Tutorial

Color Specs App

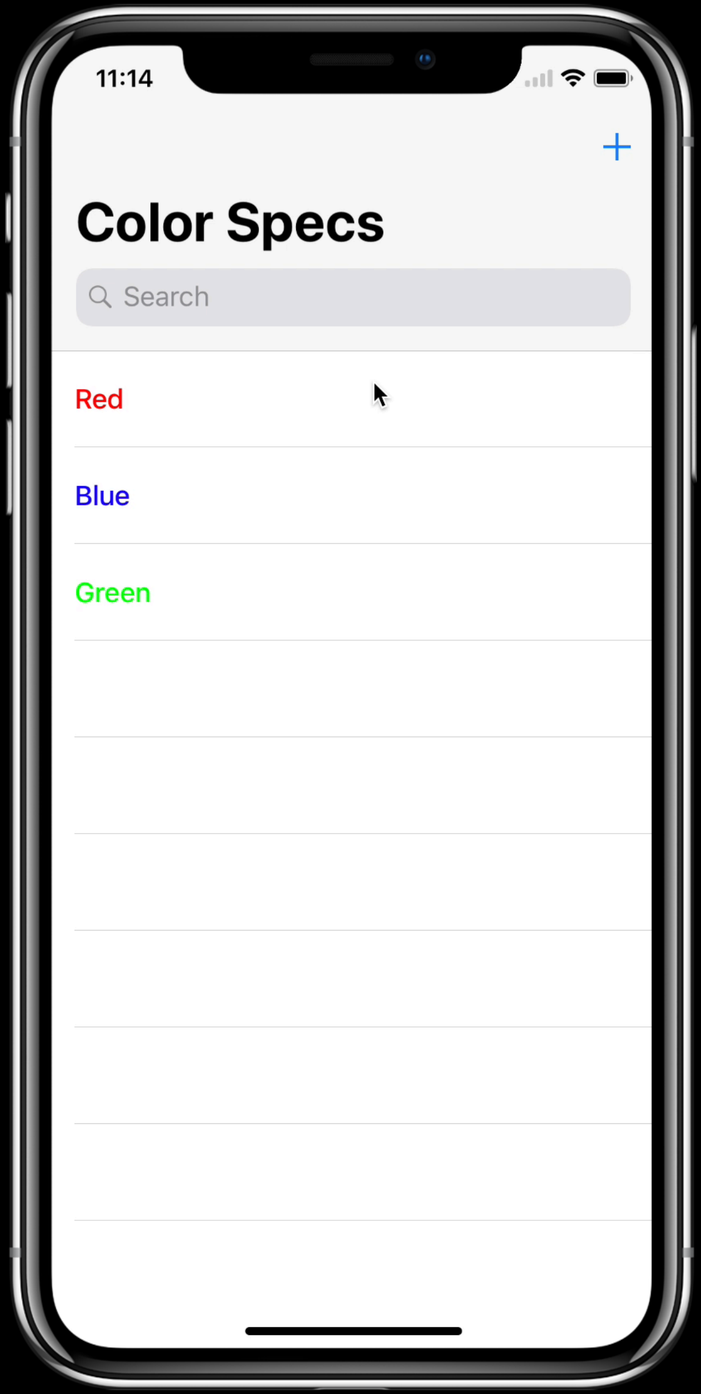
October 17, 2017

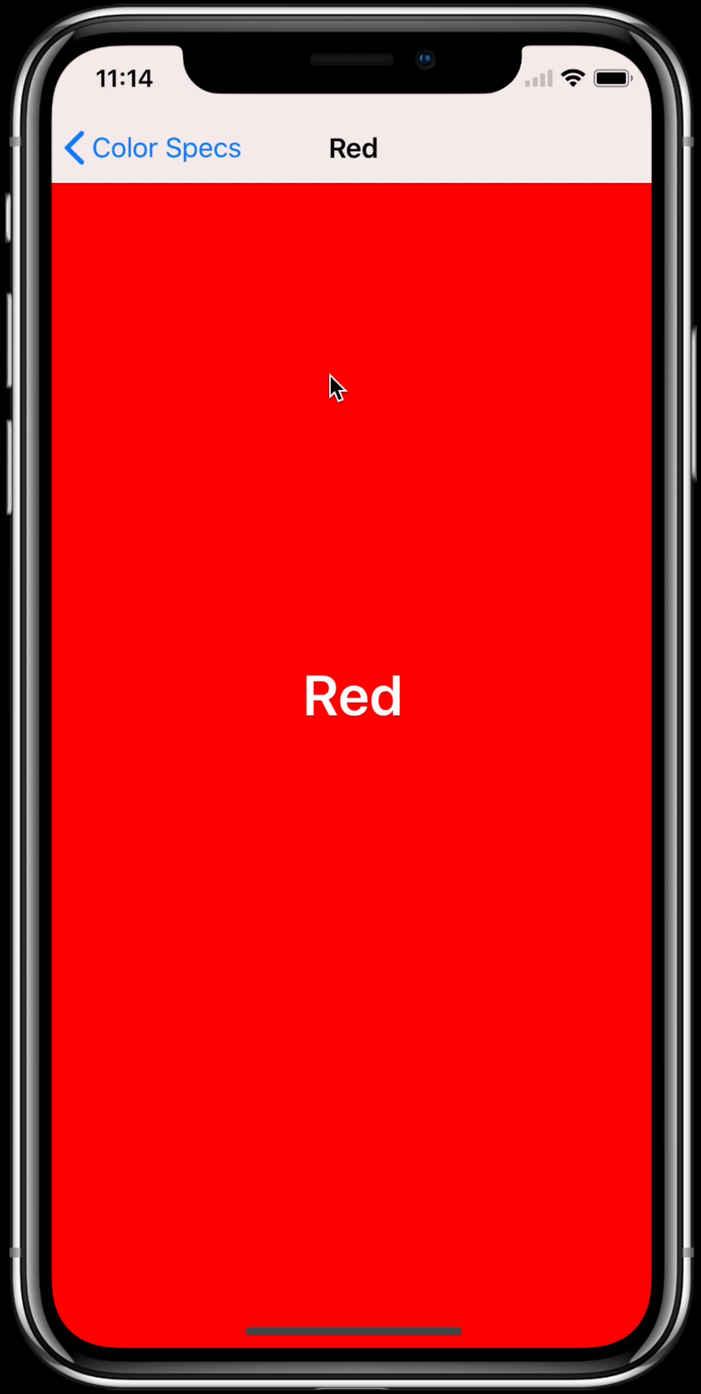
Georgia Tech's iOS Club

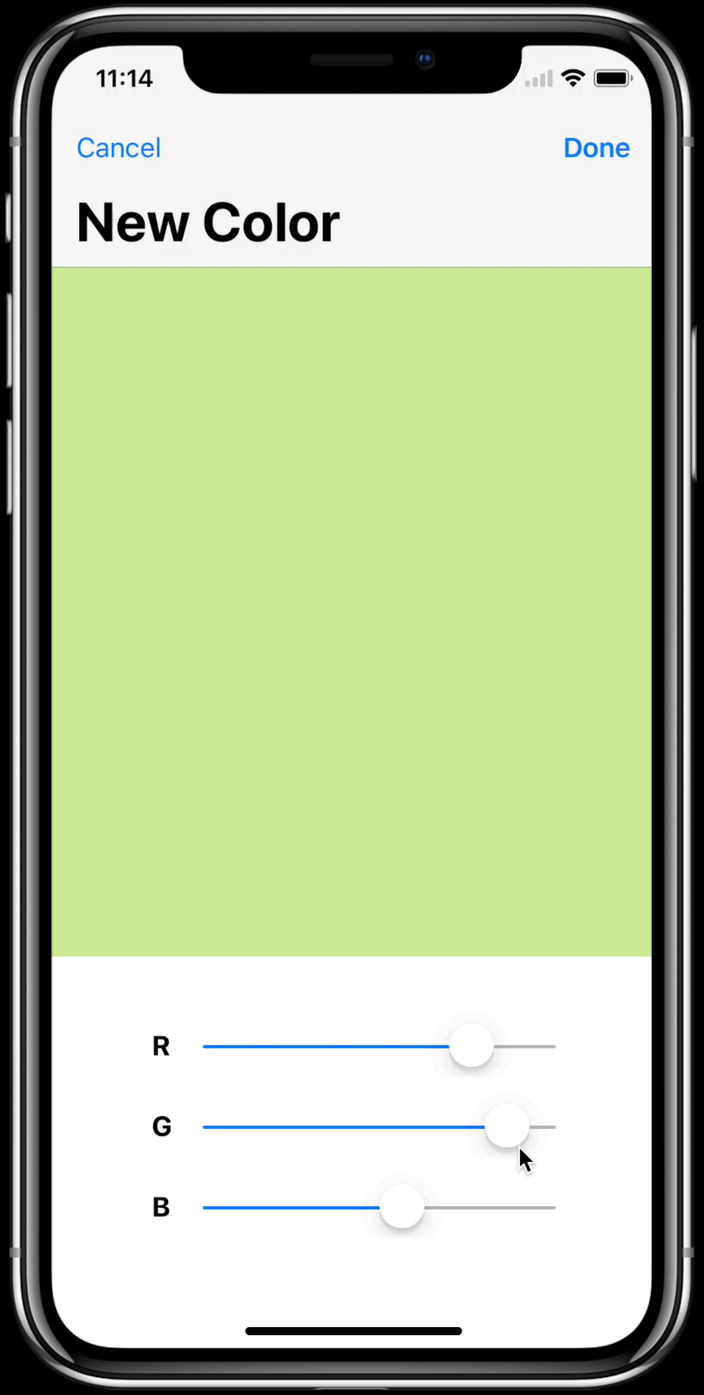
More on UITableView

UIViewController presentation and dismissal

Transferring data back and forth between view controllers


11:14
Color Specs
Search
Red
Blue
Green


11:14
Color Specs
Red
Red


11:14
Cancel
Done
New Color
R
G
B

Preparing for Segues

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let destination = segue.destination as? ColorDetailViewController,
        let tableViewCell = sender as? UITableViewCell,
        let index = tableView.indexPath(for: tableViewCell)?.row {

        destination.color = self.colors[index]
    }
}
```

Preparing for Segues

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let destination = segue.destination as? ColorDetailViewController,
        let tableViewCell = sender as? UITableViewCell,
        let index = tableView.indexPath(for: tableViewCell)?.row {

        destination.color = self.colors[index]
    }
}
```

Stubs can be found in Xcode Templates

