The Android Prep Lab (Kotlin)

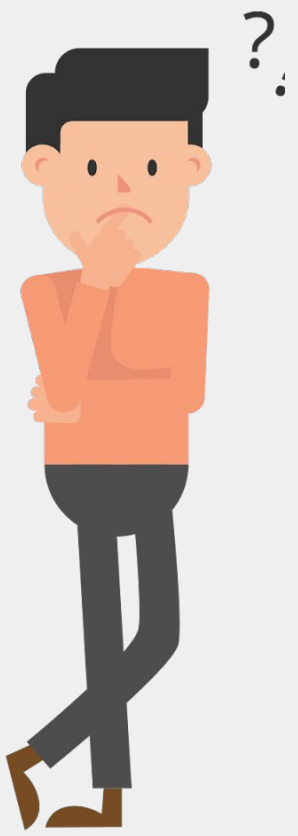
Part - 3 of 10
Kotlin Basics and Advanced Topics

What are the key features of Kotlin?

- Concise and expressive syntax.
- Null safety.
- Interoperability with Java.
- Extension functions.
- Coroutines for asynchronous programming.
- Type inference.

Explain the difference between val and var

- val (immutable): Read-only property. Its value cannot be reassigned once set.
- var (mutable): Mutable property. Its value can be changed.

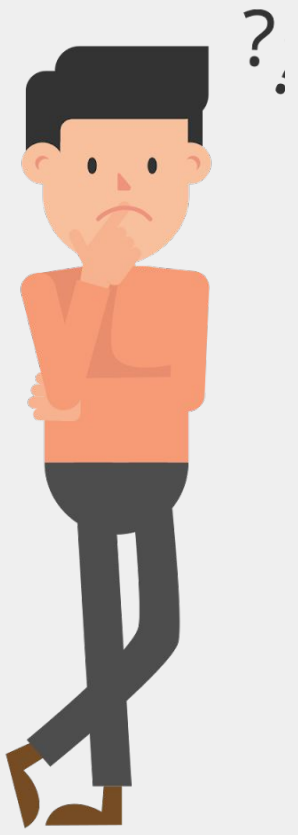


```
val name = "John" // Cannot be reassigned

var age = 25      // Can be reassigned
```

What is a data class in Kotlin? Why use it?

A data class is a special class in Kotlin used to hold data. It automatically generates methods like toString(), equals(), hashCode(), and copy().



```
data class User(val id: Int, val name: String)

val user = User(1, "Alice")
```

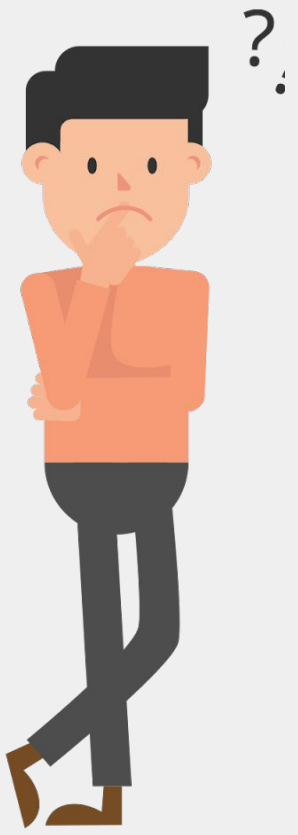
What is null safety in Kotlin? How does it work?

Kotlin helps avoid NullPointerException by distinguishing nullable (String?) and non-nullable (String) types.

- Use ?. for safe calls.
- Use !! to explicitly assert non-null.

```
val name: String? = null

println(name?.length) // Safe call
```

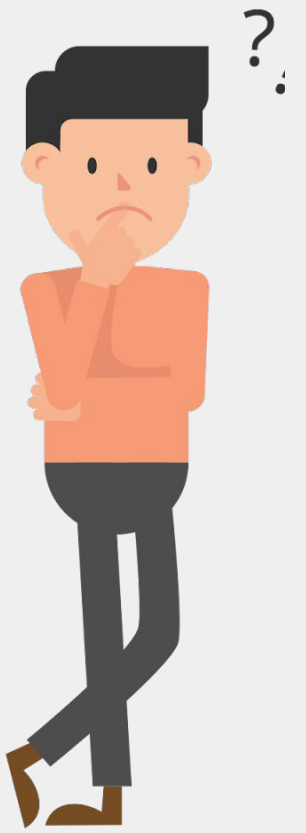


What are higher-order functions in Kotlin? Provide an example

Higher-order functions take functions as parameters or return functions.

```kotlin
fun higherOrder(func: (Int) -> Int): Int {
    return func(5)
}

val result = higherOrder { it * 2 } // Pass lambda
println(result) // Output: 10
```



Explain extension functions in Kotlin with an example.

Extension functions add functionality to existing classes.

```
fun String.greet(): String = "Hello, $this!"
println("Alice".greet()) // Output: Hello, Alice!
```

What are Kotlin lambdas? How do you use them?

A lambda is an anonymous function.

