

iOS
DeCal

lecture 0

course overview

cs198-001 : fall 2017

today's lecture

- prerequisites
- what's covered in this decal
- course logistics
- Xcode (intro) and Swift

what you need for this decal

- macbook
- **Xcode** 9 beta 6: developer.apple.com/download/
- object oriented programming experience (cs61a + cs61b or equivalent)
- willing to put in a substantial amount of time into the course
 - heavy workload for a decal
 - roughly 4-6 hours outside of class

what you will learn



Swift
(language)



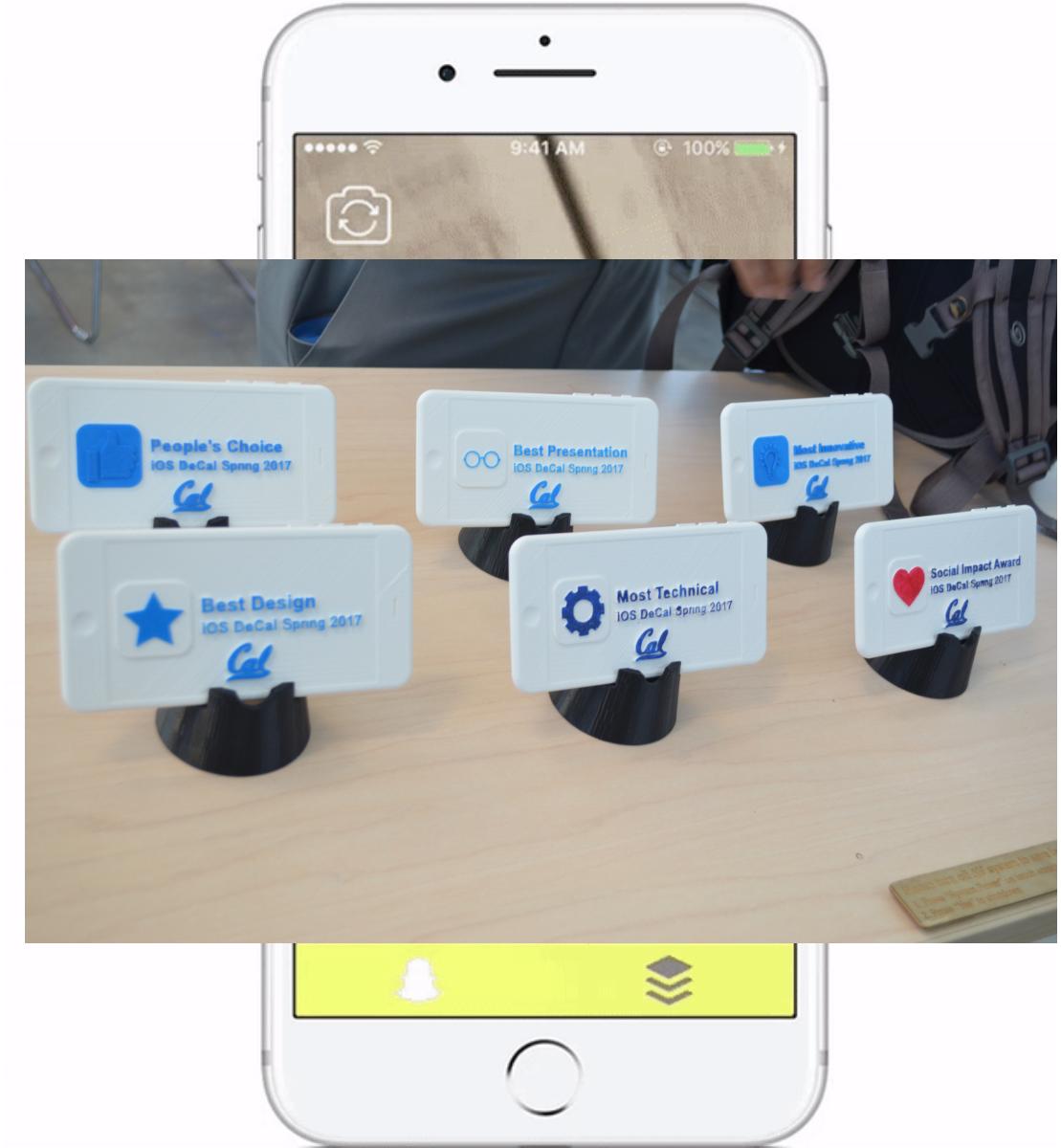
Xcode 9
(IDE)



design &
development

course overview

- weeks 1-5
 - create simple apps from scratch
 - design patterns + best practices
 - begin planning custom app
- week 6 – 10
 - advanced topics
 - begin work on custom app
- week 10 to end
 - finish custom apps
 - final app presentations



class format

lectures

- Mondays
- 6:30 to ~7:30pm
- HP Auditorium
- attendance required

labs

- Wednesdays
- 7:00-8:30pm
- various rooms (sign up on piazza tomorrow 8pm)
- attendance required

grading breakdown

- 30% projects
- 35% labs - pass / fail policy
- 35% final project

assignment submission

labs

- must get checked off during lab
- if you do not finish within the lab period, you can get checked off the following week at the beginning of lab

other assignments (homework/projects)

