Функции и поток управления в Swift

Презентация 2

10 октября 2025 г.

Содержание

Функции в Swift

- Функции — это самостоятельные блоки кода, выполняющие определенную задачу
- В Swift функции являются объектами первого класса
- Поддерживают передачу параметров и возврат значений
- Могут быть вложенными и рекурсивными

Передача параметров по значению

```swift
// Передача по значению (по умолчанию)
func greet(name: String) {
    print("Привет, \(name)!")
}

// Вызов функции
greet(name: "Анна")

// Параметры по умолчанию
func greetWithDefault(name: String = "Гость") {
    print("Привет, \(name)!")
}

greetWithDefault() // "Привет, Гость!"
greetWithDefault(name: "Петр") // "Привет, Петр!"
```

Передача по ссылке (inout)

```swift
// Передача по ссылке с помощью inout
func swapValues(_ a: inout Int, _ b: inout Int) {
    let temp = a
    a = b
    b = temp
}

var x = 5
var y = 10
print("До обмена: x = \(x), y = \(y)")

swapValues(&x, &y)
print("После обмена: x = \(x), y = \(y)")
// Вывод: До обмена: x = 5, y = 10
//        После обмена: x = 10, y = 5
```

Вариативные параметры

```swift
// Функция с вариативными параметрами
func sum(_ numbers: Int...) -> Int {
    var total = 0
    for number in numbers {
        total += number
    }
    return total
}

let result1 = sum(1, 2, 3, 4, 5) // 15
let result2 = sum(10, 20) // 30
let result3 = sum() // 0

// Функция с несколькими параметрами
func calculate(averageOf numbers: Double...) -> Double {
    guard !numbers.isEmpty else { return 0 }
    let sum = numbers.reduce(0, +)
    return sum / Double(numbers.count)
}
```

Возврат значений

```swift
// Простой возврат значения
func add(_ a: Int, _ b: Int) -> Int {
    return a + b
}

// Возврат кортежа
func getMinMax(_ numbers: [Int]) -> (min: Int, max: Int) {
    guard !numbers.isEmpty else {
        return (0, 0)
    }

    let sorted = numbers.sorted()
    return (sorted.first!, sorted.last!)
}

let numbers = [3, 1, 4, 1, 5, 9, 2, 6]
let result = getMinMax(numbers)
print("Минимум: \(result.min), Максимум: \(result.max)")
```

Опциональные возвращаемые значения

```swift
// Возврат опционального значения
func findIndex(of value: Int, in array: [Int]) -> Int? {
    for (index, item) in array.enumerated() {
        if item == value {
            return index
        }
    }
    return nil
}

let numbers = [1, 3, 5, 7, 9]
if let index = findIndex(of: 5, in: numbers) {
    print("Найдено на позиции: \(index)")
} else {
    print("Элемент не найден")
}

// Возврат нескольких значений через кортеж
func divide(_ a: Int, by b: Int) -> (quotient: Int, remainder: Int)? {
    guard b != 0 else { return nil }
    return (a / b, a % b)
}
```

Рекурсивные функции

```swift
// Классический пример - факториал
func factorial(_ n: Int) -> Int {
    if n <= 1 {
        return 1
    }
    return n * factorial(n - 1)
}

print(factorial(5)) // 120

// Числа Фибоначчи
func fibonacci(_ n: Int) -> Int {
    if n <= 1 {
        return n
    }
    return fibonacci(n - 1) + fibonacci(n - 2)
}

print(fibonacci(10)) // 55

// Более эффективная версия с мемоизацией
func fibonacciMemo(_ n: Int, memo: inout [Int: Int]) -> Int {
    if let result = memo[n] {
        return result
    }

    if n <= 1 {
        memo[n] = n
        return n
    }

    let result = fibonacciMemo(n - 1, memo: &memo) +
                 fibonacciMemo(n - 2, memo: &memo)
    memo[n] = result
    return result
```

