**YouTube Series:**

<https://www.youtube.com/watch?v=NosAkIKgA4Y&list=PLRKyZvuMYSIMW3-rSOGCkPlO1z_IYJy3G>

<https://www.youtube.com/watch?v=HfkyXuZdD_c&list=PLRKyZvuMYSIO0jLgj8g6sADnD0IBaWaw2&index=5t>

Interview Questions : <https://www.tealhq.com/interview-questions/android-developer>

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Features of Kotlin :**

* Statically typed language – Type checking will happen at compile time.
* Object oriented and functional language
* Fully interoperable with JAVA
* Cosine, Safe, and Powerful – **Safe** in the sense there are less number of chances to get runtime errors as Kotlin provide this kind of features like Null safety, Class cast exceptions. **Cosine** means the line of code is less as compared to java so that you write your logics easily,
* Open source

**Range in Kotlin :**

1. Using “..” operator – in this feature the upper bound value will considered.

e.g. number = 2

result = number in 1..9

In this case 9 will be considered

1. Using “Until” keyword – here the upper bound value will not be considered

e.g. number = 2

result = number in 1 until 9

In this case 9 will not be considered

1. Using “downTo” keyword – the numbers will gets print in reverse order

e.g. number = 2

result = number in 1 downTo 9

Now the output will be 9,8,7,6,5,4,3,2,1

**For loop :**

In for loop we use the range and steps.

* for(i in 1..9) – In this case the controller will print digits from 1 to 9.
* for(i in 1..9 step 2) – Now as we have define step as 2 so the controller will increment the range value every time by 2. The output will be 1,3,5,7,9

**Store functions in variables :**

In Kotlin we can store the function in a variable and then later on we can use the same variable to call the function. To do that we have to use “::” operator followed by function name.

e.g. - fun main(){

var Fn = ::addition

println(Fn(2, 4)) //Function call using variable

}

fun addition(a:Int, b:Int): Int {

return a + b

}

**Room Database :**

* Provide an abstraction layer over SQLite database
* Less number of lines code
* SQL queries get verifies at compile time
* Support to different libraries like **coroutines**, **live data**, Rx Java
* The interaction of SQLite becomes easy
* First create an entity (table using dataclasses).

*@Entity(tablename = “TableName”)*

*data class TableName{*

*val Columns names: datatype*

*.*

*.*

}

* Create Dao’s for CRUD operations



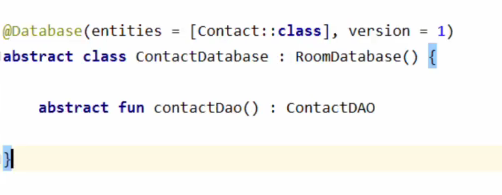
* Important\*\* – In one application there must be only one database instance. To achieve this, you typically use a singleton pattern to create and manage the database instance. By using the **getInstance()** method, you must ensure that only one instance of DatabaseClient (and thus AppDatabase) is created and used throughout your application.
* Define Database Class - Create an abstract class that extends `RoomDatabase` to define the database instance and include the entities and DAOs.

*@Database(entities = [ClassName::class], version = 1)*

*abstract class AppDatabase : RoomDatabase() {*

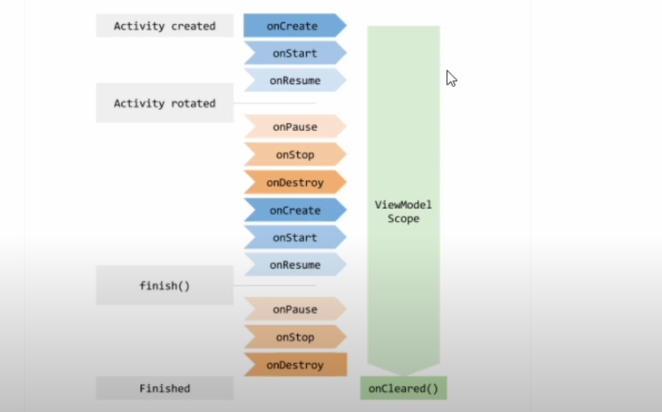
*abstract fun FunctionName(): DaoInterfaceName*

*}*



**ViewModel class :**

* A model class which holds your UI specific data so that whenever the configurations (rotating the screen, hiding the keyboard) of the app changes the data remains as it is.
* In short, it’s a model for your views like activities and fragments.
* View Models are lifecycle aware means View Model remains in a memory till the activity or fragment fully destroyed.



* Views will show the data and Models will hold the data.