Preparing for Segues

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let destination = segue.destination as? ColorDetailViewController,
        let tableViewCell = sender as? UITableViewCell,
        let index = tableView.indexPath(for: tableViewCell)?.row {

        destination.color = self.colors[index]
    }
}
```

Stubs can be found in Xcode Templates

Get destination view controller with segue.destination

Preparing for Segues

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let destination = segue.destination as? ColorDetailViewController,
        let tableViewCell = sender as? UITableViewCell,
        let index = tableView.indexPath(for: tableViewCell)?.row {

        destination.color = self.colors[index]
    }
}
```

Stubs can be found in Xcode Templates

Get destination view controller with segue.destination

Happens before **`viewDidLoad`**

Preparing for Segues

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let destination = segue.destination as? ColorDetailViewController,
        let tableViewCell = sender as? UITableViewCell,
        let index = tableView.indexPath(for: tableViewCell)?.row {

        destination.color = self.colors[index]
    }
}
```

Stubs can be found in Xcode Templates

Get destination view controller with segue.destination

Happens before **`viewDidLoad`**

Don't set UI-related variable in destination class

Preparing for Segues

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let destination = segue.destination as? ColorDetailViewController,
        let tableViewCell = sender as? UITableViewCell,
        let index = tableView.indexPath(for: tableViewCell)?.row {

        destination.color = self.colors[index]
    }
}
```

Stubs can be found in Xcode Templates

Get destination view controller with segue.destination

Happens before **`viewDidLoad`**

Don't set UI-related variable in destination class

Update UI elements in **`viewDidLoad`** from destination class

What about passing data back and forth?

```
func afterRewinding(_ segue: UIStoryboardSegue)
```

```
func afterRewinding(_ segue: UIStoryboardSegue)
```

```
segue.destination.color = newColor
```

```
func afterRewinding(_ segue: UIStoryboardSegue)
```

No, this does not exist.