- submit on Gradescope
- graded via autograder

enrollment

if you've been accepted, you'll be enrolled automatically (thursday afternoon)

waitlisted students will receive access codes to enroll on CalCentral tomorrow morning via email

attendance policy

check-in every lecture/lab via google form

- you must check in with another person in the class
(one form per pair)

excused absences – private post on piazza

unexcused absences

- students with 4+ unexcused Absences will receive an NP for the course

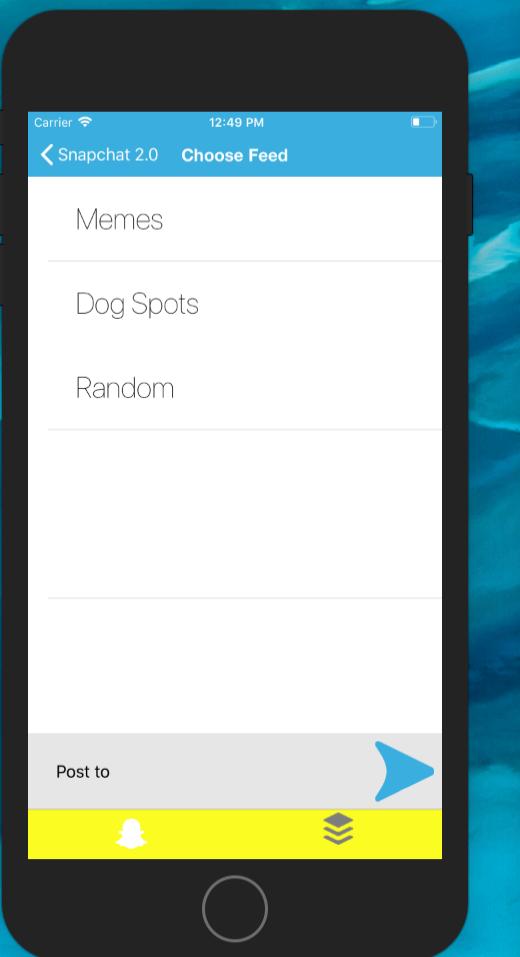
attendance policy - example

let's try it now!

introduce yourself to another student or TA, and fill out the google form found on our course website
(<http://iosdecal.com/>)

iOS development

Xcode



iPhone 7 Plus Running snapChatProject on iPhone 7 Plus 1

SnapchatClone > Main.storyboard (Base) > No Selection

ChooseThread...Controller.swift > No Selection < 5 > + X

```
1 // ChooseThreadViewController.swift
2 // SnapChatClone
3 // Created by Paige Plander on 3/8/17.
4 // Copyright © 2017 org.iosdecal. All rights reserved.
5
6 import UIKit
7
8 class ChooseThreadViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {
9
10     // Table view displaying the names of each thread
11     @IBOutlet weak var threadTableView: UITableView!
12
13     /// The image picked by the user from the image picker
14     var chosenImage: UIImage?
15
16     /// Displays the name of the thread that the user has
17     /// selected
18     @IBOutlet weak var chosenThreadLabel: UILabel!
19
20     override func viewDidLoad() {
21         super.viewDidLoad()
22         threadTableView.delegate = self
23     }
24 }
```

