ЛЕКЦИЯ 4

ОСНОВЫ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ:SWIFT И KOTLIN
ОСНОВЫ ООП В SWIFT И KOTLIN

1. Основные концепции и синтаксис Swift

1.1 Переменные и константы:

В Swift используются var для объявления переменных и let для констант.

```
var variableName = "Это переменная"
let constantName = "Это константа"
```

1.2 Типы данных:

Основные типы - Int, Double, String, Bool.

```
var integer: Int = 10
var double: Double = 10.5
var string: String = "Hello, Swift!"
var boolean: Bool = true
```

1.3 Управляющие конструкции:

Условные операторы (if, else, switch) и циклы (for-in, while).

```
if boolean {
    print("Истина")
} else {
    print("Ложь")
}

for i in 1...5 {
    print(i)
}
```

1.4 Опциональные типы:

Понятие опциональности с помощью ? и !.

```
var optionalString: String? = nil
optionalString = "Optional now has a value"
```

1.5 Функции:

Объявление и вызов функций, параметры, возвращаемые значения.

```
func greet(name: String) -> String {
    return "Hello, \(name)!"
}
print(greet(name: "Swift"))
```

2. Основные концепции и синтаксис Kotlin

2.1 Переменные:

Используйте var для объявления переменных и val для объявления неизменяемых ссылок или "констант".

```
var variableName = "Это переменная"
val constantName = "Это константа (val)"
```

2.2 Типы данных:

Основные типы - Int, Double, String, Boolean.

```
var integer: Int = 10
var double: Double = 10.5
var string: String = "Hello, Kotlin!"
var boolean: Boolean = true
```

2.3 Управляющие конструкции:
Условные операторы (if, when) и циклы (for, while).

```
if (boolean) {
    println("Истина")
} else {
    println("Ложь")
}

for (i in 1..5) {
    println(i)
}
```

2.4 Null безопасность:
Один из ключевых аспектов Kotlin - безопасная работа с null значениями через ?, !! и ?:.

```
var nullableString: String? = null
nullableString = "Now has a value"
```

2.5 Функции:
Объявление и вызов функций, параметры, возвращаемые значения и лямбда-выражения.

```
fun greet(name: String): String {
    return "Hello, $name!"
}
println(greet(name = "Kotlin"))
```

Как видно из примеров, Swift и Kotlin имеют множество схожих конструкций и синтаксических особенностей. Это делает их относительно простыми для изучения, особенно если вы уже знакомы с одним из этих языков.

Хотя Swift и Kotlin разрабатывались независимо друг от друга для разных платформ, у них есть множество схожих черт и концепций, что делает процесс перехода от одного языка к другому более гладким для разработчиков. Оба языка предлагают современные конструкции и возможности, делая акцент на краткости, читаемости и безопасности кода.

Основы ООП в Swift и Kotlin

Объектно-ориентированное программирование (ООП) — методология разработки программ, где основными концепциями являются понятия объектов и классов. Swift и Kotlin, оба предоставляют широкие возможности для ООП.

1. Концепции ООП

1.1 Классы и объекты:

Класс — это "чертёж" или "шаблон", используемый для создания объектов. Объект — это экземпляр класса.

1.2 Наследование:

Одни классы могут наследоваться от других, приобретая их свойства и методы.

1.3 Инкапсуляция:

Сокрытие деталей реализации, предоставляя публичный интерфейс.

1.4 Полиморфизм:

Способность объекта использовать методы производного класса, который он не знает.

2. ООП в Swift

2.1 Классы и объекты:
Объявление классов с помощью class и создание объектов через конструктор.

2.2 Наследование:
Используйте : для наследования от базового класса.

2.3 Инкапсуляция:
public, private, internal и другие модификаторы доступа.

2.4 Полиморфизм:
Осуществляется через наследование и переопределение методов с помощью ключевого слова override.

Определение класса и инициализация:

```
class Animal {
    var name: String

    init(name: String) {
        self.name = name
    }

    func sound() {
        print("Some generic animal sound")
    }
}

let dog = Animal(name: "Dog")
dog.sound()
```

Работа с опционалами:

```
var optionalVar: String? = "Hello"
if let unwrapped = optionalVar {
    print(unwrapped)
} else {
    print("The variable is nil")
}
```

Замыкания:

```
let numbers = [1, 2, 3, 4, 5]
let squared = numbers.map { $0 * $0 }
print(squared)
```

3. ООП в Kotlin

3.1 Классы и объекты:
Объявление классов с помощью class и создание объектов через конструктор.

