Основы данных и переменных в Swift

Презентация 1

10 октября 2025 г.

Содержание

Данные в Swift

- Swift — строго типизированный язык программирования
- Каждое значение имеет определенный тип
- Типы обеспечивают безопасность и производительность
- Поддержка как примитивных, так и пользовательских типов

Примитивные типы данных

```
1  // Целочисленные типы
2  let int8: Int8 = 127         // 8-битное целое
3  let int16: Int16 = 32767     // 16-битное целое
4  let int32: Int32 = 2147483647 // 32-битное целое
5  let int64: Int64 = 9223372036854775807 // 64-битное целое
6  let int: Int = 42            // Целое число (зависит от платформы)
7
8  // Беззнаковые целочисленные типы
9  let uint8: UInt8 = 255
10 let uint: UInt = 100
11
12 // Числа с плавающей точкой
13 let float: Float = 3.14      // 32-битное число с плавающей точкой
14 let double: Double = 3.14159265359 // 64-битное число с плавающей точкой
15
16 // Булев тип
17 let isTrue: Bool = true
18 let isFalse: Bool = false
19
20 // Символьный тип
21 let character: Character = "A"
22 let unicodeChar: Character = "      " // Поддержка Unicode
```

Строковые типы

```swift
// Строка
let string: String = "Привет, мир!"
let emptyString = ""
let multilineString = """
    Это многострочная
    строка в Swift
    """

// Интерполяция строк
let name = "Анна"
let age = 25
let greeting = "Привет, \(name)! Тебе \(age) лет."
print(greeting) // "Привет, Анна! Тебе 25 лет."

// Операции со строками
let firstName = "Иван"
let lastName = "Петров"
let fullName = firstName + " " + lastName
print(fullName) // "Иван Петров"

// Свойства строк
print(string.count) // Количество символов
print(string.isEmpty) // Пустая ли строка
print(string.hasPrefix("Привет")) // Начинается ли с подстроки
```

Пользовательские типы данных

```swift
// Структуры
struct Point {
    var x: Double
    var y: Double

    func distance(from other: Point) -> Double {
        let dx = x - other.x
        let dy = y - other.y
        return sqrt(dx * dx + dy * dy)
    }
}

let point1 = Point(x: 0, y: 0)
let point2 = Point(x: 3, y: 4)
print(point1.distance(from: point2)) // 5.0

// Перечисления
enum Direction {
    case north
    case south
    case east
    case west
}

let direction = Direction.north
switch direction {
case .north:
    print("Идем на север")
case .south:
    print("Идем на юг")
case .east:
    print("Идем на восток")
case .west:
    print("Идем на запад")
}
```

Классы и протоколы

```swift
// Классы
class Person {
    var name: String
    var age: Int

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }

    func introduce() -> String {
        return "Меня зовут \(name), мне \(age) лет"
    }
}

let person = Person(name: "Мария", age: 30)
print(person.introduce())

// Протоколы
protocol Drawable {
    func draw()
}

struct Circle: Drawable {
    var radius: Double

    func draw() {
        print("Рисуем круг радиусом \(radius)")
    }
}

let circle = Circle(radius: 5.0)
circle.draw()
```

Арифметические операции

```
1  // Основные арифметические операции
2  let a = 10
3  let b = 3
4
5  let sum = a + b         // 13
6  let difference = a - b // 7
7  let product = a * b     // 30
8  let quotient = a / b    // 3 (целочисленное деление)
9  let remainder = a % b   // 1 (остаток от деления)
10
11 // Операции с числами с плавающей точкой
12 let x = 10.0
13 let y = 3.0
14 let floatQuotient = x / y // 3.333...
15
16 // Составные операторы присваивания
17 var number = 5
18 number += 3  // number = 8
19 number -= 2  // number = 6
20 number *= 2  // number = 12
21 number /= 3  // number = 4
22
23 // Операции сравнения
24 let isEqual = (a == b)       // false
25 let isNotEqual = (a != b)    // true
26 let isGreater = (a > b)      // true
27 let isLess = (a < b)         // false
28 let isGreaterOrEqual = (a >= b) // true
29 let isLessOrEqual = (a <= b)   // false
```