View as: iPhone 7 (wC hR) 41% snapChatProject

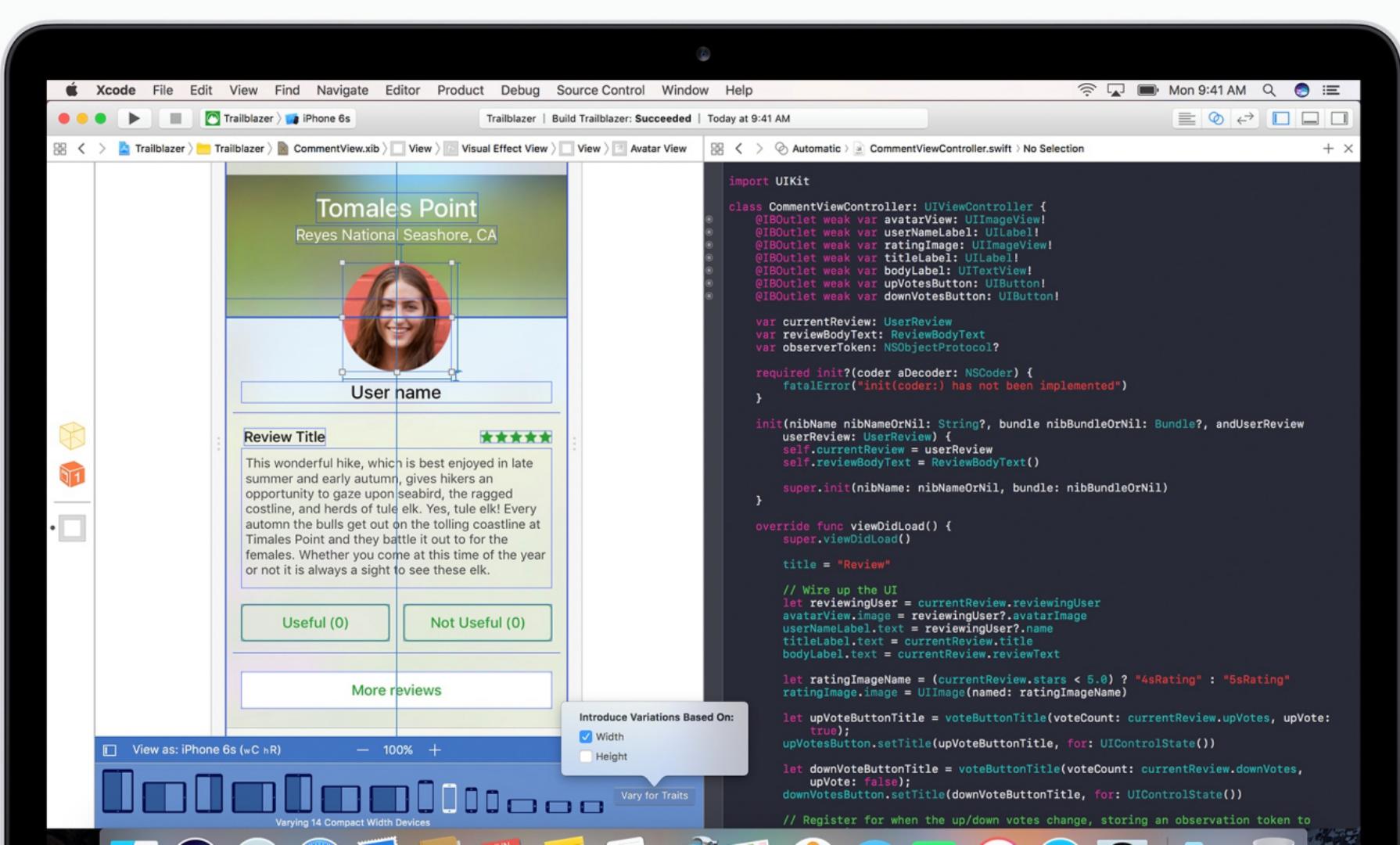
AF61386F-0E29-47EE-8B48-EF56665B3DB7/data/Containers/Shared/SystemGroup/systemgroup.com.apple.configurationprofiles
2017-09-04 12:49:37.921996-0700
snapChatProject[49022:3365634] [MC] Reading from private effective user settings.

