lecture 1

hello, swift

cs198-001 : spring 2018

today's lecture

- course logistics
- introduction to swift

facilitators

Nithi Narayanan

Daniel Phiri

Chris Zielinski

Teaching Assistants

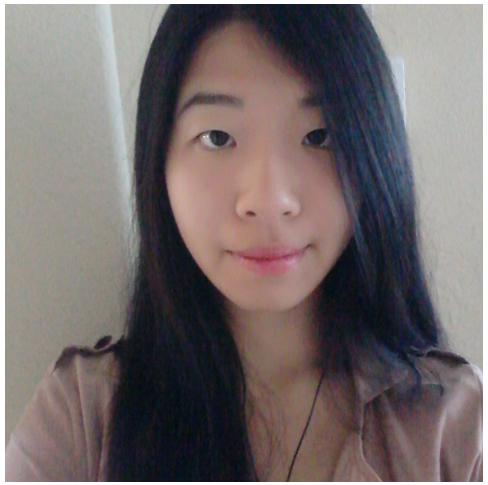
Sarah Chin

Marisa Wong

Tao Ong

Teaching Assistants

Kevin Bunarjo



Caroline

Anna

Teaching Assistants

David

RJ

Victor Korir

Teaching Assistants

Alex Yang

class format

lectures

- Mondays
- 6:30 to ~8:00pm
- 3 LeConte
- attendance required

labs

- Wednesdays
- 6:30-8:00pm
- 166 & 170 Barrows
- 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

- access codes to enroll on CalCentral will be emailed tonight

- waitlisted students will receive access codes tomorrow morning

enrollment

- access codes to enroll on CalCentral will be emailed tonight (or tomorrow, but definitely before noon)