Логические операции

```swift
// Логические операции
let isSunny = true
let isWarm = true
let isRaining = false

// Логическое И (&&)
let goodWeather = isSunny && isWarm
print(goodWeather) // true

// Логическое ИЛИ (||)
let outdoorActivity = isSunny || isWarm
print(outdoorActivity) // true

// Логическое НЕ (!)
let badWeather = !isSunny
print(badWeather) // false

// Комбинированные условия
let perfectDay = isSunny && isWarm && !isRaining
print(perfectDay) // true

// Короткое замыкание (short-circuit evaluation)
func expensiveOperation() -> Bool {
    print("Выполняется дорогая операция")
    return true
}

let result = false && expensiveOperation() // expensiveOperation не выполнится
print(result) // false
```

Операции со строками

```swift
// Конкатенация строк
let firstName = "Анна"
let lastName = "Петрова"
let fullName = firstName + " " + lastName

// Интерполяция строк
let age = 25
let message = "Привет, \(firstName)! Тебе \(age) лет."

// Сравнение строк
let str1 = "apple"
let str2 = "banana"
let str3 = "apple"

print(str1 == str3)  // true
print(str1 < str2)   // true (лексикографическое сравнение)

// Методы строк
let text = "Hello, World!"
print(text.count)                    // 13
print(text.isEmpty)                  // false
print(text.hasPrefix("Hello"))       // true
print(text.hasSuffix("World!"))      // true
print(text.uppercased())             // "HELLO, WORLD!"
print(text.lowercased())             // "hello, world!"

// Подстроки
let startIndex = text.index(text.startIndex, offsetBy: 7)
let endIndex = text.endIndex
let substring = String(text[startIndex..<endIndex])
print(substring) // "World!"
```

Явное приведение типов

```swift
// Приведение числовых типов
let intValue = 42
let doubleValue = Double(intValue)     // 42.0
let floatValue = Float(intValue)       // 42.0
let stringValue = String(intValue)   // "42"

// Приведение между целочисленными типами
let int8Value: Int8 = 100
let int16Value = Int16(int8Value)    // 100
let int32Value = Int32(int8Value)    // 100

// Приведение строк к числам
let numberString = "123"
let convertedInt = Int(numberString) // Optional(123)
let convertedDouble = Double(numberString) // Optional(123.0)

// Безопасное приведение с проверкой
if let validInt = Int(numberString) {
    print("Число: \(validInt)")
} else {
    print("Не удалось преобразовать в число")
}

// Приведение чисел к строкам
let number = 456
let stringFromNumber = String(number) // "456"
```

Неявное приведение типов

```
1  // Автоматическое определение типа
2  let inferredInt = 42          // Int
3  let inferredDouble = 3.14     // Double
4  let inferredString = "Hello" // String
5  let inferredBool = true       // Bool
6
7  // Литералы и типы
8  let binaryNumber = 0b1010     // Int (двоичная система)
9  let octalNumber = 0o755       // Int (восьмеричная система)
10 let hexNumber = 0xFF         // Int (шестнадцатеричная система)
11 let scientificNumber = 1.5e2 // Double (150.0)
12
13 // Автоматическое приведение в выражениях
14 let mixedCalculation = 10 + 3.14 // Double (13.14)
15 let intDivision = 10 / 3           // Int (3)
16 let doubleDivision = 10.0 / 3     // Double (3.333...)
17
18 // Приведение в массивах
19 let numbers = [1, 2, 3, 4, 5]     // [Int]
20 let mixedNumbers = [1, 2.5, 3]    // [Double] - автоматическое приведение
```

Сравнение типов с помощью is и as

```swift
// Проверка типа с помощью is
let value: Any = 42

if value is Int {
    print("Значение является целым числом")
} else if value is String {
    print("Значение является строкой")
} else if value is Double {
    print("Значение является числом с плавающей точкой")
}

// Приведение типа с помощью as
let anyValue: Any = "Hello, World!"

// Безопасное приведение (as?)
if let stringValue = anyValue as? String {
    print("Строка: \(stringValue)")
}

// Принудительное приведение (as!)
let forcedString = anyValue as! String
print("Принудительно приведенная строка: \(forcedString)")
```

Приведение типов в switch

```swift
// Приведение в switch
func processValue(_ value: Any) {
    switch value {
    case let intValue as Int:
        print("Целое число: \(intValue)")
    case let stringValue as String:
        print("Строка: \(stringValue)")
    case let doubleValue as Double:
        print("Число с плавающей точкой: \(doubleValue)")
    default:
        print("Неизвестный тип")
    }
}

processValue(42)          // "Целое число: 42"
processValue("Hello")     // "Строка: Hello"
processValue(3.14)        // "Число с плавающей точкой: 3.14"
```