Passing back data with callback closure

Function as Variables

```
var completionHandler: (String) -> (Int)
```

Function as Variables

```
var completionHandler: (String) -> (Int)
```

Function as a variable

Function as Variables

```
var completionHandler: (String) -> (Int)
```

Function as a variable

Function takes type **`String`** and returns **`Int`**

Function as Variables

```
var completionHandler: (String) -> (Int)
```

Function as a variable

Function takes type **`String`** and returns **`Int`**

```
let resultInt = self.completionHandler("Some String")
```

Function as Variables

```
var completionHandler: (String) -> (Int)
```

Function as a variable

Function takes type **`String`** and returns **`Int`**

```
let resultInt = self.completionHandler("Some String")
```

Call this function the same way you call any other function

Function as Variables

```
var completionHandler: (Int, String) -> Void
```

Function as Variables

```
var completionHandler: (Int, String) -> Void
```

```
var completionHandler: (Int, String?) -> Void
```

Function as Variables

```
var completionHandler: (Int, String) -> Void
```

```
var completionHandler: (Int, String?) -> Void
```

```
var completionHandler: ((Int, String?)?) -> Void
```

Function as Variables

```
var completionHandler: (Int, String) -> Void
```

```
var completionHandler: (Int, String?) -> Void
```

```
var completionHandler: ((Int, String?)?) -> Void
```

```
var completionHandler: (((Int, String?)?) -> Void)?
```

Function as Variables

```
var completionHandler: (Int, String) -> Void
```

```
var completionHandler: (Int, String?) -> Void
```

```
var completionHandler: ((Int, String?)?) -> Void
```

```
var completionHandler: (((Int, String?)?) -> Void)?
```

Parameters within function declarations can contain Optional values

Function as Variables

```
var completionHandler: (Int, String) -> Void
```

```
var completionHandler: (Int, String?) -> Void
```

```
var completionHandler: ((Int, String?)?) -> Void
```

```
var completionHandler: (((Int, String?)?) -> Void)?
```

Parameters within function declarations can contain Optional values

Function declaration itself can be an Optional value

Function as Variables

```
var completionHandler: (((Int, String?)?) -> Void)?
```

Function as Variables

```
var completionHandler: (((Int, String?)?) -> Void)?
```

Parameters within function declarations can contain Optional values

Function as Variables

```
var completionHandler: (((Int, String?)?) -> Void)?
```

Parameters within function declarations can contain Optional values

Function declaration itself can be an Optional value

Function as Variables

```
var completionHandler: (((Int, String?)?) -> Void)?
```

Parameters within function declarations can contain Optional values

Function declaration itself can be an Optional value

```
self.completionHandler?(15, "Some String")
```

Function as Variables

```
var completionHandler: (((Int, String?)?) -> Void)?
```

Parameters within function declarations can contain Optional values

Function declaration itself can be an Optional value

```
self.completionHandler?(15, "Some String")
```

Again, call this function the same way you call any other function

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

You can pass functions around just like any other variable

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

You can pass functions around just like any other variable

@escaping and **@nonescaping** (more later)

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

@nonescaping

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

`@nonescaping`

Default since Swift 3

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

@nonescaping

Default since Swift 3

Closure destroyed once the function is complete

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

`@nonescaping`

Default since Swift 3

Closure destroyed once the function is complete

Not useful for asynchronous operations or passing data between classes

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

@nonescaping

Default since Swift 3

Closure destroyed once the function is complete

Not useful for asynchronous operations or passing data between classes

@escaping

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

@nonescaping

Default since Swift 3

Closure destroyed once the function is complete

Not useful for asynchronous operations or passing data between classes

@escaping

Closure outlives the function

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

@nonescaping

Default since Swift 3

Closure destroyed once the function is complete

Not useful for asynchronous operations or passing data between classes

@escaping

Closure outlives the function

After function returns, completion handler is still retained

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

@nonescaping

Default since Swift 3

Closure destroyed once the function is complete

Not useful for asynchronous operations or passing data between classes

@escaping

Closure outlives the function

After function returns, completion handler is still retained

Allows us to perform completion callback long after presentation

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

You can pass functions around just like any other variable

`@escaping` (outlived) and **`@nonescaping`** (default, shortlived)

