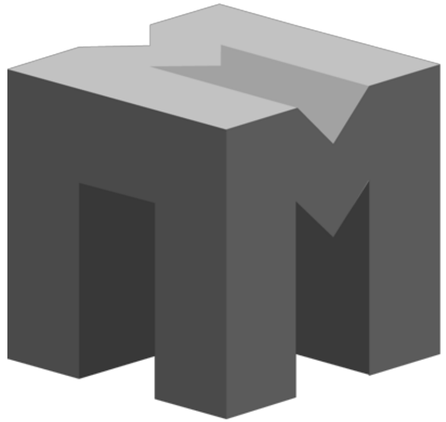

EDUCATION
Surf

iOS Школа



Александр Чаусов

Основы Swift

Основные элементы

Типы данных

Optionals

Расширенные возможности

Основные элементы

Типы данных

Optionals

Расширенные возможности

Constants and Variables

```
let pi = 3.14

var str = "Hello, playground"
```

Массивы

```
var names = ["Vasya", "Petya", "Ibragim"]

names = names + ["Svetlana"]
```

Массивы

```swift
var names = ["Vasya", "Petya", "Ibragim"]

// names = names + ["Svetlana"] - так делать не надо)
names += ["Svetlana"]
names.append("Ekaterina")

names[4] // "Ekaterina"

names.remove(at: 0)

names.count // 4
```

Словари

```
var films = ["Animation": "Angry Birds 2",
                 "Drama": "The Lion King"]

films["Crime"] = "The Gangster, the Cop, the Devil"

films["Crime"] = "John Wick"

films["Drama"] // "The Lion King"

films.removeValue(forKey: "Crime")

films.keys // ["Animation", "Drama"]
```

Tuples

```
var card = ("Red", "Hearts", 7)

card.0 // "Red"
card.1 // "Hearts"
card.2 // 7
```

Tuples

```
var card = (color: "Red", suit: "Hearts", value: 7)

card.color // "Red"
card.suit // "Hearts"
card.value // 7
```

Tuples

```
var card = (color: "Red", suit: "Hearts", value: 7)

card = ("Black", "Hearts", 10)
```

Условные операторы

```
if card.value < 5 {
    print("This is a weak card")
} else if card value <= 10 {
    print("This is a normal card")
} else {
    print("This is a strong card")
}
```

Условные операторы

```
let character = "d"
if character == "a" {
    print("this is the A letter")
} else if character == "b" {
    print("this is the B letter")
} else if character == "c" {
    print("this is the C letter")
} else {
    print("this is another letter")
}
```

Switch

```
switch значение для сопоставления {
case значение 1:
    инструкция для значения 1
case значение 2, значение 3:
    инструкция для значения 2 или значения 3
default:
    инструкция, если совпадений с шаблонами не найдено
}
```

Switch

```
let character = "d"
switch character {
case "a":
    print("this is the A letter")
case "b":
    print("this is the B letter")
case "c":
    print("this is the C letter")
default:
    print("this is another letter")
}
```

Switch

```
switch card.value {
case let x where x < 5:
    print("This is a weak card")
case 6...10:
    print("This is a normal card")
default:
    print("This is a strong card")
}
```

Guard

```
guard card.value <= 13 else {
    return
}
print(card.value)
```

Loops

```
var index = 0
while index < card.value {
    print(index)
    index += 1
}
```

Loops

```
var index = 0
repeat {
    print(index)
    index += 1
} while index < card.value
```

Loops

```
for index in 0..<card.value {
    print(index)
}
```

Loops

```
let names = ["Vasya", "Petya", "Ibragim", "Svetlana", "Ekaterina"]
for name in names {
    print(name)
}
```

Loops

```
var names = ["Vasya", "Petya", "Ibragim", "Svetlana", "Ekaterina"]
for (index, name) in names.enumerated() {
    print("\(name) in \(index) place")
}
```

Основные элементы

Типы данных

Optionals

Расширенные возможности

Основные элементы

Типы данных

Optionals

Расширенные возможности

Строгая статическая типизация

- Swift - язык со строгой статической типизацией
- Проверка типов выполняется при компиляции кода
- Swift имеет «вывод типов»

Вывод типов

```
let integerPart = 3
let fractionalPart = 0.14159
let pi = integerPart + fractionalPart
```

Вывод типов

```
let integerPart = 3
let fractionalPart = 0.14159
let pi = Double(integerPart) + fractionalPart
```