All Output Filter

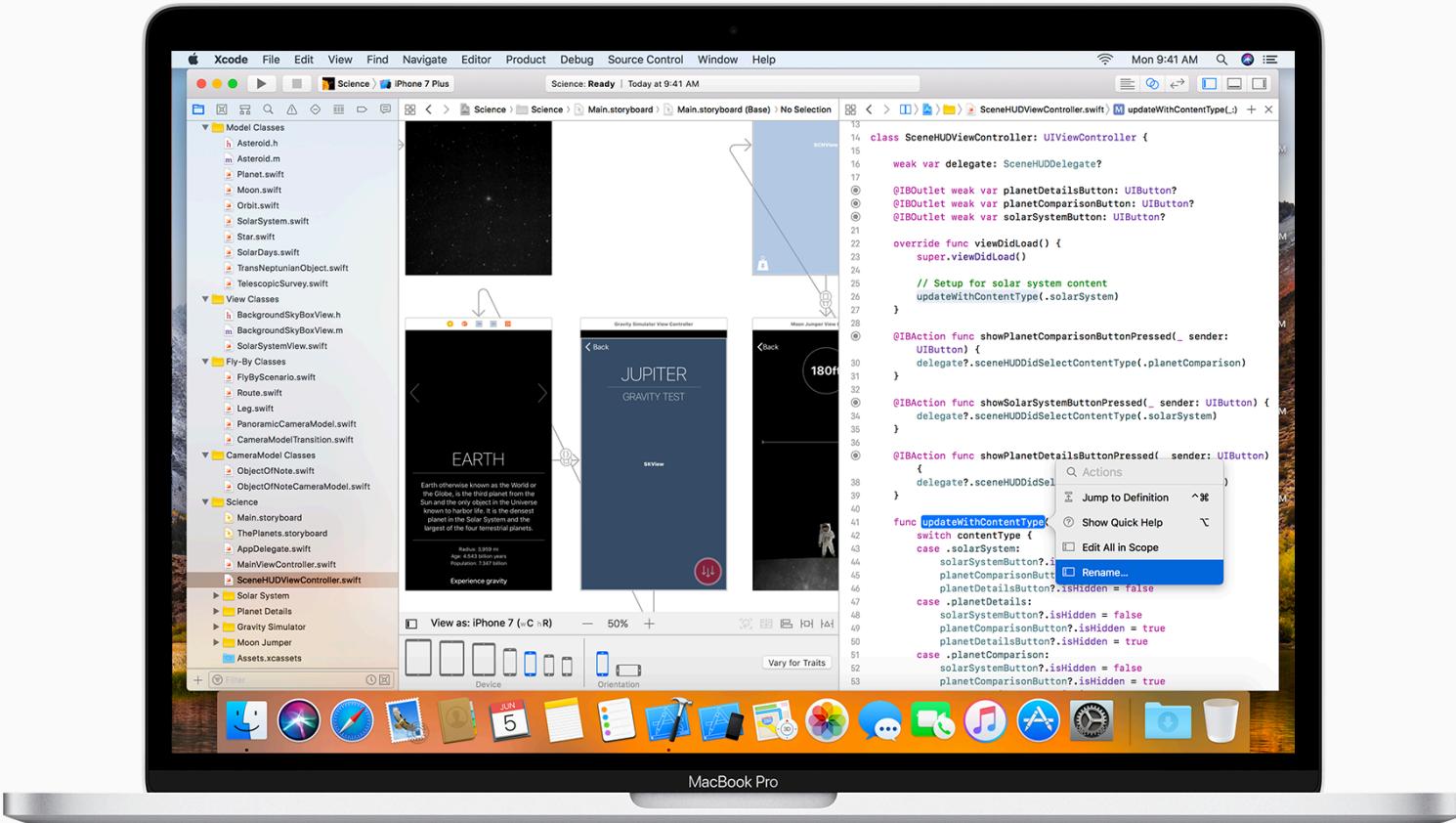
Auto Filter

The Xcode interface shows the storyboard and code for the "Choose Thread" screen. The storyboard contains two views: a main view with a placeholder image and a detail view titled "Choose Feed" with a table view. The code defines a `ChooseThreadViewController` that implements `UITableViewDelegate` and `UITableViewDataSource`. It includes outlets for the table view and a label, and handles image selection via an outlet.

Xcode 8 – currently in App Store



what we'll be using - Xcode 9



announced June 5,
2017

debugging, refactoring,
and GPU
improvements

current version:
Xcode 9 beta 6

uses the new Swift 4

swift overview

assignments with var and let

```
var x = "Hello"  
x = "world"  
  
let implicitInt = 70  
let explicitInt: Int = 70  
  
// error  
let implicitInt = 50
```

functions

```
// defining functions
func update(withNewData data: [String]) -> Bool {
    if data[0] == "Error" {
        return false
    }
    // ...
    return true
}

// calling Functions
update(withNewData: ["iOS", "DeCal"])
```

functions

```
// defining functions
func update(withNewData data: [String]) -> Bool {
    if data[0] == "Error" {
        return false
    }
    // ...
    return true
}

// calling Functions
update(withNewData: ["iOS", "DeCal"])
```

Internal Parameter (used in function) - data

External Parameter (used when calling function) - withNewData

classes and functions

```
class Dog {  
    private let name: String?  
    private let age: Int  
  
    init(age: Int, name: String?) {  
        self.age = age  
        self.name = name  
    }  
  
    func getGreeting() -> String {  
        return name + " says  
            hello!"  
    }  
}
```

classes and functions

```
class Dog {  
    private let name: String?  
    private let age: Int  
  
    init(age: Int, name: String?) {  
        self.age = age  
        self.name = name  
    }  
  
    func getGreeting() -> String {  
        return name + " says  
            hello!"  
    }  
}
```

optional type

optionals

A type that is logically allowed to have “no value”

Properties of optional type are automatically initialized with a value of nil

“Unwrap” optionals with a “!”
(Careful! If nil -> error)

```
var response: String? = "Hello"  
print(response)
```

Console Output

optionals

A type that is logically allowed to have “no value”

Properties of optional type are automatically initialized with a value of nil

“Unwrap” optionals with a “!”
(Careful! If nil -> error)

```
var response: String? = "Hello"  
print(response)
```

Console Output

Optional("Hello")

optionals

A type that is logically allowed to have “no value”

Properties of optional type are automatically initialized with a value of nil

“Unwrap” optionals with a “!”
(Careful! If nil -> error)

```
var response: String? = "Hello"  
print(response!)
```

Console Output

optionals

A type that is logically allowed to have “no value”

Properties of optional type are automatically initialized with a value of nil

“Unwrap” optionals with a “!”
(Careful! If nil -> error)

```
var response: String? = "Hello"  
print(response)
```

Console Output

“Hello”

optionals

A type that is logically allowed to have “no value”

Properties of optional type are automatically initialized with a value of nil

“Unwrap” optionals with a “!”
(Careful! If nil -> error)

```
var response: String?  
print(response)
```