Сравнение объектов

```swift
// Сравнение структур (по значению)
struct Point {
    var x: Int
    var y: Int
}

let point1 = Point(x: 1, y: 2)
let point2 = Point(x: 1, y: 2)
let point3 = Point(x: 3, y: 4)

print(point1 == point2) // true (одинаковые значения)
print(point1 == point3) // false (разные значения)

// Сравнение классов (по ссылке)
class Person {
    var name: String
    var age: Int

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }
}

let person1 = Person(name: "Анна", age: 25)
let person2 = Person(name: "Анна", age: 25)
let person3 = person1

print(person1 === person2) // false (разные объекты)
print(person1 === person3) // true (один и тот же объект)
```

Переменные в Swift

- Переменные хранят изменяемые значения
- Константы хранят неизменяемые значения
- Swift использует строгую типизацию
- Поддержка автоматического вывода типов

Объявление переменных и констант

```swift
// Константы (неизменяемые значения)
let name = "Анна"              // Тип выводится автоматически
let age: Int = 25              // Явное указание типа
let pi: Double = 3.14159       // Константа с явным типом

// Переменные (изменяемые значения)
var counter = 0                // Начальное значение
var temperature: Double        // Объявление без инициализации
temperature = 23.5             // Присваивание значения

// Множественное объявление
let x = 10, y = 20, z = 30
var a = 1, b = 2, c = 3

// Объявление с типом
let coordinates: (x: Int, y: Int) = (10, 20)
var scores: [String: Int] = ["Анна": 95, "Петр": 87]

// Пустые коллекции
var emptyArray: [Int] = []
var emptyDictionary: [String: Int] = [:]
var emptySet: Set<String> = []

// Опциональные переменные
var optionalString: String? = nil
var optionalInt: Int? = 42
```

Инициализация переменных

```
1  // Объявление с инициализацией
2  var message = "Привет, мир!"
3  var count = 0
4  var isActive = true
5
6  // Отложенная инициализация
7  var username: String
8  // username используется позже в коде
9  username = "user123"
10
11 // Инициализация массивов
12 var numbers = [1, 2, 3, 4, 5]
13 var emptyNumbers: [Int] = []
14 var initializedArray = Array(repeating: 0, count: 5) // [0, 0, 0, 0, 0]
15
16 // Инициализация словарей
17 var personInfo = ["name": "Анна", "age": "25"]
18 var emptyDict: [String: Int] = [:]
19
20 // Инициализация множеств
21 var colors: Set<String> = ["красный", "зеленый", "синий"]
22 var emptySet: Set<Int> = []
23
24 // Инициализация кортежей
25 let person = (name: "Анна", age: 25, city: "Москва")
26 let coordinates = (x: 10, y: 20)
27
28 // Инициализация структур
29 struct Point {
30     var x: Int
31     var y: Int
32 }
33 let origin = Point(x: 0, y: 0)
```

Области видимости переменных

```swift
// Глобальная область видимости
let globalConstant = "Глобальная константа"
var globalVariable = "Глобальная переменная"

func demonstrateScope() {
    // Локальная область видимости функции
    let localConstant = "Локальная константа"
    var localVariable = "Локальная переменная"

    print(globalConstant) // Доступна
    print(globalVariable) // Доступна
    print(localConstant)  // Доступна
    print(localVariable)  // Доступна

    // Вложенная область видимости
    if true {
        let nestedConstant = "Вложенная константа"
        print(nestedConstant) // Доступна
        print(localVariable)  // Доступна
    }

    // print(nestedConstant) // Ошибка! Недоступна
}

// print(localConstant) // Ошибка! Недоступна

// Область видимости в циклах
for i in 1...3 {
    let loopVariable = "Итерация \(i)"
    print(loopVariable) // Доступна только внутри цикла
}
// print(loopVariable) // Ошибка! Недоступна
```

Затенение переменных

```swift
// Затенение переменных
let name = "Глобальное имя"

func processName() {
    let name = "Локальное имя" // Затеняет глобальную переменную

    print(name) // "Локальное имя"

    // Доступ к глобальной переменной невозможен напрямую
    // Нужно использовать другое имя или self (для свойств класса)
}

// Затенение в циклах
let items = ["яблоко", "банан", "апельсин"]

for item in items {
    let item = item.uppercased() // Затеняет элемент цикла
    print(item) // "ЯБЛОКО", "БАНАН", "АПЕЛЬСИН"
}

// Затенение в замыканиях
let multiplier = 10
let numbers = [1, 2, 3, 4, 5]

let doubled = numbers.map { number in
    let multiplier = 2 // Затеняет внешнюю переменную
    return number * multiplier
}

print(doubled) // [2, 4, 6, 8, 10]
print(multiplier) // 10 (внешняя переменная не изменилась)
```