Структуры, классы и перечисления

- хранить значения
- объявлять инициализаторы (конструкторы)
- реализовывать методы
- расширять функциональность
- могут реализовывать протоколы

Структуры, классы и перечисления

Дополнительные возможности классов:

- возможность наследования
- наличие деинициализаторов
- имеет счетчик ссылок (ARC)

Автоматический подсчет ссылок (ARC)

- Swift использует **automatic reference counting** (автоматический подсчет ссылок) для отслеживания и управления памятью вашего приложения
- ARC применима только для экземпляров класса. Структуры и перечисления не участвуют в ARC (являются value type)

Структуры

```
struct Resolution {
    let width: Int
    let height: Int
}

let resolution = Resolution(width: 5, height: 10)
```

Классы

```
class Monitor {
    let resolution: Resolution
    let name: String

    init(resolution: Resolution, name: String) {
        self.resolution = resolution
        self.name = name
    }
}

let smallResolution = Resolution(width: 1280, height: 1024)
let monitor = Monitor(resolution: smallResolution,
                      name: "DELL")
```

Перечисления

```
enum CompassPoint {
    case north
    case south
    case east
    case west
}

var direction: CompassPoint = .west
```

Перечисления

```
switch direction {
case .north:
    print("north")
case .south:
    print("south")
case .east:
    print("east")
case .west:
    print("west")
}
```

Перечисления

```
enum Barcode {
    case qrCode(String)
}

let code = Barcode.qrCode("myCode")

switch code {
case .qrCode(let value):
    print(value)
}
```

Перечисления

```
enum Planet: Int {
    case mercury = 1
    case venus
    case earth
    case mars
    case jupiter
    case saturn
    case uranus
    case neptune
}

let earth = Planet(rawValue: 3)
```

Вычисляемые свойства

```
struct AlternativeRect {
    var origin = CGPoint()
    var size = CGSize()
    var center: CGPoint {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return CGPoint(x: centerX, y: centerY)
        }
        set {
            origin.x = newValue.x - (size.width / 2)
            origin.y = newValue.y - (size.height / 2)
        }
    }
}
```

Вычисляемые свойства для чтения

```
struct AlternativeRect {
    var origin = CGPoint()
    var size = CGSize()
    var center: CGPoint {
        get {
            let centerX = origin.x + (size.width / 2)
            let centerY = origin.y + (size.height / 2)
            return CGPoint(x: centerX, y: centerY)
        }
    }
}
```

Наблюдатели свойства

```
class StepCounter {
    var totalSteps: Int = 0 {
        willSet(newTotalSteps) {
            print("Вот-вот значение будет равно \(newTotalSteps)")
        }
        didSet {
            if totalSteps > oldValue  {
                print("Добавлено \(totalSteps - oldValue) шагов")
            }
        }
    }
}
```

Создание методов

```
func greet(person: String) -> String {
    return "Hello, " + person + "!"
}

let greetings = greet(person: "Anna")
```

Создание методов

```
func greet(with person: String) -> String {
    return "Hello, " + person + "!"
}

let greetings = greet(with: "Anna")
```

Создание методов

```
func greet(_ person: String) -> String {
    return "Hello, " + person + "!"
}

let greetings = greet("Anna")
```

Создание методов

```
func greet(_ person: String) -> String {
    return "Hello, " + person + "!"
}

let greetings = greet("Anna")
let _ = greet("Galina")
```

Создание методов

```
func greet(_ person: String) {
    let greetings = "Hello, " + person + "!"
    print(greetings)
}

greet("Anna")
```

