Содержание

1. НЕДЕЛЯ 6

1.1. Итоги курса

Мы практически завершили изучение материалов первого курса. Осталось лишь выполнить курсовое задание, после которого вы станете еще на один шаг ближе к мобильной разработке. Подведем итоги того, чему мы научились за это время.

1.1.1. XCode и Playground

Нам удалось познакомиться со средой разработки: xCode и Playground, который облегчает написание кода с целью обучения или тестирования.
(картинка)

1.1.2. Swift vs Objective-C

Вы узнали, что привело к созданию языка Swift и почему он приходит на смену Objective-C. Возможно, на данный момент вам понятны не все преимущества Swift, но чем больше вы будете писать код, тем лучше вы это будете осознавать и тем приятней и быстрее у вас это будет получаться.

```
Swift
extension UIColor {
var sw_hex: String? {
// ...
}
}

Objective-C
@interface UIColor (ELNUtils)
- (NSString *)eln_hex;
@end
```

1.1.3. Строительные блоки в Swift

Мы познакомились с основными блоками, из которых строятся приложения, классы, структуры, перечисления. Теперь вы знаете, в чем разница между значимыми и ссылочными типами, что такое жизненный цикл объект и в каких случаях какой тип лучше использовать. Мы научились создавать собственные типы и наполнять их функционалом с помощью методов, замыканий, функций, всевозможных свойств переменных и констант.

```
struct Resolution {
var width = 0
var height = 0
}
```

```
class VideoMode {
var resolution = Resolution()
var interlaced = false
func someFunc() {
print("Do Nothing")
}
}
```

1.1.4. Control Flow

Вы узнали о различных способах управления кодом внутри приложения и о том, какие для этого используются выражения, в чем их разница, в чем преимущество одних над другими.

```
let x = 50
if x == 1 {
print("One")
} else if x == 2 {
print("Two")
} else if x < 10 && x >= 0 {
print("X где-то от 0 до 10")
} else {
print("X больше 10")
} // X больше 10
```

1.1.5. Управление памятью

Небольшая часть курса была посвящена работе с памятью — это была лишь вводная лекция. Далее мы углубимся в изучение этого материала, но у вас уже есть понимание того, что делать во избежание проблем с захватом объектов, которые никогда не удаляются из памяти.

```
class Device {
let model: String
weak var owner: User?
init(model: String, owner: User) {
self.model = model
self.owner = owner
print("Создано устройство \(model).
Его владелец \(owneree.name)")
}
deinit {
print("Удалено устройство \(model)")
}
}
```

1.1.6. Протоколы

Протоколы — то, на чем строится большая часть работа со Swift. И коллекции являются тому подтверждением. Мы узнали, как устроены коллекции внутри, теперь мы умеем не только создавать собственные, но и проводить с ними различными манипуляции для их трансформации и итерации по

ним. Организация кода внутри приложения также очень важна. В дальнейшем это пригодится вам не раз при взаимодействии с другими разработчиками. С каждым курсом этот навык будет расти, и вы сможете писать красивый код, который будет не только работать быстро и правильно, но и приносить эстетическое удовольствие.

```
class UpperCaseStringProcessorDelegate:
StringProcessorDelegate {
var concatenateSeparator: String {
return " "
}
func transform(_ string: String) -> String {
return string.uppercased()
}
}
```

1.1.7. Типизация и Optionale

Строгая типизация в Swift могла напугать вас, если ранее вы разрабатывали приложения на Objective-C. Но создатели языка постарались максимально упросить этот процесс, и теперь при написании приложения вы делаете гораздо меньше ошибок, которые ранее могли быть выявлены только на этапе выполнения программы. И снова в этом нам помогают протоколы и Optional-типы. Дженерики позволили нам во время обучения на курсе создавать универсальный методы и функции, которые могут поддерживать различные типы, и все это, как видели, реализуется достаточно просто.

```
class SomeClass {
struct InnerStruct {
var vaiable: Int
}
var innerStruct: InnerStruct?
}
let myClass = SomeClass()
var anotherOptional =
myClass.innerStruct?.vaiable
myClass.innerStruct?.vaiable = 55
```

1.1.8. Атрибуты и операторы

Также были лекции, посвященные работе с атрибутами и созданию операторов. Скорее всего, при повседневной разработке Эти знания будут востребованы на так часто, как остальные. Однако более глубокое понимание процессов поставляет расставить вам знания по полочкам и воспринимать работу с языком на новом уровне: лучше понимаете, допускаете меньше ошибок, пишите код с более высокой скоростью. Если у вас остались вопросы или хотите закрыть какие-то пробелы в знаниях, ждем вас на форуме для обсуждения. Успехов в обучении!

```
infix operator **: MultiplicationPrecedence
extension Double {
static func ** (left: Double, right: Double) ->
Double {
return pow(left, right)
}
```