Console Output

optionals

A type that is logically allowed to have “no value”

Properties of optional type are automatically initialized with a value of nil

“Unwrap” optionals with a “!”
(Careful! If nil -> error)

```
var response: String?  
print(response)
```

Console Output

nil

optionals

A type that is logically allowed to have “no value”

Properties of optional type are automatically initialized with a value of nil

“Unwrap” optionals with a “!”
(Careful! If nil -> error)

```
var response: String?  
print(response!)
```

Console Output

optionals

A type that is logically allowed to have “no value”

Properties of optional type are automatically initialized with a value of nil

“Unwrap” optionals with a “!”
(Careful! If nil -> error)

```
var response: String?  
print(response!)
```

Console Output

```
fatal error: unexpectedly found nil while unwrapping an Optional value
```

optionals

what if you don't know
whether an optional is nil or
not?

optional binding

safe way to unwrap: "if let"

```
var someOptional: String? = "hello"

if let myOptional = someOptional {
    print(myOptional)
}
```

Console Output

hello

optional chaining

```
var optionalDog: Dog? = getDog(named:  
    "Molly")  
  
print(optionalDog.bestFriend.name)
```

Console Output

```
error: value of optional type 'Dog?'  
not unwrapped; did you mean to use  
'!' or '?'?
```

closures

```
{ (parameters) -> return type in  
      statements  
}
```

closures

```
// normal function
func isGood(string: String) -> Bool {
    return string == "dog"
}
```

closures

```
// normal function
func isGood(string: String) -> Bool {
    return string == "dog"
}
```

```
// as a closure
let isGoodClosure = { string in
    return string == "dog" }
```

closures

```
// as a closure  
let isGoodClosure = { string in  
    return string == "dog" }
```

closures

```
// as a closure  
let isGoodClosure = { string in  
    return string == "dog" }
```

```
// short hand arg name  
let isGoodClosure = {  
    return $0 == "dog" }
```

closures

```
// as a closure  
let isGoodClosure = { string in  
    return string == "dog" }
```

```
// short hand arg name  
let isGoodClosure = {  
    return $0 == "dog" }
```

```
// even better  
let isGoodClosure = { $0 == "dog" }
```

closures

```
// as a closure  
let isGoodClosure = { string in  
    return string == "dog" }
```

```
// short hand arg name  
let isGoodClosure = {  
    return $0 == "dog" }
```

```
// even better  
let isGoodClosure = { $0 == "dog" }
```

closures as completion handlers

```
func responseData(completionHandler:  
    @escaping (Response) -> Void) {  
    // ...  
}
```

closures as completion handlers

```
Alamofire.request(url).responseData({  
    response in  
    if let data = response.result.value {  
        // do something with data  
        print(data)  
    }  
})
```

closures as completion handlers

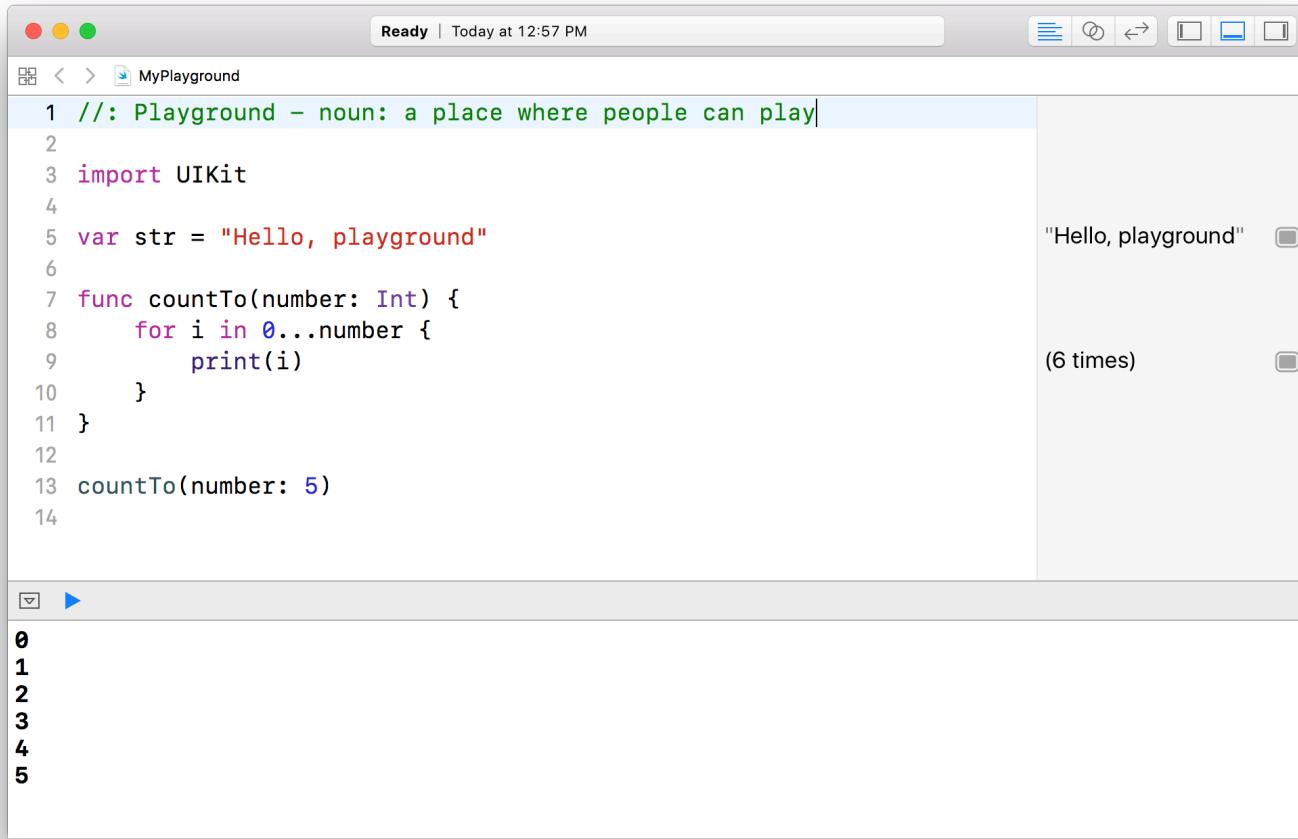
```
Alamofire.request(url).responseData {  
    response in  
    if let data = response.result.value {  
        // do something with data  
        print(data)  
    }  
}
```