**LiveData :**

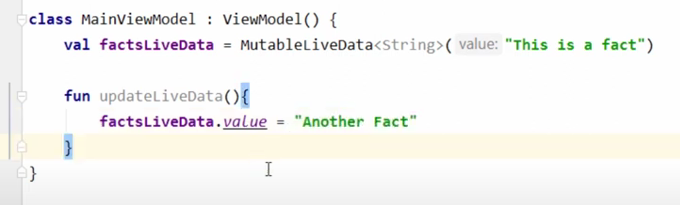
* LiveData is an observable data holder class used to hold the data with an additional feature as you can observe the live changes which are happening in data.
* Lifecycle aware – the updates of changes in a data will only send to the components which are in their active state, This prevents memory leaks and ensures that UI updates are only performed when the UI componant is visible.
* There are 2 types : **Mutable LiveData** – this stored data can be changed , **LiveData** – the data cannot be changed.
* Observe the LiveData :



A screenshot of a computer

AI-generated content may be incorrect.

* Update the LiveData : we have to use the “value” property to update the LiveData.



* Call the Update function from main activity.

**Data Binding :**

* It is a support library which is a part of Jetpack Compose.
* In this data binding, we will bind the UI components (like TextView, ImageView) with data sources

A screenshot of a computer screen

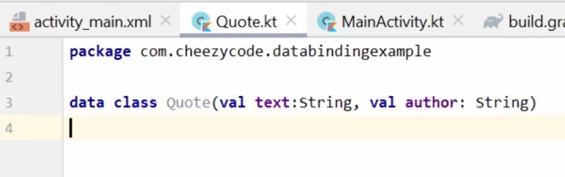
AI-generated content may be incorrect.

* It allows the declarative format instead of programmatically.
* Because of this concept we don’t have to use the findViewById as its an expensive operation which will help to improve the performance of the app.
* The binding of the data is takes place at compile time so the Null Pointer Exception will be avoided at runtime.
* Benefits in overall application performance and no memory leakage.
* To implement Data Binding we need to enable its property in build.gridle file.

A blue and white sign

AI-generated content may be incorrect.

* By adding variables :



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Binding Adapters :**

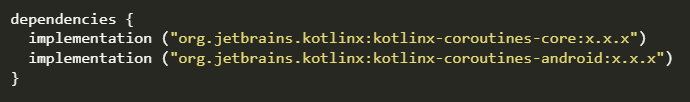
* Refer the video : (<https://www.youtube.com/watch?v=XfdaOpQ4UbI&list=PLRKyZvuMYSIO0jLgj8g6sADnD0IBaWaw2&index=9>)
* Binding adapters allows us to write a custom logic with the help of data binding libraries so that whenever the value of attribute will set this custom code/logic will be executed.
* when you need a middle man between data binding and the view to do some translation from the type passed to the binding adapter to a different type that is passed to the view we use custom setters or binding adapters.

**MVVM design pattern :**

* MVVM (Model-View-ViewModel) is a design pattern that separates the UI, business logic, and data layers in an Android app.
* The ViewModel acts as a mediator between the Model and View, handling user interactions and updating the UI.
* Data binding is often used to connect the ViewModel with the View.

**Kotlin Coroutines :**

* Kotlin Coroutines are a way to perform asynchronous programming in Android.
* Android is a single thread platform, By default, everything runs on the main thread. In Android, almost every application needs to perform some non UI operations like (Network call, I/O operations), so when coroutines concept is not introduced, what is done is that programmer dedicate this task to different threads, each thread executes the task given to it, when the task is completed, they return the result to UI thread to update the changes required.
* Though In android there is a detailed procedure given, about how to perform this task in an effective way using best practices using threads, this procedure includes lots of callbacks for passing the result among threads, which ultimately introduce tons of code in our application and the waiting time to get the result back to increases.
* On Android, every app has a main thread which handles all the UI operations like drawing views and other user interactions. If there is too much work happening on this main thread, like network calls (eg fetching a web page), the apps will appear to hang or slow down leading to poor user experience.
* Coroutines increases the amount of work that an app can execute by allowing tasks to execute in parallel.
* This makes sure that heavy tasks run in the background, keeping the UI thread responsive and improving the overall user experience.
* Coroutines defines as “lightweight threads”. Coroutines were added to Kotlin in version 1.3.



**Content Providers :**

* The role of the content provider in the android system is like a central repository in which data of the applications are stored, and it facilitates other applications to securely access and modifies that data based on the user requirements.
* Android system allows the content provider to store the application data in several ways. Users can manage to store the application data like images, audio, videos, and personal contact information by storing them in SQLite Database , in files , or even on a network . In order to share the data, content providers have certain permissions that are used to grant or restrict the rights to other applications to interfere with the data.