Замыкания (Closures)

```swift
// Простое замыкание
let greet = { (name: String) in
    return "Привет, \(name)!"
}

print(greet("Мир")) // "Привет, Мир!"

// Замыкание как параметр функции
func processNumbers(_ numbers: [Int],
                    using closure: (Int) -> Int) -> [Int] {
    return numbers.map(closure)
}

let numbers = [1, 2, 3, 4, 5]
let doubled = processNumbers(numbers) { $0 * 2 }
let squared = processNumbers(numbers) { $0 * $0 }

print(doubled) // [2, 4, 6, 8, 10]
print(squared) // [1, 4, 9, 16, 25]
```

Захват значений в замыканиях

```swift
// Захват внешних переменных
func makeCounter() -> () -> Int {
    var count = 0
    return {
        count += 1
        return count
    }
}

let counter1 = makeCounter()
let counter2 = makeCounter()

print(counter1()) // 1
print(counter1()) // 2
print(counter2()) // 1
print(counter1()) // 3

// Захват по ссылке
var multiplier = 10
let multiply = { (number: Int) in
    return number * multiplier
}

print(multiply(5)) // 50
multiplier = 20
print(multiply(5)) // 100
```

Конструкции потока управления

- Условные конструкции (if, guard, switch)
- Циклы (for, while, repeat-while)
- Итераторы и коллекции
- Блоки кода и области видимости

Условные операторы

```swift
// Оператор if
let temperature = 25
if temperature > 30 {
    print("Жарко!")
} else if temperature > 20 {
    print("Тепло")
} else {
    print("Прохладно")
}

// Тернарный оператор
let weather = temperature > 25 ? "солнечно" : "облачно"
print("Погода: \(weather)")

// Оператор guard
func processUser(_ user: String?) {
    guard let username = user, !username.isEmpty else {
        print("Неверное имя пользователя")
        return
    }
    print("Обработка пользователя: \(username)")
}

processUser("Анна") // "Обработка пользователя: Анна"
processUser(nil)    // "Неверное имя пользователя"
```

Оператор switch

```
1  // Простой switch
2  let day = "понедельник"
3  switch day {
4  case "понедельник":
5      print("Начало рабочей недели")
6  case "пятница":
7      print("Конец рабочей недели")
8  case "суббота", "воскресенье":
9      print("Выходные")
10 default:
11     print("Обычный день")
12 }
13
14 // Switch с диапазонами
15 let score = 85
16 switch score {
17 case 90...100:
18     print("Отлично")
19 case 80..<90:
20     print("Хорошо")
21 case 70..<80:
22     print("Удовлетворительно")
23 default:
24     print("Неудовлетворительно")
25 }
```

Циклы for

```swift
// Цикл for-in с диапазонами
for i in 1...5 {
    print("Число: \(i)")
}

// Цикл for-in с массивами
let fruits = ["яблоко", "банан", "апельсин"]
for fruit in fruits {
    print("Фрукт: \(fruit)")
}

// Цикл for-in со словарями
let scores = ["Анна": 95, "Петр": 87, "Мария": 92]
for (name, score) in scores {
    print("\(name): \(score) баллов")
}

// Цикл for-in с индексами
for (index, fruit) in fruits.enumerated() {
    print("\(index + 1). \(fruit)")
}
```

Циклы while

```swift
// Цикл while
var count = 0
while count < 5 {
    print("Счетчик: \(count)")
    count += 1
}

// Цикл repeat-while (аналог do-while)
var number = 0
repeat {
    print("Число: \(number)")
    number += 1
} while number < 3

// Условные операторы в циклах
for i in 1...10 {
    if i % 2 == 0 {
        continue // Пропустить четные числа
    }
    if i > 7 {
        break // Выйти из цикла
    }
    print("Нечетное число: \(i)")
}
```

Итераторы и коллекции

```
// Методы итерации для массивов
let numbers = [1, 2, 3, 4, 5]

// map - преобразование элементов
let doubled = numbers.map { $0 * 2 }
print(doubled) // [2, 4, 6, 8, 10]

// filter - фильтрация элементов
let evenNumbers = numbers.filter { $0 % 2 == 0 }
print(evenNumbers) // [2, 4]

// reduce - свертка массива
let sum = numbers.reduce(0, +)
print(sum) // 15

// forEach - выполнение действия для каждого элемента
numbers.forEach { print("Элемент: \($0)") }

// Комбинирование методов
let result = numbers
    .filter { $0 % 2 == 0 }
    .map { $0 * $0 }
    .reduce(0, +)
print(result) // 20 (4 + 16)
```