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

You can pass functions around just like any other variable

@escaping (outlived) and **@nonescaping** (default, shortlived)

```
self.completionHandler = completion
```

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

You can pass functions around just like any other variable

@escaping (outlived) and **@nonescaping** (default, shortlived)

```
self.completionHandler = completion
```

You can store the function into a class variable

Setting Up Completion Handlers

```
static func present(over presenter: UIViewController,
                    completion: @escaping ((color: UIColor, name: String)?) -> Void)?
```

You can pass functions around just like any other variable

@escaping (outlived) and **@nonescaping** (default, shortlived)

```
self.completionHandler = completion
```

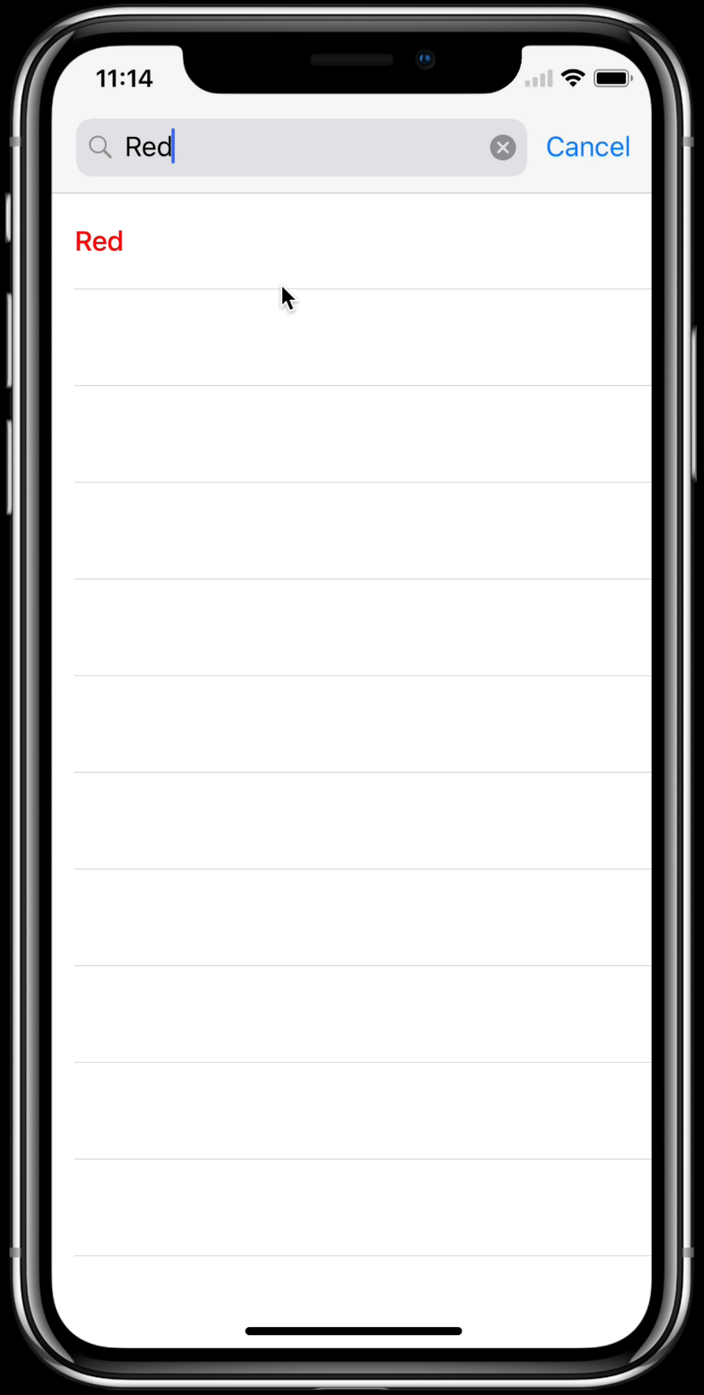
You can store the function into a class variable

Automatically makes function **@escaping**

Demo

Challenge

Implement Search


11:14
Red
Cancel
Red

Hint for Adding Support for Search

Use **`UISearchController`**

Set its **`dimsBackgroundDuringPresentation`** property to false

Why does the navigation bar disappear when navigating to detailed color view?

Sample Code

https://github.com/iosgatech/ColorSpecFall17