(tip! parentheses not required)

protocols

```
protocol DogOwner {  
    var dog: Dog { get set }  
}  
  
protocol NicePerson {  
    func pet(dog: Dog) -> Bool  
}  
  
class Sandy: DogOwner, NicePerson {  
    var dog: Dog = Dog()  
    func pet(dog: Dog) -> Bool {  
        return dog.pet()  
    }  
}
```

playgrounds



A screenshot of the Xcode playground interface. The title bar says "Ready | Today at 12:57 PM". The main area shows the following Swift code:

```
1 //: Playground - noun: a place where people can play
2
3 import UIKit
4
5 var str = "Hello, playground"
6
7 func countTo(number: Int) {
8     for i in 0...number {
9         print(i)
10    }
11 }
12
13 countTo(number: 5)
14
```

The output pane shows the results of the print statements:

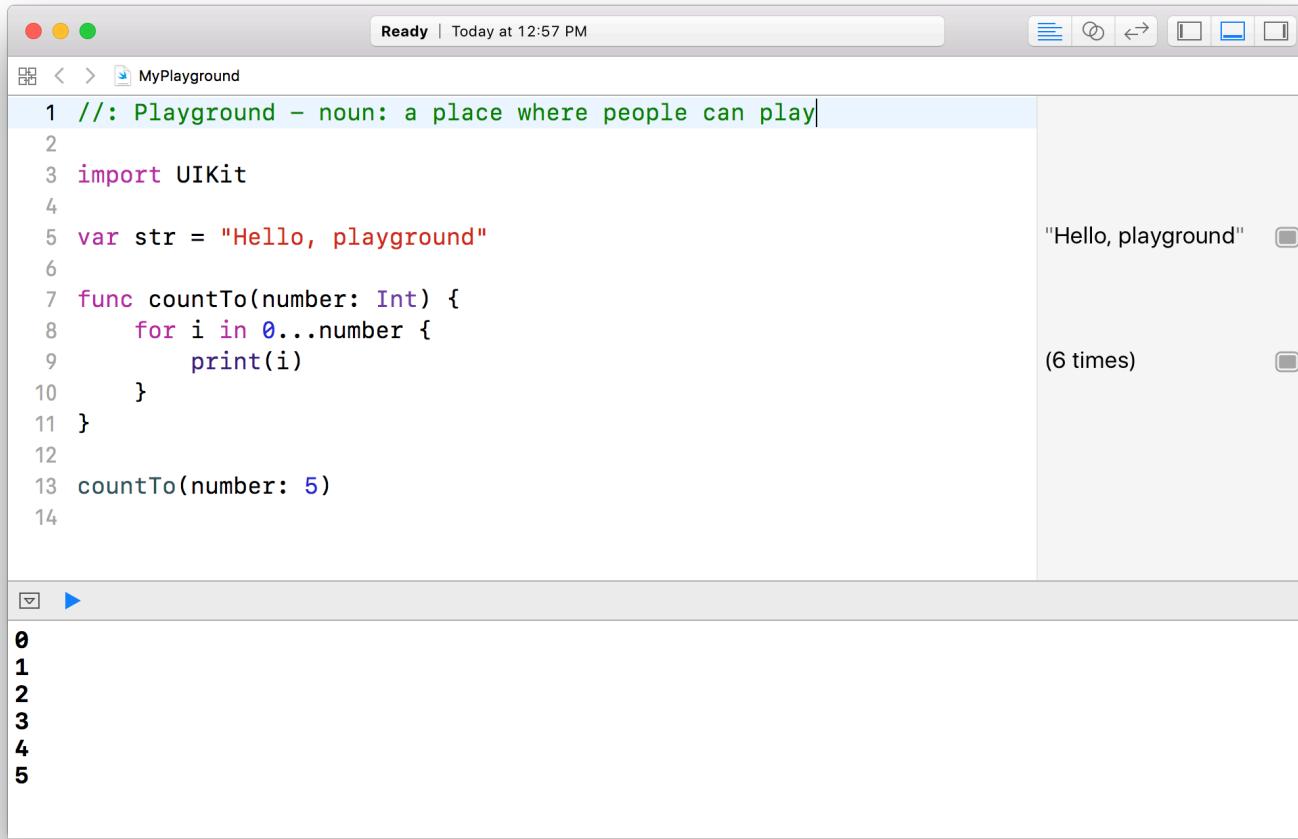
```
"Hello, playground"
(6 times)
```

Below the output pane, there is a list of numbers from 0 to 5.

lightweight
interface for small
programs

environment for
homework 1

homework 1: swift playground



A screenshot of an Xcode playground window titled "MyPlayground". The status bar at the top says "Ready | Today at 12:57 PM". The main area contains the following Swift code:

```
1 //: Playground - noun: a place where people can play
2
3 import UIKit
4
5 var str = "Hello, playground"
6
7 func countTo(number: Int) {
8     for i in 0...number {
9         print(i)
10    }
11 }
12
13 countTo(number: 5)
14
```

The output pane shows the results of the execution:

```
"Hello, playground"
(6 times)
```

The bottom left corner of the playground window shows the printed values from the code: 0, 1, 2, 3, 4, 5.

due **Monday**
before lecture
(6:30pm)