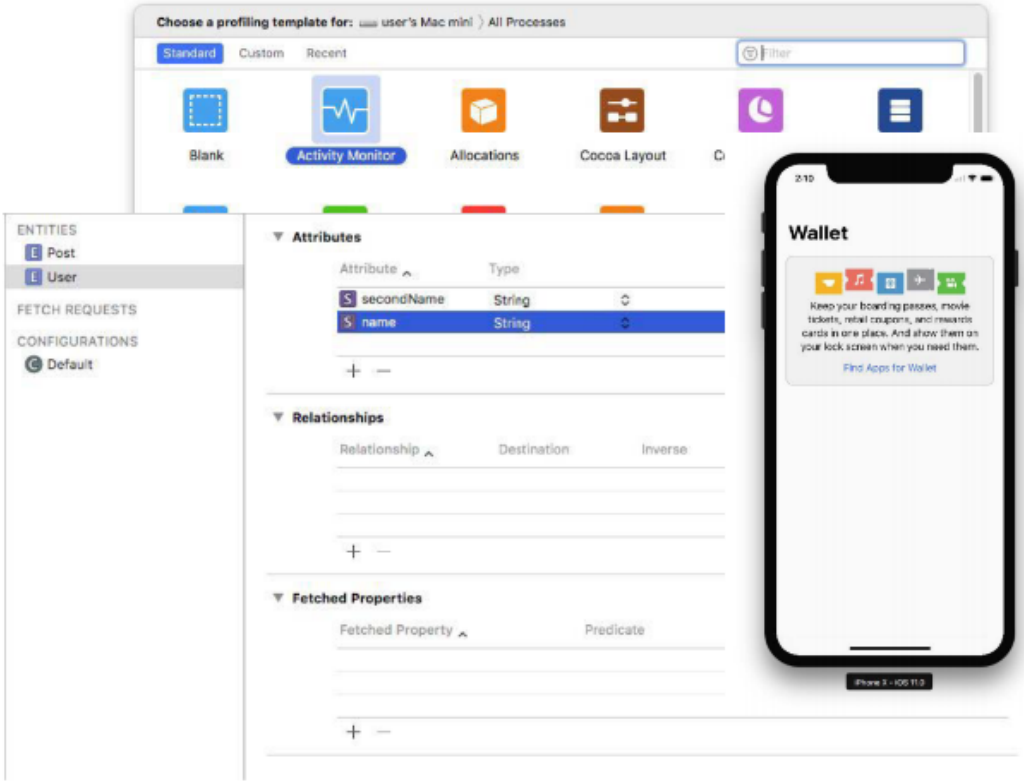
```
}
print(5 ** 5)
// 3125.0
```

1.2. Курсовой проект

Всем привет! Пришло время познакомиться с курсовым заданием, а еще я расскажу вам о том, что за приложение мы будем создавать на протяжении всей специализации. Пожалуй, начнем с общего представления.

1.2.1. Создаем приложение

Как вы помните из вводной лекции, мы хотим научить вас создавать все элементы сложного приложения, начиная от интерфейса, заканчивая работой с локальной базой данных. Но в каком приложении можно объединить и работу с сетью, и обработку графики? И в то же время оно должно быть интересным для написания и востребованным на рынке. Мы оставили свой выбор на клиенте для соцсети, который позволит вам взаимодействовать с другими пользователями, хранить данные о них в локальном хранилище для доступа при отсутствии сети, загружать и обрабатывать фотографии. Мы сможем адаптировать интерфейс под весь спектр актуальных устройств, и, конечно же, позаботимся о безопасном хранении и передачи данных пользователя. С каждым курсом, знакомясь с новыми возможностями языка и операционной системы, мы будем добавлять новые функции в приложение, производить рефакторинг и устранять ошибки, которые могли допустить на ранних этапах разработки. На первом курсе мы успели познакомиться с основами языка, но не затрагивали основные фреймворки iOS, который позволяют создавать интерфейс или работать с сетевыми данными. Однако этих знаний, которые мы получили, будет достаточно для описания модели приложения и данных, которые оно будет хранить. Более подробно цель задания и результаты, которые мы должны получить, описаны в документе, приложенном к курсу. Внимательно ознакомьтесь с ними, загрузите все необходимые файлы и приступайте к выполнению. Желаю успехов!