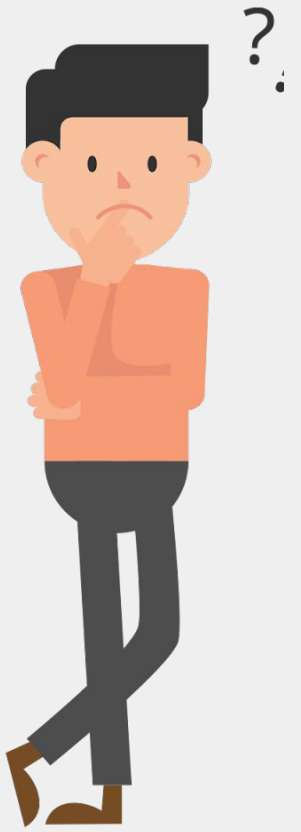
Syntax: { parameters -> body }.

```
val add = { x: Int, y: Int -> x + y }
println(add(3, 4)) // Output: 7
```

What are coroutines in Kotlin? Why use them?

Coroutines provide lightweight, non-blocking threading. They help write asynchronous code that's easier to read and maintain.

```
GlobalScope.launch {
    delay(1000L)
    println("Hello from Coroutine!")
}
```

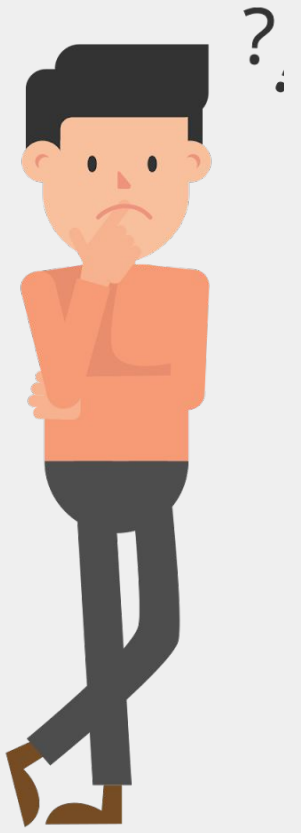


Explain the difference between launch and async in Kotlin coroutines

launch: Does not return a result. Used for fire-and-forget tasks.

async: Returns a Deferred result. Used for computations requiring a value.

```
val result = async { compute() }
println(result.await())
```

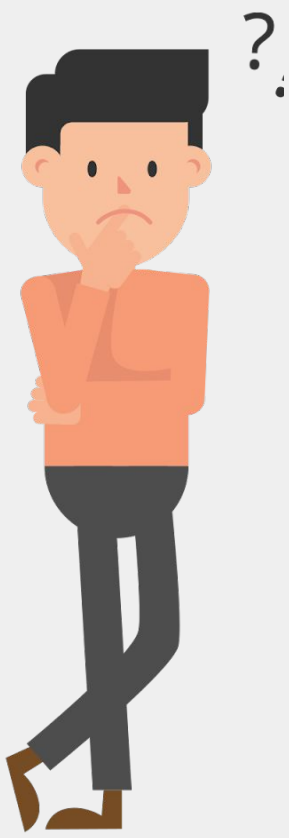


What is the apply scope function in Kotlin? How is it different from let?

apply: Works with the object itself. Returns the object.

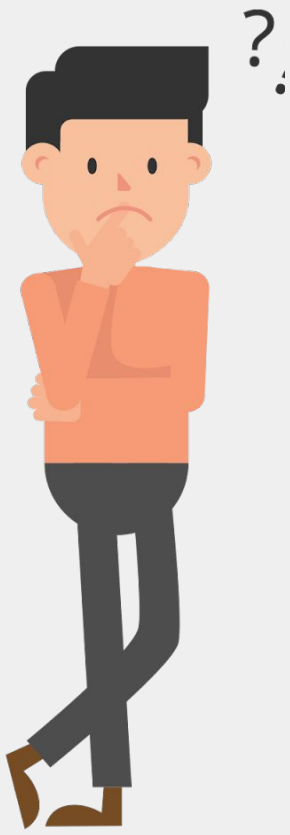
let: Used to operate on a nullable object and returns the last expression.

```
val person = Person().apply {
    name = "John"
    age = 30
}
```



What is the difference between sealed classes and abstract classes?

- **Sealed classes**: Restrict subclassing to the same file. Useful for representing restricted hierarchies (e.g., state handling).
- **Abstract classes**: Can be subclassed from anywhere.

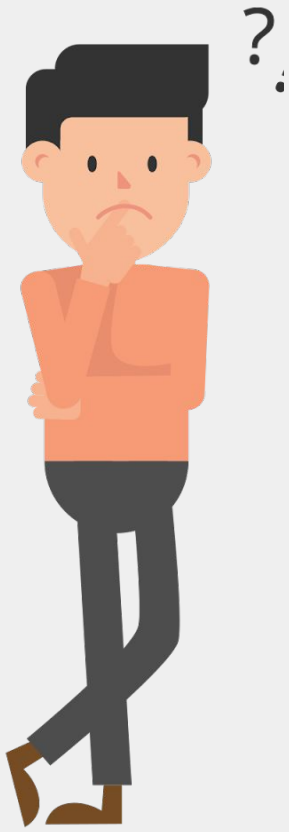


```
sealed class Result {
    data class Success(val data: String) : Result()
    object Failure : Result()
}
```

What is the use of inline functions in Kotlin?