submit on
Gradescope

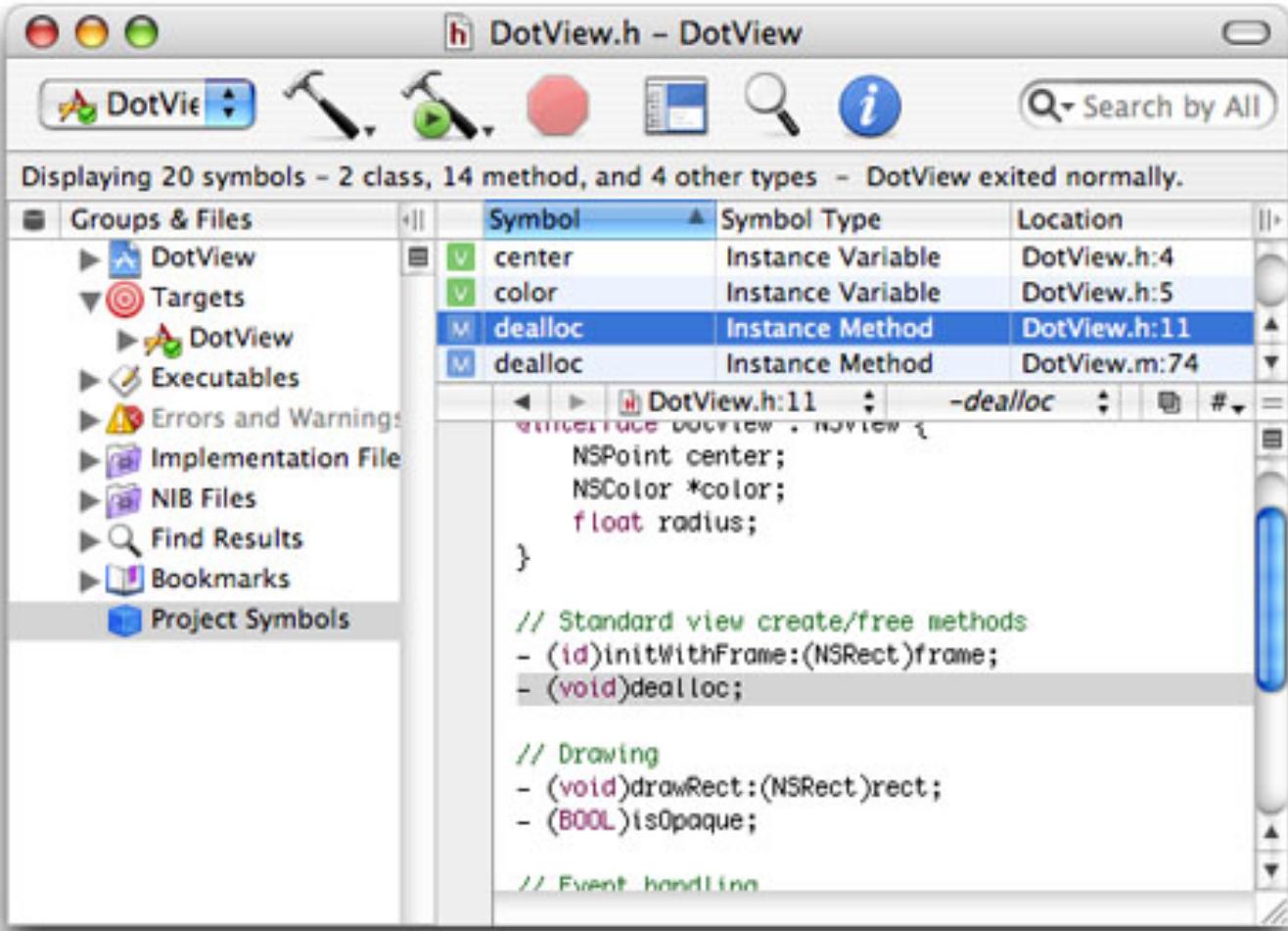
homework 1: swift playground

due monday before lecture
posted tonight

extra slides

Project Builder

- IDE for the NeXTSTEP OS (source of macOS, iOS, tvOS, etc.)
- rewritten for OS X, and rebranded as **Xcode**



The screenshot shows the Xcode Project Builder interface with the title bar "DotView.h - DotView". The main area displays a table of symbols:

Symbol	Symbol Type	Location
center	Instance Variable	DotView.h:4
color	Instance Variable	DotView.h:5
dealloc	Instance Method	DotView.h:11
dealloc	Instance Method	DotView.m:74