3.2 Наследование:
В Kotlin все классы по умолчанию final, для возможности наследования используйте open.

3.3 Инкапсуляция:
Модификаторы public, private, protected, и internal.

3.4 Полиморфизм:
Осуществляется через наследование и переопределение методов с помощью ключевого слова override.

Определение класса и инициализация:

```
class Animal(val name: String) {

fun sound() {
        println("Some generic animal sound")
    }
}

val cat = Animal("Cat")
cat.sound()
```

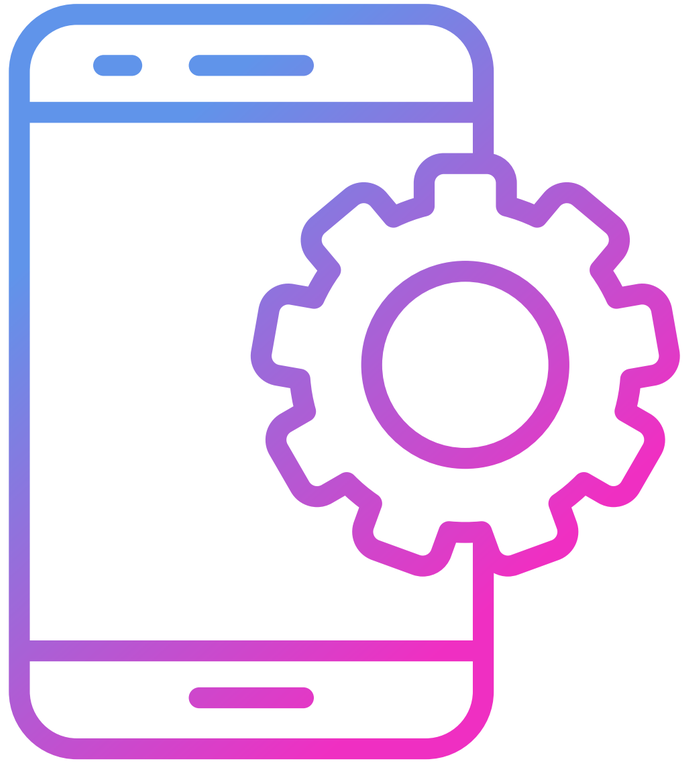
Работа с null-безопасностью:

```
var nullableVar: String? = "Hello"
nullableVar?.let {
    println(it)
} ?: println("The variable is null")
```

Расширения:

```
fun String.shout() = this.toUpperCase() + "!!!"
println("hello".shout())
```

ООП предоставляет мощные инструменты для структурирования кода, делая его более модульным, повторно используемым и легко расширяемым. Swift и Kotlin, оба предлагают современные механизмы для реализации концепций ООП, обеспечивая эффективную и безопасную разработку мобильных приложений.



Kotlin Programming Language

Kotlin is a programming language that makes coding concise, cross-platform, and fun. It is Google’s preferred language for Android app development.

Kotlin

Контрольные вопросы

1. Что такое опционалы в Swift? Какие два основных способа их развертывания?
2. Что такое замыкания, и какова их особенность в Swift?
3. Как в Swift определяются и используются перечисления (enum)?
4. Что такое расширения (extensions) в Swift, и для чего они используются?
5. Какова роль протоколов (protocol) в Swift, и как они отличаются от интерфейсов в других языках?
6. Какие основные коллекции доступны в Swift и как они отличаются друг от друга?
7. Что такое null-безопасность в Kotlin? Как работает оператор ?.?
8. Как в Kotlin создать класс с первичным конструктором?
9. Какова особенность использования when в Kotlin, и как он отличается от switch в других языках?
10. Что такое корутины в Kotlin, и в каких случаях они могут быть полезными?
11. Как в Kotlin определяются и используются расширения (extensions)?
12. Что такое "data классы" в Kotlin и для чего они нужны?
13. Каковы преимущества использования листов (list), множеств (set), и карт (map) в Kotlin?