogNoise = "Bark bark"  
    var defaultDogAge = 0  
  
    func makeNoise() {  
        print("\(self.defaultDogAge) \(Dog.dogNoise)")  
    }  
  
    static func makeNoiseStatic() {  
        print("\(Dog.dogNoise)")  
    }  
}  
  
Dog.makeNoiseStatic()  
var puppy = Dog()  
puppy.makeNoise()
```

"0 Bark bark\n"

"Bark bark\n"

Dog  
Dog

---

**INTRO TO SWIFT**

---

# **SWITCH STATEMENTS**

## INTRO TO SWIFT

---

# SWITCH STATEMENTS

```
switch some value to consider {  
  case value 1 :  
    respond to value 1  
  case value 2 ,  
    value 3 :  
    respond to value 2 or 3  
  default:  
    otherwise, do something else  
}
```

## INTRO TO SWIFT

---

# SWITCH STATEMENTS

```
var age = 30

switch age {
case 0:
    print("You are 0 years old")
case 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40:
    print("You are in your 30's")
case 40...49:
    print("you are in your 40's")
default:
    print("You are not 20 and not 30")
}
```

---

## INTRO TO SWIFT

---

# IF/ELSE, FOR LOOP, WHILE LOOP, SWITCH

- If else and switch are pretty similar
- If there are two cases, probably use an if
- If there are many cases, use a switch
- When iterating through an array use a for loop
- When iterating through a finite or known quantity use a for loop
- When iterating endlessly or an indefinite amount of time, use a while loop

---

**INTRO TO SWIFT**

---

**DELEGATE**

## INTRO TO SWIFT

---

# PROTOCOLS

- › A group of related properties and methods that can be implemented by any class
- › More flexible than a normal class interface, since they let you reuse a single API declaration in completely unrelated classes
- › Also, we don't have to override or implement a method from a superclass
- › With a protocol we can make certain methods required and others optional

## INTRO TO SWIFT

---

# PROTOCOLS

- › Here's an example:

```
protocol Swimmer {  
    func swim()  
}
```

- › Notice how we do not need to provide an implementation (any code for the function) it is just the definition



# PROTOCOLS

- › Here's how we adopt the protocol

```
class Frog: Animal, Swimmer {  
    func swim() {  
        print("I'm swimming")  
    }  
}
```

- › When adopting the protocol we need to implement the required methods

## INTRO TO SWIFT

---

# PROTOCOLS

- › Here's an example of an optional method:

```
protocol Swimmer {  
    func swim()  
    optional func drown()  
}
```

---

## INTRO TO SWIFT

---

# DELEGATE

- In English, delegating means doing something on behalf of someone else
- It's the same thing in iOS, a delegate object performs some task for another object
- The object uses the delegate to notify other objects of events
- A delegate is just some class that implements a protocol

---

## INTRO TO SWIFT

---

# DELEGATE

- The idea behind delegates is that instead of class A executing some code, it tells it's delegate to execute that code. The delegate is tied to another class (let's call it class B). In order to facilitate this, class A creates something called a protocol. This protocol has a list of methods in it (with no definition). Class A then has an instance of the protocol as a property. Class B has to implement the protocol defined in class A. Lastly, when class A is created, it's delegate property is set to class B.

<http://chrisrisner.com/31-Days-of-iOS--Day-6%E2%80%93The-Delegate-Pattern/>

---

**INTRO TO SWIFT**

---

# SEGUES

---

## INTRO TO SWIFT

---

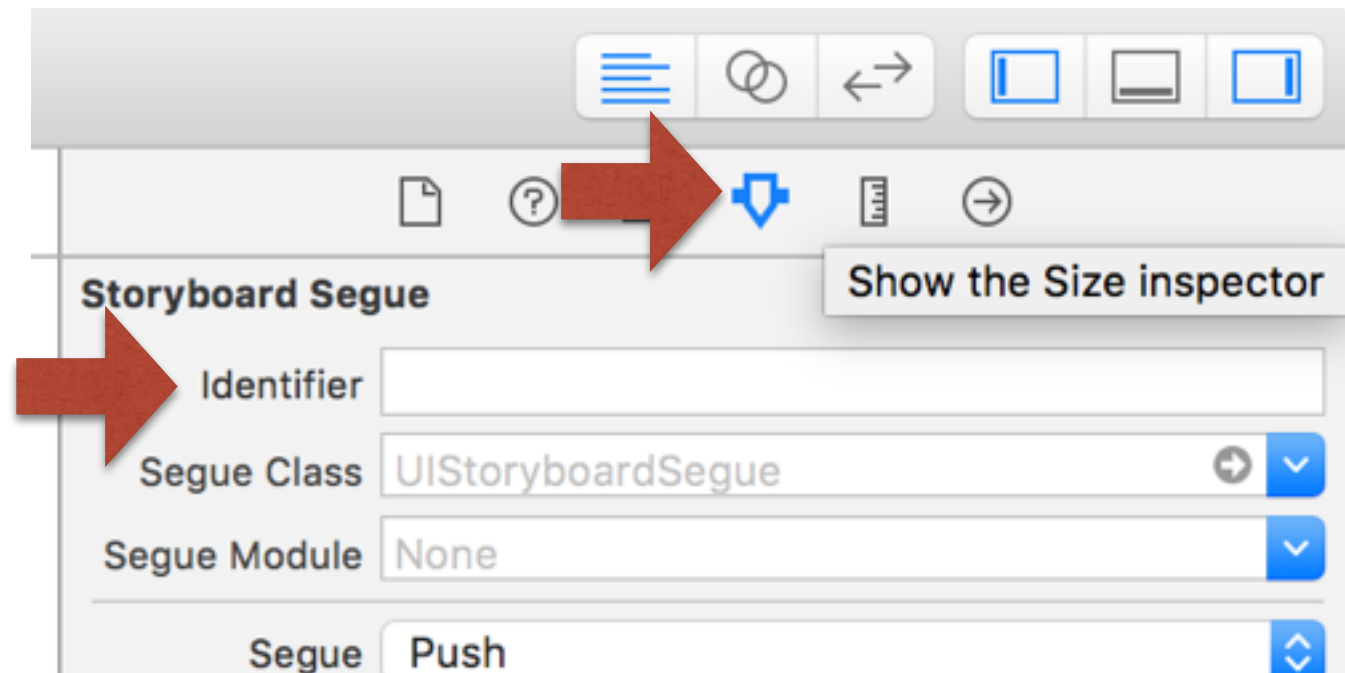
# SEGUES

- What is a segue?
- How do we create a segue?

## INTRO TO SWIFT

# SEGUES

- ▶ Can set a specific identifier/name for a segue
- ▶ We can then call the identifier in our code
- ▶ To set the name, select the Segue in storyboard and specify a name in the Attributes inspector under “Identifier”



## INTRO TO SWIFT

---

# SEGUES

- Now to perform the segue we call a special function that's built right into our view controller (it inherits it from UIViewController)
- Here, “segueName” is the name that we specified in our Storyboard, when we selected our segue and set a name
- A very common error is misspelling the name!