- waitlisted students will receive access codes tomorrow morning

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

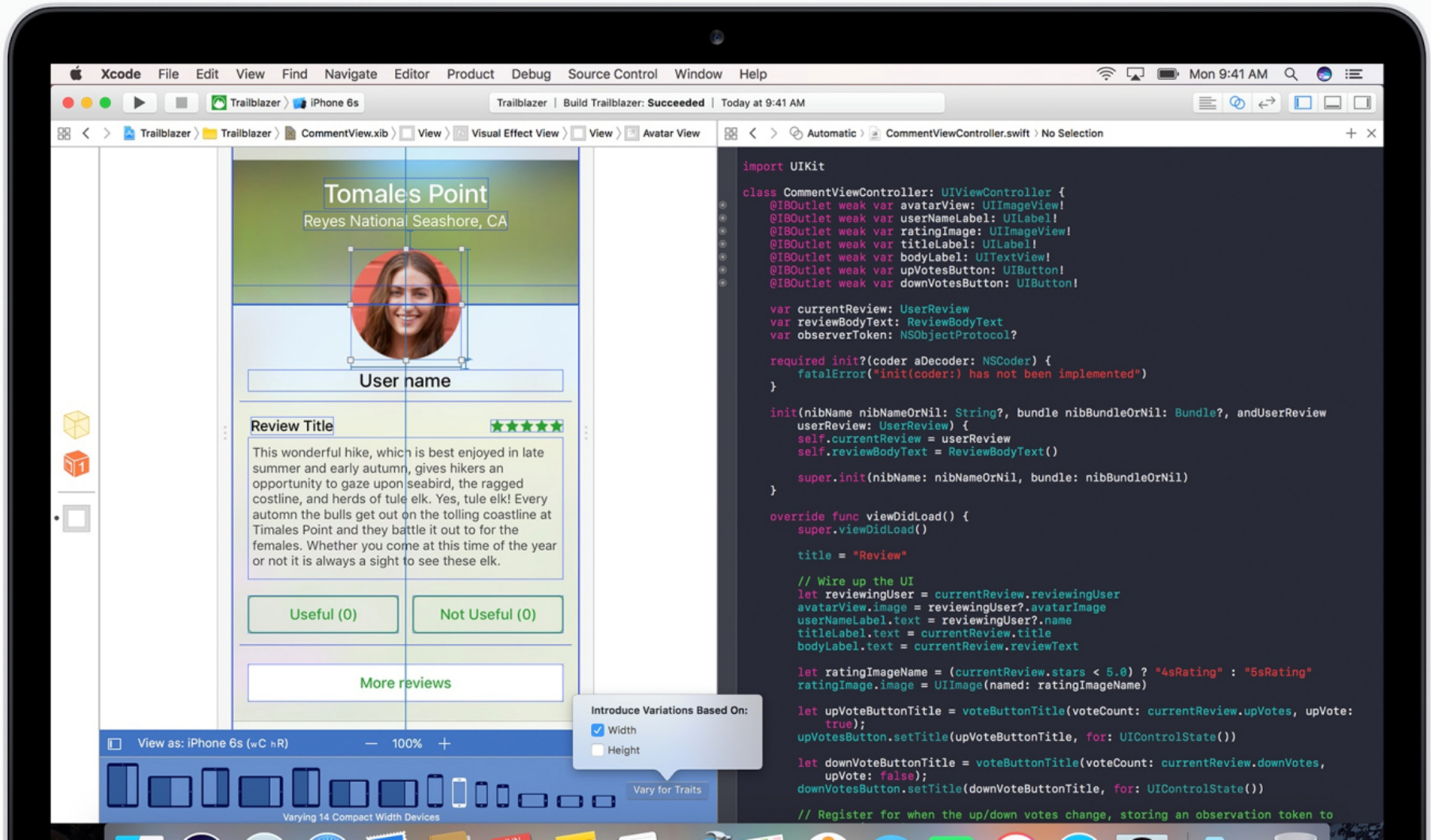
attendance policy - example

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 9 – free in Mac App Store



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

optional type

```
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!"
    }
}
```

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 optionalBear: Bear? =
getBear(named: "Oski")

print(optionalBear.bestFriend.name)
```

Console Output

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

optional chaining

```
var optionalBear: Bear? =
getBear(named: "Oski")

print(optionalBear?.bestFriend.name)
```

Console Output

```
“Oski”
```

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 -> Bool in
    return string == "dog"
}
```

closures

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

/// as a closure
let isGoodClosure = { string -> Bool in
    return string == "dog"
}

/// calling function
isGood(string: "dog")

/// execute closure
isGoodClosure("dog")
```

closures

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

closures

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

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

closures

```
/// as a closure
let isGoodClosure = { string -> Bool 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

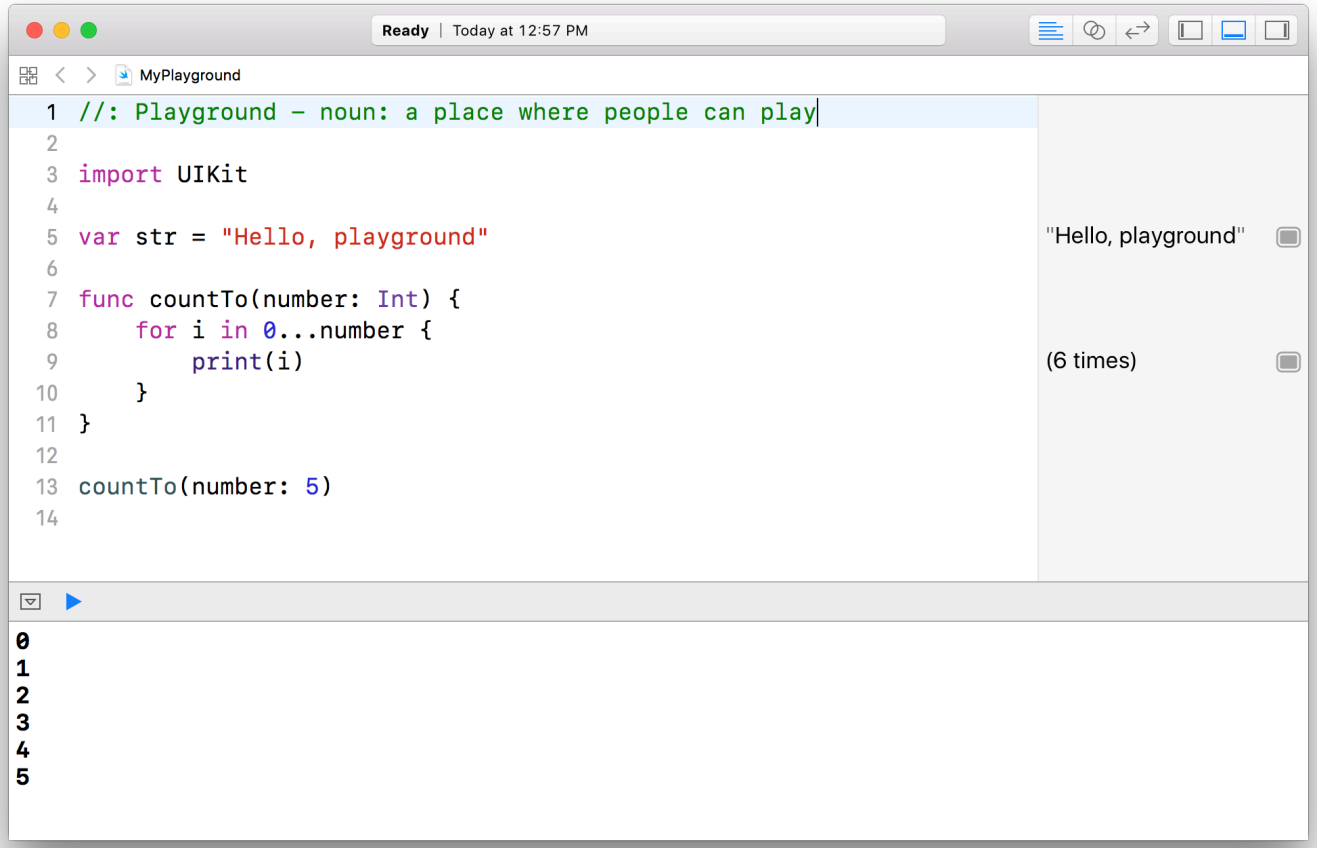
```
Alamofire.request(url).responseData {
  response in
  if let data = response.result.value {
        // do something with data
        print(data)
    }
}
```