1.2.2. Строительные блоки в Swift

Мы познакомились с основными блоками, из которых строятся приложения, классы, структуры, перечисления. Теперь вы знаете, в чем разница между значимыми и ссылочными типами, что такое жизненный цикл объект и в каких случаях какой тип лучше использовать. Мы научились создавать собственные типы и наполнять их функционалом с помощью методов, замыканий, функций, всевозможных свойств переменных и констант.

```
struct Resolution {
var width = 0
var height = 0
}
class VideoMode {
var resolution = Resolution()
var interlaced = false
func someFunc() {
print("Do Nothing")
}
}
```

1.2.3. Control Flow

Вы узнали о различных способах управления кодом внутри приложения и о том, какие для этого используются выражения, в чем их разница, в чем преимущество одних над другими.

```
let x = 50
if x == 1 {
print("One")
} else if x == 2 {
print("Two")
} else if x < 10 && x >= 0 {
print("X где-то от 0 до 10")
} else {
print("X больше 10")
} // X больше 10
```

1.2.4. Управление памятью

Небольшая часть курса была посвящена работе с памятью — это была лишь вводная лекция. Далее мы углубимся в изучение этого материала, но у вас уже есть понимание того, что делать во избежание проблем с захватом объектов, которые никогда не удаляются из памяти.

```
class Device {
let model: String
weak var owner: User?
init(model: String, owner: User) {
self.model = model
self.owner = owner
print("Создано устройство \(model).
Его владелец \(ownere.name)")
}
deinit {
print("Удалено устройство \(model)")
}
}
```

1.2.5. Протоколы

Протоколы — то, на чем строится большая часть работа со Swift. И коллекции являются тому подтверждением. Мы узнали, как устроены коллекции внутри, теперь мы умеем не только создавать собственные, но и проводить с ними различными манипуляции для их трансформации и итерации по ним. Организация кода внутри приложения также очень важна. В дальнейшем это пригодится вам не раз при взаимодействии с другими разработчиками. С каждым курсом этот навык будет расти, и вы сможете писать красивый код, который будет не только работать быстро и правильно, но и приносить эстетическое удовольствие.

```
class UpperCaseStringProcessorDelegate:
StringProcessorDelegate {
var concatenateSeparator: String {
return " "
}
func transform(_ string: String) -> String {
return string.uppercased()
}
}
```

1.2.6. Типизация и Optionale

Строгая типизация в Swift могла напугать вас, если ранее вы разрабатывали приложения на Objective-C. Но создатели языка постарались максимально упросить этот процесс, и теперь при написании приложения вы делаете гораздо меньше ошибок, которые ранее могли быть выявлены только на этапе выполнения программы. И снова в этом нам помогают протоколы и Optional-типы. Дженерики позволили нам во время обучения на курсе создавать универсальный методы и функции, которые могут поддерживать различные типы, и все это, как видели, реализуется достаточно просто.

```
class SomeClass {
struct InnerStruct {
var vaiable: Int
}
var innerStruct: InnerStruct?
}
let myClass = SomeClass()
var anotherOptional =
myClass.innerStruct?.vaiable
myClass.innerStruct?.vaiable = 55
```

1.2.7. Атрибуты и операторы

Также были лекции, посвященные работе с атрибутами и созданию операторов. Скорее всего, при повседневной разработке Эти знания будут востребованы на так часто, как остальные. Однако более глубокое понимание процессов поставляет расставить вам знания по полочкам и воспринимать работу с языком на новом уровне: лучше понимаете, допускаете меньше ошибок, пишите код с более высокой скоростью. Если у вас остались вопросы или хотите закрыть какие-то пробелы в знаниях, ждем вас на форуме для обсуждения. Успехов в обучении!

```
infix operator **: MultiplicationPrecedence
extension Double {
static func ** (left: Double, right: Double) ->
Double {
return pow(left, right)
}
}
print(5 ** 5)
// 3125.0
```

1.3. Курсовой проект

Всем привет! Пришло время познакомиться с курсовым заданием, а еще я расскажу вам о том, что за приложение мы будем создавать на протяжении всей специализации. Пожалуй, начнем с общего представления.

1.3.1. Создаем приложение

Как вы помните из вводной лекции, мы хотим научить вас создавать все элементы сложного приложения, начиная от интерфейса, заканчивая работой с локальной базой данных. Но в каком приложении можно объединить и работу с сетью, и обработку графики? И в то же время оно должно быть интересным для написания и востребованным на рынке. Мы оставили свой выбор на клиенте для соцсети, который позволит вам взаимодействовать с другими пользователями, хранить данные о них в локальном хранилище для доступа при отсутствии сети, загружать и обрабатывать фотографии. Мы сможем адаптировать интерфейс под весь спектр актуальных устройств, и, конечно же, позаботимся о безопасном хранении и передачи данных пользователя. С каждым курсом, знакомясь с новыми возможностями языка и операционной системы, мы будем добавлять новые функции в приложение, производить рефакторинг и устранять ошибки, которые могли допустить на ранних этапах разработки. На первом курсе мы успели познакомиться с основами языка, но не затрагивали основные фреймворки iOS, который позволяют создавать интерфейс или работать с сетевыми данными. Однако этих знаний, которые мы получили, будет достаточно для описания модели приложения и данных, которые оно будет хранить. Более подробно цель задания и результаты, которые мы должны получить, описаны в документе, приложенном к курсу. Внимательно ознакомьтесь с ними, загрузите все необходимые файлы и приступайте к выполнению. Желаю успехов!