```
performSegueWithIdentifier("segueName", sender: self)
```





## INTRO TO SWIFT

---

# SEGUES

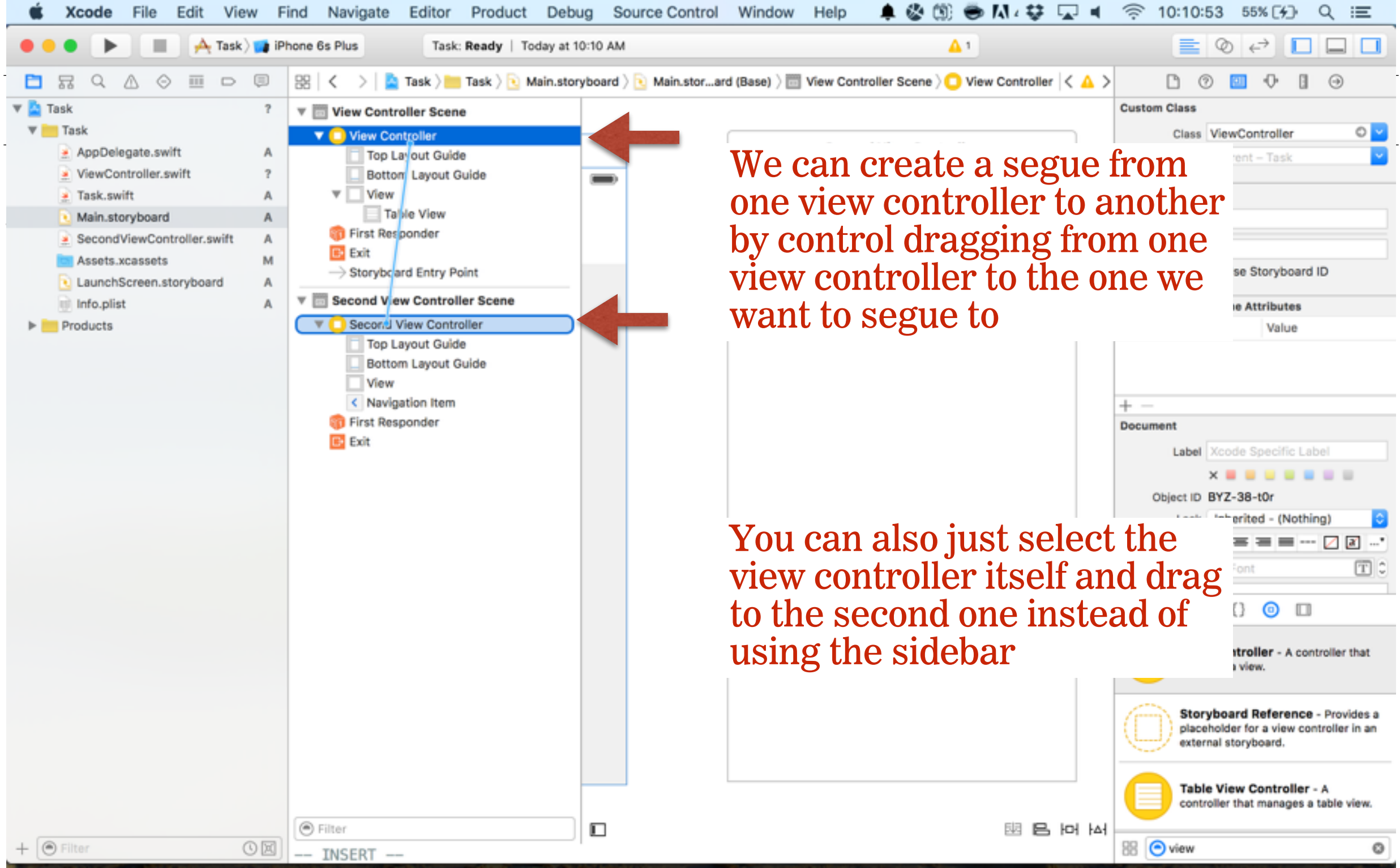
- Why might we want to perform a segue in code using `performSegueWithIdentifier` instead of performing it in Storyboard like we have been doing so far?
- When segues are performed in Storyboard, when a button was pressed for example, the only action/code the button executed was the segue
- When performing a segue in code, we can perform the segue along with additional code
- For example, we can perform the segue, and create an object or make a network request

## INTRO TO SWIFT

---

# SEGUES

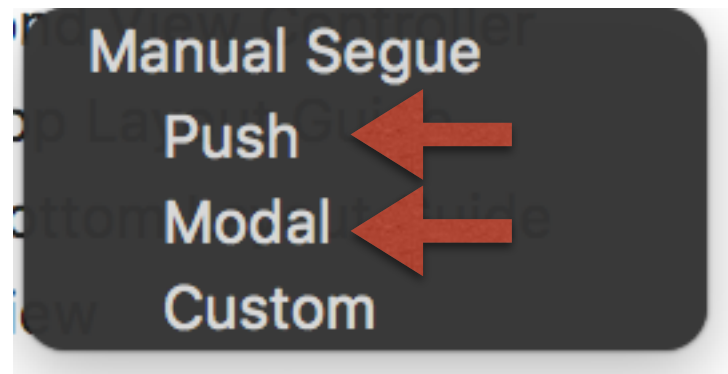
- What is an IBAction?
- How do we create an IBAction?
- Now we can create an IBAction and call `performSegueWithIdentifier` in that IBAction
- That will only perform the segue, but we can also add additional code to the same IBAction
- Or, we can call `performSegueWithIdentifier` in our `didSelectRowAtIndexPath` method if our view controller implements the `UITableViewDelegate` protocol



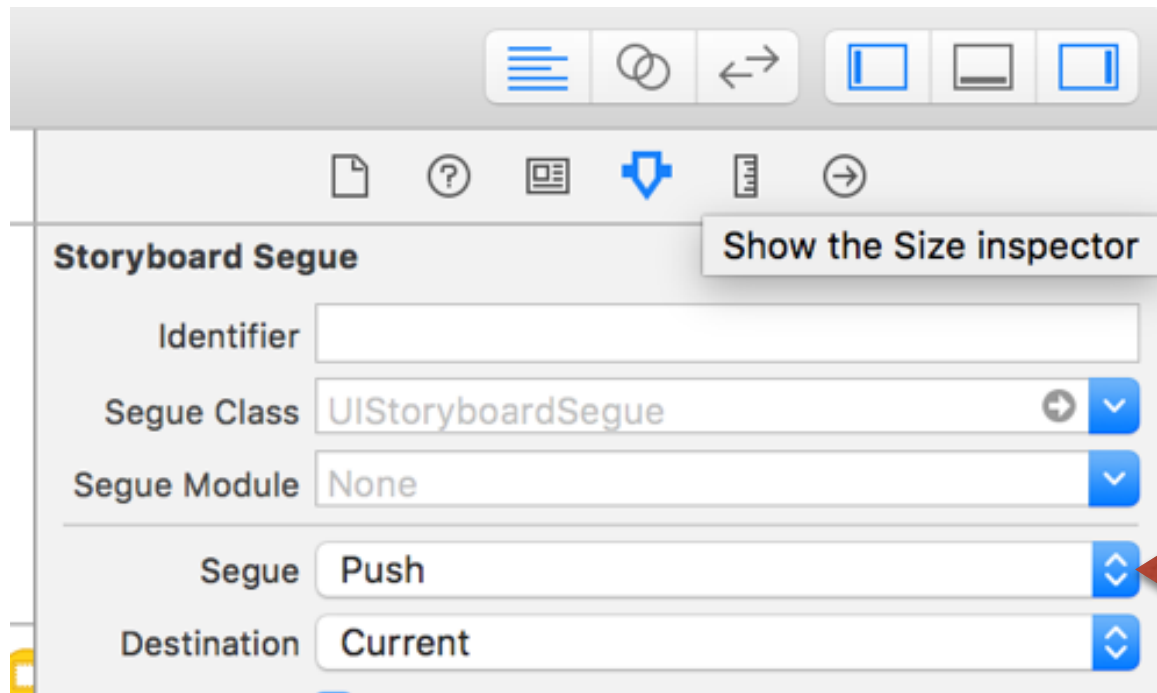
We can create a segue from one view controller to another by control dragging from one view controller to the one we want to segue to

You can also just select the view controller itself and drag to the second one instead of using the sidebar

# INTRO TO SWIFT



Remember, we can only add a “Push” segue if our view controller is inside of a UINavigationController. We’ll use the Push segue, so make sure to embed inside of a UINavigationController



If you create a “Push” segue or a “Modal” segue and need to change it, you can do so in the Attributes inspector after selecting the Segue in Storyboard

# SEGUES

- So `performSegueWithIdentifier` is called and the view controller “prepares” for the segue
- In fact, there is another method in `UIViewController` called `prepareForSegue`
- In this method, we can get a “reference”/instance of the view controller to be displayed
- Once we have a reference to this new instance, we can set properties we want on the view controller
- Since we created the view controller, we can add custom properties of some custom types we created

## INTRO TO SWIFT

---

# SEGUES

- To view the error if/when your app crashes, scroll to the top of the console (the panel at the bottom)
- Some common errors:
  - “has no segue with identifier” - there is a typo somewhere in your identifier name either in Storyboard or in your view controller
  - “Push segues can only be used when the source controller is managed by an instance of UINavigationController.” Embed the view controller inside of a Navigation Controller

---

**INTRO TO SWIFT**

---

**ANYOBJECT**

## INTRO TO SWIFT

---

# ANYOBJECT

- This is a type that can actually hold any type