(tip! parentheses not required)

protocols

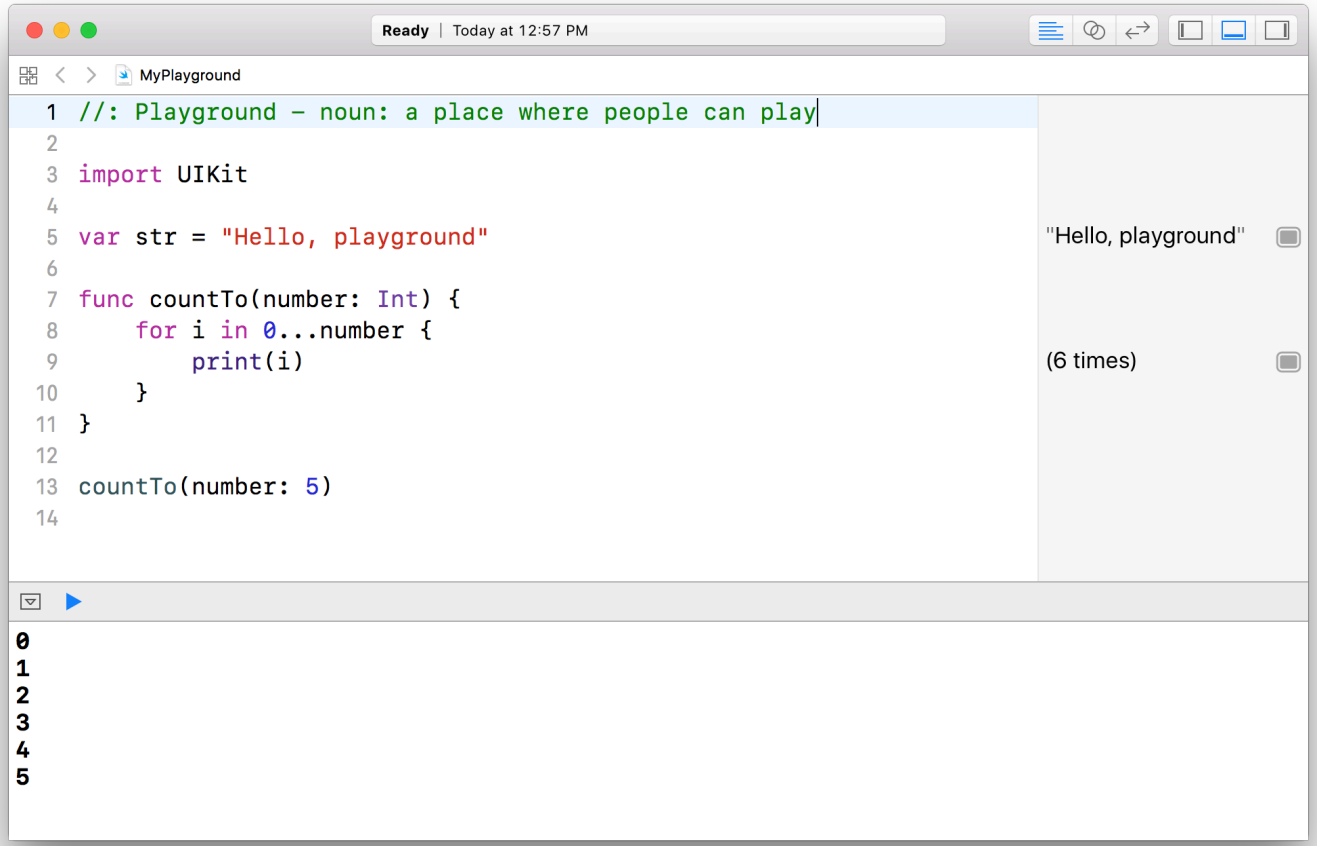
```
protocol DogOwner {
    var dog: Dog { get set }
}

protocol NicePerson {
    func pet(dog: Dog) -> Bool
}

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

playgrounds



lightweight interface for small programs

environment for homework 1



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

submit on Gradescope