Создание методов

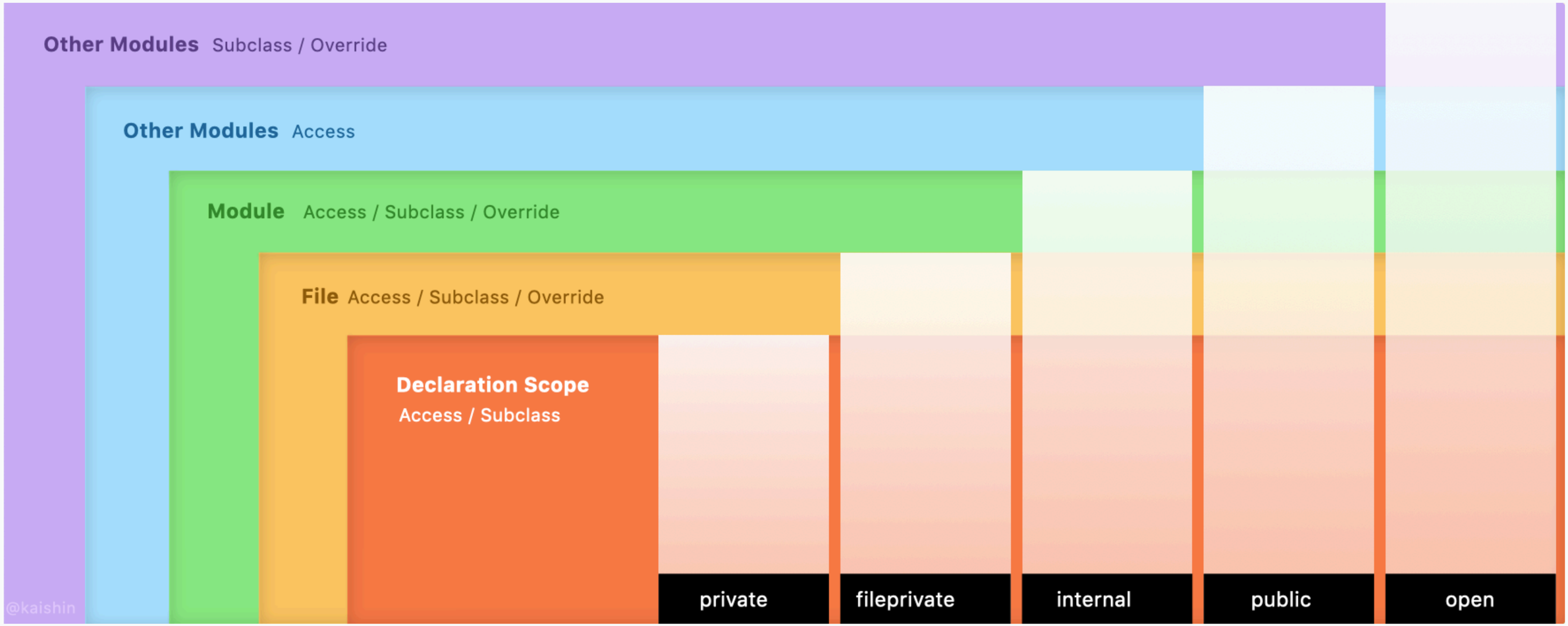
```
func greet(firstPerson: String, secondPerson: String) {
    let greetings = "Hello, " + firstPerson + " and " + secondPerson + "!"
    print(greetings)
}

greet(firstPerson: "Anna", secondPerson: "Galina")
```

Уровни доступа

- open/public - открытый и публичный доступ
- internal - внутренний
- fileprivate - уровень доступа в пределах исходного файла
- private - частный

Уровни доступа



Основные элементы

Типы данных

Optionals

Расширенные возможности

Основные элементы

Типы данных

Optionals

Расширенные возможности

Что такое Optionals?

Optionals (опционалы) — это удобный механизм обработки ситуаций, когда значение переменной может отсутствовать. Значение будет использовано, только если оно есть.

Пример

```
struct Pocket {
    let pensCount: Int
}

struct Person {
    let pocket: Pocket?

    var pens: Int? {
        return pocket?.pensCount
    }
}

let pocket = Pocket(pensCount: 2)
let person1 = Person(pocket: pocket)
let person2 = Person(pocket: nil)

person1.pens // 2
person2.pens // nil
```

Unwrapping

- optional binding
- forced unwrapping
- nil coalescing

Unwrapping

```
// optional binding
if let pens = person1.pens {
    print(pens)
}

// forced unwrapping
print(person1.pens!)

// nil coalescing
print(person1.pens ?? 0)
```

Guard

```
func joinStrings(string1: String?, string2: String?, prefix: String?) -> String? {
    if let string1 = string1, let string2 = string2 {
        let result = string1 + string2
        if let prefix = prefix {
            return prefix + result
        } else {
            return result
        }
    } else {
        return nil
    }
}
```

Guard

```
func joinStrings(string1: String?, string2: String?, prefix: String?) -> String? {
    if let string1 = string1, let string2 = string2 {
        return (prefix ?? "") + string1 + string2
    } else {
        return nil
    }
}
```

Guard

```
func joinStrings(string1: String?, string2: String?, prefix: String?) -> String? {
    guard let string1 = string1, let string2 = string2 else {
        return nil
    }
    return (prefix ?? "") + string1 + string2
}
```

Основные элементы

Типы данных

Optionals

Расширенные возможности

Основные элементы

Типы данных

Optionals

Расширенные возможности

Protocols

Протокол определяет ряд методов и переменных, которые в обязательном порядке должен наследовать определенный тип