Дополнительные итераторы

```swift
// zip - объединение двух последовательностей
let names = ["Анна", "Петр", "Мария"]
let ages = [25, 30, 28]

for (name, age) in zip(names, ages) {
    print("\(name) - \(age) лет")
}

// stride - создание последовательностей с шагом
for i in stride(from: 0, to: 10, by: 2) {
    print(i) // 0, 2, 4, 6, 8
}

// enumerated - получение индексов и значений
let colors = ["красный", "зеленый", "синий"]
for (index, color) in colors.enumerated() {
    print("\(index): \(color)")
}

// lazy - ленивая оценка
let largeNumbers = Array(1...1000000)
let lazyResult = largeNumbers.lazy
    .filter { $0 % 2 == 0 }
    .map { $0 * $0 }
    .prefix(5) // Берем только первые 5 элементов
    .map { $0 }
print(Array(lazyResult))
```

Блоки кода и области видимости

```swift
// Локальные блоки кода
func demonstrateScope() {
    let globalVar = "Глобальная переменная"

    // Блок 1
    do {
        let localVar1 = "Локальная переменная 1"
        print("В блоке 1: \(globalVar), \(localVar1)")
    }

    // Блок 2
    do {
        let localVar2 = "Локальная переменная 2"
        print("В блоке 2: \(globalVar), \(localVar2)")
        // print(localVar1) // Ошибка! Переменная недоступна
    }

    print("Вне блоков: \(globalVar)")
    // print(localVar1) // Ошибка! Переменная недоступна
}

demonstrateScope()
```

Вложенные функции

```swift
// Функции внутри функций
func outerFunction(x: Int) -> Int {
    var multiplier = 2

    func innerFunction(y: Int) -> Int {
        return y * multiplier // Захватывает multiplier из внешней функции
    }

    multiplier = 3 // Изменяем захваченную переменную
    return innerFunction(y: x)
}

print(outerFunction(x: 5)) // 15 (5 * 3)

// Рекурсивные вложенные функции
func factorial(_ n: Int) -> Int {
    func factorialHelper(_ n: Int, _ acc: Int) -> Int {
        if n <= 1 {
            return acc
        }
        return factorialHelper(n - 1, acc * n)
    }
    return factorialHelper(n, 1)
}

print(factorial(5)) // 120
```

defer - отложенное выполнение

```swift
// Блоки defer выполняются при выходе из области видимости
func processFile() {
    print("Начало обработки файла")

    defer {
        print("Очистка ресурсов")
    }

    defer {
        print("Закрытие файла")
    }

    print("Чтение файла")
    print("Обработка данных")

    // defer блоки выполняются в обратном порядке
}

processFile()
// Вывод:
// Начало обработки файла
// Чтение файла
// Обработка данных
// Закрытие файла
// Очистка ресурсов

// defer в циклах
for i in 1...3 {
    defer {
        print("Завершение итерации \(i)")
    }
    print("Итерация \(i)")
}
```

Ключевые моменты

- **Функции** — основа модульности в Swift
- **Передача параметров** может быть по значению или по ссылке (inout)
- **Рекурсия** позволяет элегантно решать задачи
- **Замыкания** обеспечивают функциональное программирование
- **Условные конструкции** (if, guard, switch) управляют логикой
- **Циклы** и **итераторы** обрабатывают коллекции
- **Блоки кода** определяют области видимости переменных

Практические рекомендации

- Используйте `guard` для раннего выхода из функций
- Предпочитайте `map`, `filter`, `reduce` циклам
- Применяйте `defer` для очистки ресурсов
- Избегайте глубокой рекурсии без мемоизации
- Используйте замыкания для обработки асинхронных операций

Спасибо за внимание!

Вопросы?