# Basic Exercises Part 4.3. Passing data back with properties or functions

## A 🡨 B (bad approach, don’t do this)

* What if do you want to pass data from a secondary view controller to the main view controller?
* Passing data between view controllers using a property on the secondary view controller (forward) is fairly straightforward and even backwards isn’t that different though.
* Cases for this situation:
  + The user has gone from view controller A to a secondary view controller B.
  + In the next view controller the user interacts with the UI and change or create some data, and then you want that data back in view controller A; ie from B 🡪 A.

### **1.1 Create a new project**

Create a basic Single View. We will create a new project on each approach, so we only going to write the steps once.

### **1.2 Add two view controller**

The easiest way to pass data back is to create a reference to view controller A on view controller B, and then call a function from view controller A within view controller B.

This is now the secondary view controller class:

class SecondaryViewController: UIViewController

{

var mainViewController:MainViewController?

@IBAction func onButtonTap()

{

mainViewController?.onUserAction(data: "loren ipsum")

}

}

Then, this function is added to MainViewController:

func onUserAction(data: String)

{

print("Data received: \(data)")

}

When the view controller is pushed onto the navigation stack, just like in the previous examples, a connection between the main view controller and the secondary view controller is made:

let vc = SecondaryViewController(nibName: "SecondaryViewController", bundle: nil)

vc.mainViewController = self

In the example above, self is assigned to property mainViewController. The secondary view controller now “knows” the main view controller, so it can call any of its functions – like onUserAction(data:).

That’s all there is to it. But this approach for passing data isn’t the most ideal. It has a few major drawbacks:

* The MainViewController and SecondaryViewController are now tightly coupled. You want to avoid tight-coupling in software design, mostly because it decreases the modularity of your code. Both classes become too entangled, and rely on each other to function properly, with often leads to spaghetti code.
* The above code example creates a retain cycle. The secondary view controller can’t be removed from memory until the main view controller is removed, but the main view controller can’t be removed from memory until the secondary view controller is removed. (A solution would be the weak property keyword.)
* Two developers can’t easily work separately on MainViewController and SecondaryViewController, because both view controllers need to have an understanding about how the other view controller works. There’s no separation of concerns.

You want to avoid directly referencing classes, instances and functions like this. Code like this simply becomes a nightmare to maintain. It often leads to spaghetti code, in which you change one piece of code that breaks another seemingly unrelated piece of code.

So, what’s a better idea? Delegation! We will see this in the next document.

### **1.3 Switch language**

Not a choice. We always need to try the same behavior on Objective C.