Protocols

```
protocol SomeProtocol {
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
    func random() -> Double
}
```

Protocols

```
protocol FullyNamed {
    var fullName: String { get }
    func rename(_ name: String)
}
class Starship: FullyNamed {
    private var prefix: String?
    private var name: String
    init(name: String, prefix: String?) {
        self.name = name
        self.prefix = prefix
    }
    var fullName: String {
        return (prefix ?? "") + " " + name
    }
    func rename(_ name: String) {
        self.name = name
    }
}
let enterprise = Starship(name: "Enterprise", prefix: "USS")
enterprise.fullName
```

Extensions

- добавлять вычисляемые свойства
- определять методы экземпляра и методы типа
- предоставлять новые конструкторы
- реализовать протокол

Синтаксис расширений

```
extension SomeType {
    // описываем новую функциональность для типа SomeType
}

extension SomeType: SomeProtocol {
    // реализация требования протокола тут
}
```

Protocols + Extensions

```
class Starship {
    private var prefix: String?
    private var name: String
    init(name: String, prefix: String?) {
        self.name = name
        self.prefix = prefix
    }
}
extension Starship: FullyNamed {
    var fullName: String {
        return (prefix ?? "") + " " + name
    }
    func rename(_ name: String) {
        self.name = name
    }
}
```

Основные элементы

Типы данных

Optionals

Расширенные возможности

Основные элементы

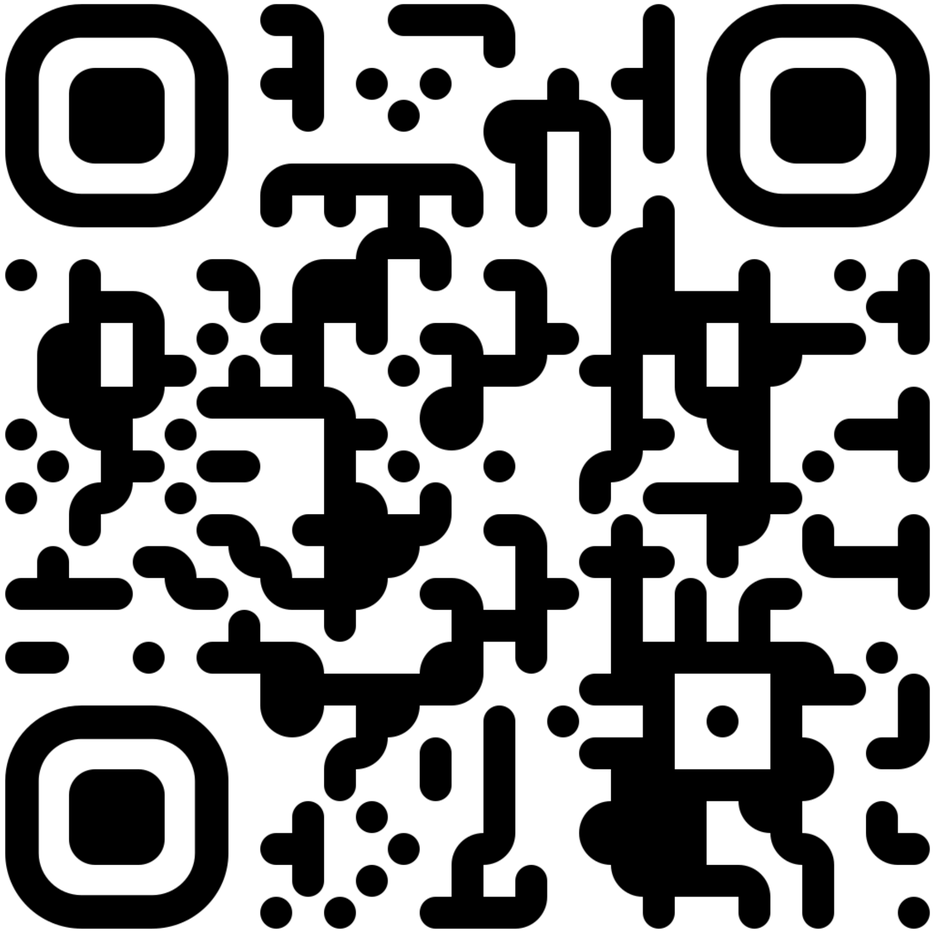
Типы данных

Optionals

Расширенные возможности

Материалы

http://bit.ly/iossurf19lecture3




EDUCATION
Surf