**1. Функции в Kotlin**

В Kotlin есть несколько способов написания функций:

Обычная функция: определяется ключевым словом fun, за которым следует имя функции, список параметров в скобках и тело функции.

fun greet(name: String) {

println("Hello, $name!")

}

**Однострочная функция (выражение):** если тело функции состоит из одного выражения, можно использовать однострочный синтаксис без ключевого слова return.

fun square(x: Int): Int = x \* x

**Функциональные аргументы и значения:** Функции могут принимать другие функции в качестве аргументов или возвращать их в качестве результата. Это позволяет создавать более гибкий и мощный код.

Множество аргументов: для передачи переменного числа аргументов используется vararg.

fun printNumbers(vararg numbers: Int) {

numbers.forEach { println(it) }

}

**Один аргумент:** если функция принимает только один аргумент, скобки можно опустить при вызове.

fun printNumber(number: Int) = println(number)

printNumber(42)

**2. Структура описания класса в Kotlin**

Классы в Kotlin определяются ключевым словом class. Внутри класса можно определить свойства (var для изменяемых и val для неизменяемых), методы и инициализаторы.

**Переменные и значения:** var используется для объявления изменяемых свойств, а val - для неизменяемых.

**Геттеры и сеттеры:** можно переопределить стандартные геттеры и сеттеры для свойств.

**Дата-классы:** Используются для хранения данных, автоматически генерируются методы equals(), hashCode() и toString().

**3. Перечисления и конструкция when**

Перечисления (enum) используются для определения типа с ограниченным набором значений. Конструкция when является аналогом switch-case в других языках программирования, но более мощной благодаря поддержке различных условий и отсутствию необходимости явного указания break.

**4. Исключения в Kotlin**

Исключения в Kotlin используются для обработки ошибок во время выполнения программы. Они создаются с помощью ключевых слов throw и try/catch.

**5. Работа с массивами и списками в Kotlin**

Массивы и списки являются основными структурами данных для хранения элементов. Массивы имеют фиксированный размер, в то время как списки динамические.

**6. Работа с множествами и ассоциативными массивами в Kotlin**

Множества (Set) используются для хранения уникальных элементов, а ассоциативные массивы (Map) - для хранения пар ключ-значение.

**7. Общая структура классов коллекций в Kotlin**

Коллекции в Kotlin представлены различными классами, такими как List, Set, Map, которые предоставляют общие методы для работы с данными.

**8. Наследование и делегация в Kotlin**

Наследование позволяет создавать новые классы на основе существующих, перенимая их свойства и методы. Делегация используется для реализации интерфейсов через другие объекты.

**9. Функции расширения и локальные функции в Kotlin**

Функции расширения позволяют добавлять новые функции к существующим классам без изменения их исходного кода. Локальные функции - это функции, объявленные внутри других функций.

**10. Регулярные выражения в Kotlin**

Регулярные выражения используются для поиска и замены текста по определенному шаблону.

**11. Интерфейсы и полиморфизм в Kotlin**

Интерфейсы определяют контракты для классов, позволяя реализовать полиморфизм.

**12. Общий обзор наследования, абстрактных классов, интерфейсов и их реализации в Kotlin**

Kotlin поддерживает наследование, абстрактные классы и интерфейсы, что позволяет создавать гибкую и масштабируемую архитектуру.

**13. Модификаторы доступа и запечатанные классы в Kotlin**

Модификаторы доступа контролируют видимость классов и членов класса. Запечатанные классы ограничивают возможность наследования.

**14. Способы задания конструкторов в Kotlin**

Конструкторы в Kotlin могут быть первичными и вторичными. Первичный конструктор объявляется в заголовке класса, вторичные - внутри тела класса.

**15. Структура классов и сравнение объектов в Kotlin**

Классы в Kotlin могут содержать свойства, методы и инициализаторы. Сравнение объектов осуществляется с помощью методов equals() и hashCode().

**16. Совместное объявление класса и экземпляра в Kotlin**

С помощью объектов можно объявить класс и его экземпляр одновременно, используя ключевые слова object и companion object.

**17. Лямбда-выражения и замыкания в Kotlin**

Лямбда-выражения позволяют создавать анонимные функции, которые могут быть переданы в качестве аргументов другим функциям. Замыкания сохраняют состояние окружающей области видимости.

**18. Методы работы с коллекциями, принимающие функции, как аргумент**

Методы filter, map, count, sort, max позволяют выполнять различные операции над коллекциями, используя лямбда-выражения.

**19. Методы работы с коллекциями, принимающие функции, как аргумент**

Методы groupBy, reduce, fold предназначены для агрегации и трансформации данных в коллекциях.

**20. Работа с последовательностями в Kotlin и Java**

Последовательности в Kotlin представляют собой лениво вычисляемые коллекции, что позволяет эффективно работать с большими объемами данных.

**21. Лямбда-выражения с получателями**

Лямбда-выражения с получателями позволяют создавать DSL (Domain-Specific Language).

**22. Поддержка значения null в Kotlin**

Kotlin имеет строгую систему типов, которая отличает nullable и non-nullable типы, помогая избежать NullPointerException.

**23. Базовые типы в Kotlin**

Kotlin поддерживает базовые типы данных, такие как Int, Double, Boolean, а также строки и символы.

**24. Перегрузка арифметических операторов в Kotlin**

Перегрузка операторов позволяет определить собственное поведение для стандартных арифметических операторов в пользовательских классах.