```
var helloWorld: AnyObject = "hello world"  
helloWorld = 2  
helloWorld = false
```

```
var helloString = "hello world"  
helloString = false
```



---

## INTRO TO SWIFT

---

# ANYOBJECT

- Why use this?
- In certain scenarios it gives us a lot of flexibility
- As a parameter type in a function, we can now have the function take in AnyObject and perform almost anything on it
- One problem with AnyObject is not all objects have the same functionality, so in order to have the variable do something we need to convert the variable to another type
- Ideally a more specific type with more functionality

---

**INTRO TO SWIFT**

---

# CASTING

# CASTING

- Casting allows us to convert a variable from one type to another
- It is most commonly used with `AnyObject`, but can be used in other instances as well
- We typically use the `as` keyword when casting from one type to another

```
var helloWorld: AnyObject = "hello world"  
var helloString = helloWorld as! String
```

# CASTING

- If a function expects us to pass in a String, we cannot pass in AnyObject, so in that case we must cast it from AnyObject to a String, then we can pass it in to the function
- If the casting fails, the app can crash

```
var helloWorld: AnyObject = 2  
var helloString = helloWorld as! String
```

# INTRO TO SWIFT

---

## CASTING

- When using `prepareForSegue` we want to cast the `destinationViewController` to a subclass of `UIViewController` that we created
- That way we can access and set its properties

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {  
    if (segue.identifier == "todo") {  
        let indexPath = sender as! NSIndexPath  
        let todoViewController = segue.destinationViewController as! SecondViewController  
        todoViewController.todo = todos[indexPath.row]  
    }  
}
```

## INTRO TO SWIFT

---

# CASTING

- Swift wants to make sure that we have actually defined the properties we pass in
- To tell Swift that we will actually pass in values, we can use the exclamation mark (force) or a question mark (an optional)

```
class SecondViewController: UIViewController {  
    var todo: Todo!  
    var todo2: Todo?  
}
```

**INTRO TO SWIFT**

---

# **VIEW CONTROLLER LIFECYCLE**

---

## INTRO TO SWIFT

---

# VIEW CONTROLLER LIFECYCLE

- The view controller has a lifecycle
- Practically, this means that it is loaded into memory and there are additional functions that are called
- We can override and insert code into these functions to “make stuff happen” when we want it to
- Some of these methods include: `viewDidLoad`, `viewWillAppear`, `viewDidAppear`, `viewWillDisappear`, etc.



## INTRO TO SWIFT

---

# VIEW CONTROLLER LIFECYCLE

- Today, we will use `viewWillAppear`
- This is a great opportunity to setup the user interface with some data
- Example:

```
override func viewWillAppear(animated: Bool) {  
    //code  
}
```

- Behind the scenes, the `UIViewController` super class automatically calls this method. Our only job is to override it and put code we want to execute inside of it

---

**INTRO TO SWIFT**

---

**LAB**