Below the table, the code for the DotView class is shown:

```
interface DotView : NSView {
    NSPoint center;
    NSColor *color;
    float radius;
}

// Standard view create/free methods
- (id)initWithFrame:(NSRect)frame;
- (void)dealloc;

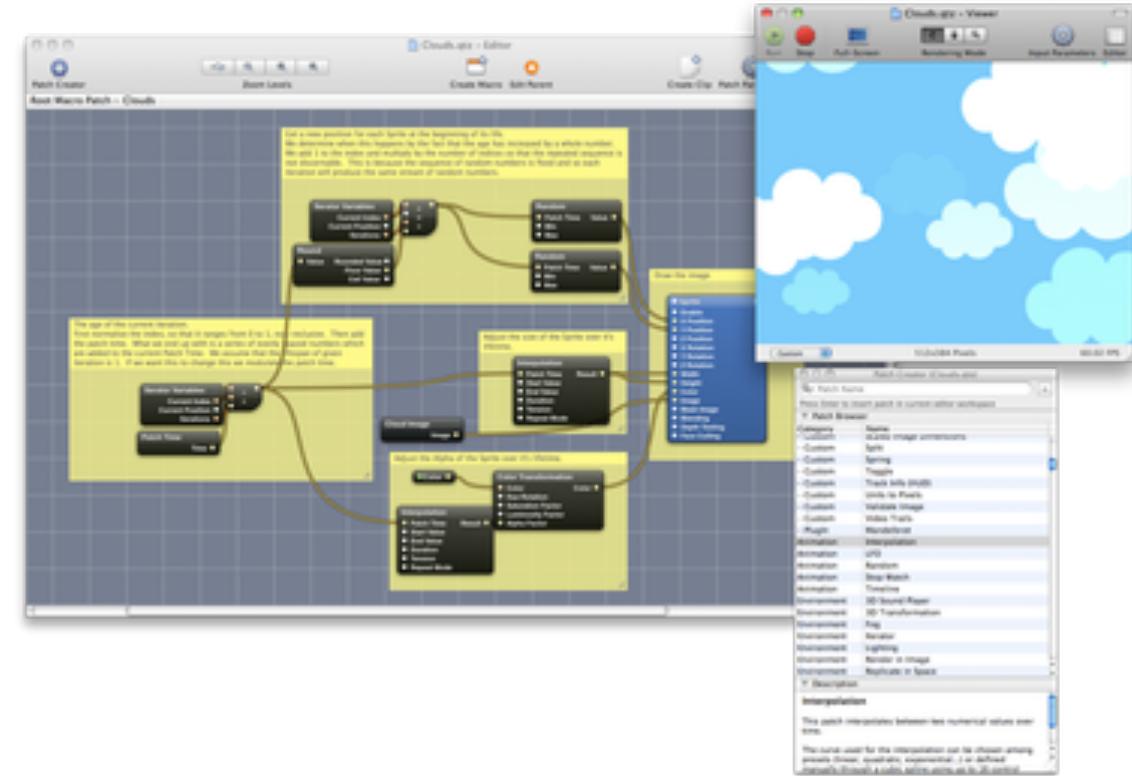
// Drawing
- (void)drawRect:(NSRect)rect;
- (BOOL)isOpaque;

// Event handling

```

Xcode versions

- 2003: Xcode 1.0
 - 2005: Xcode 2.x
 - included a visual programming language (Quartz Composer)
 - breakpoints and watchpoints
 - 2007: Xcode 3.x
 - 2008: iOS SDK released to third party developers



the Quartz Composer