Android Studio

⇒ Integrated Development Environment
[{Editor, debugger, (build system), version control}
{emulator}]

(build system) ⇓ Makefile (gradle for Android)

{emulator} ⇓ Android SDK

→ running on actual Smartphone is challenging

Emulator ⇒ so that a program can be tested or run within the computer itself

{Simulation, emulation & Testbed}

Simulation ⇓ where we utilize a model to decide how the system behaves

emulation → (more scalable than a testbed) we model the running of the hardware, and our system is evaluated by running on that hardware

Testbed → actual hardware

⇓

Can be slow,

(A personal machine has a very different architecture to a smartphone's architecture.)

During emulation, every instruction that is generated has to be re-converted into the desktop machine's instruction.

To speed it, emulate on x86 smartphone.

User Interface: ⇒ Speified by using {composable functions}

⇓

properties/limitations

⇓

[does not describe the sequence of steps to create UI; instead describes the UI itself]

{@Composable functions inside} ⇒ Should not have a {state}

⇓

Information about the status of other variables, which in turn can store prior user actions, etc.

⇓

When to execute these functions is decided

automatically, and not
by any other function.

What is the theme and additional UI design-technique used?

⇒ Google's specifications of theme ⇒ {Material}
⇓
If you want to change the look-and-feel, you only need to change a few attributes and the other attributes would be selected automatically based on the hardcoded attribute.

Arranging the UI elements

1) Change one attribute to affect other attributes

2) Modifiers to add spaces, etc.

3) You can use Row and Column classes to effectively place the individual UI elements.

```
fun myApp (Modifier modifier) {
    Column (modifier) {
        Text ("First row")
        Text ("Second row")
    }
}
```

One approach of dynamic content is that you send a suitable argument.

Changing This would automatically Trigger the composable function.

Recomposition

Changing the modifiers ⇒ how much space to retain; by making this as a function argument; we can keep readjusting depending on the content.

All the UI elements need not be created in code. All the static UI elements can be defined in XML files.

To modify the UI elements,

① Load the UI element from the Kotlin program.

② Utilize within some composable function to add the changes needed to each element.

```
layout = loadXML("layout.xml")
Text1  = fetchUIElementByAttribute("Text") ✓
Text1.setProperty(" . .. ") ✓
```

Behavior is not affected by user actions. When a user takes some action, we change a variable.

Consider a UI element like a button that needs to respond to such changes. Where would you include this?

Check the Wiki entry of Lambda functions.

Concept of Mutable and Immutable Variables

Mutable : ⇒ The content within a data might be changed even after it is assigned to a variable. Example ⇒ Lists and dictionaries (in python) are mutable; whereas tuples are immutable.

⇓

Need to recreate the tuple from scratch.

Java ⇒ Basic data types are immutable.

remember and mutable State Of are classes defined by the Android framework with the ability to manage the state.

} How the UI elements can remember the user actions.