Система владения в Swift

```swift
// Value types (типы значений) - копируются
struct Point {
    var x: Int
    var y: Int
}

var point1 = Point(x: 10, y: 20)
var point2 = point1 // Создается копия

point2.x = 30
print(point1.x) // 10 (не изменилось)
print(point2.x) // 30

// Reference types (ссылочные типы) - передаются по ссылке
class Person {
    var name: String
    var age: Int

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }
}

let person1 = Person(name: "Анна", age: 25)
let person2 = person1 // Передается ссылка

person2.age = 30
print(person1.age) // 30 (изменилось!)
print(person2.age) // 30

// Сравнение ссылок
print(person1 === person2) // true (один и тот же объект)
```

Автоматическое управление памятью

```swift
// ARC (Automatic Reference Counting) в действии
class Node {
    var value: Int
    var next: Node?

    init(value: Int) {
        self.value = value
        print("Создан узел со значением \(value)")
    }

    deinit {
        print("Удален узел со значением \(value)")
    }
}

// Создание узлов
var node1: Node? = Node(value: 1)
var node2: Node? = Node(value: 2)
var node3: Node? = Node(value: 3)

// Создание циклической ссылки
node1?.next = node2
node2?.next = node3
node3?.next = node1

// Освобождение ссылок
node1 = nil // Узел не удаляется из-за циклической ссылки
node2 = nil // Узел не удаляется из-за циклической ссылки
node3 = nil // Узел не удаляется из-за циклической ссылки
```

Решение циклических ссылок

```swift
// Решение проблемы циклических ссылок с weak
class WeakNode {
    var value: Int
    weak var next: WeakNode? // Слабая ссылка

    init(value: Int) {
        self.value = value
        print("Создан слабый узел со значением \(value)")
    }

    deinit {
        print("Удален слабый узел со значением \(value)")
    }
}

// Создание узлов с weak ссылками
var weakNode1: WeakNode? = WeakNode(value: 1)
var weakNode2: WeakNode? = WeakNode(value: 2)
var weakNode3: WeakNode? = WeakNode(value: 3)

// Создание циклической ссылки с weak
weakNode1?.next = weakNode2
weakNode2?.next = weakNode3
weakNode3?.next = weakNode1

// Освобождение ссылок
weakNode1 = nil // Узел удаляется
weakNode2 = nil // Узел удаляется
weakNode3 = nil // Узел удаляется
```

Передача владения в функциях

```swift
// Передача по значению (копирование)
func processPoint(_ point: Point) -> Point {
    var newPoint = point // Создается копия
    newPoint.x += 10
    return newPoint
}

let originalPoint = Point(x: 5, y: 10)
let modifiedPoint = processPoint(originalPoint)

print(originalPoint.x) // 5 (не изменилось)
print(modifiedPoint.x) // 15

// Передача по ссылке (inout)
func modifyPoint(_ point: inout Point) {
    point.x += 10 // Изменяет оригинальную переменную
}

var mutablePoint = Point(x: 5, y: 10)
modifyPoint(&mutablePoint)
print(mutablePoint.x) // 15 (изменилось)

// Передача ссылочных типов
func updatePerson(_ person: Person) {
    person.age += 1 // Изменяет оригинальный объект
}

let person = Person(name: "Анна", age: 25)
updatePerson(person)
print(person.age) // 26 (изменилось)

// Передача владения с помощью @escaping
func delayedExecution(completion: @escaping () -> Void) {
    DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
        completion() // Замыкание выполняется позже
```

Ключевые моменты

- **Типы данных** — основа типобезопасности Swift
- **Примитивные типы** обеспечивают эффективность
- **Пользовательские типы** позволяют моделировать предметную область
- **Операции над типами** включают арифметические, логические и строковые
- **Приведение типов** может быть явным и неявным
- **Сравнение типов** выполняется с помощью `is` и `as`
- **Области видимости** определяют доступность переменных
- **Система владения** управляет памятью автоматически

Практические рекомендации

- Используйте `let` для неизменяемых значений
- Предпочитайте явное указание типов для публичных API
- Избегайте принудительного приведения типов (`as!`)
- Используйте `weak` для предотвращения циклических ссылок
- Применяйте `inout` только когда необходимо изменить параметр
- Следите за областями видимости переменных

Спасибо за внимание!

Вопросы?