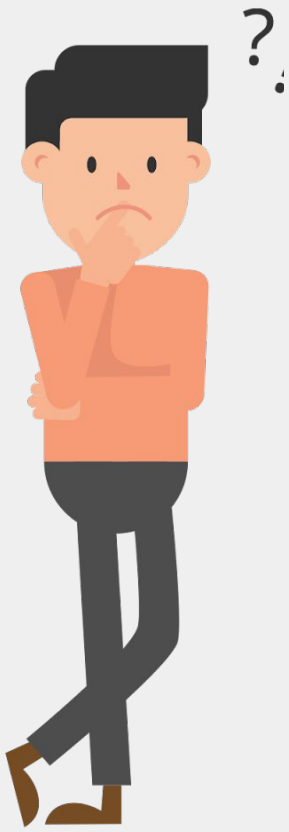
Inline functions improve performance by inlining the code, reducing the overhead of function calls.

```kotlin
inline fun calculate(operation: () -> Unit) {
    operation()
}
```



What are suspend functions in Kotlin? Why are they used?

A suspend function is a coroutine function that can be paused and resumed. Used for long-running tasks like network calls.



```
suspend fun fetchData(): String {
    delay(1000)
    return "Data"
}
```

How does Kotlin handle exceptions? How is it different from Java?

Kotlin uses try-catch-finally like Java. Unlike Java, all exceptions are unchecked, meaning you don’t need to declare them.

```
try {
    val result = 10 / 0
} catch (e: ArithmeticException) {
    println("Error: ${e.message}")
}
```

Explain the difference between with and run scope functions in Kotlin

with: Operates on an object and returns the last expression.

run: Works like with, but can be used as an extension function.

```
val person = with(Person()) {
    name = "Alice"
    age = 25
    this
}
```

What are companion objects in Kotlin? Why use them?

Companion objects act as static members of a class, enabling access without creating an instance.

```
class Utils {
    companion object {
        fun greet() = "Hello!"
    }
}

println(Utils.greet()) // Output: Hello!
```

What is the difference between == and === in Kotlin?

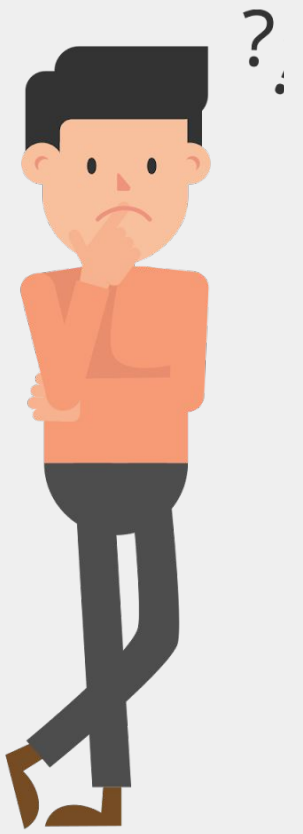
- `==`: Compares values (structural equality).
- `===`: Compares references (referential equality).

```kotlin
val a = "hello"
val b = "hello"
println(a == b) // true
println(a === b) // false
```

Explain destructuring declarations in Kotlin with an example.

- Destructuring lets you extract multiple values from an object.

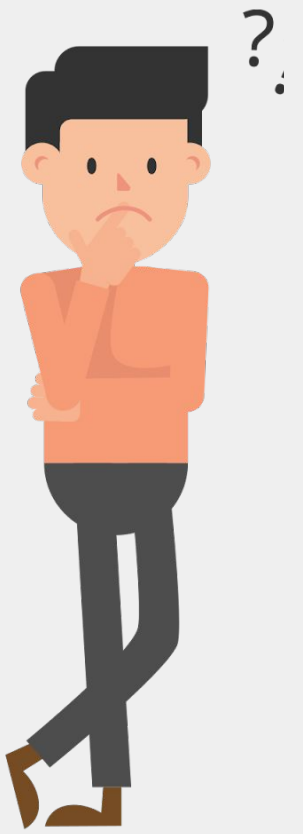
```
data class Point(val x: Int, val y: Int)
val (x, y) = Point(10, 20)
println("$x, $y") // Output: 10, 20
```



How does Kotlin handle default arguments in functions?

- Kotlin allows default values for function parameters, reducing method overloading.

```
fun greet(name: String = "Guest") = "Hello, $name!"
println(greet()) // Output: Hello, Guest!
```



What is typealias in Kotlin? How is it useful?

- typealias provides an alternate name for a type, improving readability.

```
typealias StringMap = Map<String, String>
val myMap: StringMap = mapOf("key" to "value")
```

https://www.linkedin.com/in/pavankreddy92/

Follow for More